MPC5676R Microcontroller Reference Manual

Devices Supported:

MPC5676R

MPC5676RRM
Rev 5
9/2012

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: http://www.reg.net/v2/webservices/Freescale/Docs/TermsandConditions.htm

Freescale, the Freescale logo, AltiVec, C-5, CodeTest, CodeWarrior, ColdFire, C-Ware, Energy Efficient Solutions logo, Kinetis, mobileGT, PowerQUICC, Processor Expert, QorIQ, Qorivva, StarCore, Symphony, and VortiQa are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Airfast, BeeKit, BeeStack, ColdFire+, CoreNet, Flexis, MagniV, MXC, Platform in a Package, QorIQ Qonverge, QUICC Engine, Ready Play, SafeAssure, SMARTMOS, TurboLink, Vybrid, and Xtrinsic are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.
© 2012 Freescale Semiconductor, Inc.

Document Number: MPC5676R
Rev 5
9/2012

Chapter 1
Device Overview

Chapter 2
Signal Descriptions

Chapter 3
System Integration Units (SIU, SIU_B)

Chapter 4
System Information Module

Chapter 5
Resets

Chapter 6
Boot Assist Module (BAM)

Chapter 7
Clocking

Chapter 8
Frequency Modulated Phase-Locked Loop (FMPLL)

Chapter 9
Power Management Controller (PMC)

Chapter 10
Core

Chapter 11
AMBA Crossbar Switch (XBAR)

Chapter 12
Cyclic Redundancy Checker (CRC) Unit

Chapter 13
Debug

Chapter 14
Decimation Filter

Chapter 15
Deserial Serial Peripheral Interface (DSPI)

Chapter 16
Development Trigger Semaphore (DTS)

Chapter 17
Enhanced Direct Memory Access Controller (eDMA)

Chapter 18
Enhanced Modular Input/Output Subsystem (eMIOS200)

Chapter 19
Enhanced Queued Analog-to-Digital Converter (eQADC)

Chapter 20
Enhanced Serial Communications Interface (eSCI)

Chapter 21
Enhanced Time Processing Unit (eTPU)

Chapter 22
Error Correction Status Module (ECSM)

Chapter 23
External Bus Interface (EBI)

Chapter 24
Flash Memory Array and Control

Chapter 25
FlexCAN Module

Chapter 26
FlexRay Module (FlexRay)

Chapter 27
Interrupts and Interrupt Controller (INTC)

Chapter 28
Memory Protection Unit (MPU)

Chapter 29
Periodic Interrupt Timer (PIT_RTI)

Chapter 30
Peripheral Bridge (PBRIDGE)

Chapter 31
Self-Test Control Unit (STCU)

Chapter 32
Semaphores

Chapter 33
Software Watchdog Timers (SWT_A, SWT_B)

Chapter 34
Shared Time Angle Counter Bus (STAC)

Chapter 35
System RAM (SRAM)

Chapter 36
System Timer Module (STM)

Chapter 37
Temperature Sensor

Appendix A
Revision History

Chapter 1
Device Overview

The MPC5600 family of devices is closely compatible with the MPC5500 families, while introducing new features coupled with high performance CMOS technology to provide substantial reduction of cost per feature and significant performance improvement. This document describes the features of the MPC5676R and highlights important electrical and physical characteristics of this device.

The two e200z7 host processor cores of the MPC5676R are compatible with the Power Architecture® Book E architecture. They are 100% user-mode compatible (with floating point library) with the classic PowerPC instruction set. The Book E architecture has enhancements that improve the architecture's fit in embedded applications. In addition to the standard and VLE Power Architecture instruction sets, this core has additional instruction support for digital signal processing (DSP).

The MPC5676R has two levels of memory hierarchy; separate 16 K instruction and 16 K data caches for each of two cores and 384 KB of on-chip SRAM. 6 MB of internal flash memory is provided. An external bus interface is also available for special packaged parts to support application development and calibration.

1.1 Features

This section compares the MPC5676R to the MPC5500 and MPC5600 family of devices, shows a block diagram of the device, and describes the features of the MPC5676R.

1.1.1 MPC5500 and MPC5600 Family Comparison

Table 1-1. MPC5500/MPC5600 Family Comparison

Feature	MPC5554	MPC5565	MPC5567	MPC5566	MPC5674F	MPC5676R
Process	130 nm	130 nm	130 nm	130 nm	90 nm	90 nm
Core	z6	z6	z6	z6	z7	z7[1]
Number of Cores	1	1	1	1	1	2
Single Precision Floating Point	Yes	Yes	Yes	Yes	Yes	Yes
SIMD	Yes	Yes	Yes	Yes	Yes	Yes
VLE	No	Yes	Yes	Yes	Yes	Yes
Cache	32KB Unified	8KB Unified	8KB Unified	32KB Unified	16KB Instruction + 16KB Data	16KB Instruction + 16KB Data
Non-maskable Interrupt	No	No	No	No	NMI, Critical	NMI, Critical

Table 1-1. MPC5500/MPC5600 Family Comparison

Feature	MPC5554	MPC5565	MPC5567	MPC5566	MPC5674F	MPC5676R
MMU Entries	32	32	32	32	64	32
MMU Tool Control	No	No	No	No	No	Yes
MPU	No	No	No	No	Yes	Yes
Semaphores	No	No	No	No	No	16
CRC Channels	No	No	No	No	No	3
Software Watchdog Timer	No	No	No	No	1	2
Core Nexus Class	3+	3+	3+	3+	3+	3+
SRAM	64KB (32K Standby)	64KB (32K Standby)	64KB (32K Standby)	128KB (32K Standby)	256KB (32K Standby)	384KB (48K Standby)
Flash	2MB	2MB	2MB	3MB	4MB	6MB
Flash fetch accelerator	2 x 256 bit	2 x 256 bit	2 x 256 bit	2 x 256 bit	4 x 256 bit	4 x 256 bit
External bus	32 bit	32 bit	32 bit	32 bit	Yes[2]	Yes[2]
Calibration bus	No	16 bit	16 bit	16 bit	16 bit non-mux 16, 32 bit muxed[3]	16 bit, 32 bit muxed[3]
DMA channels	64	32	32	64	64 + 32	64 + 64
DMA Nexus Class	3	3	3	3	3	3
Serial Interface (eSCI)	2	2	2	2	3	3
FlexCAN	3	3	5	4	4	4
SPI	4	3	3	4	4	5
Microsecond bus downlink	No	No	No	No	Yes	Yes
FlexRay	No	No	Yes	No	Yes	Yes
Ethernet	No	No	Yes	Yes	No	No
System Timers	No	No	No	No	1 RTI 4 PIT 4 AutoSAR	1 RTI 4 PIT 4 AutoSAR
eMIOS channels	24	24	24	24	32	32
eTPU channels	64	32	32	64	64	96
eTPU Version	2 x eTPU	1 x eTPU	1 x eTPU	2 x eTPU	2 x eTPU2	3 x eTPU2
eTPU Code memory	16KB	12KB	12KB	12KB	24KB	24KB + 12KB
eTPU Data memory	3KB	2.5KB	2.5KB	3KB	6KB	6KB + 3KB
Interrupt controller	308 sources	210 sources	210 sources	308 sources	448 sources	500 sources
ADC Input Pins	40	40	40	40	64	64
ADC Input diagnostics	No	No	No	No	Yes	Yes
ADC Resolution	12 bit	12 bit	12 bit	12 bit	12 bit	12 bit

Table 1-1. MPC5500/MPC5600 Family Comparison

Feature	MPC5554	MPC5565	MPC5567	MPC5566	MPC5674F	MPC5676R
ADC Quantity	2	2	2	2	4	4
ADC variable gain amp.	No	No	No	No	Yes	Yes
Temp. sensor	No	No	No	No	Yes	Yes
Decimation filters	No	No	No	No	8	12
Protected Port Output	No	No	No	No	No	4
Self Test Controller	No	No	No	No	No	Yes
Dev Tool Semaphores	No	No	No	No	No	32
PLL	FM	FM	FM	FM	FM	FM
Integrated linear voltage regulator	1.5V	1.5V	1.5V	1.5V	3.3V, 1.2V	3.3V, 1.2V
Integrated switch mode voltage regulator	No	No	No	No	1.2V	1.2V
External Power Supplies	5V, 3.3V	5V, 3.3V	5V, 3.3V	5V, 3.3V	5V[4]	5V[4]
Low Power Modes	No	No	No	No	Stop Mode Slow Mode	Stop Mode Slow Mode

[1] There are 2 cores on the MPC5676R. Both cores have identical features.

[2] External Bus Interface (EBI) is not available on 416 PBGA

[3] Cal bus is combined with EBI

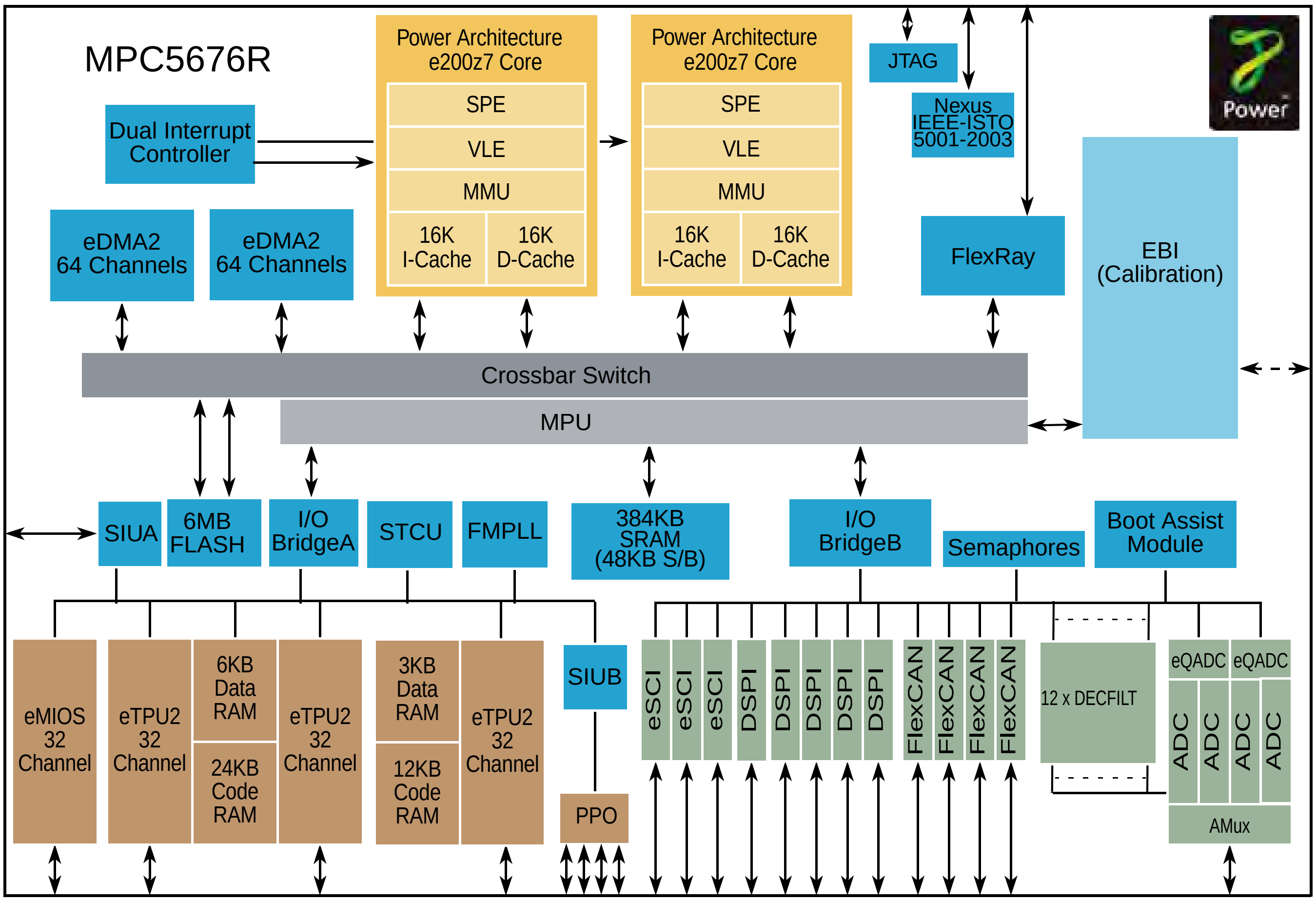
[4] External 3.3V may be needed for external 3.3V pins.

1.2 Block Diagram

Figure 1-1 shows a top-level block diagram of the MPC5676R.

1.3 Block Diagram

The following figure shows a top-level block diagram of the MPC5676R. The purpose of the block diagram is to show the general interconnection of functional modules through the crossbar switch and from the Dual Interrupt Controller, and provide an indication of the modules that connect to external pins. For clarity, the following modules are omitted from the diagram: PMU, SWT, STM, PIT, ECSM, DTS, and CRC.



LEGEND

ADC – Analog to Digital Convertor
AMux – Analog Pin Multiplexer
D-Cache – Data Cache
DECFILT – Decimation Filter
DSPI – Deserial/Serial Peripheral Interface
EBI – External Bus Interface
eDMA2 – Enhanced Direct Memory Access controller version 2
eMIOS – Enhanced Modular I/O System
eQADC – Enhanced Queued Analog to Digital Converter
eSCI – Enhanced Serial Communications Interface
eTPU2 – Enhanced Time Processing Unit version 2
FlexCAN – Flexible Controller Area Network controller
FMPLL – Frequency Modulated Phase Lock Loop clock generator
I-Cache – Instruction Cache
IRC – Internal RC Oscillator
JTAG – Joint Test Action Group controller
MMU – Memory Management Unit
MPU – Memory Protection Unit
PPO – Protected Port Output
S/B – Stand-by
SIUA – System Integration Unit A
SIUB – System Integration Unit B
SPE – Signal Processing Engine
SRAM – Static RAM
STCU – Self Test Control Unit
VLE – Variable Length instruction Encoding

Figure 1-1. MPC5676R Block Diagram

1.4 Critical Performance Parameters

The critical performance parameters of the MPC5676R feature the following:

- Maximum CPU frequency: 184MHz
- Junction temperature range: –40° to 150° C
- Nominal power dissipation is less than 1.4W, while enhancements to allow reduced power operation using clock gating are included
- Separately powered stand-by SRAM

1.5 Low-Power Modes

The MPC5676R includes software controlled reduction of application power consumption by a number of methods:

- Stopping clocks to all modules including one or both cores. A wake-up timer or external interrupt may be used to restart the system clock.
- Reducing system clock frequency to a very low speed by reprogramming the PLL
- Selectively disabling modules.
- Selecting the Internal RC oscillator as the system clock source and disabling the PLL.

1.6 Packages

The MPC5676R is offered in the following package types:

- 416-ball PBGA, 1 mm ball pitch, 27 mm × 27 mm outline (no EBI)
- 516-ball PBGA, 1 mm ball pitch, 27 mm × 27 mm outline (includes EBI)

1.7 Features Summary

On-chip modules available within the family include the following features:

- Two identical dual issue, 32-bit CPU core complexes (e200z7), each with
  - Power Architecture embedded specification compliance
  - Instruction set enhancement allowing variable length encoding (VLE), optional encoding of mixed 16-bit and 32-bit instructions, for code size footprint reduction
  - Signal processing extension (SPE) instruction support for digital sigal processing (DSP)
  - Single-precision floating point operations
  - 16 KB I-Cache and 16 KB D-Cache
  - Hardware cache coherency between cores
- 16 Hardware semaphores
- 3 channel CRC module
- 6MB on-chip flash
  - Supports read during program and erase operations, and multiple blocks allowing EEPROM emulation

- 384KB on-chip general-purpose SRAM including 48KB of standby RAM
- Two Multi channel direct memory access controllers (eDMA)
  - 64 channels per eDMA
- Dual core Interrupt controller (INTC)
- Phase-locked loop with FM modulation (FMPLL)
- Crossbar switch architecture for concurrent access to peripherals, flash, or RAM from multiple bus masters
- External Bus Interface (EBI) for calibration and application use
- System integration unit (SIU)
- Error correction status module (ECSM)
- Four protected port output pins (PPO)
- Boot assist module (BAM) supports serial bootload via CAN or SCI
- Three second-generation enhanced time processor units (eTPU2)
  - 32 channels per eTPU2
  - total of 36 KB code RAM
  - total of 9 KB parameter RAM
- Enhanced modular input output system supporting 32 unified channels (eMIOS) with each channel capable of single action, double action, pulse width modulation (PWM) and modulus counter operation
- Two enhanced queued analog-to-digital converter (eQADC) modules with
  - two separate analog converters per eQADC module
  - support for a total of 64 analog input pins, expandable to 176 inputs with off-chip multiplexers
  - one absolute reference ADC channel
  - interface to twelve hardware decimation filters
  - enhanced ‘Tap’ command to route any conversion to two separate decimation filters
- Five deserial serial peripheral interface (DSPI) modules
- Three enhanced serial communication interface (eSCI) modules
- Four controller area network (FlexCAN) modules
- Dual-channel FlexRay controller
- Nexus development interface (NDI) per IEEE-ISTO 5001-2003 standard, with some support for 2010 standard.
- Device and board test support per Joint Test Action Group (JTAG) (IEEE 1149.1)
- On-chip voltage regulator controller regulates supply voltage down to 1.2 V for core logic
- Self Test capability
- On-chip voltage regulator down to 3.3 V for PLL, flash, and I/O pre-driver logic

1.8 Memory Map

All addresses in the device, including those that are reserved, are identified in the following tables. The addresses represent the physical addresses assigned to each block. Logical addresses are translated by the MMU into physical addresses.

Under software control of the Memory Management Unit (MMU), the logical addresses allocated to IP blocks may be changed on a minimum of a 1KB boundary. Table 1-2 shows this device's memory map.

NOTE

Address ranges not declared in the Memory Map table must be considered as reserved.

Table 1-2. Memory Map

Block	Address
Flash (6MB)	0x0000_0000 - 0x005F_FFFF
Reserved	0x0060_0000 - 0x00EF_BFFF
Flash B Shadow Block	0x00EF_C000 - 0x00EF_FFFF
Reserved	0x00F0_0000 - 0x00FF_BFFF
Flash A Shadow Block	0x00FF_C000 - 0x00FF_FFFF
Emulation reMapping of Flash	0x0100_0000 - 0x1FFF_FFFF
External Bus Memory	0x2000_0000 - 0x3FFF_FFFF
Internal Standby SRAM (48 KB)	0x4000_0000 - 0x4000_BFFF
Internal SRAM (336 KB)	0x4000_C000 - 0x4005_FFFF
Reserved	0x4006_0000 - 0xC3E1_FFFF
eTPU2_C Register	0xC3E2_0000 - 0xC3E2_3FFF
Reserved	0xC3E2_4000 - 0xC3E2_7FFF
eTPU2_C Parameter RAM	0xC3E2_8000 - 0xC3E2_8BFF
Reserved	0xC3E2_8C00 - 0xC3E2_BFFF
eTPU2_C Parameter RAM Mirror	0xC3E2_C000 - 0xC3E2_CBFF
eTPU2_C Code RAM	0xC3E3_0000 - 0xC3E3_2FFF
Reserved	0xC3F0_0000 - 0xC3F0_3FFF
Reserved	0xC3F0_4000 - 0xC3F7_FFFF
PLL	0xC3F8_0000 - 0xC3F8_3FFF
EBI Configuration	0xC3F8_4000 - 0xC3F8_7FFF
Flash Configuration	0xC3F8_8000 - 0xC3F8_FFFF
SIU	0xC3F9_0000 - 0xC3F9_3FFF
Reserved	0xC3F9_4000 - 0xC3F9_7FFF

Table 1-2. Memory Map

Block	Address
SIU_B	0xC3F9_8000 - 0xC3F9_BFFF
DTS	0xC3F9_C000 - 0xC3F9_FFFF
eMIOS_A	0xC3FA_0000 - 0xC3FA_3FFF
Reserved	0xC3FA_4000 - 0xC3FB_BFFF
PMC	0xC3FB_C000 - 0xC3FB_FFFF
eTPU2_AB Register	0xC3FC_0000 - 0xC3FC_3FFF
Reserved	0xC3FC_4000 - 0xC3FC_7FFF
eTPU2_AB Parameter RAM	0xC3FC_8000 - 0xC3FC_97FF
eTPU2_AB Parameter RAM Mirror	0xC3FC_C000 - 0xC3FC_D7FF
eTPU2_AB Code RAM	0xC3FD_0000 - 0xC3FD_5FFF
Reserved	0xC3FD_6000 - 0xC3FE_FFFF
PIT / RTI	0xC3FF_0000 - 0xC3FF_3FFF
STCU	0xC3FF_4000 - 0xC3FF_7FFF
Reserved	0xC3FF_8000 - 0xC3FF_BFFF
Process Monitors (Test Only)	0xC3FF_C000 - 0xC3FF_FFFF
CRCX3	0xFFE6_8000 - 0xFFE6_BFFF
Reserved	0xFFF0_0000 - 0xFFF0_3FFF
XBAR (AXBS)	0xFFF0_4000 - 0xFFF0_7FFF
Reserved	0xFFF0_8000 - 0xFFF0_FFFF
MPU	0xFFF1_0000 - 0xFFF1_3FFF
Reserved	0xFFF1_4000 - 0xFFF2_3FFF
Semaphore Block	0xFFF2_4000 - 0xFFF2_7FFF
Reserved	0xFFF2_8000 - 0xFFF3_3FFF
SWT_B	0xFFF3_4000 - 0xFFF3_7FFF
SWT_A	0xFFF3_8000 - 0xFFF3_BFFF
STM	0xFFF3_C000 - 0xFFF3_FFFF
ECSM	0xFFF4_0000 - 0xFFF4_3FFF
eDMA_A	0xFFF4_4000 - 0xFFF4_7FFF
INTC	0xFFF4_8000 - 0xFFF4_BFFF
Reserved	0xFFF4_C000 - 0xFFF4_FFFF
CCU	0xFFF5_0000 - 0xFFF5_3FFF
eDMA_B	0xFFF5_4000 - 0xFFF5_7FFF
Reserved	0xFFF5_8000 - 0xFFF7_FFFF

Table 1-2. Memory Map

Block	Address
eQADC_A	0xFFF8_0000 - 0xFFF8_3FFF
eQADC_B	0xFFF8_4000 - 0xFFF8_7FFF
Decimation filter A	0xFFF8_8000 - 0xFFF8_87FF
Decimation filter B	0xFFF8_8800 - 0xFFF8_8FFF
Decimation filter C	0xFFF8_9000 - 0xFFF8_97FF
Decimation filter D	0xFFF8_9800 - 0xFFF8_9FFF
Decimation filter E	0xFFF8_A000 - 0xFFF8_A7FF
Decimation filter F	0xFFF8_A800 - 0xFFF8_AFFF
Decimation filter G	0xFFF8_B000 - 0xFFF8_B7FF
Decimation filter H	0xFFF8_B800 - 0xFFF8_BFFF
Decimation filter I	0xFFF8_C000 - 0xFFF8_C7FF
Decimation filter J	0xFFF8_C800 - 0xFFF8_CFFF
Decimation filter K	0xFFF8_D000 - 0xFFF8_D7FF
Decimation filter L	0xFFF8_D800 - 0xFFF8_DFFF
Reserved	0xFFF8_E000 - 0xFFF8_FFFF
DSPI_A	0xFFF9_0000 - 0xFFF9_3FFF
DSPI_B	0xFFF9_4000 - 0xFFF9_7FFF
DSPI_C	0xFFF9_8000 - 0xFFF9_BFFF
DSPI_D	0xFFF9_C000 - 0xFFF9_FFFF
DSPI_E	0xFFFA_0000 - 0xFFFA_3FFF
Reserved	0xFFFA_4000 - 0xFFFA_FFFF
SCI_A	0xFFFB_0000 - 0xFFFB_3FFF
SCI_B	0xFFFB_4000 - 0xFFFB_7FFF
SCI_C	0xFFFB_8000 - 0xFFFB_BFFF
Reserved	0xFFFB_C000 - 0xFFFB_FFFF
FlexCAN_A	0xFFFC_0000 - 0xFFFC_3FFF
FlexCAN_B	0xFFFC_4000 - 0xFFFC_7FFF
FlexCAN_C	0xFFFC_8000 - 0xFFFC_BFFF
FlexCAN_D	0xFFFC_C000 - 0xFFFC_FFFF
Reserved	0xFFFD_0000 - 0xFFFD_FFFF
FlexRay	0xFFFE_0000 - 0xFFFE_3FFF
Reserved	0xFFFE_4000 - 0xFFFE_BFFF
System Information Module (SIM)	0xFFFE_C000 - 0xFFFE_FFFF

Table 1-2. Memory Map

Block	Address
Reserved	0xFFFF_0000 - 0xFFFF_BFFF
Boot Assist Module	0xFFFF_C000 - 0xFFFF_FFFF

Chapter 2
Signal Descriptions

This chapter describes the external device signals, including a table of signal properties, detailed descriptions of the available signals, and the I/O pin power/ground segmentation.

2.1 Pin Function Selection

2.1.1 Pad Configuration Register (PCR) PA Definition

Most pins (balls) on the package support more than one function. Unlike prior devices in this class, the required pin function on this device is selected by a fixed size of the PA bit field in each SIU_PCR. The PA width is fixed at 3 bits. The pin function categories are:

Table 2-1. SIU_PCR PA Definition

PA[2:0]	Function Category
000	GPIO
001	Primary Function
010	Alternate function 1 (A1)
011	Alternate function 2 (A2)
100	Alternate function 3 (A3)

Pins that do not require selection of a function by modifying the PA field have a hard-wired value corresponding to the appropriate function available at the pin. All PA values might not be used. A definition of which values are used is provided in the SIU_PCRn settings table (Table 3-22). The “Primary Function” name is retained from previous MPC5xxx designs and indicates the name used on the ball map. The name “GPIO” is also retained for clarity. The remaining function names are new and have been chosen for simplicity and clarity.

2.1.2 LVDS Signal Selection

A pin (ball) that is driven by a signal from a 5V pad or from an LVDS pad has a single PCR controlling both signals. Different values of PA select either the 5V signal or the LVDS signal. Note that a single LVDS pad supplies two signals — a “true” and a “complement” output. This means that one LVDS pad drives two balls, but each ball has a separate PCR (and PA field). The two LVDS signals are enabled only when the PA values in both PCRs are set to select LVDS operation. Both LVDS signals are deselected when either PA is set to a value that deselects its LVDS signal. The ball that has its PA field still set to LVDS will go to an undriven state until its PA field is updated to a non-LVDS value.

Table 2-2 is an example of the options for PA on two balls, where “true” and “complement” LVDS outputs are multiplexed with SCK_C and SINC.

Table 2-2. LVDS example

PCR235[1][PA] Selection	PCR236[2][PA] Selection	SCKC Ball function	SINC Ball function
SCKC	SINC	SCKC	SINC
LVDS	LVDS	SCK_C_LVDS+	SCK_C_LVDS-
SCKC	LVDS	SCKC	undriven
LVDS	SINC	undriven	SINC

[1] SCKC_SCK_C_LVDSP_GPIO235

[2] SINC_SCK_C_LVDSM_GPIO236

Additionally, when LVDS signals are enabled on balls, their PCR SRC[1:0] bits control the differential signal output swing increase and decrease, as defined in Table 2-3.

Table 2-3. Differential Signal Output when LVDS is Enabled

SRC1	SRC0	Current flowing in the driver	Differential Voltage Across ‘true’ and ‘complement’
0	0	normal	default
0	1	increased	increased (20%)
1	0	decreased	decreased (20%)
1	1	normal	same as default

2.2 External Signal Descriptions, Pin Multiplexing, and Attributes

This section summarizes the external signal functions, their static electrical characteristics, and pad configuration settings for this device. The signal properties and their electrical characteristics are set in the System Integration Unit (SIU) Pad Configuration (PCR) registers. See the *MPC5676R Microcontroller Data Sheet* for ball-map figures.

Table 2-4. Signal Properties and Muxing Summary

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
eTPU_A											
113	TCRCLKA_IRQ7_ GPIO113	P	TCRCLKA	eTPU A TCR clock	I	MH	V_{DDEH1}	—/Up	—/Up	L1	K4
		A1	IRQ7	External interrupt request	I						
		A2	—	—	—						
		G	GPIO113	GPIO	I/O						
114	ETPUA0_ETPUA12_ GPIO114	P	ETPUA0	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	L2	L6
		A1	ETPUA12	eTPU A channel (output only)	O						
		A2	—	—	—						
		G	GPIO114	GPIO	I/O						
115	ETPUA1_ETPUA13_ GPIO115	P	ETPUA1	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	L3	J1
		A1	ETPUA13	eTPU A channel (output only)	O						
		A2	—	—	—						
		G	GPIO115	GPIO	I/O						
116	ETPUA2_ETPUA14_ GPIO116	P	ETPUA2	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	L4	J2
		A1	ETPUA14	eTPU A channel (output only)	O						
		A2	—	—	—						
		G	GPIO116	GPIO	I/O						
117	ETPUA3_ETPUA15_ GPIO117	P	ETPUA3	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	K1	H4
		A1	ETPUA15	eTPU A channel (output only)	O						
		A2	—	—	—						
		G	GPIO117	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
118	ETPUA4_ETPUA16_ GPIO118	P	ETPUA4	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	K2	J4
		A1	ETPUA16	eTPU A channel (output only)	O						
		A2	—	—	—						
		G	GPIO118	GPIO	I/O						
119	ETPUA5_ETPUA17_ GPIO119	P	ETPUA5	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	K3	H1
		A1	ETPUA17	eTPU A channel (output only)	O						
		A2	—	—	—						
		G	GPIO119	GPIO	I/O						
120	ETPUA6_ETPUA18_ GPIO120	P	ETPUA6	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	K4	K5
		A1	ETPUA18	eTPU A channel (output only)	O						
		A2	—	—	—						
		G	GPIO120	GPIO	I/O						
121	ETPUA7_ETPUA19_ GPIO121	P	ETPUA7	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	J1	H2
		A1	ETPUA19	eTPU A channel (output only)	O						
		A2	—	—	—						
		G	GPIO121	GPIO	I/O						
122	ETPUA8_ETPUA20_ GPIO122	P	ETPUA8	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	J2	H3
		A1	ETPUA20	eTPU A channel (output only)	O						
		A2	—	—	—						
		G	GPIO122	GPIO	I/O						
123	ETPUA9_ETPUA21_ GPIO123	P	ETPUA9	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	J3	J3
		A1	ETPUA21	eTPU A channel (output only)	O						
		A2	—	—	—						
		G	GPIO123	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location 416	Package Location 516
124	ETPUA10_ETPUA22_ GPIO124	P	ETPUA10	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	J4	K6
		A1	ETPUA22	eTPU A channel (output only)	O						
		A2	—	—	—						
		G	GPIO124	GPIO	I/O						
125	ETPUA11_ETPUA23_ GPIO125	P	ETPUA11	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	H1	G1
		A1	ETPUA23	eTPU A channel (output only)	O						
		A2	—	—	—						
		G	GPIO125	GPIO	I/O						
126	ETPUA12_PCSB1_ GPIO126	P	ETPUA12	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	H2	J5
		A1	PCSB1	DSPI B peripheral chip select	O						
		A2	—	—	—						
		G	GPIO126	GPIO	I/O						
127	ETPUA13_PCSB3_ GPIO127	P	ETPUA13	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	H4	G2
		A1	PCSB3	DSPI B peripheral chip select	O						
		A2	—	—	—						
		G	GPIO127	GPIO	I/O						
128	ETPUA14_PCSB4_ GPIO128	P	ETPUA14	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	H3	H5
		A1	PCSB4	DSPI B peripheral chip select	O						
		A2	—	—	—						
		G	GPIO128	GPIO	I/O						
129	ETPUA15_PCSB5_ GPIO129	P	ETPUA15	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	G1	G3
		A1	PCSB5	DSPI B peripheral chip select	O						
		A2	—	—	—						
		G	GPIO129	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
130	ETPUA16_PCSD1_GPIO130	P	ETPUA16	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	G2	H6
		A1	PCSD1	DSPI D peripheral chip select	O						
		A2	—	—	—						
		G	GPIO130	GPIO	I/O						
131	ETPUA17_PCSD2_GPIO131	P	ETPUA17	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	G3	G4
		A1	PCSD2	DSPI D peripheral chip select	O						
		A2	—	—	—						
		G	GPIO131	GPIO	I/O						
132	ETPUA18_PCSD3_GPIO132	P	ETPUA18	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	G4	G5
		A1	PCSD3	DSPI D peripheral chip select	O						
		A2	—	—	—						
		G	GPIO132	GPIO	I/O						
133	ETPUA19_PCSD4_GPIO133	P	ETPUA19	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	F1	F1
		A1	PCSD4	DSPI D peripheral chip select	O						
		A2	—	—	—						
		G	GPIO133	GPIO	I/O						
134	ETPUA20_IRQ8_GPIO134	P	ETPUA20	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	F2	F2
		A1	IRQ8	External interrupt request	I						
		A2	—	—	—						
		G	GPIO134	GPIO	I/O						
135	ETPUA21_IRQ9_GPIO135	P	ETPUA21	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	F3	F3
		A1	IRQ9	External interrupt request	I						
		A2	—	—	—						
		G	GPIO135	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
136	ETPUA22_IRQ10_ GPIO136	P	ETPUA22	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	F4	F4
		A1	IRQ10	External interrupt request	I						
		A2	—	—	—						
		G	GPIO136	GPIO	I/O						
137	ETPUA23_IRQ11_ GPIO137	P	ETPUA23	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	E1	E1
		A1	IRQ11	External interrupt request	I						
		A2	—	—	—						
		G	GPIO137	GPIO	I/O						
138	ETPUA24_IRQ12_ GPIO138	P	ETPUA24	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	E2	E2
		A1	IRQ12	External interrupt request	I						
		A2	—	—	—						
		G	GPIO138	GPIO	I/O						
139	ETPUA25_IRQ13_ GPIO139	P	ETPUA25	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	E3	E3
		A1	IRQ13	External interrupt request	I						
		A2	—	—	—						
		G	GPIO139	GPIO	I/O						
140	ETPUA26_IRQ14_ GPIO140	P	ETPUA26	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	E4	E4
		A1	IRQ14	External interrupt request	I						
		A2	—	—	—						
		G	GPIO140	GPIO	I/O						
141	ETPUA27_IRQ15_ GPIO141	P	ETPUA27	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	D1	D1
		A1	IRQ15	External interrupt request	I						
		A2	—	—	—						
		G	GPIO141	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
142	ETPUA28_PCSC1_ GPIO142	P	ETPUA28	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	D2	D2
		A1	PCSC1	DSPI C peripheral chip select	O						
		A2	—	—	—						
		G	GPIO142	GPIO	I/O						
143	ETPUA29_PCSC2_ GPIO143	P	ETPUA29	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	D3	D3
		A1	PCSC2	DSPI C peripheral chip select	O						
		A2	—	—	—						
		G	GPIO143	GPIO	I/O						
144	ETPUA30_PCSC3_ GPIO144	P	ETPUA30	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	C1	C1
		A1	PCSC3	DSPI C peripheral chip select	O						
		A2	—	—	—						
		G	GPIO144	GPIO	I/O						
145	ETPUA31_PCSC4_ GPIO145	P	ETPUA31	eTPU A channel	I/O	MH	V_{DDEH1}	—/WKPCFG	—/WKPCFG	C2	C2
		A1	PCSC4	DSPI C peripheral chip select	O						
		A2	—	—	—						
		G	GPIO145	GPIO	I/O						
eTPU_B											
146	TCRCLKB_IRQ6_ GPIO146	P	TCRCLKB	eTPU B TCR clock	I	MH	V_{DDEH6}	—/Up	—/Up	T23	V25
		A1	IRQ6	External interrupt request	I						
		A2	—	—	—						
		G	GPIO146	GPIO	I/O						
147	ETPUB0_ETPUB16_ GPIO147	P	ETPUB0	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	T24	V26
		A1	ETPUB16	eTPU B channel (output only)	O						
		A2	—	—	—						
		G	GPIO147	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
148	ETPUB1_ETPUB17_ GPIO148	P	ETPUB1	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	T25	U22
		A1	ETPUB17	eTPU B channel (output only)	O						
		A2	—	—	—						
		G	GPIO148	GPIO	I/O						
149	ETPUB2_ETPUB18_ GPIO149	P	ETPUB2	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	T26	U23
		A1	ETPUB18	eTPU B channel (output only)	O						
		A2	—	—	—						
		G	GPIO149	GPIO	I/O						
150	ETPUB3_ETPUB19_ GPIO150	P	ETPUB3	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	R23	T22
		A1	ETPUB19	eTPU B channel (output only)	O						
		A2	—	—	—						
		G	GPIO150	GPIO	I/O						
151	ETPUB4_ETPUB20_ GPIO151	P	ETPUB4	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	R24	U24
		A1	ETPUB20	eTPU B channel (output only)	O						
		A2	—	—	—						
		G	GPIO151	GPIO	I/O						
152	ETPUB5_ETPUB21_ GPIO152	P	ETPUB5	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	R25	U25
		A1	ETPUB21	eTPU B channel (output only)	O						
		A2	—	—	—						
		G	GPIO152	GPIO	I/O						
153	ETPUB6_ETPUB22_ GPIO153	P	ETPUB6	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	R26	U26
		A1	ETPUB22	eTPU B channel (output only)	O						
		A2	—	—	—						
		G	GPIO153	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
154	ETPUB7_ETPUB23_ GPIO154	P	ETPUB7	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	P23	T23
		A1	ETPUB23	eTPU B channel (output only)	O						
		A2	—	—	—						
		G	GPIO154	GPIO	I/O						
155	ETPUB8_ETPUB24_ GPIO155	P	ETPUB8	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	P24	T24
		A1	ETPUB24	eTPU B channel (output only)	O						
		A2	—	—	—						
		G	GPIO155	GPIO	I/O						
156	ETPUB9_ETPUB25_ GPIO156	P	ETPUB9	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	P25	R22
		A1	ETPUB25	eTPU B channel (output only)	O						
		A2	—	—	—						
		G	GPIO156	GPIO	I/O						
157	ETPUB10_ETPUB26_ GPIO157	P	ETPUB10	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	P26	T25
		A1	ETPUB26	eTPU B channel (output only)	O						
		A2	—	—	—						
		G	GPIO157	GPIO	I/O						
158	ETPUB11_ETPUB27_ GPIO158	P	ETPUB11	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	N24	T26
		A1	ETPUB27	eTPU B channel (output only)	O						
		A2	—	—	—						
		G	GPIO158	GPIO	I/O						
159	ETPUB12_ETPUB28_ GPIO159	P	ETPUB12	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	N25	R23
		A1	ETPUB28	eTPU B channel (output only)	O						
		A2	—	—	—						
		G	GPIO159	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
160	ETPUB13_ETPUB29_GPIO160	P	ETPUB13	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	N26	P22
		A1	ETPUB29	eTPU B channel (output only)	O						
		A2	—	—	—						
		G	GPIO160	GPIO	I/O						
161	ETPUB14_ETPUB30_GPIO161	P	ETPUB14	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	M25	R24
		A1	ETPUB30	eTPU B channel (output only)	O						
		A2	—	—	—						
		G	GPIO161	GPIO	I/O						
162	ETPUB15_ETPUB31_GPIO162	P	ETPUB15	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	M24	R25
		A1	ETPUB31	eTPU B channel (output only)	O						
		A2	—	—	—						
		G	GPIO162	GPIO	I/O						
163	ETPUB16_PCSA1_GPIO163	P	ETPUB16	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	U26	V24
		A1	PCSA1	DSPI A peripheral chip select	O						
		A2	—	—	—						
		G	GPIO163	GPIO	I/O						
164	ETPUB17_PCSA2_GPIO164	P	ETPUB17	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	U25	T21
		A1	PCSA2	DSPI A peripheral chip select	O						
		A2	—	—	—						
		G	GPIO164	GPIO	I/O						
165	ETPUB18_PCSA3_GPIO165	P	ETPUB18	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	U24	W26
		A1	PCSA3	DSPI A peripheral chip select	O						
		A2	—	—	—						
		G	GPIO165	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
166	ETPUB19_PCSA4_ GPIO166	P	ETPUB19	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	U23	W25
		A1	PCSA4	DSPI A peripheral chip select	O						
		A2	—	—	—						
		G	GPIO166	GPIO	I/O						
167	ETPUB20_ GPIO167	P	ETPUB20	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	V26	W24
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO167	GPIO	I/O						
168	ETPUB21_ GPIO168	P	ETPUB21	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	V25	V22
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO168	GPIO	I/O						
169	ETPUB22_ GPIO169	P	ETPUB22	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	V24	V23
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO169	GPIO	I/O						
170	ETPUB23_ GPIO170	P	ETPUB23	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	W26	U21
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO170	GPIO	I/O						
171	ETPUB24_ GPIO171	P	ETPUB24	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	W25	Y25
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO171	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
172	ETPUB25_ GPIO172	P	ETPUB25	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	W24	W21
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO172	GPIO	I/O						
173	ETPUB26_ GPIO173	P	ETPUB26	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	V23	Y23
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO173	GPIO	I/O						
174	ETPUB27_ GPIO174	P	ETPUB27	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	Y25	Y24
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO174	GPIO	I/O						
175	ETPUB28_ GPIO175	P	ETPUB28	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	Y24	AA24
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO175	GPIO	I/O						
176	ETPUB29_ GPIO176	P	ETPUB29	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	Y23	W22
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO176	GPIO	I/O						
177	ETPUB30_ GPIO177	P	ETPUB30	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	AA24	AB24
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO177	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
178	ETPUB31_ GPIO178	P	ETPUB31	eTPU B channel	I/O	MH	V_{DDEH6}	—/WKPCFG	—/WKPCFG	AB24	Y22
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO178	GPIO	I/O						
eTPU_C											
440	TCRCLKC_ GPIO440	P	TCRCLKC	eTPU C TCR clock	I	MH	V_{DDEH7}	—/Up	—/Up	B26	F22
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO440	GPIO	I/O						
441	ETPUC0_ GPIO441	P	ETPUC0	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	C25	C25
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO441	GPIO	I/O						
442	ETPUC1_ GPIO442	P	ETPUC1	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	C26	C26
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO442	GPIO	I/O						
443	ETPUC2_ GPIO443	P	ETPUC2	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	D25	D25
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO443	GPIO	I/O						
444	ETPUC3_ GPIO444	P	ETPUC3	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	D26	D26
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO444	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
445	ETPUC4_ PCSE1_GPIO445	P	ETPUC4	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	E24	E24
		A1		DSPI E peripheral chip select							
		A2	—	—	—						
		G	GPIO445	GPIO	I/O						
446	ETPUC5_ PCSE2_GPIO446	P	ETPUC5	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	E25	E25
		A1		DSPI E peripheral chip select							
		A2	—	—	—						
		G	GPIO446	GPIO	I/O						
447	ETPUC6_ PCSE3_GPIO447	P	ETPUC6	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	E26	E26
		A1		DSPI E peripheral chip select							
		A2	—	—	—						
		G	GPIO447	GPIO	I/O						
448	ETPUC7_ PCSE4_GPIO448	P	ETPUC7	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	F23	F23
		A1		DSPI E peripheral chip select							
		A2	—	—	—						
		G	GPIO448	GPIO	I/O						
449	ETPUC8_ PCSE5_GPIO449	P	ETPUC8	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	F24	F24
		A1		DSPI E peripheral chip select							
		A2	—	—	—						
		G	GPIO449	GPIO	I/O						
450	ETPUC9_IRQ0_ GPIO450	P	ETPUC9	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	F25	F25
		A1	IRQ0	External interrupt request	I						
		A2	—	—	—						
		G	GPIO450	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
451	ETPUC10__IRQ1_GPIO451	P	ETPUC10	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	F26	F26
		A1	IRQ1	External interrupt request	I						
		A2	—	—	—						
		G	GPIO451	GPIO	I/O						
452	ETPUC11_IRQ2_GPIO452	P	ETPUC11	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	G23	G22
		A1	IRQ2	External interrupt request	I						
		A2	—	—	—						
		G	GPIO452	GPIO	I/O						
453	ETPUC12_IRQ3_GPIO453	P	ETPUC12	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	G24	G23
		A1	IRQ3	External interrupt request	I						
		A2	—	—	—						
		G	GPIO453	GPIO	I/O						
454	ETPUC13_3_IRQ4_GPIO454	P	ETPUC13	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	G25	G24
		A1	IRQ4	External interrupt request	I						
		A2	—	—	—						
		G	GPIO454	GPIO	I/O						
455	ETPUC14_4_IRQ5_GPIO455	P	ETPUC14	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	G26	G25
		A1	IRQ5	External interrupt request	I						
		A2	—	—	—						
		G	GPIO455	GPIO	I/O						
456	ETPUC15__GPIO456	P	ETPUC15	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	H23	G26
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO456	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
457	ETPUC16_FR_A_TX_ GPIO457	P	ETPUC16	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	H24	H22
		A1	FR_A_TX	FlexRay A transfer	O						
		A2	—	—	—						
		G	GPIO457	GPIO	I/O						
458	ETPUC17_FR_A_RX_ GPIO458	P	ETPUC17	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	H25	H23
		A1	FR_A_RX	FlexRay A receive	I						
		A2	—	—	—						
		G	GPIO458	GPIO	I/O						
459	ETPUC18_FR_A_TX_EN_ GPIO459	P	ETPUC18	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	H26	H24
		A1	FR_A_TX_EN	FlexRay A transfer enable	O						
		A2	—	—	—						
		G	GPIO459	GPIO	I/O						
460	ETPUC19_TXDA_ GPIO460	P	ETPUC19	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	J23	H21
		A1	TXDA	eSCI A transmit	O						
		A2	—	—	—						
		G	GPIO460	GPIO	I/O						
461	ETPUC20_RXDA _ GPIO461	P	ETPUC20	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	J24	H25
		A1	RXDA	eSCI A receive	I						
		A2	—	—	—						
		G	GPIO461	GPIO	I/O						
462	ETPUC21_TXDB_ GPIO462	P	ETPUC21	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	J25	H26
		A1	TXDB	eSCI B transmit	O						
		A2	—	—	—						
		G	GPIO462	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
463	ETPUC22_RXDB_ GPIO463	P	ETPUC22	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	J26	J22
		A1	RXDB	eSCI B receive	I						
		A2	—	—	—						
		G	GPIO463	GPIO	I/O						
464	ETPUC23_PCSD5_ GPIO464	P	ETPUC23	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	K23	J23
		A1	PCSD5	DSPI D peripheral chip select	O						
		A2	MAA0	ADC A Mux Address 0	O						
		A3	MAB0	ADC B Mux Address 0	O						
		G	GPIO464	GPIO	I/O						
465	ETPUC24_PCSD4_ GPIO465	P	ETPUC24	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	K24	J24
		A1	PCSD4	DSPI D peripheral chip select	O						
		A2	MAA1	ADC A Mux Address 1	O						
		A4	MAB1	ADC B Mux Address 1	O						
		G	GPIO465	GPIO	I/O						
466	ETPUC25_PCSD3_ GPIO466	P	ETPUC25	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	K25	K21
		A1	PCSD3	DSPI D peripheral chip select	O						
		A2	MAA2	ADC A Mux Address 2	O						
		A3	MAB2	ADC B Mux Address 2	O						
		G	GPIO466	GPIO	I/O						
467	ETPUC26_PCSD2_ GPIO467	P	ETPUC26	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	K26	J25
		A1	PCSD2	DSPI D peripheral chip select	O						
		A2	—	—	—						
		G	GPIO467	GPIO	I/O						
468	ETPUC27_PCSD1_ GPIO468	P	ETPUC27	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	L23	J26
		A1	PCSD1	DSPI D peripheral chip select	O						
		A2	—	—	—						
		G	GPIO468	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
469	ETPUC28_PCSD0_ GPIO469	P	ETPUC28	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	L24	K22
		A1	PCSD0	DSPI D peripheral chip select	O						
		A2	—	—	—						
		G	GPIO469	GPIO	I/O						
470	ETPUC29_SCKD_ GPIO470	P	ETPUC29	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	L25	K23
		A1	SCKD	DSPI D clock	I/O						
		A2	—	—	—						
		G	GPIO470	GPIO	I/O						
471	ETPUC30_SOUTD_ GPIO471	P	ETPUC30	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	L26	K24
		A1	SOUTD	DSPI D data output	O						
		A2	—	—	—						
		G	GPIO471	GPIO	I/O						
472	ETPUC31_SIND_ GPIO472	P	ETPUC31	eTPU C channel	I/O	MH	V_{DDEH7}	—/WKPCFG	—/WKPCFG	M23	K25
		A1	SIND	DSPI D data input	I						
		A2	—	—	—						
		G	GPIO472	GPIO	I/O						
eMIOS											
179	EMIOS0_ETPUA0_ GPIO179	P	EMIOS0	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AE10	AC13
		A1	ETPUA0	eTPU A channel	O						
		A2	—	—	—						
		G	GPIO179	GPIO	I/O						
180	EMIOS1_ETPUA1_ GPIO180	P	EMIOS1	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AF10	AB13
		A1	ETPUA1	eTPU A channel	O						
		A2	—	—	—						
		G	GPIO180	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
181	EMIOS2_ETPUA2_ GPIO181	P	EMIOS2	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AD11	AD13
		A1	ETPUA2	eTPU A channel	O						
		A2	—	—	—						
		G	GPIO181	GPIO	I/O						
182	EMIOS3_ETPUA3_ GPIO182	P	EMIOS3	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AE11	AE13
		A1	ETPUA3	eTPU A channel	O						
		A2	—	—	—						
		G	GPIO182	GPIO	I/O						
183	EMIOS4_ETPUA4_ GPIO183	P	EMIOS4	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AF11	AF13
		A1	ETPUA4	eTPU A channel	O						
		A2	—	—	—						
		G	GPIO183	GPIO	I/O						
184	EMIOS5_ETPUA5_ GPIO184	P	EMIOS5	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AD12	AF14
		A1	ETPUA5	eTPU A channel	O						
		A2	—	—	—						
		G	GPIO184	GPIO	I/O						
185	EMIOS6_ETPUA6_ GPIO185	P	EMIOS6	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AE12	AE14
		A1	ETPUA6	eTPU A channel	O						
		A2	—	—	—						
		G	GPIO185	GPIO	I/O						
186	EMIOS7_ETPUA7_ GPIO186	P	EMIOS7	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AF12	AD14
		A1	ETPUA7	eTPU A channel	O						
		A2	—	—	—						
		G	GPIO186	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
187	EMIOS8_ETPUA8_ GPIO187	P	EMIOS8	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AC13	AC14
		A1	ETPUA8	eTPU A channel	O						
		A2	—	—	—						
		G	GPIO187	GPIO	I/O						
188	EMIOS9_ETPUA9_ GPIO188	P	EMIOS9	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AD13	AF15
		A1	ETPUA9	eTPU A channel	O						
		A2	—	—	—						
		G	GPIO188	GPIO	I/O						
189	EMIOS10_SCKD_ GPIO189	P	EMIOS10	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AE13	AE15
		A1	SCKD	DSPI D clock	O						
		A2	—	—	—						
		G	GPIO189	GPIO	I/O						
190	EMIOS11_SIND_ GPIO190	P	EMIOS11	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AF13	AB14
		A1	SIND	DSPI D data input	I						
		A2	—	—	—						
		G	GPIO190	GPIO	I/O						
191	EMIOS12_SOUTC_ GPIO191	P	EMIOS12	eMIOS channel	O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AF14	AD15
		A1	SOUTC	DSPI C data output	O						
		A2	—	—	—						
		G	GPIO191	GPIO	I/O						
192	EMIOS13_SOUTD_ GPIO192	P	EMIOS13	eMIOS channel	O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AE14	AC15
		A1	SOUTD	DSPI D data output	O						
		A2	—	—	—						
		G	GPIO192	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
193	EMIOS14_IRQ0_ GPIO193	P	EMIOS14	eMIOS channel	O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AC14	AF17
		A1	IRQ0	External interrupt request	I						
		A2	CNTXD	FlexCAN D transmit	O						
		G	GPIO193	GPIO	I/O						
194	EMIOS15_IRQ1_ GPIO194	P	EMIOS15	eMIOS channel	O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AD14	AE16
		A1	IRQ1	External interrupt request	I						
		A2	CNRXD	FlexCAN D receive	I						
		G	GPIO194	GPIO	I/O						
195	EMIOS16_ETPUB0_ GPIO195	P	EMIOS16	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AF15	AD16
		A1	ETPUB0	eTPU B channel	O						
		A2	FR_DBG[3]	FlexRay debug	O						
		G	GPIO195	GPIO	I/O						
196	EMIOS17_ETPUB1_ GPIO196	P	EMIOS17	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AE15	AB15
		A1	ETPUB1	eTPU B channel	O						
		A2	FR_DBG[2]	FlexRay debug	O						
		G	GPIO196	GPIO	I/O						
197	EMIOS18_ETPUB2_ GPIO197	P	EMIOS18	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AC15	AD17
		A1	ETPUB2	eTPU B channel	O						
		A2	FR_DBG[1]	FlexRay debug	O						
		G	GPIO197	GPIO	I/O						
198	EMIOS19_ETPUB3_ GPIO198	P	EMIOS19	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AD15	AB16
		A1	ETPUB3	eTPU B channel	O						
		A2	FR_DBG[0]	FlexRay debug	O						
		G	GPIO198	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
199	EMIOS20_ETPUB4_ GPIO199	P	EMIOS20	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AF16	AF16
		A1	ETPUB4	eTPU B channel	O						
		A2	—	—	—						
		G	GPIO199	GPIO	I/O						
200	EMIOS21_ETPUB5_ GPIO200	P	EMIOS21	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AE16	AE17
		A1	ETPUB5	eTPU B channel	O						
		A2	—	—	—						
		G	GPIO200	GPIO	I/O						
201	EMIOS22_ETPUB6_ GPIO201	P	EMIOS22	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AC16	AC16
		A1	ETPUB6	eTPU B channel	O						
		A2	—	—	—						
		G	GPIO201	GPIO	I/O						
202	EMIOS23_ETPUB7_ GPIO202	P	EMIOS23	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AD16	AA16
		A1	ETPUB7	eTPU B channel	O						
		A2	—	—	—						
		G	GPIO202	GPIO	I/O						
203	EMIOS24_PCSB0_ GPIO203	P	EMIOS24	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AF17	AC17
		A1	PCSB0	DSPI B peripheral chip select	I/O						
		A2	—	—	—						
		G	GPIO203	GPIO	I/O						
204	EMIOS25_PCSB1_ GPIO204	P	EMIOS25	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AE17	AF18
		A1	PCSB1	DSPI B peripheral chip select	O						
		A2	—	—	—						
		G	GPIO204	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
432	EMIOS26_PCSB2_ GPIO432	P	EMIOS26	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AD17	AE18
		A1	PCSB2	DSPI B peripheral chip select	O						
		A2	—	—	—						
		G	GPIO432	GPIO	I/O						
433	EMIOS27_PCSB3_ GPIO433	P	EMIOS27	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AC17	AD18
		A1	PCSB3	DSPI B peripheral chip select	O						
		A2	—	—	—						
		G	GPIO433	GPIO	I/O						
434	EMIOS28_PCSC0_ GPIO434	P	EMIOS28	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AF18	AC18
		A1	PCSC0	DSPI C peripheral chip select	I/O						
		A2	—	—	—						
		G	GPIO434	GPIO	I/O						
435	EMIOS29_PCSC1_ GPIO435	P	EMIOS29	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AE18	AB17
		A1	PCSC1	DSPI C peripheral chip select	O						
		A2	—	—	—						
		G	GPIO435	GPIO	I/O						
436	EMIOS30_PCSC2_ GPIO436	P	EMIOS30	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AD18	AF19
		A1	PCSC2	DSPI C peripheral chip select	O						
		A2	—	—	—						
		G	GPIO436	GPIO	I/O						
437	EMIOS31_PCSC5_ GPIO437	P	EMIOS31	eMIOS channel	I/O	MH	V_{DDEH4}	—/WKPCFG	—/WKPCFG	AC18	AA17
		A1	PCSC5	DSPI C peripheral chip select	O						
		A2	—	—	—						
		G	GPIO437	GPIO	I/O						
eQADC											
—	ANA0	P	ANA0[9]	eQADC A shared analog input	I	AE/up-down	V_{DDA_A1}	ANA0	ANA0	A4	A4

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location 416	Package Location 516
—	ANA1	P	ANA1[9]	eQADC A shared analog input	I	AE/up-down	V_{DDA_A1}	ANA1	ANA1	B5	B5
—	ANA2	P	ANA2[9]	eQADC A shared analog input	I	AE/up-down	V_{DDA_A1}	ANA2	ANA2	C5	C5
—	ANA3	P	ANA3[9]	eQADC A shared analog input	I	AE/up-down	V_{DDA_A1}	ANA3	ANA3	D6	D6
—	ANA4	P	ANA4[9]	eQADC A shared analog input	I	AE/up-down	V_{DDA_A1}	ANA4	ANA4	A5	A5
—	ANA5	P	ANA5[9]	eQADC A shared analog input	I	AE/up-down	V_{DDA_A1}	ANA5	ANA5	B6	B6
—	ANA6	P	ANA6[9]	eQADC A shared analog input	I	AE/up-down	V_{DDA_A1}	ANA6	ANA6	C6	C6
—	ANA7	P	ANA7[9]	eQADC A shared analog input	I	AE/up-down	V_{DDA_A1}	ANA7	ANA7	D7	C7
—	ANA8	P	ANA8	eQADC A analog input	I	AE	V_{DDA_A1}	ANA8	ANA8	A6	D7
—	ANA9	P	ANA9	eQADC A analog input	I	AE	V_{DDA_A1}	ANA9	ANA9	C7	A6
—	ANA10	P	ANA10	eQADC A analog input	I	AE	V_{DDA_A1}	ANA10	ANA10	B7	B7
—	ANA11	P	ANA11	eQADC A analog input	I	AE	V_{DDA_A1}	ANA11	ANA11	A7	A7
—	ANA12	P	ANA12	eQADC A analog input	I	AE	V_{DDA_A1}	ANA12	ANA12	D8	D8
—	ANA13	P	ANA13	eQADC A analog input	I	AE	V_{DDA_A1}	ANA13	ANA13	C8	C8
—	ANA14	P	ANA14	eQADC A analog input	I	AE	V_{DDA_A1}	ANA14	ANA14	B8	B8
—	ANA15	P	ANA15	eQADC A analog input	I	AE	V_{DDA_A1}	ANA15	ANA15	A8	A8
—	ANA16	P	ANA16	eQADC A analog input	I	AE	V_{DDA_A1}	ANA16	ANA16	D9	D9
—	ANA17	P	ANA17	eQADC A analog input	I	AE	V_{DDA_A1}	ANA17	ANA17	C9	C9
—	ANA18	P	ANA18	eQADC A analog input	I	AE	V_{DDA_A1}	ANA18	ANA18	D10	D10
—	ANA19	P	ANA19	eQADC A analog input	I	AE	V_{DDA_A1}	ANA19	ANA19	C10	C10
—	ANA20	P	ANA20	eQADC A analog input	I	AE	V_{DDA_A1}	ANA20	ANA20	D11	D11
—	ANA21	P	ANA21	eQADC A analog input	I	AE	V_{DDA_A1}	ANA21	ANA21	C11	C11
—	ANA22	P	ANA22	eQADC A analog input	I	AE	V_{DDA_A1}	ANA22	ANA22	D12	C12

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
—	ANA23	P	ANA23	eQADC A analog input	I	AE	V_{DDA_A1}	ANA23	ANA23	C12	D12
—	AN24	P	AN24	eQADC analog input	I	AE	V_{DDA_A0}	AN24	AN24	B12	B12
—	AN25	P	AN25	eQADC analog input	I	AE	V_{DDA_A0}	AN25	AN25	D13	C13
—	AN26	P	AN26	eQADC analog input	I	AE	V_{DDA_A0}	AN26	AN26	C13	D13
—	AN27	P	AN27	eQADC analog input	I	AE	V_{DDA_A0}	AN27	AN27	B13	B13
—	AN28	P	AN28	eQADC analog input	I	AE	V_{DDA_A0}	AN28	AN28	A13	A13
—	AN29	P	AN29	eQADC analog input	I	AE	V_{DDA_A0}	AN29	AN29	B14	A14
—	AN30	P	AN30	eQADC analog input	I	AE	V_{DDA_B1}	AN30	AN30	C14	B14
—	AN31	P	AN31	eQADC analog input	I	AE	V_{DDA_B1}	AN31	AN31	D14	C14
—	AN32	P	AN32	eQADC analog input	I	AE	V_{DDA_B1}	AN32	AN32	A14	B15
—	AN33	P	AN33	eQADC analog input	I	AE	V_{DDA_B0}	AN33	AN33	B15	D14
—	AN34	P	AN34	eQADC analog input	I	AE	V_{DDA_B0}	AN34	AN34	C15	C15
—	AN35	P	AN35	eQADC analog input	I	AE	V_{DDA_B0}	AN35	AN35	D15	D15
—	AN36	P	AN36	eQADC analog input	I	AE	V_{DDA_B1}	AN36	AN36	A15	A15
—	AN37	P	AN37	eQADC analog input	I	AE	V_{DDA_B0}	AN37	AN37	C16	C17
—	AN38	P	AN38	eQADC analog input	I	AE	V_{DDA_B0}	AN38	AN38	C17	D16
—	AN39	P	AN39	eQADC analog input	I	AE	V_{DDA_B0}	AN39	AN39	D16	C16
—	ANB0	P	ANB0	eQADC B shared analog input	I	AE/up-down	V_{DDA_B0}	ANB0	ANB0	C18	C18
—	ANB1	P	ANB1	eQADC B shared analog input	I	AE/up-down	V_{DDA_B0}	ANB1	ANB1	D17	D17
—	ANB2	P	ANB2	eQADC B shared analog input	I	AE/up-down	V_{DDA_B0}	ANB2	ANB2	D18	D18
—	ANB3	P	ANB3	eQADC B shared analog input	I	AE/up-down	V_{DDA_B0}	ANB3	ANB3	D19	D19
—	ANB4	P	ANB4	eQADC B shared analog input	I	AE/up-down	V_{DDA_B0}	ANB4	ANB4	C19	B19
—	ANB5	P	ANB5	eQADC B shared analog input	I	AE/up-down	V_{DDA_B0}	ANB5	ANB5	C20	A20

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location 416	Package Location 516
—	ANB6	P	ANB6	eQADC B shared analog input	I	AE/up-down	V_{DDA_B0}	ANB6	ANB6	B19	C20
—	ANB7	P	ANB7	eQADC B shared analog input	I	AE/up-down	V_{DDA_B0}	ANB7	ANB7	A20	C19
—	ANB8	P	ANB8	eQADC B analog input	I	AE	V_{DDA_B0}	ANB8	ANB8	B20	B20
—	ANB9	P	ANB9	eQADC B analog input	I	AE	V_{DDA_B0}	ANB9	ANB9	D20	A21
—	ANB10	P	ANB10	eQADC B analog input	I	AE	V_{DDA_B0}	ANB10	ANB10	B21	B21
—	ANB11	P	ANB11	eQADC B analog input	I	AE	V_{DDA_B0}	ANB11	ANB11	A21	C21
—	ANB12	P	ANB12	eQADC B analog input	I	AE	V_{DDA_B0}	ANB12	ANB12	C21	A22
—	ANB13	P	ANB13	eQADC B analog input	I	AE	V_{DDA_B0}	ANB13	ANB13	D21	B22
—	ANB14	P	ANB14	eQADC B analog input	I	AE	V_{DDA_B0}	ANB14	ANB14	A22	D20
—	ANB15	P	ANB15	eQADC B analog input	I	AE	V_{DDA_B0}	ANB15	ANB15	B22	C22
—	ANB16	P	ANB16	eQADC B analog input	I	AE	V_{DDA_B0}	ANB16	ANB16	C22	D21
—	ANB17	P	ANB17	eQADC B analog input	I	AE	V_{DDA_B0}	ANB17	ANB17	A23	D22
—	ANB18	P	ANB18	eQADC B analog input	I	AE	V_{DDA_B0}	ANB18	ANB18	B23	A23
—	ANB19	P	ANB19	eQADC B analog input	I	AE	V_{DDA_B0}	ANB19	ANB19	C23	B23
—	ANB20	P	ANB20	eQADC B analog input	I	AE	V_{DDA_B0}	ANB20	ANB20	D22	C23
—	ANB21	P	ANB21	eQADC B analog input	I	AE	V_{DDA_B0}	ANB21	ANB21	A24	A24
—	ANB22	P	ANB22	eQADC B analog input	I	AE	V_{DDA_B0}	ANB22	ANB22	B24	B24
—	ANB23	P	ANB23	eQADC B analog input	I	AE	V_{DDA_B0}	ANB23	ANB23	A25	E20
—	VRH_A	P	VRH_A	ADC A Voltage reference high	I	VDDINT	V_{RH_A}	VRH_A	VRH_A	A12	A12
—	VRL_A	P	VRL_A	ADC A Voltage reference low	I	VSSINT	V_{RL_A}	VRL_A	VRL_A	A11	A11
—	VRH_B	P	VRH_B	ADC B Voltage reference high	I	VDDINT	V_{RH_B}	VRH_B	VRH_B	A19	A19
—	VRL_B	P	VRL_B	ADC B Voltage reference low	I	VSSINT	V_{RL_B}	VRL_B	VRL_B	A18	A18
—	REFBYPCB	P	REFBYPCB	ADC B Reference bypass capacitor	I	AE	V_{DDA_B0}	REFBYPCB	REFBYPCB	B18	B18
—	REFBYPCA	P	REFBYPCA	ADC A Reference bypass capacitor	I	AE	V_{DDA_A1}	REFBYPCA	REFBYPCA	B11	B11
—	VDDA_A0	P	VDDA_A	Internal logic supply input	I	VDDE	V_{DDA_A0}	VDDA_A0	VDDA_A0	A9	A9

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location 416	Package Location 516
—	VDDA_A1	P	VDDA_A	Internal logic supply input	I	VDDE	V_{DDA_A1}	VDDA_A1	VDDA_A1	B9	B9
—	REFBYPCA1	P	REFBYPCA1	ADC A Reference bypass capacitor	I	AE	V_{DDA_A1}	REFBYPCA1	REFBYPCA1	A10	A10
—	VSSA_A1	P	VSSA_A	Ground	I	VSSE	V_{SSA_A1}	VSSA_A1	VSSA_A1	B10	B10
—	VDDA_B0	P	VDDA_B	Internal logic supply input	I	VDDE	V_{DDA_B0}	VDDA_B0	VDDA_B0	A16	A16
—	VDDA_B1	P	VDDA_B	Internal logic supply input	I	VDDE	V_{DDA_B1}	VDDA_B1	VDDA_B1	B16	B16
—	VSSA_B0	P	VSSA_B	Ground	I	VSSE	V_{SSA_B0}	VSSA_B0	VSSA_B0	B17	B17
—	REFBYPCB1	P	REFBYPCB1	ADC B Reference bypass capacitor	I	AE	V_{DDA_B0}	REFBYPCB1	REFBYPCB1	A17	A17
FlexRay											
248	FR_A_TX_ GPIO248	P	FR_A_TX	FlexRay A transfer	O	FS	V_{DDE2}	—/Up (–/– for Rev.1 of the device)	—/Up (–/– for Rev.1 of the device)	AD4	AD4
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO248	GPIO	I/O						
249	FR_A_RX_ GPIO249	P	FR_A_RX	FlexRay A receive	I	FS	V_{DDE2}	—/Up (–/– for Rev.1 of the device)	—/Up (–/– for Rev.1 of the device)	AE3	AE3
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO249	GPIO	I/O						
250	FR_A_TX_EN_ GPIO250	P	FR_A_TX_EN	FlexRay A transfer enable	O	FS	V_{DDE2}	—/Up (–/– for Rev.1 of the device)	—/Up (–/– for Rev.1 of the device)	AF3	AF3
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO250	GPIO	I/O						
251	FR_B_TX_ GPIO251	P	FR_B_TX	FlexRay B transfer	O	FS	V_{DDE2}	—/Up (–/– for Rev.1 of the device)	—/Up (–/– for Rev.1 of the device)	AD5	AD5
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO251	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
252	FR_B_RX_ GPIO252	P	FR_B_RX	FlexRay B receive	I	FS	V_{DDE2}	—/Up (–/– for Rev.1 of the device)	—/Up (–/– for Rev.1 of the device)	AE4	AE4
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO252	GPIO	I/O						
253	FR_B_TX_EN_ GPIO253	P	FR_B_TX_EN	FlexRay B transfer enable	O	FS	V_{DDE2}	—/Up (–/– for Rev.1 of the device)	—/Up (–/– for Rev.1 of the device)	AF4	AF4
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO253	GPIO	I/O						
FlexCAN											
83	CNTXA_TXDA_ GPIO83	P	CNTXA	FlexCAN A transmit	O	MH	V_{DDEH4}	—/Up	—/Up	AF19	AE19
		A1	TXDA	eSCI A transmit	O						
		A2	—	—	—						
		G	GPIO83	GPIO	I/O						
84	CNRXA_RXDA_ GPIO84	P	CNRXA	FlexCAN A receive	I	MH	V_{DDEH4}	—/Up	—/Up	AE19	AD19
		A1	RXDA	eSCI A receive	I						
		A2	—	—	—						
		G	GPIO84	GPIO	I/O						
85	CNTXB_PCSC3_ GPIO85	P	CNTXB	FlexCAN B transmit	O	MH	V_{DDEH4}	—/Up	—/Up	AD19	AC19
		A1	PCSC3	DSPI C peripheral chip select	O						
		A2	—	—	—						
		G	GPIO85	GPIO	I/O						
86	CNRXB_PCSC4_ GPIO86	P	CNRXB	FlexCAN B receive	I	MH	V_{DDEH4}	—/Up	—/Up	AC19	AA19
		A1	PCSC4	DSPI C peripheral chip select	O						
		A2	—	—	—						
		G	GPIO86	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
87	CNTXC_PCSD3_ GPIO87	P	CNTXC	FlexCAN C transmit	O	MH	V_{DDEH4}	—/Up	—/Up	AF20	AF20
		A1	PCSD3	DSPI D peripheral chip select	O						
		A2	—	—	—						
		G	GPIO87	GPIO	I/O						
88	CNRXC_PCSD4_ GPIO88	P	CNRXC	FlexCAN C receive	I	MH	V_{DDEH4}	—/Up	—/Up	AE20	AE20
		A1	PCSD4	DSPI D peripheral chip select	O						
		A2	—	—	—						
		G	GPIO88	GPIO	I/O						
246	CNTXD_ GPIO246	P	CNTXD	FlexCAN D transmit	O	MH	V_{DDEH4}	—/Up	—/Up	AD20	AD20
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO246	GPIO	I/O						
247	CNRXD_ GPIO247	P	CNRXD	FlexCAN D receive	I	MH	V_{DDEH4}	—/Up	—/Up	AC20	AC20
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO247	GPIO	I/O						
eSCI											
89	TXDA_ GPIO89	P	TXDA	eSCI A transmit	O	MH	V_{DDEH1}	—/Up	—/Up	M2	K2
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO89	GPIO	I/O						
90	RXDA _ GPIO90	P	RXDA	eSCI A receive	I	MH	V_{DDEH1}	—/Up	—/Up	M3	K3
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO90	GPIO	I						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
91	TXDB_PCSD1_ GPIO91	P	TXDB	eSCI B transmit	O	MH	V_{DDEH1}	—/Up	—/Up	P1	K1
		A1	PCSD1	DSPI D peripheral chip select	O						
		A2	—	—	—						
		G	GPIO91	GPIO	I/O						
92	RXDB_PCSD5_ GPIO92	P	RXDB	eSCI B receive	I	MH	V_{DDEH1}	—/Up	—/Up	N1	L5
		A1	PCSD5	DSPI D peripheral chip select	O						
		A2	—	—	—						
		G	GPIO92	GPIO	I/O						
244	TXDC_ETRIG0_ GPIO244	P	TXDC	eSCI C transmit	O	MH	V_{DDEH4}	—/Up	—/Up	AF23	AF23
		A1	ETRIG0	eQADC trigger input	I						
		A2	—	—	—						
		G	GPIO244	GPIO	I/O						
245	RXDC_ GPIO245	P	RXDC	eSCI C receive	I	MH	V_{DDEH5}	—/Up	—/Up	AD22	AD22
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO245	GPIO	I/O						
DSPI											
93	SCKA_PCSC1_ GPIO93	P	SCKA	DSPI A clock	I/O	MH	V_{DDEH3}	—/Up	—/Up	AD8	AB8
		A1	PCSC1	DSPI C peripheral chip select	O						
		A2	—	—	—						
		G	GPIO93	GPIO	I/O						
94	SINA_PCSC2_ GPIO94	P	SINA	DSPI A data input	I	MH	V_{DDEH3}	—/Up	—/Up	AF7	AE7
		A1	PCSC2	DSPI C peripheral chip select	O						
		A2	—	—	—						
		G	GPIO94	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
95	SOUTA_PCSC5_ GPIO95	P	SOUTA	DSPI A data output	O	MH	V_{DDEH3}	—/Up	—/Up	AD7	AC7
		A1	PCSC5	DSPI C peripheral chip select	O						
		A2	—	—	—						
		G	GPIO95	GPIO	I/O						
96	PCSA0_PCSD2_ GPIO96	P	PCSA0	DSPI A peripheral chip select	I/O	MH	V_{DDEH3}	—/Up	—/Up	AE6	AD6
		A1	PCSD2	DSPI D peripheral chip select	O						
		A2	—	—	—						
		G	GPIO96	GPIO	I/O						
97	PCSA1_ PCSE0_GPIO97	P	PCSA1	DSPI A peripheral chip select	O	MH	V_{DDEH3}	—/Up	—/Up	AC6	AC6
		A1		DSPI E peripheral chip select							
		A2	—	—	—						
		G	GPIO97	GPIO	I/O						
98	PCSA2_ SOUTE_GPIO98	P	PCSA2	DSPI A peripheral chip select	O	MH	V_{DDEH3}	—/Up	—/Up	AC7	AF6
		A1		DSPI E data output							
		A2	—	—	—						
		G	GPIO98	GPIO	I/O						
99	PCSA3_ SINE_GPIO99	P	PCSA3	DSPI A peripheral chip select	O	MH	V_{DDEH3}	—/Up	—/Up	AE7	AD7
		A1		DSPI E data input							
		A2	—	—	—						
		G	GPIO99	GPIO	I/O						
100	PCSA4_ SCKE_GPIO100	P	PCSA4	DSPI A peripheral chip select	O	MH	V_{DDEH3}	—/Up	—/Up	AE5	AE5
		A1		DSPI E clock							
		A2	—	—	—						
		G	GPIO100	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
101	PCSA5_ETRIG1_ GPIO101	P	PCSA5	DSPI A peripheral chip select	O	MH	V_{DDEH3}	—/Up	—/Up	AD6	AA8
		A1	ETRIG1	eQADC trigger input	I						
		A2	—	—	—						
		G	GPIO101	GPIO	I/O						
102	SCKB_ GPIO102	P	SCKB	DSPI B clock	I/O	MH	V_{DDEH3}	—/Up	—/Up	AE8	AC8
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO102	GPIO	I/O						
103	SINB_ GPIO103	P	SINB	DSPI B data input	I	MH	V_{DDEH3}	—/Up	—/Up	AE9	AB9
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO103	GPIO	I/O						
104	SOUTB_ GPIO104	P	SOUTB	DSPI B data output	O	MH	V_{DDEH3}	—/Up	—/Up	AF9	AA10
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO104	GPIO	I/O						
105	PCSB0_PCSD2_ GPIO105	P	PCSB0	DSPI B peripheral chip select	I/O	MH	V_{DDEH3}	—/Up	—/Up	AD9	AF8
		A1	PCSD2	DSPI D peripheral chip select	O						
		A2	—	—	—						
		G	GPIO105	GPIO	I/O						
106	PCSB1_PCSD0_ GPIO106	P	PCSB1	DSPI B peripheral chip select	O	MH	V_{DDEH3}	—/Up	—/Up	AC9	AE8
		A1	PCSD0	DSPI D peripheral chip select	I/O						
		A2	—	—	—						
		G	GPIO106	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
107	PCSB2_SOUTC_ GPIO107	P	PCSB2	DSPI B peripheral chip select	O	MH	V_{DDEH3}	—/Up	—/Up	AF8	AD8
		A1	SOUTC	DSPI C data output	O						
		A2	—	—	—						
		G	GPIO107	GPIO	I/O						
108	PCSB3_SINC_ GPIO108	P	PCSB3	DSPI B peripheral chip select	O	MH	V_{DDEH3}	—/Up	—/Up	AD10	AC9
		A1	SINC	DSPI C data input	I						
		A2	—	—	—						
		G	GPIO108	GPIO	I/O						
109	PCSB4_SCKC_ GPIO109	P	PCSB4	DSPI B peripheral chip select	O	MH	V_{DDEH3}	—/Up	—/Up	AC8	AF7
		A1	SCKC	DSPI C clock	I/O						
		A2	—	—	—						
		G	GPIO109	GPIO	I/O						
110	PCSB5_PCSC0_ GPIO110	P	PCSB5	DSPI B peripheral chip select	O	MH	V_{DDEH3}	—/Up	—/Up	AF6	AE6
		A1	PCSC0	DSPI C peripheral chip select	I/O						
		A2	—	—	—						
		G	GPIO110	GPIO	I/O						
235	SCKC_SCK_C_LVDSP_ GPIO235	P	SCKC	DSPI C clock	I/O	MH+ LVDS	V_{DDEH4}	—/Up	—/Up	AD21	AD21
		A1	SCK_C_LVDSP	LVDS+ downstream signal positive output clock	O						
		A2	—	—	—						
		G	GPIO235	GPIO	I/O						
236	SINC_SCK_C_LVDSM_ GPIO236	P	SINC	DSPI C data input	I	MH+ LVDS	V_{DDEH4}	—/Up	—/Up	AE22	AE22
		A1	SCK_C_LVDSM	LVDS– downstream signal negative output clock	O						
		A2	—	—	—						
		G	GPIO236	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
237	SOUTC_SOUT_C_LVDSP_ GPIO237	P	SOUTC	DSPI C data output	O	MH+ LVDS	V_{DDEH4}	—/Up	—/Up	AF21	AF21
		A1	SOUT_C_LVDSP	LVDS+ downstream signal positive output data	O						
		A2	—	—	—						
		G	GPIO237	GPIO	I/O						
238	PCSC0_SOUT_C_LVDSM_ GPIO238	P	PCSC0	DSPI C peripheral chip select	I/O	MH+ LVDS	V_{DDEH4}	—/Up	—/Up	AE21	AE21
		A1	SOUT_C_LVDSM	LVDS– downstream signal negative output data	O						
		A2	—	—	—						
		G	GPIO238	GPIO	I/O						
239	PCSC1_ GPIO239	P	PCSC1	DSPI C peripheral chip select	O	MH	V_{DDEH4}	—/Up	—/Up	AC22	AC22
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO239	GPIO	I/O						
240	PCSC2_GPIO240	P	PCSC2	DSPI C peripheral chip select	O	MH	V_{DDEH5}	—/Up	—/Up	AE23	AE23
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO240	GPIO	I/O						
241	PCSC3_GPIO241	P	PCSC3	DSPI C peripheral chip select	O	MH	V_{DDEH5}	—/Up	—/Up	AD23	AD23
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO241	GPIO	I/O						
242	PCSC4_GPIO242	P	PCSC4	DSPI C peripheral chip select	O	MH	V_{DDEH5}	—/Up	—/Up	AF24	AF24
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO242	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
243	PCSC5_GPIO243	P	PCSC5	DSPI C peripheral chip select	O	MH	V_{DDEH5}	—/Up	—/Up	AE24	AE24
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO243	GPIO	I/O						
EBI											
256	D_CS0_ GPIO256	P	D_CS0	EBI chip select 0	O	F	V_{DDE9}	—/Up	—/Up	—	AD9
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO256	GPIO	I/O						
257	D_CS2_D_ADD_DAT31_ GPIO257	P	D_CS2	EBI chip select 2	O	F	V_{DDE8}	—/Up	—/Up	—	U1
		A1	D_ADD_DAT31	Address and data in mux mode.	I/O						
		A2	—	—	—						
		G	GPIO257	GPIO	I/O						
258	D_CS3_D_TEA_ GPIO258	P	D_CS3	EBI chip select 3	O	F	V_{DDE8}	—/Up	—/Up	—	T6
		A1	D_TEA	EBI transfer error acknowledge	I/O						
		A2	—	—	—						
		G	GPIO258	GPIO	I/O						
259	D_ADD12_ GPIO259	P	D_ADD12	EBI address bus	O	F	V_{DDE8}	—/Up	—/Up	—	R1
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO259	GPIO	I/O						
260	D_ADD13_ GPIO260	P	D_ADD13	EBI address bus	O	F	V_{DDE8}	—/Up	—/Up	—	R2
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO260	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
261	D_ADD14_ GPIO261	P	D_ADD14	EBI address bus	O	F	V_{DDE8}	—/Up	—/Up	—	R3
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO261	GPIO	I/O						
262	D_ADD15_ GPIO262	P	D_ADD15	EBI address bus	O	F	V_{DDE8}	—/Up	—/Up	—	R4
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO262	GPIO	I/O						
263	D_ADD16_D_ADD_DAT16_ GPIO263	P	D_ADD16	EBI address bus	O	F	V_{DDE8}	—/Up	—/Up	—	R5
		A1	D_ADD_DAT16	Address and data in mux mode.	I/O						
		A2	—	—	—						
		G	GPIO263	GPIO	I/O						
264	D_ADD17_D_ADD_DAT17_ GPIO264	P	D_ADD17	EBI address bus	O	F	V_{DDE8}	—/Up	—/Up	—	T5
		A1	D_ADD_DAT17	Address and data in mux mode.	I/O						
		A2	—	—	—						
		G	GPIO264	GPIO	I/O						
265	D_ADD18_D_ADD_DAT18_ GPIO265	P	D_ADD18	EBI address bus	O	F	V_{DDE8}	—/Up	—/Up	—	T2
		A1	D_ADD_DAT18	Address and data in mux mode.	I/O						
		A2	—	—	—						
		G	GPIO265	GPIO	I/O						
266	D_ADD19_D_ADD_DAT19_ GPIO266	P	D_ADD19	EBI address bus	O	F	V_{DDE8}	—/Up	—/Up	—	T3
		A1	D_ADD_DAT19	Address and data in mux mode.	I/O						
		A2	—	—	—						
		G	GPIO266	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
267	D_ADD20_D_ADD_DAT20_ GPIO267	P	D_ADD20	EBI address bus	O	F	V_{DDE8}	—/Up	—/Up	—	T4
		A1	D_ADD_DAT20	Address and data in mux mode.	I/O						
		A2	—	—	—						
		G	GPIO267	GPIO	I/O						
268	D_ADD21_D_ADD_DAT21_ GPIO268	P	D_ADD21	EBI address bus	O	F	V_{DDE9}	—/Up	—/Up	—	AB11
		A1	D_ADD_DAT21	Address and data in mux mode.	I/O						
		A2	—	—	—						
		G	GPIO268	GPIO	I/O						
269	D_ADD22_D_ADD_DAT22_ GPIO269	P	D_ADD22	EBI address bus	O	F	V_{DDE9}	—/Up	—/Up	—	AD10
		A1	D_ADD_DAT22	Address and data in mux mode.	I/O						
		A2	—	—	—						
		G	GPIO269	GPIO	I/O						
270	D_ADD23_D_ADD_DAT23_ GPIO270	P	D_ADD23	EBI address bus	O	F	V_{DDE9}	—/Up	—/Up	—	AE10
		A1	D_ADD_DAT23	Address and data in mux mode.	I/O						
		A2	—	—	—						
		G	GPIO270	GPIO	I/O						
271	D_ADD24_D_ADD_DAT24_ GPIO271	P	D_ADD24	EBI address bus	O	F	V_{DDE9}	—/Up	—/Up	—	AF10
		A1	D_ADD_DAT24	Address and data in mux mode.	I/O						
		A2	—	—	—						
		G	GPIO271	GPIO	I/O						
272	D_ADD25_D_ADD_DAT25_ GPIO272	P	D_ADD25	EBI address bus	O	F	V_{DDE9}	—/Up	—/Up	—	AD11
		A1	D_ADD_DAT25	Address and data in mux mode.	I/O						
		A2	—	—	—						
		G	GPIO272	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
273	D_ADD26_D_ADD_DAT26_ GPIO273	P	D_ADD26	EBI address bus	O	F	V_{DDE9}	—/Up	—/Up	—	AE11
		A1	D_ADD_DAT26	Address and data in mux mode.	I/O						
		A2	—	—	—						
		G	GPIO273	GPIO	I/O						
274	D_ADD27_D_ADD_DAT27_ GPIO274	P	D_ADD27	EBI address bus	O	F	V_{DDE9}	—/Up	—/Up	—	AF11
		A1	D_ADD_DAT27	Address and data in mux mode.	I/O						
		A2	—	—	—						
		G	GPIO274	GPIO	I/O						
275	D_ADD28_D_ADD_DAT28_ GPIO275	P	D_ADD28	EBI address bus	O	F	V_{DDE9}	—/Up	—/Up	—	AD12
		A1	D_ADD_DAT28	Address and data in mux mode.	I/O						
		A2	—	—	—						
		G	GPIO275	GPIO	I/O						
276	D_ADD29_D_ADD_DAT29_ GPIO276	P	D_ADD29	EBI address bus	O	F	V_{DDE9}	—/Up	—/Up	—	AB12
		A1	D_ADD_DAT29	Address and data in mux mode.	I/O						
		A2	—	—	—						
		G	GPIO276	GPIO	I/O						
277	D_ADD30_D_ADD_DAT30_ GPIO277	P	D_ADD30	EBI address bus	O	F	V_{DDE9}	—/Up	—/Up	—	AE12
		A1	D_ADD_DAT30	Address and data in mux mode.	I/O						
		A2	—	—	—						
		G	GPIO277	GPIO	I/O						
278	D_ADD_DAT0_ GPIO278	P	D_ADD_DAT0	EBI data only in non-mux mode. Address and data in mux mode.	I/O	F	V_{DDE10}	—/Up	—/Up	—	P25
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO278	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
279	D_ADD_DAT1_ GPIO279	P	D_ADD_DAT1	EBI data only in non-mux mode. Address and data in mux mode.	I/O	F	V_{DDE10}	—/Up	—/Up	—	P26
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO279	GPIO	I/O						
280	D_ADD_DAT2_ GPIO280	P	D_ADD_DAT2	EBI data only in non-mux mode. Address and data in mux mode.	I/O	F	V_{DDE10}	—/Up	—/Up	—	N24
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO280	GPIO	I/O						
281	D_ADD_DAT3_ GPIO281	P	D_ADD_DAT3	EBI data only in non-mux mode. Address and data in mux mode.	I/O	F	V_{DDE10}	—/Up	—/Up	—	N25
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO281	GPIO	I/O						
282	D_ADD_DAT4_ GPIO282	P	D_ADD_DAT4	EBI data only in non-mux mode. Address and data in mux mode.	I/O	F	V_{DDE10}	—/Up	—/Up	—	N26
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO282	GPIO	I/O						
283	D_ADD_DAT5_ GPIO283	P	D_ADD_DAT5	EBI data only in non-mux mode. Address and data in mux mode.	I/O	F	V_{DDE10}	—/Up	—/Up	—	M25
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO283	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
284	D_ADD_DAT6_ GPIO284	P	D_ADD_DAT6	EBI data only in non-mux mode. Address and data in mux mode.	I/O	F	V_{DDE10}	—/Up	—/Up	—	N22
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO284	GPIO	I/O						
285	D_ADD_DAT7_ GPIO285	P	D_ADD_DAT7	EBI data only in non-mux mode. Address and data in mux mode.	I/O	F	V_{DDE10}	—/Up	—/Up	—	M24
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO285	GPIO	I/O						
286	D_ADD_DAT8_ GPIO286	P	D_ADD_DAT8	EBI data only in non-mux mode. Address and data in mux mode.	I/O	F	V_{DDE10}	—/Up	—/Up	—	M23
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO286	GPIO	I/O						
287	D_ADD_DAT9_ GPIO287	P	D_ADD_DAT9	EBI data only in non-mux mode. Address and data in mux mode.	I/O	F	V_{DDE10}	—/Up	—/Up	—	M22
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO287	GPIO	I/O						
288	D_ADD_DAT10_ GPIO288	P	D_ADD_DAT10	EBI data only in non-mux mode. Address and data in mux mode.	I/O	F	V_{DDE10}	—/Up	—/Up	—	L26
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO288	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
289	D_ADD_DAT11_ GPIO289	P	D_ADD_DAT11	EBI data only in non-mux mode. Address and data in mux mode.	I/O	F	V_{DDE10}	—/Up	—/Up	—	L25
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO289	GPIO	I/O						
290	D_ADD_DAT12_ GPIO290	P	D_ADD_DAT12	EBI data only in non-mux mode. Address and data in mux mode.	I/O	F	V_{DDE10}	—/Up	—/Up	—	L24
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO290	GPIO	I/O						
291	D_ADD_DAT13 _GPIO291	P	D_ADD_DAT13	EBI data only in non-mux mode. Address and data in mux mode.	I/O	F	V_{DDE10}	—/Up	—/Up	—	L23
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO291	GPIO	I/O						
292	D_ADD_DAT14_GPIO292	P	D_ADD_DAT14	EBI data only in non-mux mode. Address and data in mux mode.	I/O	F	V_{DDE10}	—/Up	—/Up	—	L22
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO292	GPIO	I/O						
293	D_ADD_DAT15_GPIO293	P	D_ADD_DAT15	EBI data only in non-mux mode. Address and data in mux mode.	I/O	F	V_{DDE10}	—/Up	—/Up	—	K26
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO293	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
294	D_RD_WR_GPIO294	P	D_RD_WR	EBI read/write	O	F	V_{DDE10}	—/Up	—/Up	—	R26
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO294	GPIO	I/O						
295	D_WE0_GPIO295	P	D_WE0	EBI write enable	O	F	V_{DDE8}	—/Up	—/Up	—	N1
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO295	GPIO	I/O						
296	D_WE1_GPIO296	P	D_WE1	EBI write enable	O	F	V_{DDE8}	—/Up	—/Up	—	P5
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO296	GPIO	I/O						
297	D_OE_GPIO297	P	D_OE	EBI output enable	O	F	V_{DDE10}	—/Up	—/Up	—	P23
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO297	GPIO	I/O						
298	D_TS_GPIO298	P	D_TS	EBI transfer start	O	F	V_{DDE9}	—/Up	—/Up	—	AE9
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO298	GPIO	I/O						
299	D_ALE_GPIO299	P	D_ALE	EBI Address Latch Enable	O	F	V_{DDE10}	—/Up	—/Up	—	P24
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO299	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
300	D_TA_GPIO300	P	D_TA	EBI transfer acknowledge	I/O	F	V_{DDE9}	—/Up	—/Up	—	AF9
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO300	GPIO	I/O						
301	D_CS1_GPIO301	P	D_CS1	EBI chip select	O	F	V_{DDE9}	—/Up	—/Up	—	AB10
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO301	GPIO	I/O						
302	D_BDIP_GPIO302	P	D_BDIP	EBI burst data in progress	O	F	V_{DDE8}	—/Up	—/Up	—	M2
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO302	GPIO	I/O						
303	D_WE2_GPIO303	P	D_WE2	EBI write enable	O	F	V_{DDE8}	—/Up	—/Up	—	N2
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO303	GPIO	I/O						
304	D_WE3_GPIO304	P	D_WE3	EBI write enable	O	F	V_{DDE8}	—/Up	—/Up	—	N3
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO304	GPIO	I/O						
305	D_ADD9_GPIO305	P	D_ADD9	EBI address bus	O	F	V_{DDE8}	—/Up	—/Up	—	P1
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO305	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
306	D_ADD10_GPIO306	P	D_ADD10	EBI address bus	O	F	V_{DDE8}	—/Up	—/Up	—	P2
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO306	GPIO	I/O						
307	D_ADD11_GPIO307	P	D_ADD11	EBI address bus	O	F	V_{DDE8}	—/Up	—/Up	—	P3
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO307	GPIO	I/O						
Reset and Clocks											
—	RESET	P	RESET	External reset input	I	MH	V_{DDEH1}	RESET/Up	RESET/Up	R2	N5
230	RSTOUT	P	RSTOUT	External reset output	O	MH	V_{DDEH1}	RSTOUT/Low	RSTOUT/ High	A3	A3
211	BOOTCFG0_IRQ2_ GPIO211	P	BOOTCFG0	Boot configuration	I	MH	V_{DDEH1}	BOOTCFG/ Down	—/Down	—	L4
		A1	IRQ2		I						
		A2	—	—	—						
		G	GPIO211	GPIO	I/O						
212	BOOTCFG1_IRQ3_ GPIO212	P	BOOTCFG1	Boot configuration	I	MH	V_{DDEH1}	BOOTCFG/ Down	—/Down	N2	L3
		A1	IRQ3	External interrupt request	I						
		A2	—	—	—						
		G	GPIO212	GPIO	I/O						
213	WKPCFG_NMI_ GPIO213[10]	P	WKPCFG	Weak pull configuration input	I	MH	V_{DDEH1}	WKPCFG/Up	—/Up	N3	M5
		A1									
		A2	—	—	—						
		G	GPIO213	GPIO	I						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
208	PLLCFG0_IRQ4_ GPIO208	P	PLLCFG0	FMPLL mode configuration input	I	MH	V_{DDEH1}	PLLCFG/Up	—/Up	R3	M3
		A1	IRQ4	External interrupt request	I						
		A2	—	—	—						
		G	GPIO208	GPIO	I/O						
209	PLLCFG1_IRQ5_GPIO209	P	PLLCFG1	FMPLL mode configuration input	I	MH	V_{DDEH1}	PLLCFG/Up	—/Up	P2	L1
		A1	IRQ5	External interrupt request	I						
		A2	SOUTD	DSPI D data output	O						
		G	GPIO209	GPIO	I/O						
—	PLLCFG2	P	PLLCFG2	FMPLL mode configuration input	I	MH	V_{DDEH1}	PLLCFG/ Down	—/ Down	P3	L2
—	XTAL	P	XTAL	Crystal oscillator output	O	AE	V_{DD33}	XTAL	XTAL	AC26	AC26
—	EXTAL	P	EXTAL	Crystal oscillator input	I	AE	V_{DD33}	EXTAL	EXTAL	AB26	AB26
229	D_CLKOUT	P	D_CLKOUT	EBI system clock output	O	F	V_{DDE9}	CLKOUT/ Enabled	CLKOUT/ Enabled	—	AF12
214	ENGCLK	P	ENGCLK	EBI engineering clock output Note: EXTCLK (External clock input) selected through SIU register)	O	F	V_{DDE2}	ENGCLK/ Enabled	ENGCLK/ Enabled	AD1	AD1
JTAG and Nexus **(see footnote[11] about resets)**											
—	$\overline{\text{EVTI}}$	–[12]	EVTI	Nexus event in	I	F	V_{DDE2}	—/Up	EVTI/Up	T4	V1
227	$\overline{\text{EVTO}}$ (the BAM uses this pin to select if auto baud rate is on or off)	–[12]	EVTO	Nexus event out	O	F	V_{DDE2}	ABS/Up	EVTO/HI	U1	V2
219	MCKO	–[12]	MCKO	Nexus message clock out	O	F	V_{DDE2}	O/Low	Disabled[13]	T2	U4
220	MDO0_GPIO220	–[12]	MDO0[14]	Nexus message data out	O	F	V_{DDE2}	See Note[15]	See Note[15]	U3	V3
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO220	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
221	MDO1_GPIO221	–[12]	MDO1[14]	Nexus message data out	O	F	V_{DDE2}	O/Low	—/Down	U4	W6
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO221	GPIO	I/O						
222	MDO2_GPIO222	–[12]	MDO2[14]	Nexus message data out	O	F	V_{DDE2}	O/Low	—/Down	V1	V4
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO222	GPIO	I/O						
223	MDO3_GPIO223	–[12]	MDO3[14]	Nexus message data out	O	F	V_{DDE2}	O/Low	—/Down	V2	V5
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO223	GPIO	I/O						
75	MDO4_GPIO75	–[12]	MDO4[14]	Nexus message data out	O	F	V_{DDE2}	O/Low	—/Down	V3	W1
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO75	GPIO	I/O						
76	MDO5_GPIO76	–[12]	MDO5[14]	Nexus message data out	O	F	V_{DDE2}	O/Low	—/Down	V4	W2
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO76	GPIO	I/O						
77	MDO6_GPIO77	–[12]	MDO6[14]	Nexus message data out	O	F	V_{DDE2}	O/Low	—/Down	W1	W3
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO77	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
78	MDO7_GPIO78	–[12]	MDO7[14]	Nexus message data out	O	F	V_{DDE2}	O/Low	—/Down	W2	Y1
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO78	GPIO	I/O						
79	MDO8_GPIO79	–[12]	MDO8[14]	Nexus message data out	O	F	V_{DDE2}	O/Low	—/Down	W3	W5
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO79	GPIO	I/O						
80	MDO9_GPIO80	–[12]	MDO9[14]	Nexus message data out	O	F	V_{DDE2}	O/Low	—/Down	Y1	Y2
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO80	GPIO	I/O						
81	MDO10_GPIO81	–[12]	MDO10[14]	Nexus message data out	O	F	V_{DDE2}	O/Low	—/Down	Y2	Y3
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO81	GPIO	I/O						
82	MDO11_GPIO82	–[12]	MDO11[14]	Nexus message data out	O	F	V_{DDE2}	O/Low	—/Down	Y3	Y4
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO82	GPIO	I/O						
231	MDO12_GPIO231	–[12]	MDO12[14]	Nexus message data out	O	F	V_{DDE2}	O/Low	—/Down	AA1	Y5
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO231	GPIO	I/O						

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
232	MDO13_GPIO232	–[12]	MDO13[14]	Nexus message data out	O	F	V_{DDE2}	O/Low	—/Down	AA2	AA1
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO232	GPIO	I/O						
233	MDO14_GPIO233	–[12]	MDO14[14]	Nexus message data out	O	F	V_{DDE2}	O/Low	—/Down	AA3	AA2
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO233	GPIO	I/O						
234	MDO15_GPIO234	–[12]	MDO15[14]	Nexus message data out	O	F	V_{DDE2}	O/Low	—/Down	Y4	AA3
		A1	—	—	—						
		A2	—	—	—						
		G	GPIO234	GPIO	I/O						
224	$\overline{\text{MSEO0}}$	–[12]	MSEO0[14]	Nexus message start/end out	O	F	V_{DDE2}	O/Low	MSEO/HI	U2	U6
225	$\overline{\text{MSEO1}}$	–[12]	MSEO1[14]	Nexus message start/end out	O	F	V_{DDE2}	O/Low	MSEO/HI	T3	U5
226	RDY	–[12]	RDY	Nexus ready output	O	F	V_{DDE2}	O/Low	RDY/HI	R4	U3
—	TCK	–[12]	TCK	JTAG test clock input	I	F	V_{DDE2}	TCK/Down	TCK/Down	AB2	AB2
—	TDI	–[12]	TDI	JTAG test data input	I	F	V_{DDE2}	TDI/Up	TDI/Up	AC2	AC2
228	TDO	–[12]	TDO	JTAG test data output	O	F	V_{DDE2}	TDO/Up	TDO/Up	AB1	AB1
—	TMS	–[12]	TMS	JTAG test mode select input	I	F	V_{DDE2}	TMS/Up	TMS/Up	AB3	AB3
—	JCOMP	–[12]	JCOMP	JTAG TAP controller enable	I	F	V_{DDE2}	JCOMP/Down	JCOMP/Down	R1	U2
—	TEST	—	TEST	Test mode select (not for customer use)	I	F	V_{DDEH1}	TEST/Down	TEST/Down	B4	B4
—	VDDSYN	—	VDDSYN	Clock synthesizer power input	I/O	VDDE	V_{DDSYN}	VDDSYN	VDDSYN	AD26	AD26
—	VSSSYN	—	VSSSYN	Clock synthesizer ground input	I	VSSE	V_{DDSYN}	VSSSYN	VSSSYN	AA26	AA26
—	VSTBY	—	VSTBY	SRAM standby power input	I	VHV	V_{DDEH1}	VSTBY	VSTBY	M4	M4
—	REGSEL	—	REGSEL	Selects regulator mode (Linear/Switch mode)	I	AE	V_{DDREG}	REGSEL	REGSEL	W23	W23

Table 2-4. Signal Properties and Muxing Summary (continued)

GPIO/PCR[1]	Signal Name[2]	P/A/G[3]	Function[4]	Function Summary	Direction	Pad Type[5]	Voltage[6]	State during RESET[7]	State after RESET[8]	Package Location	
										416	516
—	REGCTL	—	REGCTL	Regulator controller output to base/gate of power transistor	O	AE	V_{DDREG}	REGCTL	REGCTL	Y26	Y26
—	VSSFL	—	VSSFL	Tie to V_{SS}	I	VSS	V_{DDREG}	VSSFL	VSSFL	AB25	AB25
—	VDDREG	—	VDDREG	Source voltage for on-chip regulators and Low voltage detect circuits	I	VDDINT	V_{DDREG}	VDDREG	VDDREG	AA25	AA25

[1] The GPIO number is the same as the corresponding pad configuration register (SIU_PCRn) number in pins that have GPIO functionality. For pins that do not have GPIO functionality, this number is the PCR number.

[2] The primary signal name is used as the pin label on the BGA map for identification purposes. However, the primary signal function is not available on all devices and is indicated by a dash in the following table columns: Signal Functions, P/F/G, and I/O Type.

[3] P/A/G stands for Primary/Alternate/GPIO . This column indicates which function on a pin is Primary, Alternate 1, Alternate 2, (Alternate *n*) and GPIO.

[4] Each line in the Function column corresponds to a separate signal function on the pin. For all device I/O pins, the primary, alternate, or GPIO signal functions are designated in the PA field of the SIU_PCRn registers except where explicitly noted.

[5] MH = High voltage, medium speed
F = Fast speed
FS = Fast speed with slew
AE = Analog with ESD protection circuitry (up/down = pull up and pull down circuits included in the pad)
VHV = Very high voltage

[6] VDDE (fast I/O) and VDDEH (slow I/O) power supply inputs are grouped into segments. Each segment of VDDEH pins can connect to a separate 3.3–5.0 V (+5%/–10%) power supply input. Each segment of VDDE pins can connect to a separate 1.8–3.3 V (±10%) power supply.

[7] All pins are sampled after the internal POR is negated. The terminology used in this column is: O – output, I – input, Up – weak pull up enabled, Down – weak pulldown enabled, Low – output driven low, High – output driven high, ABS — Auto Baud Select (during Reset or until JCOMP assertion). A dash on the left side of the slash denotes that both the input and output buffers for the pin are off. A dash on the right side of the slash denotes that there is no weak pull up/down enabled on the pin. The signal name to the left or right of the slash indicates the pin is enabled.

[8] The Function After Reset of a GPI function is general purpose input. A dash on the left side of the slash denotes that both the input and output buffers for the pin are off. A dash on the right side of the slash denotes that there is no weak pull up/down enabled on the pin.

[9] During and just after POR negates, internal pull resistors can be enabled, resulting in as much as 4 mA of current draw. The pull resistors are disabled when the system clock propagates through the device.

[10] NMI function is selected using the SIU_IREER/SIU_IFEER registers and has priority over any other function on this pin.

[11] Nexus reset is different than system reset; MDO0-11 are enabled in RPM or FPM trace modes, while MDO12-15 are enabled in FPM trace mode only. MSEO and MCKO are also dependent on trace (RPM or FPM) being enabled.

[12] The Nexus pins don't have a "primary" function as they are not configured by the SIU. The pins are selected by asserting JCOMP and configuring the NPC. SIU values have no effect on the function of these pins once enabled.

[13] MCKO is disabled from reset; it can be enabled from the tool (controlled by Nexus NPC_PCR register).

[14] Do not connect pin directly to a power supply or ground.

[15] While JCOMP is negated, the MDO0 pad is pulled up because of the default values in its SIU PCR. When JCOMP is asserted, the MDO0 pad is enabled as an output and goes low when the system clock is present.

Table 2-5 lists the pin locations of the power and ground signals on the 416 TEPBGA package.

Table 2-5. 416-pin Power Supply Locations

VDD

A2	B3	C4	D5	N4	AB4	AB23	AC3	AC12	AC24	AD2	AD25	AE1	AE26

VDD33

M1	AA4	AA23

VDDE2

N10	P10	P11	R10	R11	T1	T10	T11	T12	U10	U11	U12	W4	AC1	AC5	AF2

VDDEH1

B1	P4

VDDEH3

AC10	AF5

VDDEH4

AC11	AF22

VDDEH5

AC21	AF25

VDDEH6

N23	AC25

VDDEH7

D24	E23	M26

VSS

A1	A26	B2	B25	C3	C24	D4	D23	K10	K11	K12	K13	K14	K15	K16	K17	L10	L11
L12	L13	L14	L15	L16	L17	M10	M11	M12	M13	M14	M15	M16	M17	N11	N12	N13	N14
N15	N16	N17	P12	P13	P14	P15	P16	P17	R12	R13	R14	R15	R16	R17	T13	T14	T15
T16	T17	U13	U14	U15	U16	U17	AC4	AC23	AD3	AD24	AE2	AE25	AF1	AF26			

Table 2-6 lists the pin locations of the power and ground signals on the 516 TEPBGA package.

Table 2-6. 516-pin Power Supply Locations

VDD

A2	B3	C4	D5	E6	N4	AB4	AB23	AC3	AC12	AC24	AD2	AD25	AE1	AE26

VDD33

M1	P6	L21	AA4	AA11	AA14	AA23

VDDE10

F16	F17	F19	F21	N21	P21	AA22

VDDE2

N10	P10	P11	R10	R11	T1	T10	T11	T12	U10	U11	U12	W4	AC1	AC5	AF2

VDDE8

F6	F8	F10	F11	N6	AA5

VDDE9

AA13	AB6	AB7	AB18	AB19	AB20	AB21

VDDEH1

B1	P4

VDDEH3

AC10	AF5

VDDEH4

AC11	AF22

VDDEH5

AC21	AF25

VDDEH6

N23	AC25

VDDEH7

D24	E23	M26

VSS

A25	B2	B25	B26	C3	C24	D4	D23	E5	E7	E8	E9	E10	E11	E12	E13	E14	E15
E16	E17	E18	E19	E21	E22	F5	F13	F14	K10	K11	K12	K13	K14	K15	K16	K17	L10
L11	L12	L13	L14	L15	L16	L17	M10	M11	M12	M13	M14	M15	M16	M17	N11	N12	N13
N14	N15	N16	N17	P12	P13	P14	P15	P16	P17	R12	R13	R14	R15	R16	R17	T13	T14
T15	T16	T17	U13	U14	U15	U16	U17	AA6	AA21	AB5	AB22	AC4	AC23	AD3	AD24	AE2	AE25

Refer to the *MPC5676R Microcontroller Data Sheet* for ball-map figures.

2.3 Detailed Signal Description

This section provides detailed descriptions of the signal functions available for the device.

2.3.1 eTPU Signals

Table 2-7. eTPU Signals

Signal Name	Description
TCRCLKA_$\overline{\text{IRQ7}}$_GPIO113	TCRCLKA_$\overline{\text{IRQ}}$[7]_GPIO[113] is the TCR clock input for the eTPU A module. The first alternate function is an external interrupt request input for the SIU module.
ETPUA[0:11]_ETPUA[12:23]_GPIO[114:125]	ETPUA[0:11]_ETPUA[12:23]_GPIO[114:125] is an input/output channel pin for the eTPU A module. ETPUA[n] is the primary function and is an input/output channel for the eTPU A module. The alternate function, ETPUA[m], is an output channel for the eTPU A module. When configured as ETPUA[m], the pin functions as output only.
ETPUA12_PCSB1_GPIO126	Input/output channel pin for the eTPU A module. The first alternate function is a peripheral chip select for the DSPI B module.
ETPUA13_PCSB3_GPIO127	
ETPUA14_PCSB4_GPIO128	
ETPUA15_PCSB5_GPIO129	
ETPUA16_PCSD1_GPIO130	Input/output channel pin for the eTPU A module. The first alternate function is a peripheral chip select for the DSPI D module.
ETPUA17_PCSD2_GPIO131	
ETPUA18_PCSD3_GPIO132	
ETPUA19_PCSD4_GPIO133	
ETPUA20_$\overline{\text{IRQ8}}$_GPIO134	Input/output channel pin for the eTPU A module. The first alternate function is an external interrupt request input for the SIU module.
ETPUA21_$\overline{\text{IRQ9}}$_GPIO135	
ETPUA22_$\overline{\text{IRQ10}}$_GPIO136	
ETPUA23_$\overline{\text{IRQ11}}$_GPIO137	
ETPUA24_$\overline{\text{IRQ12}}$_GPIO138	
ETPUA25_$\overline{\text{IRQ13}}$_GPIO139	
ETPUA26_$\overline{\text{IRQ14}}$_GPIO140	
ETPUA27_$\overline{\text{IRQ15}}$_GPIO141	
ETPUA28_PCSC1_GPIO142	Input/output channel pin for the eTPU A module. The first alternate function is a peripheral chip select for the DSPI C module.
ETPUA29_PCSC2_GPIO143	
ETPUA30_PCSC3_GPIO144	
ETPUA31_PCSC4_GPIO145	
TCRCLKB_$\overline{\text{IRQ6}}$_GPIO146	TCRCLKB_$\overline{\text{IRQ}}$[6]_GPIO[146] is the TCR B clock input for the eTPU module. The alternate function is an external interrupt request input for the SIU module.
ETPUB[0:15]_ETPUB[16:31]_GPIO[147:162]	ETPUB[0:15]_ETPUB[16:31]_GPIO[147:162] are 16 input/output channel pins for the eTPU B module. The alternate functions are output channels for the eTPU B module; that is, when configured as ETPUB[16:31], the pins function as outputs only.
ETPUB[16:19]_PCSA[1:4]_GPIO[163:166]	ETPUB[16:19]_PCSA[1:4]_GPIO[163:166] are input/output channel pins for the eTPU B module. The alternate functions are peripheral chip select signals for DSPI A.

Table 2-7. eTPU Signals

Signal Name	Description
ETPUB[20:31]_GPIO[167:178]	ETPUB[20:31]_GPIO[167:178] are input/output channel pins for the eTPU B module.
TCRCLKC_GPIO440	TCRCLKC_GPIO[440] is the TCR clock input for the eTPU_C module.
ETPUC[0:3]_GPIO[441:444]	Input/output channel pins for the eTPU_C module.
ETPUC[4:8]_PCSE[1:5]_ GPIO[445:449]	Input/output channel pins for the eTPU_C module. The alternate function is a peripheral chip select for the DSPI_E module.
ETPUC[9:14]_IRQ[0:5]_ GPIO[450:455]	Input/output channel pins for the eTPU_C module. The alternate function is an external interrupt request input for the SIU module.
ETPUC15_GPIO456	Input/output channel pin for the eTPU_C module.
ETPUC16_FR_A_TX_ GPIO457	Input/output channel pins for the eTPU_C module. The alternate function is FlexRay Channel A transmit.
ETPUC17_FR_A_RX_ GPIO458	Input/output channel pin for the eTPU_C module. The alternate function is FlexRay Channel A receive.
ETPUC18_FR_A_TX_EN_ GPIO459	Input/output channel pin for the eTPU_C module. The alternate function is FlexRay Channel A transmit enable.
ETPUC19_TXDA_GPIO460	Input/output channel pin for the eTPU_C module. The alternate function is the transmit pin for the eSCI A module
ETPUC20_RXDA _GPIO461	Input/output channel pin for the eTPU_C module. The alternate function is the receive pin for the eSCI A module
ETPUC21_TXDB_GPIO462	Input/output channel pin for the eTPU_C module. The alternate function is the transmit pin for the eSCI B module
ETPUC22_RXDB_GPIO463	Input/output channel pin for the eTPU_C module. The alternate function is the receive pin for the eSCI B module
ETPUC[23:28]_PCSD[5:0]_ GPIO[464:469]	Input/output channel pins for the eTPU_C module. The alternate function is a peripheral chip select for the DSPI D module.
ETPUC29_SCKD_GPIO470	Input/output channel pin for the eTPU_C module. The alternate function is a DSPI clock pin for the DSPI D module.
ETPUC30_SOUTD_GPIO471	Input/output channel pin for the eTPU_C module. The alternate function is a data output pin for the DSPI D module.
ETPUC31_SIND_GPIO472	Input/output channel pin for the eTPU_C module. The alternate function is a data input pin for the DSPI D module.

2.3.2 IRQ Signals

All IRQ signals are alternate functions on the signal names listed in Table 2-8. IRQ signals are inputs to the SIU module.

Table 2-8. IRQ Signals

Signal Name	Description
TCRCLKA_$\overline{\text{IRQ7}}$_GPIO113	Primary function is TCR clock input for the eTPU_A module.
ETPUA[20:27]_$\overline{\text{IRQ[8:15]}}$_GPIO[134:141]	Primary functions are input/output channel pins for the eTPU_A module.
TCRCLKB_$\overline{\text{IRQ6}}$_GPIO146	Primary function is the TCR_B clock input for the eTPU module.

Table 2-8. IRQ Signals

Signal Name	Description
ETPUC[9:14]_$\overline{\text{IRQ}}$[0:5]_GPIO[450:455]	Primary function is input/output channel pin for the eTPU_C module.
EMIOS[14:15]_$\overline{\text{IRQ}}$[0:1]_GPIO[193:194]	Primary functions are output channel pins for the eMIOS module.
BOOTCFG[0:1]_$\overline{\text{IRQ}}$[2:3]_GPIO[211:212]	Reset functions to select boot configuration.
PLLCFG[0:1]_$\overline{\text{IRQ}}$[4:5]_GPIO[208:209]	Reset functions to select PLL configuration.

2.3.3 eMIOS Signals

Table 2-9. eMIOS Signals

Signal Name	Description
EMIOS[0:9]_ETPUA[0:9]_GPIO[179:188]	EMIOS[0:9]_ETPUA[0:9]_GPIO[179:188] is an input/output channel pin for the eMIOS module. The alternate functions are output channels for the eTPU A module; that is, when configured as ETPUA[0:9], the pins function as outputs only.
EMIOS10_SCKD_GPIO189	EMIOS10_SCKD_GPIO189 is an input/output channel pin for the eMIOS module. The alternate function is a DSPI clock pin for the DSPI D module.
EMIOS11_SIND_GPIO190	EMIOS11_SIND_GPIO190 is an input/output channel pin for the eMIOS module. The alternate function is a data input pin for the DSPI D module
EMIOS12_SOUTC_GPIO191	EMIOS[12]_SOUTC_GPIO[191] is an output channel pin for the eMIOS module. The alternate function is the data output signal for the DSPI C module.
EMIOS13_SOUTD_GPIO192	EMIOS[13]_SOUTD_GPIO[192] is an output channel pin for the eMIOS module. The alternate function is the data output signal for the DSPI D module.
EMIOS14_$\overline{\text{IRQ0}}$_GPIO193	EMIOS[14]_$\overline{\text{IRQ}}$[0]_GPIO[193] is an output channel pin for the eMIOS module. The alternate function is an external interrupt request input.
EMIOS15_$\overline{\text{IRQ1}}$_GPIO194	EMIOS[15]_$\overline{\text{IRQ}}$[1]_GPIO[194] is an output channel pin for the eMIOS module. The alternate function is an external interrupt request input.
EMIOS[16:23]_ETPUB[0:7]_GPIO[195:202]	Input/Output channel pins for the eMIOS module. Alternate function is an eTPU B output channel
EMIOS24_PCSB0_GPIO203	Input/Output channel pins for the eMIOS module. Alternate function peripheral chip select for the DSPI B module.
EMIOS25_PCSB1_GPIO204	Input/Output channel pins for the eMIOS module. Alternate function peripheral chip select for the DSPI B module.
EMIOS26_PCSB2_GPIO432	Input/Output channel pins for the eMIOS module. Alternate function peripheral chip select for the DSPI B module.
EMIOS27_PCSB3_GPIO433	Input/Output channel pins for the eMIOS module. Alternate function peripheral chip select for the DSPI B module.
EMIOS28_PCSC0_GPIO434	Input/Output channel pins for the eMIOS module. Alternate function peripheral chip select for the DSPI C module.
EMIOS29_PCSC1_GPIO435	Input/Output channel pins for the eMIOS module. Alternate function peripheral chip select for the DSPI C module.

Table 2-9. eMIOS Signals

Signal Name	Description
EMIOS30_PCSC2_GPIO436	Input/Output channel pins for the eMIOS module. Alternate function peripheral chip select for the DSPI C module.
EMIOS31_PCSC5_GPIO437	Input/Output channel pins for the eMIOS module. Alternate function peripheral chip select for the DSPI C module.

2.3.4 eQADC Signals

Table 2-10. eQADC Signals

Signal Name	Description
ANA0–ANA23[1]	ANA[n] is an analog input to eQADC_A’s converter pair.
AN24–AN39	AN[n] is an analog input that goes to both eQADC’s converter pairs.
ANB0–ANB23	ANB[n] is an analog input to eQADC_B’s converter pair.
VRH_A	V_{RHA} is the voltage reference high input pin for the eQADC_A.
VRL_A	V_{RLA} is the voltage reference low input pin for the eQADC_A.
VRH_B	V_{RHB} is the voltage reference high input pin for the eQADC_B.
VRL_B	V_{RLB} is the voltage reference low input pin for the eQADC_B.
REFBYPCB	REFBYPCB is a bypass capacitor input for the eQADC_B. The REFBYPCB pin is used to connect an external bias capacitor between the REFBYPCB pin and V_{RLB}. The value of this capacitor must be 100nF. This bypass capacitor is used to provide a stable reference voltage for the ADC.
REFBYPCA	REFBYPCA is a bypass capacitor input for the eQADC_A. The REFBYPCA pin is used to connect an external bias capacitor between the REFBYPCA pin and V_{RLA}. The value of this capacitor must be 100nF. This bypass capacitor is used to provide a stable reference voltage for the ADC.
VDDA_A0	V_{DDAn} is the analog supply input pin for the eQADC_A.
VDDA_A1	V_{DDAn} is the analog supply input pin for the eQADC_A.
REFBYPCA1	REFBYPCA1 is a bypass capacitor input for the eQADC_A. The REFBYPCA1 pin is used to connect an external bias capacitor providing a stable reference voltage for the ADC.
VSSA_A1	V_{SSAn} is the analog ground reference input pin for the eQADC_A.
VDDA_B0	V_{DDAn} is the analog supply input pin for the eQADC_B.
VDDA_B1	V_{DDAn} is the analog supply input pin for the eQADC_B.
VSSA_B0	V_{SSAn} is the analog ground reference input pin for the eQADC_B.
REFBYPCB1	REFBYPCB1 is a bypass capacitor input for the eQADC_B. The REFBYPCB1 pin is used to connect an external bias capacitor providing a stable reference voltage for the ADC.

[1] For ANA[0:7] pins, during and just after POR negates, internal pull resistors can be enabled, resulting in as much as 4 mA of current draw. The pull resistors are disabled when the system clock propagates through the device.

2.3.5 FlexRay Signals

Table 2-11. FlexRay Signals

Signal Name	Description
FR_A_TX_GPIO248	FlexRay Channel A transmit pin.
FR_A_RX_GPIO249	FlexRay Channel A receive pin.
FR_A_TX_EN_GPIO250	FlexRay Channel A transmit enable pin.
FR_B_TX_GPIO251	FlexRay Channel B transmit pin.
FR_B_RX_GPIO252	FlexRay Channel B receive pin.
FR_B_TX_EN_GPIO253	FlexRay Channel B transmit enable pin.
ETPUC16_FR_A_TX_GPIO45	The alternate function is FlexRay Channel A transmit.
ETPUC17_FR_A_RX_GPIO458	The alternate function is FlexRay Channel A receive.
ETPUC18_FR_A_TX_EN_GPIO459	The alternate function is FlexRay Channel A transmit enable.

2.3.6 FlexCAN Signals

Table 2-12. FlexCAN Signals

Signal Name	Description
CNTXA_TXDA_GPIO83	Transmit pin for the FlexCAN A module. Alternate function is the transmit pin for the eSCI A module
CNRXA_RXDA_GPIO84	Receive pin for the FlexCAN A module. Alternate function is the receive pin for the eSCI A module
CNTXB_PCSC3_GPIO85	Transmit pin for the FlexCAN B module. Alternate function is a peripheral chip select output for the DSPI C module.
CNRXB_PCSC4_GPIO86	Receive pin for the FlexCAN B module. Alternate function is a peripheral chip select output for the DSPI C module.
CNTXC_PCSD3_GPIO87	Transmit pin for the FlexCAN C module. Alternate function is a peripheral chip select output for the DSPI C module.
CNRXC_PCSD4_GPIO88	Receive pin for the FlexCAN C module. Alternate function is a peripheral chip select output for the DSPI C module.
CNTXD_GPIO246	Transmit pin for the FlexCAN D module.
CNRXD_GPIO247	Receive pin for the FlexCAN D module.

2.3.7 eSCI Signals

Table 2-13. eSCI Signals

Signal Name	Description
TXDA_GPIO89	Transmit data pin for the eSCI A module.
RXDA _GPIO90	Receive pin for the eSCI A module.

Table 2-13. eSCI Signals

Signal Name	Description
TXDB_PCSD1_GPIO91	Transmit data pin for the eSCI B module. The alternate function is a peripheral chip select output for the DSPI D module.
RXDB_PCSD5_GPIO92	Receive pin for the eSCI B module. The alternFate function is a peripheral chip select output for the DSPI D module.
TXDC_ETRIG0_GPIO244	Transmit data pin for the eSCI C module. The alternate function is external trigger signal to trigger a software or hardware event. The eQADC can detect rising edge, falling edge, high level, and low level on each of the external trigger signals. The eQADC also supports configurable digital filters for these external trigger signals. The eQADC external trigger input pins can be connected to the eTPU, the eMIOS, or an external signal. The source is selected by configuring the eQADC trigger source in the SIU_ISEL4-7 registers. ETRIG[0] is the external trigger for CFIFO0, CFIFO2, and CFIFO4
RXDC_GPIO245	Receive pin for the eSCI C module.
ETPUC19_TXDA_GPIO460	The alternate function is the transmit pin for the eSCI A module
ETPUC20_RXDA _GPIO461	The alternate function is the receive pin for the eSCI A module
ETPUC21_TXDB_GPIO462	The alternate function is the transmit pin for the eSCI B module
ETPUC22_RXDB_GPIO463	The alternate function is the receive pin for the eSCI B module

2.3.8 DSPI Signals

Table 2-14. DSPI Signals

Signal Name	Description
SCKA_PCSC1_GPIO93	SPI clock pin for the DSPI A module. The alternate signal function is a peripheral chip select for the DSPI C module
SINA_PCSC2_GPIO94	Data input pin for the DSPI A module. The alternate signal function is a peripheral chip select for the DSPI C module
SOUTA_PCSC5_GPIO95	Data output pin for the DSPI A module. The alternate signal function is a peripheral chip select for the DSPI C module
PCSA0_PCSD2_GPIO96	Peripheral chip select for the DSPI A module. The alternate signal function is a peripheral chip select for the DSPI D module
PCSA1_GPIO97	Peripheral chip selects for the DSPI A module.
PCSA2_GPIO98	
PCSA3_GPIO99	
PCSA4_GPIO100	
PCSA5_ETRIG1_GPIO101	Peripheral chip select for the DSPI A module. Alternate function is external trigger signal to trigger a software or hardware event. The eQADC can detect rising edge, falling edge, high level, and low level on each of the external trigger signals. The eQADC also supports configurable digital filters for these external trigger signals. The eQADC external trigger input pins can be connected to the eTPU, the eMIOS, or an external signal. The source is selected by configuring the eQADC trigger source in the SIU_ISEL4-7 registers. ETRIG[1] serves as the external trigger for CFIFO1, CFIFO3, and CFIFO5.
SCKB_GPIO102	SPI clock pin for the DSPI B module.

Table 2-14. DSPI Signals

Signal Name	Description
SINB_GPIO103	Data input pin for the DSPI B module.
SOUTB_GPIO104	Data output pin for the DSPI B module.
PCSB0_PCSD2_GPIO105	Peripheral chip select for the DSPI B module. The alternate function is peripheral chip select for the DSPI D module.
PCSB1_PCSD0_GPIO106	Peripheral chip select for the DSPI B module.The alternate function is peripheral chip select for the DSPI D module.
PCSB2_SOUTC_GPIO107	Peripheral chip select for the DSPI B module. The alternate function is data output pin for the DSPI C module
PCSB3_SINC_GPIO108	Peripheral chip select for the DSPI B module. The alternate function is data input pin for the DSPI C module.
PCSB4_SCKC_GPIO109	Peripheral chip select for the DSPI B module. The alternate function is SPI clock pin for the DSPI C module.
PCSB5_PCSC0_GPIO110	Peripheral chip select for the DSPI B module. The alternate function is peripheral chip select for the DSPI C module.
SCKC_SCK_C_LVDSP_ GPIO235	SPI clock pin for the DSPI C module. The alternate function is the LVDS+ version of SCKC.
SINC_SCK_C_LVDSM_ GPIO236	Data input pin for the DSPI C module. The alternate function is the LVDS- version of SCKC.
SOUTC_SOUT_C_LVDSP_ GPIO237	Data output pin for the DSPI C module. The alternate function is the LVDS+ version of SOUTC.
PCSC0_SOUT_C_LVDSM_ GPIO238	Peripheral chip select for the DSPI C module. The alternate function is the LVDS- version of SOUTC
PCSC1_GPIO239	Peripheral chip select for the DSPI B module.
PCSC2_GPIO240	Peripheral chip select for the DSPI B module.
PCSC3_GPIO241	Peripheral chip select for the DSPI B module.
PCSC4_GPIO242	Peripheral chip select for the DSPI B module.
PCSC5_GPIO243	Peripheral chip select for the DSPI B module.
ETPUC23_PCSD5_GPIO464	The alternate function is a peripheral chip select for the DSPI D module.
ETPUC24_PCSD4_GPIO465	The alternate function is a peripheral chip select for the DSPI D module.
ETPUC25_PCSD3_GPIO466	The alternate function is a peripheral chip select for the DSPI D module.
ETPUC26_PCSD2_GPIO467	The alternate function is a peripheral chip select for the DSPI D module.
ETPUC27_PCSD1_GPIO468	The alternate function is a peripheral chip select for the DSPI D module.
ETPUC28_PCSD0_GPIO469	The alternate function is a peripheral chip select for the DSPI D module.
ETPUC29_SCKD_GPIO470	The alternate function is a DSPI clock pin for the DSPI D module.
ETPUC30_SOUTD_GPIO471	The alternate function is a data output pin for the DSPI D module.
ETPUC31_SIND_GPIO472	The alternate function is a data input pin for the DSPI D module.

2.3.9 EBI Signals

Table 2-15. EBI Signals (Development Bus Only)

Signal Name	Description
D_CS0_GPIO256	EBI chip select output signal.
D_CS2_D_DAT31_GPIO257	EBI chip select output signal. The alternate function is data signal 31.
D_CS3_D_TEA_GPIO258	EBI chip select output signal. The alternate function is TEA which indicates that an error occurred in the current external bus transfer.
D_ADD12_GPIO259	EBI address signals.
D_ADD13_GPIO260	
D_ADD14_GPIO261	
D_ADD15_GPIO262	
D_ADD16_D_DAT16_GPIO263	EBI address signals with alternate functions of EBI data signals.
D_ADD17_D_DAT17_GPIO264	
D_ADD18_D_DAT18_GPIO265	
D_ADD19_D_DAT19_GPIO266	
D_ADD20_D_DAT20_GPIO267	
D_ADD21_D_DAT21_GPIO268	
D_ADD22_D_DAT22_GPIO269	
D_ADD23_D_DAT23_GPIO270	
D_ADD24_D_DAT24_GPIO271	
D_ADD25_D_DAT25_GPIO272	
D_ADD26_D_DAT26_GPIO273	
D_ADD27_D_DAT27_GPIO274	
D_ADD28_D_DAT28_GPIO275	
D_ADD29_D_DAT29_GPIO276	
D_ADD30_D_DAT30_GPIO277	
D_DAT[0:15]_GPIO[278:293]	EBI data signals.
D_RD_$\overline{\text{WR}}$_GPIO294	Indicates whether an external bus transfer is a read or write operation.
D_$\overline{\text{WE0}}$_GPIO295	Write/Byte enable specify which data pins contain valid data for an external bus transfer.
D_$\overline{\text{WE1}}$_GPIO296	
D_$\overline{\text{OE}}$_GPIO297	Output enable indicates that the EBI is ready to accept read data.
D_$\overline{\text{TS}}$_GPIO298	Transfer start is asserted by the EBI owner to indicate the start of a transfer.
D_ALE_GPIO299	Address latch enable is used to demultiplex the address from data bus. It is asserted while the least significant 16 bits of the address are present in the multiplexed address/data bus.
D_$\overline{\text{TA}}$_GPIO300	Transfer acknowledge is asserted by the EBI owner to acknowledge that the slave has completed the current transfer.
D_$\overline{\text{CS1}}$_GPIO301	EBI chip select output signal.
D_$\overline{\text{BDIP}}$_GPIO302	Burst Data In Progress indicates that an EBI burst transfer is in progress.

Table 2-15. EBI Signals (Development Bus Only)

Signal Name	Description
D_$\overline{\text{WE2}}$_GPIO303	Write/Byte enable specify which data pins contain valid data for an external bus transfer.
D_$\overline{\text{WE3}}$_GPIO304	
D_ADD9_GPIO305	EBI address signals.
D_ADD10_GPIO306	
D_ADD11_GPIO307	

2.3.10 Reset, Configuration and Clock Signals

Table 2-16. Reset, Configuration and Clock Signals

Signal Name	Description
$\overline{\text{RESET}}$	The $\overline{\text{RESET}}$ input is asserted by an external device to reset all modules of the device MCU. The $\overline{\text{RESET}}$ pin must be asserted during a power-on reset.
$\overline{\text{RSTOUT}}$	The $\overline{\text{RSTOUT}}$ output is a push/pull output that is asserted during an internal device reset. The pin can also be asserted by software without causing an internal reset of the device MCU. **Note:** During a power-on-reset (POR), $\overline{\text{RSTOUT}}$ is tri-stated.
BOOTCFG[0:1]_$\overline{\text{IRQ}}$[2:3]_ GPIO[211:212]	BOOTCFG[0:1] signals are sampled on every reset. The values are used by the Boot Assist Module (BAM) program to determine the boot configuration of the device. The alternate functions are the external interrupt request inputs (IRQs).
WKPCFG_NMI_GPIO213	WKPCFG (sampled at every reset) determines whether specific eTPU and eMIOS pins are connected to a weak pullup or weak pulldown during and immediately after reset. The alternate function (NMI) is a critical interrupt to the core.
PLLCFG0_$\overline{\text{IRQ4}}$_GPIO208	PLLCFG*n* are sampled at every reset. These values are used to configure the FMPLL mode of operation. The alternate function is an external interrupt request input.
PLLCFG1_$\overline{\text{IRQ5}}$_GPIO209	PLLCFG*n* are sampled at every reset. These values are used to configure the FMPLL operation mode. The alternate functions are an external interrupt request input and data output for the DSPI module D.
PLLCFG2	PLLCFG*n* are sampled at every reset. These values are used to configure the FMPLL operation mode. PLLCFG2 configures the crystal oscillator range.
XTAL	XTAL is the output pin for an external crystal oscillator.
EXTAL	EXTAL is the input pin for an external crystal oscillator or an external clock source.
D_CLKOUT	CLKOUT is the device system clock output (for the development EBI).
ENGCLK	ENGCLK is a 50% duty cycle output clock with a maximum frequency of the device system clock divided by two. ENGCLK is not synchronous to CLKOUT.

2.3.11 JTAG and Nexus Signals

Table 2-17. JTAG and Nexus Signals

Signal Name	Description
$\overline{\text{EVTI}}$	$\overline{\text{EVTI}}$ is an input that is read during a debug port reset to enable or disable the Nexus Auxiliary port for data trace. After reset, the $\overline{\text{EVTI}}$ pin is used to initiate program and data trace synchronization messages or generate a breakpoint.
$\overline{\text{EVTO}}$	$\overline{\text{EVTO}}$ is an output that provides timing to a development tool for a single watchpoint or breakpoint occurrence. The BAM uses this pin to select if auto baud rate is on or off.
MCKO	MCKO is a free running clock output to the development tools which is used for timing of the MDO and $\overline{\text{MSEO}}$ signals.
MDO0_GPIO220[1]	MDO[0] is a trace message output to the development tools. In addition, MDO[0] indicates the presence of the system clock following a power-on reset. MDO[0] is driven high following a power-on-reset until a system clock is present, at which time it is then negated. There is an internal pullup on MDO[0]. This pin functions as GPIO when Nexus messaging is disabled.
MDO[3:1]_GPIO[223:221][1]	MDO[3:1] are the trace message outputs to the development tools for reduced port mode. These pins function as GPIO when Nexus messaging is disabled.
MDO[11:4]_GPIO[82:75][1]	MDO[11:4] are the trace message outputs to the development tools for reduced port mode. These pins function as GPIO when Nexus messaging is disabled
MDO[15:12]_GPIO[234:231[1]	Trace message outputs to the development tools for full port mode. These pins function as GPIO when the Nexus Development Interface (NDI) functions in reduced port mode or when Nexus messaging is disabled
$\overline{\text{MSEO}}$[1:0][1]	$\overline{\text{MSEO}}$[1:0] are output signals that indicate when messages start and end on the MDO pins.
$\overline{\text{RDY}}$	$\overline{\text{RDY}}$ is an output signal that indicates to the development tools the data is ready to be read from or written to the Nexus read/write access registers.
TCK	TCK provides the clock input for the on-chip test logic.
TDI	TDI provides the serial test instruction and data input for the on-chip test logic.
TDO	TDO provides the serial test data output for the on-chip test logic.
TMS	TMS controls test mode operations for the on-chip test logic.
JCOMP	JCOMP enables the JTAG TAP controller.
TEST	TEST places the chip into test mode. You must tie this pin to VSS.

[1] Do not connect MDO[0:15] and MSEO[0:1] pins directly to a power supply or ground.

2.3.12 PMC and Power/Voltage Signals

Table 2-18. PMC and Power/Voltage Signals

Signal Name	Description
REGSEL	Selects PMC regulator mode (Linear/Switch mode).
REGCTL	Core voltage regulator output control pin to the external bipolar/MOS transistor.
VDDSYN	V_{DDSYN} is the power supply input for the FMPLL. It is also the 3.3V regulator output, when enabled (see the PMC chapter).
VSSSYN	V_{SSSYN} is the ground reference input for the FMPLL.
VDD33	IO pre-drivers and flash power supply.
VSTBY	V_{STBY} is the power supply input that is used to maintain a portion of the contents of internal SRAM during power down. If not used, tie V_{STBY} to V_{SS}.
VSSFL	VSSFL must be tied to V_{SS}.
VDDREG	Source voltage for on-chip regulators and low voltage detect (LVD) circuits.
VDDEH*n*, VDDE*n*	I/O supply input.
VRH_A	V_{RHA} is the voltage reference high input pin for the eQADC_A.
VRL_A	V_{RLA} is the voltage reference low input pin for the eQADC_A.
VRH_B	V_{RHB} is the voltage reference high input pin for the eQADC_B.
VRL_B	V_{RLB} is the voltage reference low input pin for the eQADC_B.
VDDA_A0	V_{DDAn} is the analog supply input pin for the eQADC_A.
VDDA_A1	V_{DDAn} is the analog supply input pin for the eQADC_A.
VSSA_A1	V_{SSAn} is the analog ground reference input pin for the eQADC_A.
VDDA_B0	V_{DDAn} is the analog supply input pin for the eQADC_B.
VDDA_B1	V_{DDAn} is the analog supply input pin for the eQADC_B.
VSSA_B0	V_{SSAn} is the analog ground reference input pin for the eQADC_B.

Chapter 3
System Integration Units (SIU, SIU_B)

3.1 Introduction

This chapter describes the device system integration units (SIU, and SIU_B) that configures and initializes the following features in the SIU:

- MCU reset configuration
- System reset operation
- Pad configuration
- External interrupts
- General-purpose I/O (GPIO)
- Internal peripheral multiplexing
- GPDI and GPDO I/Os of the DSPI modules
- TLB Entry Selector

and the following feature in SIU_B

- Mirrored pad configuration and GPIO registers for Protected Port Output mode

3.1.1 Block Diagram

Figure 3-1 is a block diagram of the SIU. The SIU register memory map is shown in Table 3-3.

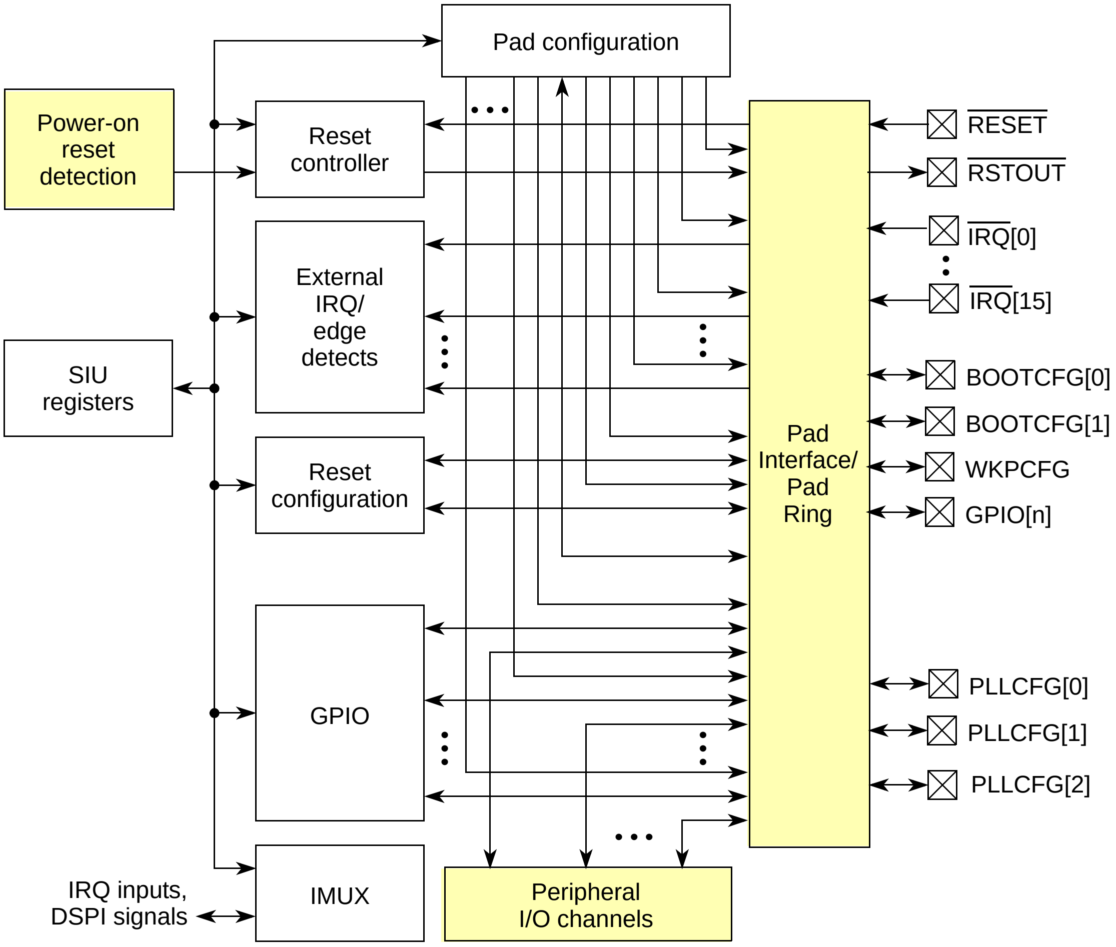


Figure 3-1. SIU Block Diagram

NOTE

The power-on reset detection module, pad interface/pad ring module, and peripheral I/O channels shown shaded in Figure 3-1 are external to the SIU.

3.1.2 Overview

The system integration units are accessed through the system bus crossbar switch (XBAR) and the peripheral bridge A (PBRIDGE_A). Table 3-1 lists the features that are configured:

Table 3-1. Features

Feature	Description
MCU reset operations	Controls the external pin boot logic
System reset operations	Monitors internal and external reset sources, and drives the $\overline{\text{RSTOUT}}$ signal • Power-on reset support • Reset status register providing last reset source to software • Glitch detection on reset input • Software controlled reset assertion
Pad configuration registers	Enables the configuration and initialization of the I/O pin electrical characteristics using software to select the following: • Active function from the set of multiplexed functions • Pullup and pulldown characteristics of the pin • Slew rate for slow and medium pads • Open drain mode for output pins • Hysteresis for input pins • Drive strength of bus signals for fast pads
External interrupt operations	• 16 interrupt requests • Rising- or falling-edge event detection • Programmable digital filter for glitch rejection • NMI and critical interrupt control
General-purpose I/O (GPIO)	Provides uniform and discrete I/O control of MCU general-purpose I/O pins, where each GPIO signal has an input register and an output register.
Internal peripheral multiplexing	Provides flexibility to customize signal/pin assignments for application development that allows: • Serial and parallel chaining of DSPIs • Flexible selection of eQADC trigger inputs • Assignment of interrupt requests (IRQs) between external pins and DSPI
TLB Entry Selector	Allows an external tool to non-intrusively change the TLB entry that is used by the cores to access memory.
Protected Port Output	Allows up to four GPIO pins to be reserved by any one of the two cores.

3.1.3 Modes of Operation

The operating modes for this module are listed in Table 3-2.

Table 3-2. SIU Modes of Operation

Operating Mode	Description
Normal	In normal mode, the SIU provides the register interface and logic that controls the device and system configuration, the reset controller, and GPIO. The SIU continues operation with no changes in stop mode.
Debug	SIU operation in debug mode is identical to operation in normal mode.

3.2 Memory Map and Register Definition

Table 3-3 is the address map for the SIU registers. All register addresses shown are an offset of the SIU base address.

Table 3-3. SIU Memory Map

Address	Name	Description	Bits
SIU_BASE = 0xC3F9_0000	Reserved	—	—
SIU_BASE + 0x0004	SIU_MIDR	MCU ID register	32
SIU_BASE + 0x0008	Reserved	—	—
SIU_BASE + 0x000C	SIU_RSR	Reset status register	32
SIU_BASE + 0x0010	SIU_SRCR	System reset control register	32
SIU_BASE + 0x0014	SIU_EISR	SIU external interrupt status register	32
SIU_BASE + 0x0018	SIU_DIRER	DMA/interrupt request enable register	32
SIU_BASE + 0x001C	SIU_DIRSR	DMA/interrupt request select register	32
SIU_BASE + 0x0020	SIU_OSR	Overrun status register	32
SIU_BASE + 0x0024	SIU_ORER	Overrun request enable register	32
SIU_BASE + 0x0028	SIU_IREER	IRQ rising-edge event enable register	32
SIU_BASE + 0x002C	SIU_IFEER	IRQ falling-edge event enable register	32
SIU_BASE + 0x0030	SIU_IDFR	IRQ digital filter register	32
SIU_BASE + 0x0034	SIU_IFIR	IRQ Filtered Input Register	32
SIU_BASE + (0x0038–0x003F)	Reserved	—	—
SIU_BASE + (0x0040–0x043E)	SIU_PCR0–SIU_PCR511	Pad configuration registers 0–511	16
SIU_BASE + (0x0440–0x05FF)	Reserved	—	—
SIU_BASE + (0x0600–0x07FF)	SIU_GPDO0 – SIU_GPDO511	GPIO pin data output registers 0–511	8
SIU_BASE + (0x0800–0x08FF)	SIU_GPDI0 – 255	GPIO input data registers 0–255 (Legacy support only, new implementation at SIU_BASE + 0x0E00)	8
SIU_BASE + (0x0900–0x0903)	Reserved	—	—
SIU_BASE + (0x0904–0x0907)	SIU_EIISR (sometimes referred to as ISEL1)	External IRQ input select register	32
SIU_BASE + (0x0908–0x090B)	SIU_DISR (sometimes referred to as ISEL2)	DSPI input select register	32
SIU_BASE + (0x090C–0x090F)	Reserved	—	—
SIU_BASE + (0x0910–0x091F)	SIU_ISEL4–7	eQADC Command FIFO Trigger Source Select— IMUX Select Registers	32
SIU_BASE + 0x0920	SIU_ISEL8	eTPU Input Select Register	32
SIU_BASE + 0x0924	SIU_ISEL9	eQADC Advance Trigger Selection	32

Table 3-3. SIU Memory Map

Address	Name	Description	Bits
SIU_BASE + 0x0928	SIU_DECFIL1	Decimation Filter Register 1	32
SIU_BASE + 0x092C	SIU_DECFIL2	Decimation Filter Register 2	32
SIU_BASE + 0x0930	SIU_DECFIL3	Decimation Filter Register 3	32
SIU_BASE + 0x0934	SIU_DECFIL4	Decimation Filter Register 4	32
SIU_BASE + 0x0938	SIU_DECFIL5	Decimation Filter Register 5	32
SIU_BASE + (0x093C–0x097F)	Reserved	—	—
SIU_BASE + (0x0930–0x097F)	Reserved	—	—
SIU_BASE + 0x0980	SIU_CCR	Chip Configuration Register	32
SIU_BASE + 0x0984	SIU_ECCR	External clock control Register	32
SIU_BASE + 0x0988	Reserved	—	—
SIU_BASE + 0x098C	Reserved	—	—
SIU_BASE + 0x0990	SIU_CBRH	Compare B High Register	32
SIU_BASE + 0x0994	SIU_CBRL	Compare B Low Register	32
SIU_BASE + (0x0998–0x099F)	Reserved	—	—
SIU_BASE + 0x9A0	SIU_SYSDIV	System Clock Register	32
SIU_BASE + 0x9A4	SIU_HLT	Halt Register	32
SIU_BASE + 0x9A8	SIU_HLTACK	Halt Acknowledge Register	32
SIU_BASE + 0x9AC	SIU_RSTVEC0	Reset Vector Register for Core 0	32
SIU_BASE + 0x9B0	SIU_RSTVEC1	Reset Vector Register for Core 1	32
SIU_BASE + 0x9B4	SIU_C0PID	Core 0 PID mapping control register	32
SIU_BASE + 0x9B8	SIU_C1PID	Core 1PID mapping control register	32
SIU_BASE + (0x09BC–0x0BFF)	Reserved	—	—
SIU_BASE + (0x0C00–0x0C3C)	SIU_PGPDO0–SIU_PGPDO15	Parallel GPIO Pin Data Output Register 0–15	32
SIU_BASE + (0x0C40–0x0C7C)	SIU_PGPDI0–SIU_PGPDI15	Parallel GPIO Pin Data input Register0–15	32
SIU_BASE + (0x0C80–0x0CFC)	SIU_MPGPDO0–SIU_MPGPDO31	Masked Parallel GPIO Pin Data Output Register 0–31	32
SIU_BASE+0x0D00	SIU_DSPIAH	DSPIA GP Mask-Output High Register	32
SIU_BASE+0x0D04	SIU_DSPIAL	DSPIA GP Mask-Output Low Register	32
SIU_BASE+0x0D08	SIU_DSPIBH	DSPIB GP Mask-Output High Register	32
SIU_BASE+0x0D0C	SIU_DSPIBL	DSPIB GP Mask-Output Low Register	32
SIU_BASE+0x0D10	SIU_DSPICH	DSPIC GP Mask-Output High Register	32
SIU_BASE+0x0D14	SIU_DSPICL	DSPIC GP Mask-Output Low Register	32

Table 3-3. SIU Memory Map

Address	Name	Description	Bits
SIU_BASE+0x0D18	SIU_DSPIDH	DSPID GP Mask-Output High Register	32
SIU_BASE+0x0D1C	SIU_DSPIDL	DSPID GP Mask-Output Low Register	32
SIU_BASE+0x0D20– SIU_BASE+0x0D3F	Reserved	—	—
SIU_BASE+0x0D40	SIU_ETPUBA	DSPIA eTPUB Select Register	32
SIU_BASE+0x0D44	SIU_EMIOSA	DSPIA eMIOS Select Register	32
SIU_BASE+0x0D48	SIU_DSPIAHLA	DSPIA SIU_DSPIAH/L Select Register	32
SIU_BASE+0x0D4C	Reserved	—	—
SIU_BASE+0x0D50	SIU_ETPUAB	DSPIB eTPUA Select Register	32
SIU_BASE+0x0D54	SIU_EMIOSB	DSPIB eMIOS Select Register	32
SIU_BASE+0x0D58	SIU_DSPIBHLB	DSPIB SIU_DSPIBH/L Select Register	32
SIU_BASE+0x0D5C	Reserved	—	—
SIU_BASE+0x0D60	SIU_ETPUAC	DSPIC eTPUA Select Register	32
SIU_BASE+0x0D64	SIU_EMIOSC	DSPIC eMIOS Select Register	32
SIU_BASE+0x0D68	SIU_DSPICHLC	DSPIC SIU_DSPICH/L Select Register	32
SIU_BASE+0x0D6C	Reserved	—	—
SIU_BASE+0x0D70	SIU_ETPUBD	DSPID eTPUB Select Register	32
SIU_BASE+0x0D74	SIU_EMIOSD	DSPID eMIOS Select Register	32
SIU_BASE+0x0D78	SIU_DSPIDHLD	DSPIC SIU_DSPICH/L Select Register	32
SIU_BASE+0x0D7C– SIU_BASE+0x0DFF	Reserved	—	—
SIU_BASE+0x0E00– SIU_BASE+0x0FDC	SIU_GPDI0–SIU_GPDI511	GPIO Pin Data Input Register 0–511	8

Table 3-4 is the address map for the SIU_B registers. All register addresses shown are an offset of the SIU_B base address.

Table 3-4. SIU_B Memory Map

Address	Name	Description	Bits
SIU_B_BASE = 0xC3F9_8000	Reserved	—	—
SIU_B_BASE + (0x0001–0x021F)	Reserved	—	—
SIU_B_BASE + (0x0220–0x0226)	SIU_PCR240M–SIU_PCR243M	Mirrored Pad configuration registers 240–243	16
SIU_B_BASE + (0x0228–0x06EF)	Reserved	—	—
SIU_B_BASE + (0x06F0–0x06F3)	SIU_GPDO240M–SIU_GPDO243M	Mirrored GPIO pin data output registers 240–243	8
SIU_B_BASE + (0x06F4–0x08EF)	Reserved	—	—

Table 3-4. SIU_B Memory Map

Address	Name	Description	Bits
SIU_B_BASE + (0x08F0–0x08F3)	SIU_GPDI240M–SIU_GPDI243M	Mirrored GPIO input data registers 240–243	8
SIU_B_BASE + (0x08F4–0x3FFF)	Reserved	—	—

3.2.1 Register Descriptions

3.2.1.1 MCU ID Register (SIU_MIDR)

The SIU_MIDR contains the part identification number and mask revision number specific to the device. The part number is a read-only field that is mask programmed with the part number of the device. The part number changes depending on the module versions contained in a device. The part number does not change for bug fixes or process changes.

The mask number is a read-only field that is mask programmed with the specific mask revision level of the device. The current value applies to revision 0 and is updated for each mask revision.

The MCU ID register is 32-bits. Figure 3-2 shows the MCU ID register values.

Address: SIU_BASE + 0x0004

Access: Read Only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PARTNUM															
W																
MPC5676R part number	0	1	0	1	0	1	1	0	0	1	1	1	0	1	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PKG				0	0	0	0	MASKNUM_MAJOR				MASKNUM_MINOR			
W																
Default reset values	416= 0b0110 516=0b1110				0	0	0	0	0b0000				0b0000			

Figure 3-2. MPC5676R MCU ID Register (SIU_MIDR)

The following table describes the MIDR fields:

Table 3-5. SIU_MIDR Bit Field Descriptions

Field	Description
0–15 PARTNUM	MCU part number. Read-only, mask-programmed part identification number of the MCU. MPC5676R reads 0x5676.
16–19 PKG	Package settings. PKG selects the pin package for the device. 0110 Select the 416 package 1110 Select the 516 package All other settings are reserved.
20–23	Reserved
24–27 MASKNUM_MAJOR [0:3]	Major revision number of MCU mask. Read-only, mask programmed mask number of the MCU. Reads 0x0 for the initial mask set of the device, and changes sequentially for each mask set.
28–31 MASKNUM_MINOR [0:3]	Minor revision number of MCU mask. Read-only, mask programmed mask number of the MCU. Reads 0x0 for the initial mask set of the device, and changes sequentially for each mask set.

3.2.1.2 Reset Status Register (SIU_RSR)

The SIU_RSR contains the sources of the most recent reset, and the state of the configuration pins at reset. Except for a POR request or a software external reset, all reset requests, regardless of priority, are not serviced until the current reset completes.

This register contains one reset status bit for each reset source (see Table 3-6).

A reset status bit set to 1 indicates a reset request by that source. After the reset status bits are set, they remain set until another reset occurs. Simultaneous reset requests are prioritized. When reset requests with different priorities occur on the same clock cycle, the reset request with the highest priority is serviced and the status bit of only that reset request is set.

The following table lists the reset sources and arbitration priorities:

Table 3-6. Reset Source Priorities

Reset Source	Priority	Group
• Power on reset (POR) • External reset	Highest	0
• Software system reset	Higher	1
• Loss-of-clock • Loss-of-lock • Core Watchdog or Debug • Platform Watchdog	Lower	2
• Software external reset	Lowest	3

The WKPCFG bit retains the latest value of the WKPCFG signal before reset. The BOOTCFG field retains the latest values of the BOOTCFG[0:1] signals before reset.

Address: SIU_BASE + 0x000C

Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PORS	ERS	LLRS	LCRS	WDRS0	0	SWTRS0	SWTRS1	WDRS1	CPURS	0	0	0	STCURS	SSRS	SERF
W																
Reset[1]	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	WKPCFG[2]	0	0	0	0	0	0	0	0	0	0	0	ABR	BOOTCFG		RGF
W																
Reset[1]	U[2]	0	0	0	0	0	0	0	0	0	0	0	U	U[3]		0

[1] The reset status register receives the reset values during power-on reset.

[2] The reset value of the WKPCFG bit is the value on the WKPCFG pin at the time of the last reset.

[3] The reset value of the BOOTCFG bits is the value on the BOOTCFG[0:1] pins at he time of the last reset.

Figure 3-3. Reset Status Register (SIU_RSR)

The following table lists and describes the fields of the reset status register:

Table 3-7. SIU_RSR Bit Field Descriptions

Field	Description
0 PORS	Power-on reset status. 0 Another reset source was acknowledged by the reset controller since the last assertion of the power-on reset input. 1 The power-on reset input to the reset controller was asserted and no other reset source was acknowledged since the assertion of the power-on reset input except an external reset.
1 ERS	External reset status. 0 The last reset source acknowledged by the reset controller was *not* a valid assertion of the $\overline{\text{RESET}}$ pin. 1 The last reset source acknowledged by the reset controller was a valid assertion of the $\overline{\text{RESET}}$ pin.
2 LLRS	Loss-of-lock reset status. 0 The last reset source acknowledged by the reset controller was *not* a loss-of-PLL lock reset. 1 The last reset source acknowledged by the reset controller was a loss-of-PLL lock reset.
3 LCRS	Loss-of-clock reset status. 0 The last reset source acknowledged by the reset controller was *not* a loss-of-clock reset. 1 The last reset source acknowledged by the reset controller was a loss-of-clock reset.
4 WDRS0	Core 0 Watchdog timer/debug reset status. 0 The last reset source acknowledged by the reset controller was *not* a core 0 watchdog timer or debug reset. 1 The last reset source acknowledged by the reset controller was a core 0 watchdog timer or debug reset.
6 SWTRS0	Software watchdog 0 (SWT0) reset status. 0 The last reset source acknowledged by the reset controller was *not* a software watchdog timer or debug reset. 1 The last reset source acknowledged by the reset controller was a software watchdog timer or debug reset.

Table 3-7. SIU_RSR Bit Field Descriptions

Field	Description
7 SWTRS1	Software watchdog 1 (SWT1) reset status. 0 The last reset source acknowledged by the reset controller was *not* a software watchdog timer or debug reset. 1 The last reset source acknowledged by the reset controller was a software watchdog timer or debug reset.
8 WDRS1	Core 1 Watchdog timer/debug reset status. 0 The last reset source acknowledged by the reset controller was *not* a core 1 watchdog timer or debug reset. 1 The last reset source acknowledged by the reset controller was a core 1 watchdog timer or debug reset.
9 CPURS	Simultaneous Core 0 and Core 1 reset status. 0 The last reset source acknowledged by the reset controller was *not* a simultaneous reset attempt of both cores. 1 The last reset source acknowledged by the reset controller was a simultaneous reset attempt of both cores.
10–12	Reserved
13 STCURS	Self Test control unit reset status. 0 The last reset source acknowledged by the reset controller was *not* an STCU reset. 1 The last reset source acknowledged by the reset controller was an STCU reset.
14 SSRS	Software system reset status. 0 The last reset source acknowledged by the reset controller was *not* a software system reset. 1 The last reset source acknowledged by the reset controller was a software system reset.
15 SERF	Software external reset flag. 0 The software external reset input to the reset controller was *not* asserted, or this bit has been cleared by writing a 1 to it. 1 The software external reset input to the reset controller was asserted while this bit was 0.
16 WKPCFG	Weak pull configuration pin status 0 The WKPCFG pin value latched during the last reset was a *logical 0 and weak pulldown* is the default setting. 1 The WKPCFG pin value latched during the last reset was a *logical 1 and weak pullup* is the default setting.
17–27	Reserved
28 ABR	Auto Baud Rate 0 Auto Baud Rate Disabled 1 Auto Baud Rate Enabled
29–30 BOOTCFG	Reset configuration pin status. 00 Internal boot mode 01 Serial boot mode 10 External boot mode (16 bit data, non-muxed address bus) 11 External boot mode (16 bit data, muxed address bus)
31 RGF	Reset glitch flag. Set by the reset controller when a glitch is detected on the $\overline{\text{RESET}}$ pin. This bit is cleared by the assertion of the power-on reset input to the reset controller, or a write of 1 to the RGF bit. 0 No glitch has been detected on the $\overline{\text{RESET}}$ pin. 1 A glitch has been detected on the $\overline{\text{RESET}}$ pin.

Except for a POR request or writing a 1 to the software external reset flag (SERF) bit, all reset requests, regardless of priority are not serviced until the current reset completes.

In the following cases, more than one reset bit is set in the reset status register (SIU_RSR):

Table 3-8. Causes That Set Multiple Reset Status Bits

Case 1	
Condition	• POR request negates and the device remains in the reset • External reset requested • POR and external reset status bits are set
Reason	POR request started the reset sequence, but an external reset request was received before the POR reset sequence ended.
Case 2	
Condition	• Software external reset requested • SERF flag bit set but no previously set bits in the SIU_RSR are cleared
Reason	The SERF flag bit is cleared by writing a 1 (write 1 to clear) to the bit location or when another reset source is asserted.
Case 3	
Condition	• Loss-of-clock reset requested • Loss-of-lock reset requested • Watchdog reset requested
Reason	More than one reset request occurred on the same clock cycle with no reset request by a higher-priority reset source, therefore the status bits for all the requesting resets are set. Refer to Table 3-6.

3.2.1.3 System Reset Control Register (SIU_SRCR)

The system reset control register configures whether a software system reset or a software external reset is generated. An software system reset uses an internal system reset. An software external reset asserts $\overline{\text{RSTOUT}}$.

Address: SIU_BASE + 0x0010

Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	SER	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	SSR[1]	see note[2]														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] The SSR bit always reads 0. A write of 0 to this bit has no effect.

[2] Write 1 to the SER bit to generate a software external reset. A write of 0 to this bit has no effect. When the reset completes, the SER bit is cleared to 0.

Figure 3-4. System Reset Control Register (SIU_SRCR)

The following table describes the fields in the system reset control register:

Table 3-9. SIU_SRCR Bit Field Descriptions

Field	Description
0 SSR	Software system reset. The software system reset is processed as a synchronous reset. Except for a software external reset, the bit automatically clears if any other reset source asserts. 0 No software system reset. 1 Generate an software internal system reset.
1 SER	Software external reset. Used to generate a software external reset. Writing a 1 to this bit asserts $\overline{\text{RSTOUT}}$ for 2400 clocks, and the internal reset is not asserted. The bit automatically clears when the software external reset completes or any other reset source is asserted. After a software external reset has been initiated, $\overline{\text{RSTOUT}}$ negates if this bit is cleared before the 2400 clock period expires. 0 Do not generate a software external reset. 1 Generate a software external reset.
2–31	Reserved

3.2.1.4 External Interrupt Status Register (SIU_EISR)

The external interrupt status register is used to record edge-triggered events on the IRQ[0]–IRQ[15] inputs and edge-triggered critical or NMI interrupts to the cores. When an edge-detect enable bit is set in the SIU_IREER or SIU_IFEER registers for an IRQ and an IRQ edge-event occurs and is detected, the IRQ flag bit is set in the SIU_EISR. The IRQ flag bits are cleared by writing a 1 to the bit. A write of 0 has no effect.

The IRQ flag bit is set regardless of the state of the DMA or interrupt request enable bit in SIU_DIRER. The IRQ flag bit remains set until cleared by software or through the servicing of a DMA or interrupt request.

Address: SIU_BASE + 0x0014 Access: R/w1c

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NMI0	NMI1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	w1c	w1c														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EIF15	EIF14	EIF13	EIF12	EIF11	EIF10	EIF9	EIF8	EIF7	EIF6	EIF5	EIF4	EIF3	EIF2	EIF1	EIF0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-5. External Interrupt Status Register (SIU_EISR)

The following table describes the fields in the external interrupt status register:

Table 3-10. SIU_EISR Bit Field Descriptions

Field	Description
0 NMI0	Non-Maskable Interrupt Flag. This bit is set when core 0 receives an interrupt from the NMI input pin. 0 No core 0 interrupt has occurred on the NMI input 1 A core 0 interrupt has occurred on the NMI input
1 NMI1	Non-Maskable Interrupt Flag. This bit is set when core 1 receives an interrupt from the NMI input pin. 0 No core 1 interrupt has occurred on the NMI input 1 A core 1 interrupt has occurred on the NMI input
2–15	Reserved
16–31 EIF*n*	External interrupt request flag *n*. This bit is set when an edge-triggered event occurs on the corresponding $\overline{\mathrm{IRQ}}[n]$ input. Cleared by writing a 1. 0 No edge-triggered event has occurred on the corresponding $\overline{\mathrm{IRQ}}[n]$ input. 1 An edge-triggered event has occurred on the corresponding $\overline{\mathrm{IRQ}}[n]$ input.

3.2.1.5 DMA/Interrupt Request Enable Register (SIU_DIRER)

The SIU_DIRER enables external signals on selected pins to generate DMA or interrupt requests, and the SWT modules to generate interrupt requests. EIRE[3:0] bits enable requests to either the eDMA or INTC module, based on the value of the DIRSn bits in the SIU_DIRSR register. EIRE[15:4] bits enable requests only to the INTC module. The NMISELn bits enable requests to either the critical or non-maskable interrupt inputs of the cores. For a pin to generate an interrupt, the appropriate bit in either the SIU_IREER or SIU_IFEER register must additionally be set.

Address: SIU_BASE + 0x0018 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NMI SEL8	NMI SEL9	0	0	0	0	0	0	NMI SEL0	NMI SEL1	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EIRE 15	EIRE 14	EIRE 13	EIRE 12	EIRE 11	EIRE 10	EIRE 9	EIRE 8	EIRE 7	EIRE 6	EIRE 5	EIRE 4	EIRE 3	EIRE 2	EIRE 1	EIRE 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-6. DMA/Interrupt Request Enable Register (SIU_DIRER)

The following table describes the fields in the DMA interrupt request enable register:

Table 3-11. SIU_DIRER Bit Field Descriptions

Field	Description
NMISEL8	Non Maskable Interrupt / Critical Interrupt Selection (from external pin). SIU generates two specific sources of interrupt to core 0, one of them is defined as critical interrupt and the other is defined as non maskable interrupt (NMI). The NMISEL bit selects which signal receives the IRQ from the pin. 0 NMI is enabled (IVOR1 core 0 exception) 1 Critical interrupt is enabled (IVOR0 core 0 exception) **Note:** NMISEL8 is a write once bit.
NMISEL9	Non Maskable Interrupt / Critical Interrupt Selection (from external pin). SIU generates two specific sources of interrupt to core 1, one of them is defined as critical interrupt and the other is defined as non maskable interrupt (NMI). The NMISEL bit selects which signal receives the IRQ from the pin. 0 NMI is enabled (IVOR1 core 1 exception) 1 Critical interrupt is enabled (IVOR0 core 1 exception) **Note:** NMISEL9 is a write once bit.
NMISEL0	Non Maskable Interrupt / Critical Interrupt Selection (from the watchdog timer, SWT0). SIU generates two specific sources of interrupt to core 0, one of them is defined as a critical interrupt and the other is defined as a non maskable interrupt (NMI). 0 NMI is enabled (IVOR1 core 0 exception) 1 Critical interrupt is enabled (IVOR0 core 0 exception) **Note:** NMISEL0 is a write once bit.
NMISEL1	Non Maskable Interrupt / Critical Interrupt Selection (from the watchdog timer, SWT1). SIU generates two specific sources of interrupt to core 1, one of them is defined as a critical interrupt and the other is defined as a non maskable interrupt (NMI). 0 NMI is enabled (IVOR1 core 1 exception) 1 Critical interrupt is enabled (IVOR0 core 1 exception) **Note:** NMISEL1 is a write once bit.
EIRE*n*	External interrupt request enable *n*. Enables the assertion of the interrupt request from the SIU to the interrupt controller when an edge-triggered event occurs on the $\overline{\text{IRQ}}$[*n*] pin. 0 External interrupt request is disabled. 1 External interrupt request is enabled. **Note:** EIRE[0:3] can optionally enable DMA requests instead of IRQs.

3.2.1.6 DMA/Interrupt Request Select Register (SIU_DIRSR)

This register selects the source of the control signals for the integrators of decimation filters A to D. Each ZSELx field of this register may be programmed to route a single eTPU channel simultaneously to the Zero, Integrate and Read control inputs of one integrator. Each HSELx field may be programmed to route a single eTPU channel to the Halt control input of one integrator. Refer to Table 3-42 for details of the source selection codes. See Chapter 14, "Decimation Filter," for details on the Zero, Integrate, Read and Halt control configuration and functionality.

Address: SIU_BASE + 0x001C Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	DIRS 3	DIRS 2	DIRS 1	DIRS 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-7. DMA/Interrupt Request Select Register (SIU_DIRSR)

The following table describes the fields of the SIU_DIRSR register:

Table 3-12. SIU_DIRSR Bit Field Descriptions

Field	Description
0–27	Reserved
28 DIRS3	DMA/Interrupt request select 3. Selects between a DMA or external interrupt request when an edge-triggered event occurs on $\overline{\text{IRQ}}$[3] pin. 0 Interrupt request is selected. 1 DMA request to eDMA_A channel 51 is selected.
29 DIRS2	DMA/Interrupt request select 2. Selects between a DMA or external interrupt request when an edge-triggered event occurs on $\overline{\text{IRQ}}$[2] pin. 0 Interrupt request is selected. 1 DMA request to eDMA_A channel 50 is selected.
30 DIRS1	DMA/Interrupt request select 1. Selects between a DMA or external interrupt request when an edge-triggered event occurs on $\overline{\text{IRQ}}$[1] pin. 0 Interrupt request is selected. 1 DMA request to eDMA_A channel 49 is selected.
31 DIRS0	DMA/Interrupt request select 0. Selects between a DMA or external interrupt request when an edge-triggered event occurs on $\overline{\text{IRQ}}$[0] pin. 0 Interrupt request is selected. 1 DMA request to eDMA_A channel 48 is selected.

3.2.1.7 Overrun Status Register (SIU_OSR)

The SIU_OSR flag bits indicate that an overrun has occurred.

Address: SIU_BASE + 0x0020 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	OVF 15	OVF 14	OVF 13	OVF 12	OVF 11	OVF 10	OVF 9	OVF 8	OVF 7	OVF 6	OVF 5	OVF 4	OVF 3	OVF 2	OVF 1	OVF 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-8. Overrun Status Register (SIU_OSR)

The following table describes the fields in the overrun status register:

Table 3-13. SIU_OSR Bit Field Descriptions

Field	Description
0–15	Reserved
16–31 OVF*n*	Overrun flag *n.* This bit is set when an overrun occurs on $\overline{\text{IRQ}}$[*n*]. Bit 31 (OVF0) is the overrun flag for $\overline{\text{IRQ}}$[0]; bit 16 (OVF15) is overrun flag for $\overline{\text{IRQ}}$[15]. 0 No overrun occurred. 1 An overrun occurred.

3.2.1.8 Overrun Request Enable Register (SIU_ORER)

The SIU_ORER contains bits to enable an overrun if the corresponding flag bit is set in the SIU_OSR. If the overrun request enable bit and the flag bit are set, the single combined overrun request from the SIU to the interrupt controller is asserted.

Address: SIU_BASE + 0x0024 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ORE 15	ORE 14	ORE 13	ORE 12	ORE 11	ORE 10	ORE 9	ORE 8	ORE 7	ORE 6	ORE 5	ORE 4	ORE 3	ORE 2	ORE 1	ORE 0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-9. Overrun Request Enable Register (SIU_ORER)

The following table describes the fields in the overrun request enable register:

Table 3-14. SIU_ORER Bit Field Descriptions

Field	Function
0–15	Reserved
16–31 ORE*n*	Overrun request enable *n*. Enables the overrun request when an overrun occurs on the $\overline{\text{IRQ}}$[*n*] pin. Bit 31 (ORE0) is the enable overrun flag for $\overline{\text{IRQ}}$[0]; bit 16 (ORE15) is overrun flag for $\overline{\text{IRQ}}$[15]. 0 Overrun request is disabled. 1 Overrun request is enabled.

3.2.1.9 IRQ Rising-Edge Event Enable Register (SIU_IREER)

The SIU_IREER enables rising edge-triggered events on $\overline{\text{IRQ}}[n]$. Rising- and falling-edge events are enabled by setting the bits in SIU_IREER and SIU_IFEER.

SIU_IREER bits used for NMI events are write once after a reset.

Address: SIU_BASE + 0x0028 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	IREE_NMI8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IREE 15	IREE 14	IREE 13	IREE 12	IREE 11	IREE 10	IREE9	IREE8	IREE7	IREE6	IREE5	IREE4	IREE3	IREE2	IREE1	IREE0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-10. IRQ Rising-Edge Event Enable Register (SIU_IREER)

The following table describes the fields in the IRQ rising-edge event enable register:

Table 3-15. SIU_IREER Bit Field Descriptions

Field	Function
IREE_ NMI8	IRQ rising-edge event enable for NMI from external NMI pin. 0 Rising-edge event is disabled. 1 Rising-edge event is enabled.
IREE*n*	IRQ rising-edge event enable *n*. Enables rising-edge-triggered events on the corresponding $\overline{\text{IRQ}}[n]$ pin. 0 Rising-edge event is disabled. 1 Rising-edge event is enabled.

3.2.1.10 IRQ Falling-Edge Event Enable Register (SIU_IFEER)

The SIU_IFEER enables falling edge-triggered events on $\overline{\text{IRQ}}$[*n*]. Rising- and falling-edge events are enabled by setting the bits in both SIU_IREER and SIU_IFEER.

SIU_IFEER bits used for NMI events are write once after a reset.

Address: SIU_BASE + 0x002C Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	IFEE_ NMI8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	IFEE 15	IFEE 14	IFEE 13	IFEE 12	IFEE 11	IFEE 10	IFEE 9	IFEE 8	IFEE 7	IFEE 6	IFEE 5	IFEE 4	IFEE 3	IFEE 2	IFEE 1	IFEE 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-11. IRQ Falling-Edge Event Enable Register (SIU_IFEER)

The following table describes the fields in the IRQ falling-edge event enable register:

Table 3-16. SIU_IFEER Bit Field Descriptions

Field	Function
IFEE_ NMI8	IRQ falling-edge event enable for NMI from external NMI pin. 0 Falling-edge event is disabled. 1 Falling-edge event is enabled.
IFEE*n*	IRQ falling-edge event enable *n.* Enables falling-edge-triggered events on the corresponding $\overline{\text{IRQ}}$[*n*] pin. 0 Falling-edge event is disabled. 1 Falling-edge event is enabled.

3.2.1.11 IRQ Digital Filter Register (SIU_IDFR)

The SIU_IDFR specifies the amount of digital filtering on $\overline{\text{IRQ}}$[0]–$\overline{\text{IRQ}}$[15]. The digital filter length field specifies the number of system clocks that define the period of the digital filter and the minimum time an IRQ signal must hold the active state to qualify as an edge-triggered event.

Address: SIU_BASE + 0x0030

Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	DFL			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-12. IRQ Digital Filter Register (SIU_IDFR)

The following table describes the field in the IRQ digital filter register:

Table 3-17. SIU_IDFR Bit Field Descriptions

Field	Function
0–27	Reserved
28–31 DFL	Digital filter length. Defines the digital filter period on the $\overline{\text{IRQ}}$[*n*] inputs according to the following equation: $\text{Filter Period} = (\text{SystemClockPeriod} \times 2^{\text{DFL}}) + 1(\text{SystemClockPeriod})$ For a 100 MHz system clock, this gives a range of 20 ns to 328 µs. The minimum time of three clocks accounts for synchronization of the IRQ input pins with the system clock.

3.2.1.12 IRQ Filtered Input Register (SIU_IFIR)

The SIU_IFIR is a read only register where the filtered values of the NMI and $\overline{\text{IRQ}}$[0]–$\overline{\text{IRQ}}$[15] pins are captured.

Address: SIU_BASE + 0x0034 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	IFI_ NMI	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	IFI15	IFI14	IFI13	IFI12	IFI11	IFI10	IFI9	IFI8	IFI7	IFI6	IFI5	IFI4	IFI3	IFI2	IFI1	IFI0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-13. IRQ Filtered Input Register (SIU_IFIR)

Table 3-18. SIU_IFIR Bit Field Descriptions

Field	Function
0–31 IFI*n*	Filtered Input n—set/cleared for the corresponding filtered IRQ pin. 0 A logic one has passed through the IRQ digital filter for the corresponding IRQ pin. 1 A logic zero has passed through the IRQ digital filter for the corresponding IRQ pin.

3.2.1.13 Pad Configuration Registers (SIU_PCR)

The following subsections define pad configuration registers (PCR) in the SIU_PCR segment. These registers define the pad configuration for all configurable device pins that specify that active function, direction, and electrical attributes for the pin. The information presented pertains to which bits and fields are active for a given pin or group of pins, and the reset state of the register.

The reset state of SIU_PCRs given in this section is the value before the BAM program executes. The BAM program can change some pad configuration registers based on the reset configuration. Refer to the BAM chapter for more detailed information.

The SIU_PCRs are 16-bit registers that are read from or written to as:

- 16-bit values aligned on 16-bit boundaries, or
- 32-bit values aligned on 32-bit address boundaries.

The PCRs allow software to control the following range of electrical characteristics of the external pads:

- Weak pullup/down enable/disable
- Weak pullup/down selection
- Slew-rate selection for outputs
- Drive strength selection for outputs
- Input buffer enable (when direction is configured for output)
- Input hysteresis enable/disable
- Open drain/push-pull output selection
- Multiplexed function selection
- Data direction selection

NOTE

The actual characteristics available in any SIU_PCR depend on the type of pad it controls. Refer to the SIU_PCR definition tables in this section.

All device pin names begin with the primary function, followed by the alternate function, and then GPIO. In some cases, the third function can be a secondary alternate, which supersedes the GPIO. Those exceptions are noted in the documentation. For example, SIU_PCR85 configures the CNTXB_PCSC[3]_GPIO[85] muxed signal, where CNTXB is the primary function, PCSC[3] is the alternate function. For identification of the source module for primary and alternate functions, and the description of these signals, refer to Chapter 2, “Signal Descriptions” of this manual. Refer to the chapter for the specific module that uses the signal for an additional signal description.

Address: SIU_BASE + offset (see SIU_PCRn Settings table)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PA			OBE	IBE	DSC		ODE	HYS	SRC		WPE	WPS
W																
Reset	See SIU_PCRn Settings table for reset values															

Figure 3-14. Pad Configuration Register (SIU_PCR*n*)

The following table describes the fields in the pad configuration control registers:

Table 3-19. SIU_PCR Bit Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>0–2</td><td>Reserved</td></tr>
<tr><td>3–5
PA
[0:2]</td><td>Pin assignment. Selects the function of a multiplexed pad. A separate port enable output signal from the SIU is asserted for each value of this register. The size of the field can be from 1 to 3 bits, depending on the amount of multiplexing on the pad.
<table>
<tr><th colspan="3">PA</th><th>Pin Function [1]</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>GPIO</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>Primary function</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>Alternate 1</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>Alternate 2</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>Alternate 3</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>Invalid value</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>Invalid value</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>Invalid value</td></tr>
</table>
[1] For any SIU_PCR that does not comply with these rules, the PA definition is given explicitly with the SIU_PCR definition.</td></tr>
<tr><td>6
OBE</td><td>Output buffer enable. Enables the pad as an output and drives the output buffer enable signal.
0 Disable output buffer for the pad.
1 Enable output buffer for the pad is enabled.</td></tr>
<tr><td>7
IBE</td><td>Input buffer enable. Enables the pad as an input and drives the input buffer enable signal.
0 Disable input buffer for the pad.
1 Enable input buffer for the pad is enabled.</td></tr>
<tr><td>8–9
DSC
[0:1]</td><td>Drive strength control. Controls the pad drive strength. Drive strength control pertains to pins with the fast I/O pad type.
00 10 pF drive strength
01 20 pF drive strength
10 30 pF drive strength
11 50 pF drive strength</td></tr>
<tr><td>10
ODE</td><td>Open drain output enable. Controls output driver configuration for the pads. Either open drain or push/pull driver configurations can be selected. This feature applies to output pins only.
0 Disable open drain for the pad (push/pull driver enabled).
1 Enable open drain for the pad.</td></tr>
<tr><td>11
HYS</td><td>Input hysteresis. Controls whether hysteresis is enabled for the pad.
0 Disable hysteresis for the pad.
1 Enable hysteresis for the pad.</td></tr>
</table>

Table 3-19. SIU_PCR Bit Field Descriptions

Field	Description
12–13 SRC [0:1]	Slew rate control. Controls slew rate for the pad. Slew rate control pertains to pins with slow or medium I/O pad types, and the output signals are driven according to the value of this field. Actual slew rate depends on the pad type and load. Refer to the electrical specifications for this information. 00 Minimum slew rate 01 Medium slew rate 10 Invalid value 11 Maximum slew rate
14 WPE	Weak pullup/down enable. Controls whether the weak pullup/down devices are enabled/disabled for the pad. Pullup/down devices are enabled by default. 0 Disable weak pull device for the pad. 1 Enable weak pull device for the pad.
15 WPS	Weak pullup/down select. Controls whether weak pullup or weak pulldown devices are used for the pad when weak pullup/down devices are enabled. The WKPCFG pin determines whether pullup or pulldown devices are enabled during reset. The WPS bit determines whether weak pullup or pulldown devices are used after reset, or for pads in which the WKPCFG pin does not determine the reset weak pullup/down state. 0 Pulldown is enabled for the pad. 1 Pullup is enabled for the pad.

There are a number of input signals on the device that have multiple external pins as input sources. Only one input source can be active at any time for these pins. To achieve this, there is a predefined priority for the PCR registers controlling these pins, which is used to mux the input sources and allow only one active input. The multiple source inputs with PCR priority is given in Table 3-20.

Table 3-20. PCR priority for multiple source inputs

Function	PCR number		
	highest priority	→	lowest priority
IRQ0	193	450	—
IRQ1	194	451	—
IRQ2	211	452	—
IRQ3	212	453	—
IRQ4	208	454	—
IRQ5	209	455	—
PCSB0	105	203	—
SCKC	235	109	—
SINC	236	108	—
PCSC0	238	110	434
SCKD	189	470	—
SIND	190	472	—
PCSD0	106	469	—
CANRXD	247	194	—
TXDA	89	83	460
RXDA	90	84	461
TXDB	91	462	—
RXDB	92	463	—
FR_RX_A	249	458	—

The symbols in the SIU_PCRn bit fields in table on the following pages represent the reset state of the bits. The meaning of each symbol is given in Table 3-21.

Table 3-21. SIU_PCRn Bit Field Symbols

Symbol	Meaning
—	Not implemented in the PCR. Software should write a default value of 0.
0	Function is implemented and disabled on reset.
1	Function is implemented and enabled on reset.
U	Function is implemented and is defined by the WKPCFG pin at reset.

Table 3-22. SIU_PCRn Settings

PCRn	Address Offset	GPIO	Primary Function	A2	A3	A4	SIU_PCRn[3:15] (SIU_PCRn[0:2] = Reserved, should be cleared)												
							3	4	5	6	7	8	9	10	11	12	13	14	15
							PA2	PA1	PA0	OBE	IBE	DSC1	DSC0	ODE	HYS	SRC1	SRC0	WPE	WPS
75	0x00D6	GPIO75	MDO4	—	—	—	—	—	—	0	0	1	1	0	0	—	—	1	0
76	0x00D8	GPIO76	MDO5	—	—	—	—	—	—	0	0	1	1	0	0	—	—	1	0
77	0x00DA	GPIO77	MDO6	—	—	—	—	—	—	0	0	1	1	0	0	—	—	1	0
78	0x00DC	GPIO78	MDO7	—	—	—	—	—	—	0	0	1	1	0	0	—	—	1	0
79	0x00DE	GPIO79	MDO8	—	—	—	—	—	—	0	0	1	1	0	0	—	—	1	0
80	0x00E0	GPIO80	MDO9	—	—	—	—	—	—	0	0	1	1	0	0	—	—	1	0
81	0x00E2	GPIO81	MDO10	—	—	—	—	—	—	0	0	1	1	0	0	—	—	1	0
82	0x00E4	GPIO82	MDO11	—	—	—	—	—	—	0	0	1	1	0	0	—	—	1	0
83	0x00E6	GPIO83	CNTXA	TXDA	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
84	0x00E8	GPIO84	CNRXA	RXDA	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
85	0x00EA	GPIO85	CNTXB	PCSC3	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
86	0x00EC	GPIO86	CNRXB	PCSC4	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
87	0x00EE	GPIO87	CNTXC	PCSD3	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
88	0x00F0	GPIO88	CNRXC	PCSD4	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
89	0x00F2	GPIO89	TXDA	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	1
90	0x00F4	GPIO90	RXDA	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	1
91	0x00F6	GPIO91	TXDB	PCSD1	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
92	0x00F8	GPIO92	RXDB	PCSD5	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
93	0x00FA	GPIO93	SCKA	PCSC1	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
94	0x00FC	GPIO94	SINA	PCSC2	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
95	0x00FE	GPIO95	SOUTA	PCSC5	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
96	0x0100	GPIO96	PCSA0	PCSD2	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1

Table 3-22. SIU_PCRn Settings

PCRn	Address Offset	GPIO	Primary Function	A2	A3	A4	SIU_PCRn[3:15] (SIU_PCRn[0:2] = Reserved, should be cleared)												
							3	4	5	6	7	8	9	10	11	12	13	14	15
							PA2	PA1	PA0	OBE	IBE	DSC1	DSC0	ODE	HYS	SRC1	SRC0	WPE	WPS
97	0x0102	GPIO97	PCSA1	PCSE0	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
98	0x0104	GPIO98	PCSA2	SOUTE	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
99	0x0106	GPIO99	PCSA3	SINE	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
100	0x0108	GPIO100	PCSA4	SCKE	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
101	0x010A	GPIO101	PCSA5	ETRIG1	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
102	0x010C	GPIO102	SCKB	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	1
103	0x010E	GPIO103	SINB	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	1
104	0x0110	GPIO104	SOUTB	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	1
105	0x0112	GPIO105	PCSB0	PCSD2	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
106	0x0114	GPIO106	PCSB1	PCSD0	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
107	0x0116	GPIO107	PCSB2	SOUTC	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
108	0x0118	GPIO108	PCSB3	SINC	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
109	0x011A	GPIO109	PCSB4	SCKC	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
110	0x011C	GPIO110	PCSB5	PCSC0	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
113	0x0122	GPIO113	TCRCLKA	IRQ7	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
114	0x0124	GPIO114	ETPUA0	ETPUA12	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
115	0x0126	GPIO115	ETPUA1	ETPUA13	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
116	0x0128	GPIO116	ETPUA2	ETPUA14	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
117	0x012A	GPIO117	ETPUA3	ETPUA15	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
118	0x012C	GPIO118	ETPUA4	ETPUA16	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
119	0x012E	GPIO119	ETPUA5	ETPUA17	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
120	0x0130	GPIO120	ETPUA6	ETPUA18	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
121	0x0132	GPIO121	ETPUA7	ETPUA19	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U

Table 3-22. SIU_PCRn Settings

PCRn	Address Offset	GPIO	Primary Function	A2	A3	A4	SIU_PCRn[3:15] (SIU_PCRn[0:2] = Reserved, should be cleared)												
							3	4	5	6	7	8	9	10	11	12	13	14	15
							PA2	PA1	PA0	OBE	IBE	DSC1	DSC0	ODE	HYS	SRC1	SRC0	WPE	WPS
122	0x0134	GPIO122	ETPUA8	ETPUA20	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
123	0x0136	GPIO123	ETPUA9	ETPUA21	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
124	0x0138	GPIO124	ETPUA10	ETPUA22	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
125	0x013A	GPIO125	ETPUA11	ETPUA23	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
126	0x013C	GPIO126	ETPUA12	PCSB1	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
127	0x013E	GPIO127	ETPUA13	PCSB3	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
128	0x0140	GPIO128	ETPUA14	PCSB4	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
129	0x0142	GPIO129	ETPUA15	PCSB5	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
130	0x0144	GPIO130	ETPUA16	PCSD1	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
131	0x0146	GPIO131	ETPUA17	PCSD2	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
132	0x0148	GPIO132	ETPUA18	PCSD3	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
133	0x014A	GPIO133	ETPUA19	PCSD4	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
134	0x014C	GPIO134	ETPUA20	IRQ8	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
135	0x014E	GPIO135	ETPUA21	IRQ9	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
136	0x0150	GPIO136	ETPUA22	IRQ10	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
137	0x0152	GPIO137	ETPUA23	IRQ11	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
138	0x0154	GPIO138	ETPUA24	IRQ12	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
139	0x0156	GPIO139	ETPUA25	IRQ13	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
140	0x0158	GPIO140	ETPUA26	IRQ14	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
141	0x015A	GPIO141	ETPUA27	IRQ15	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
142	0x015C	GPIO142	ETPUA28	PCSC1	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
143	0x015E	GPIO143	ETPUA29	PCSC2	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
144	0x0160	GPIO144	ETPUA30	PCSC3	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U

Table 3-22. SIU_PCRn Settings

PCRn	Address Offset	GPIO	Primary Function	A2	A3	A4	SIU_PCRn[3:15] (SIU_PCRn[0:2] = Reserved, should be cleared)												
							3	4	5	6	7	8	9	10	11	12	13	14	15
							PA2	PA1	PA0	OBE	IBE	DSC1	DSC0	ODE	HYS	SRC1	SRC0	WPE	WPS
145	0x0162	GPIO145	ETPUA31	PCSC4	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
146	0x0164	GPIO146	TCRCLKB	IRQ6	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
147	0x0166	GPIO147	ETPUB0	ETPUB16	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
148	0x0168	GPIO148	ETPUB1	ETPUB17	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
149	0x016A	GPIO149	ETPUB2	ETPUB18	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
150	0x016C	GPIO150	ETPUB3	ETPUB19	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
151	0x016E	GPIO151	ETPUB4	ETPUB20	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
152	0x0170	GPIO152	ETPUB5	ETPUB21	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
153	0x0172	GPIO153	ETPUB6	ETPUB22	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
154	0x0174	GPIO154	ETPUB7	ETPUB23	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
155	0x0176	GPIO155	ETPUB8	ETPUB24	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
156	0x0178	GPIO156	ETPUB9	ETPUB25	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
157	0x017A	GPIO157	ETPUB10	ETPUB26	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
158	0x017C	GPIO158	ETPUB11	ETPUB27	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
159	0x017E	GPIO159	ETPUB12	ETPUB28	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
160	0x0180	GPIO160	ETPUB13	ETPUB29	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
161	0x0182	GPIO161	ETPUB14	ETPUB30	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
162	0x0184	GPIO162	ETPUB15	ETPUB31	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
163	0x0186	GPIO163	ETPUB16	PCSA1	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
164	0x0188	GPIO164	ETPUB17	PCSA2	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
165	0x018A	GPIO165	ETPUB18	PCSA3	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
166	0x018C	GPIO166	ETPUB19	PCSA4	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
167	0x018E	GPIO167	ETPUB20	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	U

Table 3-22. SIU_PCRn Settings

PCRn	Address Offset	GPIO	Primary Function	A2	A3	A4	SIU_PCRn[3:15] (SIU_PCRn[0:2] = Reserved, should be cleared)												
							3	4	5	6	7	8	9	10	11	12	13	14	15
							PA2	PA1	PA0	OBE	IBE	DSC1	DSC0	ODE	HYS	SRC1	SRC0	WPE	WPS
168	0x0190	GPIO168	ETPUB21	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	U
169	0x0192	GPIO169	ETPUB22	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	U
170	0x0194	GPIO170	ETPUB23	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	U
171	0x0196	GPIO171	ETPUB24	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	U
172	0x0198	GPIO172	ETPUB25	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	U
173	0x019A	GPIO173	ETPUB26	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	U
174	0x019C	GPIO174	ETPUB27	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	U
175	0x019E	GPIO175	ETPUB28	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	U
176	0x01A0	GPIO176	ETPUB29	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	U
177	0x01A2	GPIO177	ETPUB30	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	U
178	0x01A4	GPIO178	ETPUB31	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	U
179	0x01A6	GPIO179	EMIOS0	ETPUA0	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
180	0x01A8	GPIO180	EMIOS1	ETPUA1	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
181	0x01AA	GPIO181	EMIOS2	ETPUA2	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
182	0x01AC	GPIO182	EMIOS3	ETPUA3	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
183	0x01AE	GPIO183	EMIOS4	ETPUA4	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
184	0x01B0	GPIO184	EMIOS5	ETPUA5	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
185	0x01B2	GPIO185	EMIOS6	ETPUA6	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
186	0x01B4	GPIO186	EMIOS7	ETPUA7	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
187	0x01B6	GPIO187	EMIOS8	ETPUA8	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
188	0x01B8	GPIO188	EMIOS9	ETPUA9	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
189	0x01BA	GPIO189	EMIOS10	SCKD	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
190	0x01BC	GPIO190	EMIOS11	SIND	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U

Table 3-22. SIU_PCRn Settings

PCRn	Address Offset	GPIO	Primary Function	A2	A3	A4	SIU_PCRn[3:15] (SIU_PCRn[0:2] = Reserved, should be cleared)												
							3	4	5	6	7	8	9	10	11	12	13	14	15
							PA2	PA1	PA0	OBE	IBE	DSC1	DSC0	ODE	HYS	SRC1	SRC0	WPE	WPS
191	0x01BE	GPIO191	EMIOS12	SOUTC	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
192	0x01C0	GPIO192	EMIOS13	SOUTD	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
193	0x01C2	GPIO193	EMIOS14	IRQ0	CNTXD	—	—	0	0	0	0	—	—	0	0	0	0	1	U
194	0x01C4	GPIO194	EMIOS15	IRQ1	CNRXD	—	—	0	0	0	0	—	—	0	0	0	0	1	U
195	0x01C6	GPIO195	EMIOS16	ETPUB0	FR_DBG[3]	—	—	0	0	0	0	—	—	0	0	0	0	1	U
196	0x01C8	GPIO196	EMIOS17	ETPUB1	FR_DBG[2]	—	—	0	0	0	0	—	—	0	0	0	0	1	U
197	0x01CA	GPIO197	EMIOS18	ETPUB2	FR_DBG[1]	—	—	0	0	0	0	—	—	0	0	0	0	1	U
198	0x01CC	GPIO198	EMIOS19	ETPUB3	FR_DBG[0]	—	—	0	0	0	0	—	—	0	0	0	0	1	U
199	0x01CE	GPIO199	EMIOS20	ETPUB4	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
200	0x01D0	GPIO200	EMIOS21	ETPUB5	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
201	0x01D2	GPIO201	EMIOS22	ETPUB6	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
202	0x01D4	GPIO202	EMIOS23	ETPUB7	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
203	0x01D6	GPIO203	EMIOS24	PCSB0	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
204	0x01D8	GPIO204	EMIOS25	PCSB1	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
208	0x01E0	GPIO208	PLLCFG0	IRQ4	—	—	—	0	1	0	0	—	—	0	1	0	0	1	1
209	0x01E2	GPIO209	PLLCFG1	IRQ5	SOUTD	—	—	0	1	0	0	—	—	0	1	0	0	1	1
211	0x01E6	GPIO211	BOOTCFG0	IRQ2	—	—	—	0	1	0	0	—	—	0	1	0	0	1	0
212	0x01E8	GPIO212	BOOTCFG1	IRQ3	—	—	—	0	1	0	0	—	—	0	1	0	0	1	0
213	0x01EA	GPIO213	WKPCFG	NMI	—	—	—	—	1	0	0	—	—	—	1	0	0	1	1
214	0x01EC	—	ENGCLK	—	—	—	—	—	—	1	—	1	1	—	—	—	—	—	—
219	0x01F6	—	MCKO	—	—	—	—	—	—	—	—	1	1	—	—	—	—	—	—
220	0x01F8	GPIO220	MDO0	—	—	—	—	—	—	0	0	1	1	0	0	—	—	1	1
221	0x01FA	GPIO221	MDO1	—	—	—	—	—	—	0	0	1	1	0	0	—	—	1	0

Table 3-22. SIU_PCRn Settings

PCRn	Address Offset	GPIO	Primary Function	A2	A3	A4	SIU_PCRn[3:15] (SIU_PCRn[0:2] = Reserved, should be cleared)												
							3	4	5	6	7	8	9	10	11	12	13	14	15
							PA2	PA1	PA0	OBE	IBE	DSC1	DSC0	ODE	HYS	SRC1	SRC0	WPE	WPS
222	0x01FC	GPIO222	MDO2	—	—	—	—	—	—	0	0	1	1	0	0	—	—	1	0
223	0x01FE	GPIO223	MDO3	—	—	—	—	—	—	0	0	1	1	0	0	—	—	1	0
224	0x0200	—	MSEO0	—	—	—	—	—	—	—	—	1	1	—	—	—	—	—	—
225	0x0202	—	MSEO1	—	—	—	—	—	—	—	—	1	1	—	—	—	—	—	—
226	0x0204	—	RDY	—	—	—	—	—	—	—	—	1	1	—	—	—	—	—	—
227	0x0206	—	EVTO	—	—	—	—	—	—	—	—	1	1	—	—	—	—	—	—
228	0x0208	—	TDO	—	—	—	—	—	—	—	—	1	1	—	—	—	—	—	—
229	0x020A	—	D_CLKOUT	—	—	—	—	—	—	1	—	1	1	—	—	—	—	—	—
230	0x020C	—	RSTOUT	—	—	—	—	—	—	—	—	—	—	—	—	1	1	—	—
231	0x020E	GPIO231	MDO12	—	—	—	—	—	—	0	0	1	1	0	0	—	—	1	0
232	0x0210	GPIO232	MDO13	—	—	—	—	—	—	0	0	1	1	0	0	—	—	1	0
233	0x0212	GPIO233	MDO14	—	—	—	—	—	—	0	0	1	1	0	0	—	—	1	0
234	0x0214	GPIO234	MDO15	—	—	—	—	—	—	0	0	1	1	0	0	—	—	1	0
235	0x0216	GPIO235	SCKC	SCK_C_LVDS+	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
236	0x0218	GPIO236	SINC	SCK_C_LVDS—	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
237	0x021A	GPIO237	SOUTC	SOUT_C_LVDS+	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
238	0x021C	GPIO238	PCSC0	SOUT_C_LVDS—	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
239	0x021E	GPIO239	PCSC1	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	1
240	0x0220	GPIO240	PCSC2	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	1
241	0x0222	GPIO241	PCSC3	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	1
242	0x0224	GPIO242	PCSC4	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	1
243	0x0226	GPIO243	PCSC5	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	1

Table 3-22. SIU_PCRn Settings

PCRn	Address Offset	GPIO	Primary Function	A2	A3	A4	SIU_PCRn[3:15] (SIU_PCRn[0:2] = Reserved, should be cleared)												
							3	4	5	6	7	8	9	10	11	12	13	14	15
							PA2	PA1	PA0	OBE	IBE	DSC1	DSC0	ODE	HYS	SRC1	SRC0	WPE	WPS
244	0x0228	GPIO244	TXDC	ETRIG0	—	—	—	0	0	0	0	—	—	0	0	0	0	1	1
245	0x022A	GPIO245	RXDC	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	1
246	0x022C	GPIO246	CNTXD	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	1
247	0x022E	GPIO247	CNRXD	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	1
248	0x0230	GPIO248	FR_A_TX	—	—	—	—	—	0	0	0	—	—	0	0	1	1	0	1
249	0x0232	GPIO249	FR_A_RX	—	—	—	—	—	0	0	0	—	—	0	0	1	1	1	0
250	0x0234	GPIO250	FR_A_TX_EN	—	—	—	—	—	0	0	0	—	—	0	0	1	1	1	1
251	0x0236	GPIO251	FR_B_TX	—	—	—	—	—	0	0	0	—	—	0	0	1	1	0	1
252	0x0238	GPIO252	FR_B_RX	—	—	—	—	—	0	0	0	—	—	0	0	1	1	1	0
253	0x023A	GPIO253	FR_B_TX_EN	—	—	—	—	—	0	0	0	—	—	0	0	1	1	1	1
256	0x0240	GPIO256	D_CS0	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
257	0x0242	GPIO257	D_CS2	D_ADD_DAT31	—	—	—	0	0	0	0	1	1	0	0	—	—	1	1
258	0x0244	GPIO258	D_CS3	D_TEA	—	—	—	0	0	0	0	1	1	0	0	—	—	1	1
259	0x0246	GPIO259	D_ADD12	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
260	0x0248	GPIO260	D_ADD13	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
261	0x024A	GPIO261	D_ADD14	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
262	0x024C	GPIO262	D_ADD15	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
263	0x024E	GPIO263	D_ADD16	D_ADD_DAT16	—	—	—	0	0	0	0	1	1	0	0	—	—	1	1
264	0x0250	GPIO264	D_ADD17	D_ADD_DAT17	—	—	—	0	0	0	0	1	1	0	0	—	—	1	1
265	0x0252	GPIO265	D_ADD18	D_ADD_DAT18	—	—	—	0	0	0	0	1	1	0	0	—	—	1	1
266	0x0254	GPIO266	D_ADD19	D_ADD_DAT19	—	—	—	0	0	0	0	1	1	0	0	—	—	1	1
267	0x0256	GPIO267	D_ADD20	D_ADD_DAT20	—	—	—	0	0	0	0	1	1	0	0	—	—	1	1
268	0x0258	GPIO268	D_ADD21	D_ADD_DAT21	—	—	—	0	0	0	0	1	1	0	0	—	—	1	1

Table 3-22. SIU_PCRn Settings

PCRn	Address Offset	GPIO	Primary Function	A2	A3	A4	SIU_PCRn[3:15] (SIU_PCRn[0:2] = Reserved, should be cleared)												
							3	4	5	6	7	8	9	10	11	12	13	14	15
							PA2	PA1	PA0	OBE	IBE	DSC1	DSC0	ODE	HYS	SRC1	SRC0	WPE	WPS
269	0x025A	GPIO269	D_ADD22	D_ADD_DAT22	—	—	—	0	0	0	0	1	1	0	0	—	—	1	1
270	0x025C	GPIO270	D_ADD23	D_ADD_DAT23	—	—	—	0	0	0	0	1	1	0	0	—	—	1	1
271	0x025E	GPIO271	D_ADD24	D_ADD_DAT24	—	—	—	0	0	0	0	1	1	0	0	—	—	1	1
272	0x0260	GPIO272	D_ADD25	D_ADD_DAT25	—	—	—	0	0	0	0	1	1	0	0	—	—	1	1
273	0x0262	GPIO273	D_ADD26	D_ADD_DAT26	—	—	—	0	0	0	0	1	1	0	0	—	—	1	1
274	0x0264	GPIO274	D_ADD27	D_ADD_DAT27	—	—	—	0	0	0	0	1	1	0	0	—	—	1	1
275	0x0266	GPIO275	D_ADD28	D_ADD_DAT28	—	—	—	0	0	0	0	1	1	0	0	—	—	1	1
276	0x0268	GPIO276	D_ADD29	D_ADD_DAT29	—	—	—	0	0	0	0	1	1	0	0	—	—	1	1
277	0x026A	GPIO277	D_ADD30	D_ADD_DAT30	—	—	—	0	0	0	0	1	1	0	0	—	—	1	1
278	0x026C	GPIO278	D_ADD_DAT0	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
279	0x026E	GPIO279	D_ADD_DAT1	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
280	0x0270	GPIO280	D_ADD_DAT2	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
281	0x0272	GPIO281	D_ADD_DAT3	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
282	0x0274	GPIO282	D_ADD_DAT4	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
283	0x0276	GPIO283	D_ADD_DAT5	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
284	0x0278	GPIO284	D_ADD_DAT6	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
285	0x027A	GPIO285	D_ADD_DAT7	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
286	0x027C	GPIO286	D_ADD_DAT8	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
287	0x027E	GPIO287	D_ADD_DAT9	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
288	0x0280	GPIO288	D_ADD_DAT10	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
289	0x0282	GPIO289	D_ADD_DAT11	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
290	0x0284	GPIO290	D_ADD_DAT12	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
291	0x0286	GPIO291	D_ADD_DAT13	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1

Table 3-22. SIU_PCRn Settings

PCRn	Address Offset	GPIO	Primary Function	A2	A3	A4	**SIU_PCRn[3:15]** (SIU_PCRn[0:2] = Reserved, should be cleared)												
							3	4	5	6	7	8	9	10	11	12	13	14	15
							PA2	PA1	PA0	OBE	IBE	DSC1	DSC0	ODE	HYS	SRC1	SRC0	WPE	WPS
292	0x0288	GPIO292	D_ADD_DAT14	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
293	0x028A	GPIO293	D_ADD_DAT15	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
294	0x028C	GPIO294	D_RD_WR	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
295	0x028E	GPIO295	D_WE0	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
296	0x0290	GPIO296	D_WE1	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
297	0x0292	GPIO297	D_OE	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
298	0x0294	GPIO298	D_TS	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
299	0x0296	GPIO299	D_ALE	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
300	0x0298	GPIO300	D_TA	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
301	0x029A	GPIO301	D_CS1	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
302	0x029C	GPIO302	D_BDIP	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
303	0x029E	GPIO303	D_WE2	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
304	0x02A0	GPIO304	D_WE3	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
305	0x02A2	GPIO305	D_ADD9	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
306	0x02A4	GPIO306	D_ADD10	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
307	0x02A6	GPIO307	D_ADD11	—	—	—	—	—	0	0	0	1	1	0	0	—	—	1	1
432	0x03A0	GPIO432	EMIOS26	PCSB2	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
433	0x03A2	GPIO433	EMIOS27	PCSB3	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
434	0x03A4	GPIO434	EMIOS28	PCSC0	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
435	0x03A6	GPIO435	EMIOS29	PCSC1	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
436	0x03A8	GPIO436	EMIOS30	PCSC2	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
437	0x03AA	GPIO437	EMIOS31	PCSC5	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
440	0x03B0	GPIO440	TCRCLKC	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	1

Table 3-22. SIU_PCRn Settings

PCRn	Address Offset	GPIO	Primary Function	A2	A3	A4	SIU_PCRn[3:15] (SIU_PCRn[0:2] = Reserved, should be cleared)												
							3	4	5	6	7	8	9	10	11	12	13	14	15
							PA2	PA1	PA0	OBE	IBE	DSC1	DSC0	ODE	HYS	SRC1	SRC0	WPE	WPS
441	0x03B2	GPIO441	ETPUC0	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	U
442	0x03B4	GPIO442	ETPUC1	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	U
443	0x03B6	GPIO443	ETPUC2	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	U
444	0x03B8	GPIO444	ETPUC3	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	U
445	0x03BA	GPIO445	ETPUC4	PCSE1	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
446	0x03BC	GPIO446	ETPUC5	PCSE2	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
447	0x03BE	GPIO447	ETPUC6	PCSE3	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
448	0x03C0	GPIO448	ETPUC7	PCSE4	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
449	0x03C2	GPIO449	ETPUC8	PCSE5	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
450	0x03C4	GPIO450	ETPUC9	IRQ0		—	—	0	0	0	0	—	—	0	0	0	0	1	U
451	0x03C6	GPIO451	ETPUC10	IRQ1		—	—	0	0	0	0	—	—	0	0	0	0	1	U
452	0x03C8	GPIO452	ETPUC11	IRQ2		—	—	0	0	0	0	—	—	0	0	0	0	1	U
453	0x03CA	GPIO453	ETPUC12	IRQ3		—	—	0	0	0	0	—	—	0	0	0	0	1	U
454	0x03CC	GPIO454	ETPUC13	IRQ4		—	—	0	0	0	0	—	—	0	0	0	0	1	U
455	0x03CE	GPIO455	ETPUC14	IRQ5		—	—	0	0	0	0	—	—	0	0	0	0	1	U
456	0x03D0	GPIO456	ETPUC15	—	—	—	—	—	0	0	0	—	—	0	0	0	0	1	U
457	0x03D2	GPIO457	ETPUC16	FR_A_TX	—	—	—	0	0	0	0	—	—	0	0	0	0	0	0
458	0x03D4	GPIO458	ETPUC17	FR_A_RX	—	—	—	0	0	0	0	—	—	0	0	0	0	1	0
459	0x03D6	GPIO459	ETPUC18	FR_A_TX_EN	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
460	0x03D8	GPIO460	ETPUC19	TXDA	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
461	0x03DA	GPIO461	ETPUC20	RXDA	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
462	0x03DC	GPIO462	ETPUC21	TXDB	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
463	0x03DE	GPIO463	ETPUC22	RXDB	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U

Table 3-22. SIU_PCRn Settings

PCRn	Address Offset	GPIO	Primary Function	A2	A3	A4	SIU_PCRn[3:15] (SIU_PCRn[0:2] = Reserved, should be cleared)												
							3	4	5	6	7	8	9	10	11	12	13	14	15
							PA2	PA1	PA0	OBE	IBE	DSC1	DSC0	ODE	HYS	SRC1	SRC0	WPE	WPS
464	0x03E0	GPIO464	ETPUC23	PCSD5	MAA0	MAB0	0	0	0	0	0	—	—	0	0	0	0	1	U
465	0x03E2	GPIO465	ETPUC24	PCSD4	MAA1	MAB1	0	0	0	0	0	—	—	0	0	0	0	1	U
466	0x03E4	GPIO466	ETPUC25	PCSD3	MAA2	MAB2	0	0	0	0	0	—	—	0	0	0	0	1	U
467	0x03E6	GPIO467	ETPUC26	PCSD2	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
468	0x03E8	GPIO468	ETPUC27	PCSD1	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
469	0x03EA	GPIO469	ETPUC28	PCSD0	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
470	0x03EC	GPIO470	ETPUC29	SCKD	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
471	0x03EE	GPIO471	ETPUC30	SOUTD	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U
472	0X03F0	GPIO472	ETPUC31	SIND	—	—	—	0	0	0	0	—	—	0	0	0	0	1	U

3.2.1.14 GPIO Pin Data Output Registers 0–512 (SIU_GPDO*n*)

The 8-bit SIU_GPDO*n* registers defined in Figure 3-15 each specify the output data for the function assigned to the GPIO[*n*] pin. The *n* notation in the SIU_GPDO*n* register name relate to the [*n*] in GPIO[*n*] signal name. For example, SIU_GPDO246 contains the PDO246 bit for CNTXD_GPIO246. The address for a GPDO pin is the GPIO number plus an offset of SIU_BASE + 0x0600.

Software writes to the SIU_GPDO*n* registers to drive data out on the external pin. Each register drives one external pin, which allows independent control of the pin. Writes to the SIU_GPDO*n* registers have no effect if an input function is assigned to the pin by the pad configuration register.

If the direction of a GPIO signal changes from input to output, the SIU_GPDO*n* register value is automatically driven out to the external pin without a software update.

Writes to the SIU_GPDO*n* registers have no effect when a primary or alternate function is assigned.

Address: SIU_BASE + 0x0600 + n R/W

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W								PDO*n*
Reset	0	0	0	0	0	0	0	0

Figure 3-15. General Purpose Data Output (GPDO) Registers 0–512 (SIU_GPDO*n*)

Table 3-23. SIU_GPDO Bit Field Descriptions

Name	Description
PDO*n*	Pin data out. Stores the data to drive out the external GPIO. If the register is read, it returns the value written. 0 A logic 0 is driven on the external GPIO pin when the pin is configured as an output. 1 A logic 1 is driven on the external GPIO pin when the pin is configured as an output.

3.2.1.15 GPIO Pin Data Input Registers 0–255 (SIU_GPDI*n*)

NOTE

> This register is implemented for legacy purposes and is limited to 256 general purpose input registers. For full access to all 512 general purpose input registers, the SIU_GPDIx registers in Section 3.2.1.39, “GPIO Pin Data Input Registers (SIU_GPDI0_3 - SIU_GPDI508_511),” should be used.

The 8-bit read-only SIU_GPDI*n* registers defined in Figure 3-16 each specify the input state for the function assigned to the GPDI[*n*] pin. The *n* notation in the SIU_GPDI*n* register name relates to the [*n*] in GPIO[*n*] signal name. For example, SIU_GPDI246 contains the PDI246 bit for CNTXD_GPIO246. The GPDI address for a particular pin is the GPIO number plus an offset of SIU_BASE + 0x0800. Gaps exist in the memory where GPIO pins are not implemented in the package.

Software reads the SIU_GPDI*n* registers to get the input state of the external GPIO pin. Each GPDI register contains the input state of one external GPIO pin. If a GPDI register is configured as output, and the input buffer enable bit is set to one in the PCR register, the SIU_GPDI*n* register reflects the state of the output pin.

Address: SIU_BASE + 0x0800 + n

Read Only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	PDI*n*
W								
Reset	0	0	0	0	0	0	0	0

Figure 3-16. General Purpose Data Input (GPDI) Registers 0–255 (SIU_GPDI*n*)

Table 3-24. SIU_GPDI Bit Field Descriptions

Name	Description
PDI*n*	Pin data in. This bit reflects the input state on the external GPIO pin for the register. If PCR*n*[IBE] = 1, then: 0 Signal on pin is a logic 0. 1 Signal on pin is a logic 1.

3.2.1.16 External IRQ Input Select Register (SIU_EIISR)

The SIU_EIISR selects the source for the external interrupt/DMA inputs.

Address: SIU_BASE + 0x0904 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ESEL15		ESEL14		ESEL13		ESEL12		ESEL11		ESEL10		ESEL9		ESEL8	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ESEL7		ESEL6		ESEL5		ESEL4		ESEL3		ESEL2		ESEL1		ESEL0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-17. External IRQ Input Select Register (SIU_EIISR)

The following table describes the external IRQ input select fields:

Table 3-25. SIU_EIISR Bit Field Descriptions

Bits	Name	Description
0–1	ESEL15 [0:1]	External IRQ input select 15. Specifies the input for $\overline{\text{IRQ}}$[15]. 00 $\overline{\text{IRQ}}$[15] 01 PCSB[15] 10 PCSC[0] 11 PCSD[1] serialized input (ETPUA[20])
2–3	ESEL14 [0:1]	External IRQ input select 14. Specifies the input for $\overline{\text{IRQ}}$[14]. 00 $\overline{\text{IRQ}}$[14] 01 PCSB[14] 10 PCSC[15] 11 PCSD[0] serialized input (ETPUA[21])
4–5	ESEL13 [0:1]	External IRQ input select 13. Specifies the input for $\overline{\text{IRQ}}$[13]. 00 $\overline{\text{IRQ}}$[13] 01 PCSB[13] 10 PCSC[14] 11 PCSD[15] serialized input (ETPUA[24])
6–7	ESEL12 [0:1]	External IRQ input select 12. Specifies the input for $\overline{\text{IRQ}}$[12]. 00 $\overline{\text{IRQ}}$[12] pin 01 PCSB[12] 10 PCSC[13] 11 PCSD[14] serialized input (ETPUA[25])
8–9	ESEL11 [0:1]	External IRQ input select 11. Specifies the input for $\overline{\text{IRQ}}$[11]. 00 $\overline{\text{IRQ}}$[11] 01 PCSB[11] 10 PCSC[12] 11 PCSD[13] serialized input (ETPUA[26])

Table 3-25. SIU_EIISR Bit Field Descriptions

Bits	Name	Description
10–11	ESEL10 [0:1]	External IRQ input select 10. Specifies the input for $\overline{\text{IRQ}}$10]. 00 $\overline{\text{IRQ}}$[10] 01 PCSB[10] 10 PCSC[11] 11 PCSD[12] serialized input (ETPUA[27])
12–13	ESEL9 [0:1]	External IRQ input select 9. Specifies the input for $\overline{\text{IRQ}}$[9]. 00 $\overline{\text{IRQ}}$[9] 01 PCSB[9] 10 PCSC[10] 11 PCSD[11] serialized input (ETPUA[28])
14–15	ESEL8 [0:1]	External IRQ input select 8. Specifies the input for $\overline{\text{IRQ}}$[8]. 00 $\overline{\text{IRQ}}$[8] 01 PCSB[8] 10 PCSC[9] 11 PCSD[10] serialized input (ETPUA[29])
16–17	ESEL7 [0:1]	External IRQ input select 7. Specifies the input for $\overline{\text{IRQ}}$[7]. 00 $\overline{\text{IRQ}}$[7] 01 PCSB[7] 10 PCSC[8] 11 PCSD[9] serialized input (EMIOS[12])
20–21	ESEL5 [0:1]	External IRQ input select 5. Specifies the input for $\overline{\text{IRQ}}$[5]. 00 $\overline{\text{IRQ}}$[5] 01 PCSB[5] 10 PCSC[6] 11 PCSD[7] serialized input (EMIOS[10])
22–23	ESEL4 [0:1]	External IRQ input select 4. Specifies the input for $\overline{\text{IRQ}}$[4]. 00 $\overline{\text{IRQ}}$[4] 01 PCSB[4] 10 PCSC[5] 11 PCSD[6] serialized input (EMIOS[11])
24–25	ESEL3 [0:1]	External IRQ input select 3. Specifies the input for $\overline{\text{IRQ}}$[3]. 00 $\overline{\text{IRQ}}$[3] 01 PCSB[3] 10 PCSC[4] 11 PCSD[5] serialized input (ETPUA[16])
26–27	ESEL2 [0:1]	External IRQ input select 2. Specifies the input for $\overline{\text{IRQ}}$[2]. 00 $\overline{\text{IRQ}}$[2] 01 PCSB[2] 10 PCSC[3] 11 PCSD[4] serialized input (ETPUA[17])

Table 3-25. SIU_EIISR Bit Field Descriptions

Bits	Name	Description
28–29	ESEL1 [0:1]	External IRQ input select 1. Specifies the input for $\overline{\text{IRQ}}$[1]. 00 $\overline{\text{IRQ}}$[1] 01 PCSB[1] input (EMIOS[10]) 10 PCSC[2] 11 EMIOS[15]
30–31	ESEL0 [0:1]	External IRQ input select 0. Specifies the input for $\overline{\text{IRQ}}$[0]. 00 $\overline{\text{IRQ}}$[0] 01 PCSB[0] input (EMIOS[11]) 10 PCSC[1] 11 EMIOS[14]

3.2.1.17 DSPI Input Select Register (SIU_DISR)

The SIU_DISR specifies the following operations for each DSPI:

- Data input source
- Slave select
- Clock input

Trigger input to allow serial and parallel chaining of the DSPI modules.

Address: SIU_BASE + 0x0908 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SINSELA		SSSELA		SCKSELA		TRIGSELA		SINSELB		SSSELB		SCKSELB		TRIGSELB	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	SINSELC		SSSELC		SCKSELC		TRIGSELC		SINSELD		SSSELD		SCKSELD		TRIGSELD	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-18. DSPI Input Select Register (SIU_DISR)

The following table describes the DSPI input select fields:

Table 3-26. SIU_DISR Bit Field Descriptions

Bits	Name	Description
0–1	SINSELA [0:1]	DSPI A data input select. Specifies the source of the DSPI A data input. 00 SINA_PCSC[2]_GPIO[94] pin 01 SOUTB 10 SOUTC 11 SOUTD
2–3	SSSELA [0:1]	DSPI A slave select input select. Specifies the source of the DSPI A slave select input. 00 PCSA[0]_PCSD[2]_GPIO[96] pin 01 PCSB[0] (master) 10 PCSC[0] (master) 11 PCSD[0] (master)
4–5	SCKSELA [0:1]	DSPI A clock input select. Specifies the source of the DSPI A clock input. 00 SCKA_PCSC[1]_GPIO[93] pin 01 SCKB (master) 10 SCKC (master) 11 SCKD (master)
6–7	TRIGSELA [0:1]	DSPI A trigger input select. Specifies the source of the DSPI A trigger input. 00 No Trigger 01 PCSB[4] 10 PCSC[4] 11 PCSD[4]
8–9	SINSELB [0:1]	DSPI B data input select. Specifies the source of DSPI B data input. 00 SINB_GPIO[103] pin 01 SOUTA 10 SOUTC 11 SOUTD
10–11	SSSELB [0:1]	DSPI B slave select input select. Specifies the source of the DSPI B slave select input. 00 PCSB[0]_PCSD[2]_GPIO[105] pin 01 PCSA[0] (master) 10 PCSC[0] (master) 11 PCSD[0] (master)
12–13	SCKSELB [0:1]	DSPI B clock input select. Specifies the source of the DSPI B clock input. 00 SCKB_GPIO[102] pin 01 SCKA (master) 10 SCKC (master) 11 SCKD (master)
14–15	TRIGSELB [0:1]	DSPI B trigger input select. Specifies the source of the DSPI B trigger input for master or slave mode. 00 Invalid value 01 PCSA[4] 10 PCSC[4] 11 PCSD[4]

Table 3-26. SIU_DISR Bit Field Descriptions

Bits	Name	Description
16–17	SINSELC [0:1]	DSPI C data input select. Specifies the source of the DSPI C data input. 00 PCSB[3]_SINC_GPIO[108] pin 01 SOUTA 10 SOUTB 11 SOUTD
18–19	SSSELC [0:1]	DSPI C slave select input select. Specifies the source of the DSPI C slave select input. 00 PCSB[5]_PCSC[0]_GPIO[110] pin 01 PCSA[0] (master) 10 PCSB[0] (master) 11 PCSD[0] (master)
20–21	SCKSELC [0:1]	DSPI C clock input select. Specifies the source of the DSPI C clock input when in slave mode. 00 PCSB[4]_SCKC_GPIO[109] pin 01 SCKA (master) 10 SCKB (master) 11 SCKD (master)
22–23	TRIGSELC [0:1]	DSPI C trigger input select. Specifies the source of the DSPI C trigger input for master or slave mode. 00 Invalid value 01 PCSA[4] 10 PCSB[4] 11 PCSD[4]
24–25	SINSELD [0:1]	DSPI D data input select. Specifies the source of the DSPI D data input. 00 PCSA[3]_GPIO[99] pin 01 SOUTA 10 SOUTB 11 SOUTC
26–27	SSSELD [0:1]	DSPI D slave select input select. Specifies the source of the DSPI D slave select input. 00 PCSB[1]_PCSD[0]_GPIO[106] pin 01 PCSA0 (master) 10 PCSB0 (master) 11 PCSC0 (master)
28–29	SCKSELD [0:1]	DSPI D clock input select. Specifies the source of the DSPI D clock input in slave mode. 00 PCSA[2]_SCKD_GPIO[98] pin 01 Invalid value 10 SCKB (master) 11 SCKC (master)
30–31	TRIGSELD [0:1]	DSPI D trigger input select. Specifies the source of the DSPI D trigger input for master or slave mode. 00 Invalid value 01 PCSA4 10 PCSB4 11 PCSC4

3.2.1.18 eQADC CFIFO Trigger Source Select - IMUX Select Registers (SIU_ISEL[4-7])

The IMUX select registers (SIU_ISEL[4 -7]) are used to select a trigger source for a command FIFO. SIU_ISEL select registers [4:5] are used to select a trigger source for command FIFOs in one eQADC. SIU_ISEL select registers [6:7] are used to select a trigger source for command FIFOs in a second eQADC. The cTSEL (combined Trigger Select) field is used to configure one of many possible trigger sources for each command FIFO.

To trigger the eQADC, the trigger source must change to the state that the input to the command FIFO has been programmed to recognize. A command FIFO trigger input can be programmed to recognize either rising or falling edges, and low or high gated trigger types.

SIU_ISEL4: eTRIG_A[5:2]

Address: SIU_BASE + 0x0910

Access: Read / write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	cTSEL5_0							0	cTSEL4_0						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	cTSEL3_0							0	cTSEL2_0						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SIU_ISEL5: eTRIG_A[1:0]

Address: SIU_BASE + 0x0914

Access: Read / write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	cTSEL1_0							0	cTSEL0_0						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SIU_ISEL6: eTRIG_B[5:2]

Address: SIU_BASE + 0x0918 Access: Read / write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	cTSEL5_1							0	cTSEL4_1						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	cTSEL3_1							0	cTSEL2_1						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

SIU_ISEL7: eTRIG_B[1:0]

Address: SIU_BASE + 0x091C Access: Read / write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	cTSEL1_1							0	cTSEL0_1						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3-27. eQADC_A Command FIFO 0 Trigger Sources

	cTSEL0_0							eQADC_A Trigger Inputs
eTSEL	0	0	0	0	0	0	0	Not Connected (default)
			0	0	0	0	1	RTI Trigger
			0	0	0	1	0	PIT0 Trigger
			0	0	0	1	1	PIT1 Trigger
			0	0	1	0	0	PIT2 Trigger
			0	0	1	0	1	PIT3 Trigger
			0	0	1	1	0	Reserved
			0	0	1	1	1	eTRIG1 pin
			0	1	0	x	x	Reserved
			0	1	1	0	0	eTPUA28
			0	1	1	0	1	eTPUA29
			0	1	1	1	0	eTPUA30
			0	1	1	1	1	eTPUA31
			1	0	0	0	0	eTPUB28
			1	0	0	0	1	eTPUB29
			1	0	0	1	0	eTPUB30
			1	0	0	1	1	eTPUB31
			1	0	1	0	0	eTPUC28
			1	0	1	0	1	eTPUC29
			1	0	1	1	0	eTPUC30
			1	0	1	1	1	eTPUC31
			1	1	0	0	0	eMIOS16
			1	1	0	0	1	eMIOS17
			1	1	0	1	0	eMIOS18
			1	1	0	1	1	eMIOS19
			1	1	1	0	0	eMIOS20
			1	1	1	0	1	eMIOS21
			1	1	1	1	0	eMIOS22
			1	1	1	1	1	eMIOS23
eTPU	0	1	x	x	x	x	x	eTPUA30
eMIOS	1	0	x	x	x	x	x	eMIOS10
eTRIG	1	1	x	x	x	x	x	ETRIG0 Pin

Table 3-28. eQADC_A Command FIFO 1 Trigger Sources

	cTSEL1_0							eQADC_A Trigger Inputs
eTSEL	0	0	0	0	0	0	0	Not Connected (default)
			0	0	0	0	1	RTI Trigger
			0	0	0	1	0	PIT0 Trigger
			0	0	0	1	1	PIT1 Trigger
			0	0	1	0	0	PIT2 Trigger
			0	0	1	0	1	PIT3 Trigger
			0	0	1	1	0	eTPUA7
			0	0	1	1	1	eTRIG0 pin
			0	1	0	x	x	Reserved
			0	1	1	0	0	eTPUA28
			0	1	1	0	1	eTPUA29
			0	1	1	1	0	eTPUA30
			0	1	1	1	1	eTPUA31
			1	0	0	0	0	eTPUB28
			1	0	0	0	1	eTPUB29
			1	0	0	1	0	eTPUB30
			1	0	0	1	1	eTPUB31
			1	0	1	0	0	eTPUC28
			1	0	1	0	1	eTPUC29
			1	0	1	1	0	eTPUC30
			1	0	1	1	1	eTPUC31
			1	1	0	0	0	eMIOS16
			1	1	0	0	1	eMIOS17
			1	1	0	1	0	eMIOS18
			1	1	0	1	1	eMIOS19
			1	1	1	0	0	eMIOS20
			1	1	1	0	1	eMIOS21
			1	1	1	1	0	eMIOS22
			1	1	1	1	1	eMIOS23
eTPU	0	1	x	x	x	x	x	eTPUA31
eMIOS	1	0	x	x	x	x	x	eMIOS11
eTRIG	1	1	x	x	x	x	x	ETRIG1 Pin

Table 3-29. eQADC_A Command FIFO 2 Trigger Sources

	cTSEL2_0							eQADC_A Trigger Inputs
eTSEL	0	0	0	0	0	0	0	Not Connected (default)
			0	0	0	0	1	RTI Trigger
			0	0	0	1	0	PIT0 Trigger
			0	0	0	1	1	PIT1 Trigger
			0	0	1	0	0	PIT2 Trigger
			0	0	1	0	1	PIT3 Trigger
			0	0	1	1	0	eTPUA14
			0	0	1	1	1	eTRIG1 pin
			0	1	0	x	x	Reserved
			0	1	1	0	0	eTPUA28
			0	1	1	0	1	eTPUA29
			0	1	1	1	0	eTPUA30
			0	1	1	1	1	eTPUA31
			1	0	0	0	0	eTPUB28
			1	0	0	0	1	eTPUB29
			1	0	0	1	0	eTPUB30
			1	0	0	1	1	eTPUB31
			1	0	1	0	0	eTPUC28
			1	0	1	0	1	eTPUC29
			1	0	1	1	0	eTPUC30
			1	0	1	1	1	eTPUC31
			1	1	0	0	0	eMIOS16
			1	1	0	0	1	eMIOS17
			1	1	0	1	0	eMIOS18
			1	1	0	1	1	eMIOS19
			1	1	1	0	0	eMIOS20
			1	1	1	0	1	eMIOS21
			1	1	1	1	0	eMIOS22
			1	1	1	1	1	eMIOS23
eTPU	0	1	x	x	x	x	x	eTPUA29
eMIOS	1	0	x	x	x	x	x	eMIOS15
eTRIG	1	1	x	x	x	x	x	ETRIG0 Pin

Table 3-30. eQADC_A Command FIFO 3 Trigger Sources

	cTSEL3_0							eQADC_A Trigger Inputs
eTSEL	0	0	0	0	0	0	0	Not Connected (default)
			0	0	0	0	1	RTI Trigger
			0	0	0	1	0	PIT0 Trigger
			0	0	0	1	1	PIT1 Trigger
			0	0	1	0	0	PIT2 Trigger
			0	0	1	0	1	PIT3 Trigger
			0	0	1	1	0	eTPUA22
			0	0	1	1	1	eTRIG0 pin
			0	1	0	x	x	Reserved
			0	1	1	0	0	eTPUA28
			0	1	1	0	1	eTPUA29
			0	1	1	1	0	eTPUA30
			0	1	1	1	1	eTPUA31
			1	0	0	0	0	eTPUB28
			1	0	0	0	1	eTPUB29
			1	0	0	1	0	eTPUB30
			1	0	0	1	1	eTPUB31
			1	0	1	0	0	eTPUC28
			1	0	1	0	1	eTPUC29
			1	0	1	1	0	eTPUC30
			1	0	1	1	1	eTPUC31
			1	1	0	0	0	eMIOS16
			1	1	0	0	1	eMIOS17
			1	1	0	1	0	eMIOS18
			1	1	0	1	1	eMIOS19
			1	1	1	0	0	eMIOS20
			1	1	1	0	1	eMIOS21
			1	1	1	1	0	eMIOS22
			1	1	1	1	1	eMIOS23
eTPU	0	1	x	x	x	x	x	eTPUA28
eMIOS	1	0	x	x	x	x	x	eMIOS14
eTRIG	1	1	x	x	x	x	x	ETRIG1 Pin

Table 3-31. eQADC_A Command FIFO 4 Trigger Sources

	cTSEL4_0							**eQADC_A Trigger Inputs**
eTSEL	0	0	0	0	0	0	0	Not Connected (default)
			0	0	0	0	1	RTI Trigger
			0	0	0	1	0	PIT0 Trigger
			0	0	0	1	1	PIT1 Trigger
			0	0	1	0	0	PIT2 Trigger
			0	0	1	0	1	PIT3 Trigger
			0	0	1	1	0	eTPUA30
			0	0	1	1	1	eTRIG1 pin
			0	1	0	x	x	Reserved
			0	1	1	0	0	eTPUA28
			0	1	1	0	1	eTPUA29
			0	1	1	1	0	eTPUA30
			0	1	1	1	1	eTPUA31
			1	0	0	0	0	eTPUB28
			1	0	0	0	1	eTPUB29
			1	0	0	1	0	eTPUB30
			1	0	0	1	1	eTPUB31
			1	0	1	0	0	eTPUC28
			1	0	1	0	1	eTPUC29
			1	0	1	1	0	eTPUC30
			1	0	1	1	1	eTPUC31
			1	1	0	0	0	eMIOS16
			1	1	0	0	1	eMIOS17
			1	1	0	1	0	eMIOS18
			1	1	0	1	1	eMIOS19
			1	1	1	0	0	eMIOS20
			1	1	1	0	1	eMIOS21
			1	1	1	1	0	eMIOS22
			1	1	1	1	1	eMIOS23
eTPU	0	1	x	x	x	x	x	eTPUA27
eMIOS	1	0	x	x	x	x	x	eMIOS13
eTRIG	1	1	x	x	x	x	x	ETRIG0 Pin

Table 3-32. eQADC_A Command FIFO 5 Trigger Sources

	cTSEL5_0							eQADC_A Trigger Inputs
eTSEL	0	0	0	0	0	0	0	Not Connected (default)
			0	0	0	0	1	RTI Trigger
			0	0	0	1	0	PIT0 Trigger
			0	0	0	1	1	PIT1 Trigger
			0	0	1	0	0	PIT2 Trigger
			0	0	1	0	1	PIT3 Trigger
			0	0	1	1	0	Reserved
			0	0	1	1	1	eTRIG0 pin
			0	1	0	x	x	Reserved
			0	1	1	0	0	eTPUA28
			0	1	1	0	1	eTPUA29
			0	1	1	1	0	eTPUA30
			0	1	1	1	1	eTPUA31
			1	0	0	0	0	eTPUB28
			1	0	0	0	1	eTPUB29
			1	0	0	1	0	eTPUB30
			1	0	0	1	1	eTPUB31
			1	0	1	0	0	eTPUC28
			1	0	1	0	1	eTPUC29
			1	0	1	1	0	eTPUC30
			1	0	1	1	1	eTPUC31
			1	1	0	0	0	eMIOS16
			1	1	0	0	1	eMIOS17
			1	1	0	1	0	eMIOS18
			1	1	0	1	1	eMIOS19
			1	1	1	0	0	eMIOS20
			1	1	1	0	1	eMIOS21
			1	1	1	1	0	eMIOS22
			1	1	1	1	1	eMIOS23
eTPU	0	1	x	x	x	x	x	eTPUA26
eMIOS	1	0	x	x	x	x	x	eMIOS12
eTRIG	1	1	x	x	x	x	x	ETRIG1 Pin

Table 3-33. eQADC_B Command FIFO 0 Trigger Sources

	cTSEL0_1							eQADC_B Trigger Inputs
eTSEL	0	0	0	0	0	0	0	Not Connected (default)
			0	0	0	0	1	RTI Trigger
			0	0	0	1	0	PIT0 Trigger
			0	0	0	1	1	PIT1 Trigger
			0	0	1	0	0	PIT2 Trigger
			0	0	1	0	1	PIT3 Trigger
			0	0	1	1	0	Reserved
			0	0	1	1	1	eTRIG1 pin
			0	1	0	x	x	Reserved
			0	1	1	0	0	eTPUA28
			0	1	1	0	1	eTPUA29
			0	1	1	1	0	eTPUA30
			0	1	1	1	1	eTPUA31
			1	0	0	0	0	eTPUB28
			1	0	0	0	1	eTPUB29
			1	0	0	1	0	eTPUB30
			1	0	0	1	1	eTPUB31
			1	0	1	0	0	eTPUC28
			1	0	1	0	1	eTPUC29
			1	0	1	1	0	eTPUC30
			1	0	1	1	1	eTPUC31
			1	1	0	0	0	eMIOS16
			1	1	0	0	1	eMIOS17
			1	1	0	1	0	eMIOS18
			1	1	0	1	1	eMIOS19
			1	1	1	0	0	eMIOS20
			1	1	1	0	1	eMIOS21
			1	1	1	1	0	eMIOS22
			1	1	1	1	1	eMIOS23
eTPU	0	1	x	x	x	x	x	eTPUA30
eMIOS	1	0	x	x	x	x	x	eMIOS10
eTRIG	1	1	x	x	x	x	x	ETRIG0 Pin

Table 3-34. eQADC_B Command FIFO 1 Trigger Sources

	cTSEL1_1							eQADC_B Trigger Inputs
eTSEL	0	0	0	0	0	0	0	Not Connected (default)
			0	0	0	0	1	RTI Trigger
			0	0	0	1	0	PIT0 Trigger
			0	0	0	1	1	PIT1 Trigger
			0	0	1	0	0	PIT2 Trigger
			0	0	1	0	1	PIT3 Trigger
			0	0	1	1	0	eTPUA7
			0	0	1	1	1	eTRIG0 pin
			0	1	0	x	x	Reserved
			0	1	1	0	0	eTPUA28
			0	1	1	0	1	eTPUA29
			0	1	1	1	0	eTPUA30
			0	1	1	1	1	eTPUA31
			1	0	0	0	0	eTPUB28
			1	0	0	0	1	eTPUB29
			1	0	0	1	0	eTPUB30
			1	0	0	1	1	eTPUB31
			1	0	1	0	0	eTPUC28
			1	0	1	0	1	eTPUC29
			1	0	1	1	0	eTPUC30
			1	0	1	1	1	eTPUC31
			1	1	0	0	0	eMIOS16
			1	1	0	0	1	eMIOS17
			1	1	0	1	0	eMIOS18
			1	1	0	1	1	eMIOS19
			1	1	1	0	0	eMIOS20
			1	1	1	0	1	eMIOS21
			1	1	1	1	0	eMIOS22
			1	1	1	1	1	eMIOS23
eTPU	0	1	x	x	x	x	x	eTPUA31
eMIOS	1	0	x	x	x	x	x	eMIOS11
eTRIG	1	1	x	x	x	x	x	ETRIG1 Pin

Table 3-35. eQADC_B Command FIFO 2 Trigger Sources

	cTSEL2_1							eQADC_B Trigger Inputs
eTSEL	0	0	0	0	0	0	0	Not Connected (default)
			0	0	0	0	1	RTI Trigger
			0	0	0	1	0	PIT0 Trigger
			0	0	0	1	1	PIT1 Trigger
			0	0	1	0	0	PIT2 Trigger
			0	0	1	0	1	PIT3 Trigger
			0	0	1	1	0	eTPUA14
			0	0	1	1	1	eTRIG1 pin
			0	1	0	x	x	Reserved
			0	1	1	0	0	eTPUA28
			0	1	1	0	1	eTPUA29
			0	1	1	1	0	eTPUA30
			0	1	1	1	1	eTPUA31
			1	0	0	0	0	eTPUB28
			1	0	0	0	1	eTPUB29
			1	0	0	1	0	eTPUB30
			1	0	0	1	1	eTPUB31
			1	0	1	0	0	eTPUC28
			1	0	1	0	1	eTPUC29
			1	0	1	1	0	eTPUC30
			1	0	1	1	1	eTPUC31
			1	1	0	0	0	eMIOS16
			1	1	0	0	1	eMIOS17
			1	1	0	1	0	eMIOS18
			1	1	0	1	1	eMIOS19
			1	1	1	0	0	eMIOS20
			1	1	1	0	1	eMIOS21
			1	1	1	1	0	eMIOS22
			1	1	1	1	1	eMIOS23
eTPU	0	1	x	x	x	x	x	eTPUA29
eMIOS	1	0	x	x	x	x	x	eMIOS15
eTRIG	1	1	x	x	x	x	x	ETRIG0 Pin

Table 3-36. eQADC_B Command FIFO 3 Trigger Sources

	cTSEL3_1							eQADC_B Trigger Inputs
eTSEL	0	0	0	0	0	0	0	Not Connected (default)
			0	0	0	0	1	RTI Trigger
			0	0	0	1	0	PIT0 Trigger
			0	0	0	1	1	PIT1 Trigger
			0	0	1	0	0	PIT2 Trigger
			0	0	1	0	1	PIT3 Trigger
			0	0	1	1	0	eTPUA22
			0	0	1	1	1	eTRIG0 pin
			0	1	0	x	x	Reserved
			0	1	1	0	0	eTPUA28
			0	1	1	0	1	eTPUA29
			0	1	1	1	0	eTPUA30
			0	1	1	1	1	eTPUA31
			1	0	0	0	0	eTPUB28
			1	0	0	0	1	eTPUB29
			1	0	0	1	0	eTPUB30
			1	0	0	1	1	eTPUB31
			1	0	1	0	0	eTPUC28
			1	0	1	0	1	eTPUC29
			1	0	1	1	0	eTPUC30
			1	0	1	1	1	eTPUC31
			1	1	0	0	0	eMIOS16
			1	1	0	0	1	eMIOS17
			1	1	0	1	0	eMIOS18
			1	1	0	1	1	eMIOS19
			1	1	1	0	0	eMIOS20
			1	1	1	0	1	eMIOS21
			1	1	1	1	0	eMIOS22
			1	1	1	1	1	eMIOS23
eTPU	0	1	x	x	x	x	x	eTPUA28
eMIOS	1	0	x	x	x	x	x	eMIOS14
eTRIG	1	1	x	x	x	x	x	ETRIG1 Pin

Table 3-37. eQADC_B Command FIFO 4 Trigger Sources

	cTSEL4_1							eQADC_B Trigger Inputs
eTSEL	0	0	0	0	0	0	0	Not Connected (default)
			0	0	0	0	1	RTI Trigger
			0	0	0	1	0	PIT0 Trigger
			0	0	0	1	1	PIT1 Trigger
			0	0	1	0	0	PIT2 Trigger
			0	0	1	0	1	PIT3 Trigger
			0	0	1	1	0	eTPUA30
			0	0	1	1	1	eTRIG1 pin
			0	1	0	x	x	Reserved
			0	1	1	0	0	eTPUA28
			0	1	1	0	1	eTPUA29
			0	1	1	1	0	eTPUA30
			0	1	1	1	1	eTPUA31
			1	0	0	0	0	eTPUB28
			1	0	0	0	1	eTPUB29
			1	0	0	1	0	eTPUB30
			1	0	0	1	1	eTPUB31
			1	0	1	0	0	eTPUC28
			1	0	1	0	1	eTPUC29
			1	0	1	1	0	eTPUC30
			1	0	1	1	1	eTPUC31
			1	1	0	0	0	eMIOS16
			1	1	0	0	1	eMIOS17
			1	1	0	1	0	eMIOS18
			1	1	0	1	1	eMIOS19
			1	1	1	0	0	eMIOS20
			1	1	1	0	1	eMIOS21
			1	1	1	1	0	eMIOS22
			1	1	1	1	1	eMIOS23
eTPU	0	1	x	x	x	x	x	eTPUA27
eMIOS	1	0	x	x	x	x	x	eMIOS13
eTRIG	1	1	x	x	x	x	x	ETRIG0 Pin

Table 3-38. eQADC_B Command FIFO 5 Trigger Sources

	cTSEL5_1							eQADC_B Trigger Inputs
eTSEL	0	0	0	0	0	0	0	Not Connected (default)
			0	0	0	0	1	RTI Trigger
			0	0	0	1	0	PIT0 Trigger
			0	0	0	1	1	PIT1 Trigger
			0	0	1	0	0	PIT2 Trigger
			0	0	1	0	1	PIT3 Trigger
			0	0	1	1	0	Reserved
			0	0	1	1	1	eTRIG0 pin
			0	1	0	x	x	Reserved
			0	1	1	0	0	eTPUA28
			0	1	1	0	1	eTPUA29
			0	1	1	1	0	eTPUA30
			0	1	1	1	1	eTPUA31
			1	0	0	0	0	eTPUB28
			1	0	0	0	1	eTPUB29
			1	0	0	1	0	eTPUB30
			1	0	0	1	1	eTPUB31
			1	0	1	0	0	eTPUC28
			1	0	1	0	1	eTPUC29
			1	0	1	1	0	eTPUC30
			1	0	1	1	1	eTPUC31
			1	1	0	0	0	eMIOS16
			1	1	0	0	1	eMIOS17
			1	1	0	1	0	eMIOS18
			1	1	0	1	1	eMIOS19
			1	1	1	0	0	eMIOS20
			1	1	1	0	1	eMIOS21
			1	1	1	1	0	eMIOS22
			1	1	1	1	1	eMIOS23
eTPU	0	1	x	x	x	x	x	eTPUA26
eMIOS	1	0	x	x	x	x	x	eMIOS12
eTRIG	1	1	x	x	x	x	x	ETRIG1 Pin

3.2.1.19 eTPUA Input Select Register (SIU_ISEL8)

The SIU_ISEL 8 register is used to multiplex the eTPUA[24:29] inputs. These 6 eTPUA channels can come from the output of DSPI_B or corresponding pad. When SIU_ISEL8 is in its default state, the eTPU pins listed in Figure 3-19 will not be enabled as input, irrespective of the SIU_PCR[PA] field (PCR138-143).

Address: SIU_BASE + 0x0920 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	eTPU A29	0	0	0	eTPU A28
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	eTPU A27	0	0	0	eTPU A26	0	0	0	eTPU A25	0	0	0	eTPU A24
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-19. eTPUA Input Select Register (SIU_ISEL8)

Table 3-39. SIU_ISEL8 Bit Field Descriptions

Bits	Name	Description
11	eTPUA29	eTPUA29 input select. Specifies the source of the eTPUA29 channel input. 0 DSPI_B Serialized input 8 1 eTPUA29 channel input pad
15	eTPUA28	eTPUA28 input select. Specifies the source of the eTPUA28 channel input. 0 DSPI_B Serialized input 9 1 eTPUA28 channel input pad
19	eTPUA27	eTPUA27 input select. Specifies the source of the eTPUA27 channel input. 0 DSPI_B Serialized input 10 1 eTPUA27 channel input pad
23	eTPUA26	eTPUA26 input select. Specifies the source of the eTPUA26 channel input. 0 DSPI_B Serialized input 11 1 eTPUA26 channel input pad
27	eTPUA25	eTPUA25 input select. Specifies the source of the eTPUA25 channel input. 0 DSPI_B Serialized input 12 1 eTPUA25 channel input pad
31	eTPUA24	eTPUA24 input select. Specifies the source of the eTPUA24 channel input. 0 DSPI_B Serialized input 13 1 eTPUA24 channel input pad

3.2.1.20 eQADC Advance Trigger Selection (SIU_ISEL9)

The eQADC’s streaming mode requires a second trigger for Queue 0. The source for this trigger can come from ETPU, EMIOS or PIT channels. This mux select register selects the source of the Queue 0 trigger.

Address: SIU_BASE + 0x0924 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	eTSEL0ADV_A				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	eTSEL0ADV_B				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-20. eQADC Advance Trigger Selection Register (SIU_ISEL9)

Table 3-40. eTSEL0ADV_A Bit Field Descriptions

eTSEL0A					eQADC Trigger Input
0	0	0	0	0	Reserved
0	0	0	0	1	RTI Trigger
0	0	0	1	0	PIT0 Trigger
0	0	0	1	1	PIT1 Trigger
0	0	1	0	0	PIT2 Trigger
0	0	1	0	1	PIT3 Trigger
0	0	1	1	0	Reserved
0	0	1	1	1	Reserved
0	1	0	0	0	eTPUA30 AND PIT0
0	1	0	0	1	eTPUA30 AND PIT1
0	1	0	1	0	Reserved
0	1	0	1	1	Reserved
0	1	1	0	0	eTPUA28
0	1	1	0	1	eTPUA29
0	1	1	1	0	eTPUA30
0	1	1	1	1	eTPUA31
1	0	0	0	0	eTPUC28
1	0	0	0	1	eTPUC29
1	0	0	1	0	eTPUC30
1	0	0	1	1	eTPUC31
1	0	1	0	0	eMIOS10 AND PIT2

Table 3-40. eTSEL0ADV_A Bit Field Descriptions

eTSEL0A					eQADC Trigger Input
1	0	1	0	1	eMIOS10 AND PIT3
1	0	1	1	0	Reserved
1	0	1	1	1	Reserved
1	1	0	0	0	Reserved
1	1	0	0	1	Reserved
1	1	0	1	0	Reserved
1	1	0	1	1	Reserved
1	1	1	0	0	Reserved
1	1	1	0	1	Reserved
1	1	1	1	0	Reserved
1	1	1	1	1	eMIOS23

Table 3-41. eTSEL0ADV_B Bit Field Descriptions

eTSEL0A					eQADC Trigger Input
0	0	0	0	0	Reserved
0	0	0	0	1	RTI Trigger
0	0	0	1	0	PIT0 Trigger
0	0	0	1	1	PIT1 Trigger
0	0	1	0	0	PIT2 Trigger
0	0	1	0	1	PIT3 Trigger
0	0	1	1	0	Reserved
0	0	1	1	1	Reserved
0	1	0	0	0	eTPUA30 AND PIT0
0	1	0	0	1	eTPUA30 AND PIT1
0	1	0	1	0	Reserved
0	1	0	1	1	Reserved
0	1	1	0	0	eTPUA28
0	1	1	0	1	eTPUA29
0	1	1	1	0	eTPUA30
0	1	1	1	1	eTPUA31
1	0	0	0	0	eTPUC28
1	0	0	0	1	eTPUC29
1	0	0	1	0	eTPUC30
1	0	0	1	1	eTPUC31
1	0	1	0	0	eMIOS10 AND PIT2
1	0	1	0	1	eMIOS10 AND PIT3
1	0	1	1	0	Reserved
1	0	1	1	1	Reserved
1	1	0	0	0	Reserved

Table 3-41. eTSEL0ADV_B Bit Field Descriptions

eTSEL0A					eQADC Trigger Input
1	1	0	0	1	Reserved
1	1	0	1	0	Reserved
1	1	0	1	1	Reserved
1	1	1	0	0	Reserved
1	1	1	0	1	Reserved
1	1	1	1	0	Reserved
1	1	1	1	1	eMIOS23

3.2.1.21 Decimation Filter Register 1 (SIU_DECFIL1)

This register selects the source of the control signals for the integrators of decimation filters A to D. Each ZSELx field of this register may be programmed to route a single eTPU channel simultaneously to the Zero, Integrate and Read control inputs of one integrator. Each HSELx field may be programmed to route a single eTPU channel to the Halt control input of one integrator. Refer to Table 3-42 for details of the source selection codes. See Chapter 14, “Decimation Filter,” for details on the Zero, Integrate, Read and Halt control configuration and functionality.

Address: SIU_BASE + 0x0928 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	HSELD				ZSELD				HSELC				ZSELC			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	HSELB				ZSELB				HSELA				ZSELA			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-21. Decimation Filter Register 1 (SIU_DECFIL1)

Table 3-42. SIU_DECFIL1 Bit Field Descriptions

Bits	Name	Description
0–3	HSELD	Halt Input Select for Decimation Filter D 0000 Unused 0001 eTPUB Channel 22 0010 eTPUB Channel 23 0011 eTPUB Channel 24 0100 eTPUB Channel 25 0101 eTPUC Channel 22 0110 eTPUC Channel 23 0111 eTPUC Channel 24 1000eTPUC Channel 25
4–7	ZSELD	ZIR Input Select for Decimation Filter D 0000 Unused 0001 eTPUB Channel 22 0010 eTPUB Channel 23 0011 eTPUB Channel 24 0100 eTPUB Channel 25 0101 eTPUC Channel 22 0110 eTPUC Channel 23 0111 eTPUC Channel 24 1000eTPUC Channel 25
8–11	HSELC	Halt Input Select for Decimation Filter C 0000 Unused 0001 eTPUB Channel 22 0010 eTPUB Channel 23 0011 eTPUB Channel 24 0100 eTPUB Channel 25 0101 eTPUC Channel 22 0110 eTPUC Channel 23 0111 eTPUC Channel 24 1000 eTPUC Channel 25
12–15	ZSELC	ZIR Input Select for Decimation Filter C 0000 Unused 0001 eTPUB Channel 22 0010 eTPUB Channel 23 0011 eTPUB Channel 24 0100 eTPUB Channel 25 0101 eTPUC Channel 22 0110 eTPUC Channel 23 0111 eTPUC Channel 24 1000 eTPUC Channel 25
16–19	HSELB	Halt Input Select for Decimation Filter B 0000 Unused 0001 eTPUA Channel 22 0010 eTPUA Channel 23 0011 eTPUA Channel 24 0100 eTPUA Channel 25 0101 eTPUC Channel 22 0110 eTPUC Channel 23 0111 eTPUC Channel 24 1000 eTPUC Channel 25

Table 3-42. SIU_DECFIL1 Bit Field Descriptions

Bits	Name	Description
20–23	ZSELB	ZIR Input Select for Decimation Filter B 0000 Unused 0001 eTPUA Channel 22 0010 eTPUA Channel 23 0011 eTPUA Channel 24 0100 eTPUA Channel 25 0101 eTPUC Channel 22 0110 eTPUC Channel 23 0111 eTPUC Channel 24 1000 eTPUC Channel 25
24–27	HSELA	Halt Input Select for Decimation Filter A 0000 Unused 0001 eTPUA Channel 22 0010 eTPUA Channel 23 0011 eTPUA Channel 24 0100 eTPUA Channel 25 0101 eTPUC Channel 22 0110 eTPUC Channel 23 0111 eTPUC Channel 24 1000 eTPUC Channel 25
28–31	ZSELA	ZIR Input Select for Decimation Filter A 0000 Unused 0001 eTPUA Channel 22 0010 eTPUA Channel 23 0011 eTPUA Channel 24 0100 eTPUA Channel 25 0101 eTPUC Channel 22 0110 eTPUC Channel 23 0111 eTPUC Channel 24 1000 eTPUC Channel 25

3.2.1.22 Decimation Filter Register 2 (SIU_DECFIL2)

This register selects the source of the control signals for the integrators of decimation filters E to H. Each ZSELx field of this register may be programmed to route a single eTPU channel simultaneously to the Zero, Integrate and Read control inputs of one integrator. Each HSELx field may be programmed to route a single eTPU channel to the Halt control input of one integrator. Refer to Table 3-43 for details of the source selection codes. See Chapter 14, "Decimation Filter," for details on the Zero, Integrate, Read and Halt control configuration and functionality.

Address: SIU_BASE + 0x092C Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	HSELH				ZSELH				HSELG				ZSELG			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	HSELF				ZSELF				HSELE				ZSELE			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-22. Decimation Filter Register 2 (SIU_DECFIL2)

Table 3-43. SIU_DECFIL2 Bit Field Descriptions

Bits	Name	Description
0–3	HSELH	Halt Input Select for Decimation Filter H 0000 Unused 0001 eTPUB Channel 18 0010 eTPUB Channel 19 0011 eTPUB Channel 20 0100 eTPUB Channel 21 0101 eTPUC Channel 18 0110 eTPUC Channel 19 0111 eTPUC Channel 20 1000eTPUC Channel 21
4–7	ZSELH	ZIR Input Select for Decimation Filter H 0000 Unused 0001 eTPUB Channel 18 0010 eTPUB Channel 19 0011 eTPUB Channel 20 0100 eTPUB Channel 21 0101 eTPUC Channel 18 0110 eTPUC Channel 19 0111 eTPUC Channel 20 1000 eTPUC Channel 21

Table 3-43. SIU_DECFIL2 Bit Field Descriptions

Bits	Name	Description
8–11	HSELG	Halt Input Select for Decimation Filter G 0000 Unused 0001 eTPUB Channel 18 0010 eTPUB Channel 19 0011 eTPUB Channel 20 0100 eTPUB Channel 21 0101 eTPUC Channel 18 0110 eTPUC Channel 19 0111 eTPUC Channel 20 1000eTPUC Channel 21
12–15	ZSELG	ZIR Input Select for Decimation Filter G 0000 Unused 0001 eTPUB Channel 18 0010 eTPUB Channel 19 0011 eTPUB Channel 20 0100 eTPUB Channel 21 0101 eTPUC Channel 18 0110 eTPUC Channel 19 0111 eTPUC Channel 20 1000eTPUC Channel 21
16–19	HSELF	Halt Input Select for Decimation Filter F 0000 Unused 0001 eTPUA Channel 18 0010 eTPUA Channel 19 0011 eTPUA Channel 20 0100 eTPUA Channel 21 0101 eTPUC Channel 18 0110 eTPUC Channel 19 0111 eTPUC Channel 20 1000 eTPUC Channel 21
20–23	ZSELF	ZIR Input Select for Decimation Filter F 0000 Unused 0001 eTPUA Channel 18 0010 eTPUA Channel 19 0011 eTPUA Channel 20 0100 eTPUA Channel 21 0101 eTPUC Channel 18 0110 eTPUC Channel 19 0111 eTPUC Channel 20 1000 eTPUC Channel 21

Table 3-43. SIU_DECFIL2 Bit Field Descriptions

Bits	Name	Description
24–27	HSELE	Halt Input Select for Decimation Filter E 0000 Unused 0001 eTPUA Channel 18 0010 eTPUA Channel 19 0011 eTPUA Channel 20 0100 eTPUA Channel 21 0101 eTPUC Channel 18 0110 eTPUC Channel 19 0111 eTPUC Channel 20 1000 eTPUC Channel 21
28–31	ZSELE	ZIR Input Select for Decimation Filter E 0000 Unused 0001 eTPUA Channel 18 0010 eTPUA Channel 19 0011 eTPUA Channel 20 0100 eTPUA Channel 21 0101 eTPUC Channel 18 0110 eTPUC Channel 19 0111 eTPUC Channel 20 1000 eTPUC Channel 21

3.2.1.23 Decimation Filter Register 3 (SIU_DECFIL3)

This register selects the source of the control signals for the integrators of decimation filters I to L. Each ZSELx field of this register may be programmed to route a single eTPU channel simultaneously to the Zero, Integrate and Read control inputs of one integrator. Each HSELx field may be programmed to route a single eTPU channel to the Halt control input of one integrator. Refer to Table 3-44 for details of the source selection codes. See Chapter 14, “Decimation Filter,” for details on the Zero, Integrate, Read and Halt control configuration and functionality.

Address: SIU_BASE + 0x092C

Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	HSELL				ZSELL				HSELK				ZSELK			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	HSELJ				ZSELJ				HSELI				ZSELI			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3-44. SIU_DECFIL3 Bit Field Descriptions

Bits	Name	Description
0–3	HSELL	Halt Input Select for Decimation Filter L 0000 Unused 0001 eTPUB Channel 14 0010 eTPUB Channel 15 0011 eTPUB Channel 16 0100 eTPUB Channel 17 0101 eTPUC Channel 14 0110 eTPUC Channel 15 0111 eTPUC Channel 16 1000eTPUC Channel 17
4–7	ZSELL	ZIR Input Select for Decimation Filter L 0000 Unused 0001 eTPUB Channel 14 0010 eTPUB Channel 15 0011 eTPUB Channel 16 0100 eTPUB Channel 17 0101 eTPUC Channel 14 0110 eTPUC Channel 15 0111 eTPUC Channel 16 1000eTPUC Channel 17
8–11	HSELK	Halt Input Select for Decimation Filter K 0000 Unused 0001 eTPUB Channel 14 0010 eTPUB Channel 15 0011 eTPUB Channel 16 0100 eTPUB Channel 17 0101 eTPUC Channel 14 0110 eTPUC Channel 15 0111 eTPUC Channel 16 1000eTPUC Channel 17
12–15	ZSELK	ZIR Input Select for Decimation Filter K 0000 Unused 0001 eTPUB Channel 14 0010 eTPUB Channel 15 0011 eTPUB Channel 16 0100 eTPUB Channel 17 0101 eTPUC Channel 14 0110 eTPUC Channel 15 0111 eTPUC Channel 16 1000 eTPUC Channel 17
16–19	HSELJ	Halt Input Select for Decimation Filter J 0000 Unused 0001 eTPUA Channel 14 0010 eTPUA Channel 15 0011 eTPUA Channel 16 0100 eTPUA Channel 17 0101 eTPUC Channel 14 0110 eTPUC Channel 15 0111 eTPUC Channel 16 1000eTPUC Channel 17

Table 3-44. SIU_DECFIL3 Bit Field Descriptions

Bits	Name	Description
20–23	ZSELJ	ZIR Input Select for Decimation Filter J 0000 Unused 0001 eTPUA Channel 14 0010 eTPUA Channel 15 0011 eTPUA Channel 16 0100 eTPUA Channel 17 0101 eTPUC Channel 14 0110 eTPUC Channel 15 0111 eTPUC Channel 16 1000 eTPUC Channel 17
24–27	HSELI	Halt Input Select for Decimation Filter I 0000 Unused 0001 eTPUA Channel 14 0010 eTPUA Channel 15 0011 eTPUA Channel 16 0100 eTPUA Channel 17 0101 eTPUC Channel 14 0110 eTPUC Channel 15 0111 eTPUC Channel 16 1000 eTPUC Channel 17
28–31	ZSELI	ZIR Input Select for Decimation Filter I 0000 Unused 0001 eTPUA Channel 14 0010 eTPUA Channel 15 0011 eTPUA Channel 16 0100 eTPUA Channel 17 0101 eTPUC Channel 14 0110 eTPUC Channel 15 0111 eTPUC Channel 16 1000 eTPUC Channel 17

3.2.1.24 Decimation Filter Register 4, 5 (SIU_DECFIL4, SIU_DECFIL5)

The SIU_DECFIL4 register contains bit fields that specify which eTPU output is used to trigger the decimation filter result output buffer for decimation filters A to H. The register is shown in Figure 3-23. The encodings for the bit fields are shown in Table 3-45. The SIU_DECFIL5 register contains the triggered output source selections for decimation filters I to L, and is defined in Figure 3-24 and Table 3-46.

Address: SIU_BASE + 0x0934 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TRIG_SRCH				TRIG_SRCG				TRIG_SRCF				TRIG_SRCE			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TRIG_SRCD				TRIG_SRCC				TRIG_SRCB				TRIG_SRCA			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-23. Decimation Filter A - H Triggered Output Source Select Register (SIU_DECFIL4)

Table 3-45. Decimation Filter A - H Triggered Output Source Selection

Field Name	Code[1]				Source
TRIG_SRCH	0	0	0	0	Not Connected (default)
	0	0	0	1	eTPUA22
	0	0	1	0	eTPUA23
	0	0	1	1	eTPUA24
	0	1	0	0	eTPUA25
TRIG_SRCG	0	0	0	0	Not Connected (default)
	0	0	0	1	eTPUA22
	0	0	1	0	eTPUA23
	0	0	1	1	eTPUA24
	0	1	0	0	eTPUA25
TRIG_SRCF	0	0	0	0	Not Connected (default)
	0	0	0	1	eTPUA22
	0	0	1	0	eTPUA23
	0	0	1	1	eTPUA24
	0	1	0	0	eTPUA25
TRIG_SRCE	0	0	0	0	Not Connected (default)
	0	0	0	1	eTPUA22
	0	0	1	0	eTPUA23
	0	0	1	1	eTPUA24
	0	1	0	0	eTPUA25
TRIG_SRCD	0	0	0	0	Not Connected (default)
	0	0	0	1	eTPUA22
	0	0	1	0	eTPUA23
	0	0	1	1	eTPUA24
	0	1	0	0	eTPUA25
TRIG_SRCC	0	0	0	0	Not Connected (default)
	0	0	0	1	eTPUA22
	0	0	1	0	eTPUA23
	0	0	1	1	eTPUA24
	0	1	0	0	eTPUA25
TRIG_SRCB	0	0	0	0	Not Connected (default)
	0	0	0	1	eTPUA22
	0	0	1	0	eTPUA23
	0	0	1	1	eTPUA24
	0	1	0	0	eTPUA25
TRIG_SRCA	0	0	0	0	Not Connected (default)
	0	0	0	1	eTPUA22
	0	0	1	0	eTPUA23
	0	0	1	1	eTPUA24
	0	1	0	0	eTPUA25

[1] Selecting a code from 0b0101 to 0b1111 results in the same behavior as selecting 0b0000.

Address: SIU_BASE + 0x0938 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TRIG_SRCL				TRIG_SRCK				TRIG_SRCJ				TRIG_SRCI			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-24. Decimation Filter I - LTriggered Output Source Select Register (SIU_DECFIL5)

Table 3-46. Decimation Filter I - L Triggered Output Source Selection

Field Name	Code[1]				Source
TRIG_SRCL	0	0	0	0	Not Connected (default)
	0	0	0	1	eTPUA22
	0	0	1	0	eTPUA23
	0	0	1	1	eTPUA24
	0	1	0	0	eTPUA25
TRIG_SRCK	0	0	0	0	Not Connected (default)
	0	0	0	1	eTPUA22
	0	0	1	0	eTPUA23
	0	0	1	1	eTPUA24
	0	1	0	0	eTPUA25
TRIG_SRCJ	0	0	0	0	Not Connected (default)
	0	0	0	1	eTPUA22
	0	0	1	0	eTPUA23
	0	0	1	1	eTPUA24
	0	1	0	0	eTPUA25
TRIG_SRCI	0	0	0	0	Not Connected (default)
	0	0	0	1	eTPUA22
	0	0	1	0	eTPUA23
	0	0	1	1	eTPUA24
	0	1	0	0	eTPUA25

[1] Selecting a code from 0b0101 to 0b1111 results in the same behavior as selecting 0b0000.

3.2.1.25 Chip Configuration Register (SIU_CCR)

The SIU_CCR controls the chip configuration for enabling and disabling Nexus on the external bus interface.

Address: SIU_BASE + 0x0980

Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MATCH	DISNEX
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X[1]

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TEST
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] When system $\overline{\text{RESET}}$ negates, the value in this bit depends on the censorship control word and the boot configuration bits.

Figure 3-25. Chip Configuration Register (SIU_CCR)

The following table describes the chip configuration fields.

Table 3-47. SIU_CCR Bit Field Descriptions

Bits	Name	Description
0–13	—	Reserved
14	MATCH	Compare register match. Holds the value of the match input signal to the SIU. The match input is asserted if the values in SIU_CBRH and SIU_CBRL are the same as the public password stored in flash, 0xFEED_FACE_CAFE_BEEF. The MATCH bit is reset by the internal reset condition. 0 Match input signal is negated 1 Match input signal is asserted
15	DISNEX	Disable Nexus. Holds the value of the Nexus disable input signal to the SIU. When system reset negates, the value in this bit depends on the censorship control word and the boot configuration bits. 0 Nexus disable input signal is negated. 1 Nexus disable input signal is asserted.
16–30	—	Reserved
31	TEST	Test mode enable. Allows reads or writes to undocumented registers used only for production tests. Since these production test registers are undocumented, estimating the impact of errant accesses to them is impossible. Do not change this bit from its negated state at reset. 0 Undocumented production test registers cannot be read or written. 1 Undocumented production test registers can be read or written.

Figure 3-26. Chip Configuration Register (SIU_CCR)

3.2.1.26 External Clock Control Register (SIU_ECCR)

The SIU_ECCR controls the timing relationship between the system clock and the external clocks ENGCLK and D_CLKOUT[1]. All bits and fields in the SIU_ECCR are read/write and are reset by the internal reset condition.

Address: SIU_BASE + 0x0984 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ENGDIV								ECSS	0	0	0	EBTS	0	EBDF	
W																
Reset	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1

Figure 3-27. External Clock Control Register (SIU_ECCR)

The following table describes the external clock control fields:

Table 3-48. SIU_ECCR Bit Field Descriptions

Bits	Name	Description
0–15	—	Reserved
16–23	ENGDIV [0:7]	Engineering clock division factor. Specifies the frequency ratio between f_{periph} (also referred to as f_{platf} on this device) and ENGCLK. The ENGCLK frequency is divided from f_{platf} according to the following equation: $$\text{Engineering clock frequency} = \frac{f_{periph}}{\text{ENGDIV} \times 2}$$ Setting ENGDIV to 0 makes the ENGCLK frequency equal to the f_{periph} clock frequency. **Note:** Maximum ENGCLK frequency is limited by the pad performance. Refer to the Electrical Specifications in the Data Sheet document for this device. Synchronization between ENGCLK and D_CLKOUT cannot be guaranteed when ENGDIV is 0.
24	ECSS	Engineering clock (ENGCLK) source select. 0 The system clock is the source of the ENGCLK. 1 The external clock (the EXTAL frequency of the oscillator) is the source of the ENGCLK.
25–27	—	Reserved

---

[1] D_CLKOUT is not available in all packages.

Table 3-48. SIU_ECCR Bit Field Descriptions

Bits	Name	Description
28	EBTS	External bus tap select. Changes the phase relationship between the system clock and D_CLKOUT. Changing the phase relationship so that D_CLKOUT is advanced in relation to the system clock increases the output hold time of the external bus signals to a non-zero value. It also increases the output delay times, increases the input hold times to non-zero values, and decreases the input setup times. Refer to the *Electrical Specifications* for how the EBTS bit affects the external bus timing. 0 External bus signals have zero output hold times. 1 External bus signals have non-zero output hold times. **Note:** Do not change EBTS while an external bus transaction is in process.
29	—	Reserved
30–31	EBDF [0:1]	External bus division factor. Specifies the frequency ratio between the system clock and the external clock, D_CLKOUT. Do not change EBDF during an external bus access or while an access is pending. The D_CLKOUT frequency is divided from the system clock frequency according to the descriptions below. When operating in full mode (1:1), set the divider to 0b01 (divide-by-2). 00 Divide by 1 01 Divide by 2 10 Divide by 3 11 Divide by 4

3.2.1.27 Compare B Register High (SIU_CBRH)

The SIU_CBRH holds the 32-bit value that is compared against the public password in flash (0xFEED_FACE_CAFE_BEEF). The CMPBH field is read/write and is reset by the internal reset condition.

Address: SIU_BASE + 0x0990 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CMPBH															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CMPBH															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-28. Compare B Register High (SIU_CBRH)

3.2.1.28 Compare B Register Low (SIU_CBRL)

The SIU_CBRL holds the 32-bit value that is compared against the public password in flash (0xFEED_FACE_CAFE_BEEF). The CMPBL field is read/write and is reset by the internal reset condition.

Address: SIU_BASE + 0x0994 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CMPBL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CMPBL															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-29. Compare B Register Low (SIU_CBRL)

3.2.1.29 System Clock Register (SIU_SYSDIV)

The SIU_SYSDIV field is read/write and is reset by the internal reset condition.

SIU_BASE+0x9A0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LCK															
W		▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R			SYS-CLKSEL				IPCLKDIV					BY-PASS	SYS-CLKDIV			
W	▨	▨			▨	▨			▨	▨	▨				▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

▨ = Unimplemented or Reserved

Figure 3-30. System Clock Register (SIU_SYSDIV)

Table 3-49. System Clock Register Field Descriptions

Bit/Field	Description
0	SIU_SYSDIV write lock bit. When the LCK is written with a value of 1, writes to all other bits and fields in the SIU_SYSDIV register are disabled. The LCK bit can be set/cleared by software at any time. 0 = SIU_SYSDIV register is unlocked and writes are permitted. 1 = SIU_SYSIDIV register is locked, and only writes to the LCK bit are permitted.
1-17	Reserved
18-19	System Clock Select. The SYSCLKSEL bits select the source of the system clock. 00 The system clock is driven by the 16MHz Internal RC oscillator (IRC). 01 The system clock is driven by the external oscillator (XOSC). 10 The system clock is driven by the output clock from the PLL. 11 Reserved
20-21	Reserved
22-23	IP Clock Divider. The IPCLKDIV bits select the divider value for the Platform, peripheral and eTPU clocks in respect to the core clock. 00 CPU frequency is doubled (Max 200Mhz). Platform, peripheral, and eTPU clocks are 1/2 of CPU frequency 01 CPU and eTPU frequency is doubled (Max 200Mhz). Platform and peripheral clocks are 1/2 of CPU frequency 10 Reserved 11 Reserved
24-26	Reserved
27	Bypass bit. 0 = system clock divider is not bypassed 1 = system clock divider is bypassed
28-29	System Clock Divider. The SYSCLKDIV bits select the divider value for the system clock (m_clk). Note that the SYSCLKDIV divider is required in addition to the RFD to allow the other source for the system clock (OSC) to be divided down to slowest frequencies to improve power. The output of the clock divider is nominally a 50% duty cycle. 00 = Divide by 2 01 = Divide by 4 10 = Divide by 8 11 = Divide by 16
30-31	Reserved

Figure 3-31. System Clock Register (SIU_SYSDIV)

3.2.1.30 Halt Register (SIU_HLT)

The SIU_HLT register is used to disable the clocks to various modules. Each bit drives a separate stop output of the SIU. These outputs are connected as shown in Table 3-50.

NOTE

Some peripherals have an MDIS (module disable) bit in the module control register that can be set to disable the module clock, reducing power consumption. In most cases the peripheral registers are still readable and writeable. However, using the SIU_HALT register also disables the read/write functions on the disabled peripheral's registers for additional power saving.

See Section 29.5.2.2, "Low Power Mode With RTI Wakeup" for more information on how to use the SIU_HLT and SIU_HLTACK registers.

SIU_BASE + 0x9A4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	HLT															
RESET	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	HLT															
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-32. Halt Register (SIU_HLT)

Table 3-50. SIU_HLT Register Field Descriptions

Field	Description
0-31 HLT	Halt Selects The HLT bits halt specific modules. Each bit corresponds to a separate module as mapped below: 0 Core 0[1] 1 Core 1[1] 2 rsvd 3 rsvd 4 eTPU_C 5 eTPU_A, eTPU_B 6 NPC 7 EBI 8 eQADCs: eQADC_A and eQADC_B 9 rsvd 10 eMIOS_A 11 DECFILT (decimation filters) 12 rsvd 13 PIT 14 rsvd 15 rsvd 16FlexCAN_D 17 FlexCAN_C 18 FlexCAN_B 19 FlexCAN_A 20 DSPI_D 21 DSPI_C 22 DSPI_B 23 DSPI_A 24 DSPI_E 25 rsvd 26 rsvd 27 rsvd 28 rsvd 29 eSCI_C 30 eSCI_B 31 eSCI_A

---

[1] Stops the core clocks but only after the core executes a WAIT instruction. The interrupt controller and SWT0 and SWT1 clocks are never stopped.

3.2.1.31 Halt Acknowledge Register (SIU_HLTACK)

The SIU_HLTACK bits indicate that the module requested to halt via the HLT bit has completed the halt process and has entered a halted state with the module clocks disabled. The HLTACK bits are read-only and writes have no effect, it is reset by the internal reset condition. The input signals from each module will be connected as shown in Table 3-51., "HALT Acknowledge Register Field Descriptions.

SIU_BASE + 0x9A8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	HLTACK															
W																
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	HLTACK															
W																
RESET	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-33. Halt Acknowledge Register (SIU_HLTACK)

Table 3-51. HALT Acknowledge Register Field Descriptions

Field	Description
HLTACK	Halt Acknowledge The HLTACK bits acknowledge halt for specific modules. Each bit corresponds to a separate module as mapped below: 0 Core 0 1 Core 1 2 rsvd 3 rsvd 4 eTPU_C 5 eTPU_A, eTPU_B 6 NPC 7 EBI 8 eQADCs: eQADC_A and eQADC_B 9 rsvd 10 eMIOS_A 11 DECFILT (decimation filters) 12 rsvd 13 PIT 14 rsvd 15 rsvd 16 FlexCAN_D 17 FlexCAN_C 18FlexCAN_B 19 FlexCAN_A 20 DSPI_D 21 DSPI_C 22 DSPI_B 23 DSPI_A 24 DSPI_E 25 rsvd 26 rsvd 27 rsvd 28 rsvd 29 eSCI_C 30 eSCI_B 31 eSCI_A

3.2.1.32 Core 0 and Core 1 Reset Vectors

Figure 3-34, SIU_RSTVEC0, shows the reset vector, reset control and instruction execution type selection for Core 0. Figure 3-35, SIU_RSTVEC1, show the equivalent details for core 1. The RST bit in each register is qualified by the corresponding core halt bit in the SIU halt register, such that both must be asserted for a core reset to occur. The order in which the RST and core halt bits is asserted does not matter. It is recommended that the RST bit is set immediately after the corresponding halt bit is set, to minimize the opportunity for a “single upset event” to prematurely reset a core.

Note that after any reset, Core 0’s halt and RST bits are negated by default to allow Core 0 to execute, whilst Core 1’s halt and RST bits remain asserted by default to prevent Core 1 executing.

After any core’s halt and RST bits are asserted, internal logic places a request to halt that core’s clocks, but the core may continue to run until such time that it terminates any pending crossbar master bus cycles and interrupt controller interaction that might prevent their use by other bus masters.

Any attempt to reset both cores by setting both RST bits will result in an immediate system reset, with the cause of the reset recorded in the SIU_RSR[CPURS] bit.

Note that the core halt bits in the SIU Halt register have additional functionality not related to the RST bits in these reset vector registers.

Offset: SIU_A_BASE + 0x09AC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	RSTVEC																														RST	VLE
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0

Figure 3-34. Reset Vector Register for Core 0 (SIU_RSTVEC0)

Table 3-52. SIU_RSTVEC0 Field Descriptions

Field	Description
0-29 RSTVEC	Core 0 Reset Vector. The RSTVEC value determines the initial program counter for core 0 upon exiting reset. On POR, the value contained in the register defaults to 0xFFFF_FFFC, so that core 0 fetches BookE code from the BAM starting at address 0xFFFF_FFFC. User code may change this value to select a different fetch address on exit from a user initiated reset of this core. Core 1 MMU TLB entry 0 effective and real page number (EPN/RPN) is automatically loaded with RSTVEC[0:21] when reset is released and the TLB entry size is set to 4KB. In the *e200z759 Core Reference Manual* the reset vector is referred to as “p_rstbase[0-21]”.
30 RST	Controls the assertion of RESET to core 0. Writing 1 to this bit causes core 0 to enter reset, provided core 0 halt bit is also set. Reads of this bit indicate whether the core is being held in reset. 0 core 0 not in reset 1 core 0 reset requested
31 VLE	VLE Select. The VLE bit selects whether the core executes VLE or Book E code at the reset vector address. 0 Book E 1 VLE

Offset: SIU_A_BASE + 0x09B0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RSTVEC																														RST	VLE
W																																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Figure 3-35. Reset Vector Register for Core 1 (SIU_RSTVEC1)

Table 3-53. SIU_RSTVEC1 Field Descriptions

Field	Description
0-29 RSTVEC	Core 1 Reset Vector. The RSTVEC value determines the initial program counter for core 1 upon exiting reset. On POR, the value contained in the register defaults to 0xFFFF_FFFC, so that core 1 fetches BookE code from the BAM starting at address 0xFFFF_FFFC. User code may change this value to select a different fetch address on exit from a user initiated reset of this core. Core 1 MMU TLB entry 0 effective and real page number (EPN/RPN) is automatically loaded with RSTVEC[0:21] when reset is released and the TLB entry size is set to 4KB. In the *e200z759 Core Reference Manual* the reset vector is referred to as “p_rstbase[0-21]”.
30 RST	Controls the assertion of RESET to core1. Writing 1 to this bit causes core1 to enter reset, provided core1 halt bit is also set. Reads of this bit indicate if the core is being held in reset. 0 core 1 not in reset 1 core 1 reset requested
31 VLE	VLE Select. The VLE bit selects whether the core executes VLE or Book E code at the reset vector address. 0 Book E 1 VLE

3.2.1.33 Core 0/1 PID mapping control register (SIU_C0PID, SIU_C1PID)

These registers allow an external tool to modify the logical to physical address mapping without interrupting normal application code execution in either core. An optional synchronization mechanism using the nex_wevto[2] watchpoint event output from each core allows the mapping that is selected by the tool to change deterministically, when a specified instruction address is reached or a specified load/store address is accessed. If the synchronization mechanism is not enabled, a write to the register is propagated immediately to the MMU.

Offset: SIU_A_BASE + 0x09B4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EXT_PID_EN	EXT_PID_SYNC	Reserved																												EXT_PID6	EXT_PID7
W																																
Reset	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0

Figure 3-36. Core 0 PID Mapping Control Register (SIU_C0PID)

Table 3-54. SIU_C0PID Field Descriptions

Field	Description
0 EXT_PID_EN	External PID Selection Enable 0 EXT_PID[6:7] are not used to select an alternate MMU address mapping in core 0 1 EXT_PID[6:7] are used to select an alternate MMU address mapping in core 0
1 EXT_PID_SYNC	External PID Synchronization 0 Nexus watchpoint event 2 (nex_wevto[2]) is not enabled to transfer the EXT_PID[6:7] values to core 0 MMU 1 Nexus watchpoint event 2 (nex_wevto[2]) is enabled to transfer the EXT_PID[6:7] values to core 0 MMU
2-29 Reserved	Must be written with 0s for future compatibility. Writes have no effect.
30 EXT_PID6	External PID6 for core 0
31 EXT_PID7	External PID7 for core 0

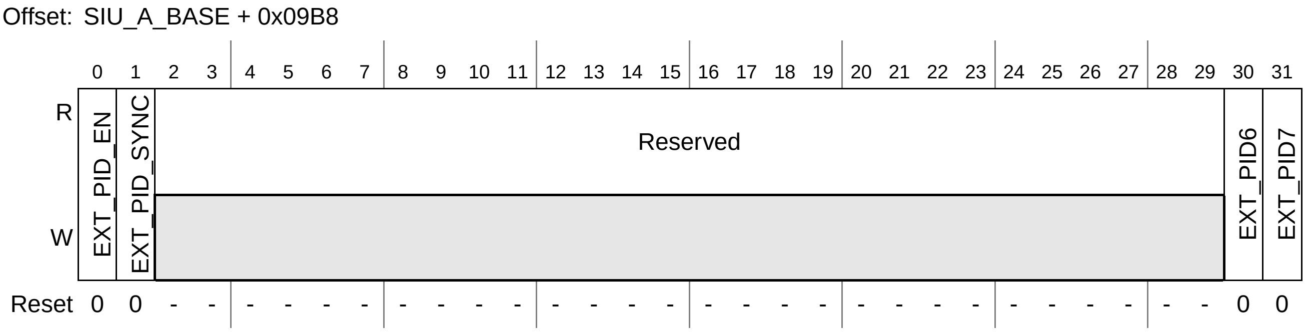


Figure 3-37. Core 1 PID Mapping Control Register (SIU_C1PID)

Table 3-55. SIU_C1PID Field Descriptions

Field	Description
0 EXT_PID_EN	External PID Selection Enable 0 EXT_PID[6:7] are not used to select an alternate MMU address mapping in core 1 1 EXT_PID[6:7] are used to select an alternate MMU address mapping in core 1
1 EXT_PID_SYNC	External PID Synchronization 0 Nexus watchpoint event 2 (nex_wevto[2]) is not enabled to transfer the EXT_PID[6:7] values to core 1MMU 1 Nexus watchpoint event 2 (nex_wevto[2]) is enabled to transfer the EXT_PID[6:7] values to core 1 MMU
2-29 Reserved	Must be written with 0s for future compatibility. Writes have no effect.
30 EXT_PID6	External PID6 for core 1
31 EXT_PID7	External PID7 for core 1

3.2.1.34 Parallel GPIO Output Register (SIU_PGPDO0 - SIU_PGPDO15)

The PGPDO*x* registers are written to by software to drive data out on the external GPIO pin. These registers access the same GPIO pins accessed by SIU_GPDO0–SIU_GPDO511 bit registers. The SIU_GPDO registers should map directly to these registers. For example, SIU_PGPDO0 bit 31 is SIU_GPDO28_31 bit 31, SIU_PGPDO0 bit 30 is SIU_GPDO28_31 bit 23, ..., SIU_PGPD07 bit 0 is SIU_GPDO224_227, bit 7.

The parallel GPIO read/write should be decode the logical addresses to the same physical address of the normal GPIO. This way both GPDO and corresponding PGPDO registers will be updated by a single write to a either register. The parallel GPIO read/write function should be enabled/disabled entirely by setting an input parameter to the SIU.

SIU_BASE + 0xC00 - SIU_BASE + 0xC3C (16)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PGPDO 0	PGPDO 1	PGPDO 2	PGPDO 3	PGPDO 4	PGPDO 5	PGPDO 6	PGPDO 7	PGPDO 8	PGPDO 9	PGPDO 10	PGPDO 11	PGPDO 12	PGPDO 13	PGPDO 14	PGPDO 15
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PGPDO 16	PGPDO 17	PGPDO 18	PGPDO 19	PGPDO 20	PGPDO 21	PGPDO 22	PGPDO 23	PGPDO 24	PGPDO 25	PGPDO 26	PGPDO 27	PGPDO 28	PGPDO 29	PGPDO 30	PGPDO 31
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-38. Parallel GPIO Pin Data Output Register (SIU_PGPDO0 - SIU_PGPDO15)

Table 3-56. SIU_PGPDO0 - SIU_PGPDO15 Field Descriptions

Field	Description
PGPDO*x*	Pin Data Out. Stores the data to be driven out on the external GPIO pin controlled by this register. 0 Logic low value is driven on the data out signal for the corresponding GPOI pin when the pin is configured as an output. 1 Logic high value is driven on the data out signal for the corresponding GPOI pin when the pin is configured as an output.

3.2.1.35 Parallel GPIO Input Register (SIU_PGPDI0 - SIU_PGPDI15)

The GPDI*x* registers are read-only registers that allow reading of the input state of an external GPIO pin. These registers access the same GPIO pins accessed by SIU_GPDI0 - SIU_GPDI511 bit registers. The SIU_GPDI registers should map directly to these registers. For example, SIU_PGPDI0 bit 31 is SIU_GPDI28_31 bit 24, SIU_PGPDI0 bit 30 is SIU_GPDI28_31 bit 23, ..., SIU_PGPDI7 bit 0 is SIU_GPDI224_227, bit 7.

The GPIO read/write should decode logical addresses to the same physical address of the normal GPIO. This way both the GPDI and corresponding PGPDI register should be updated on a pin state change when the IBE is asserted in the corresponding PCR.

The parallel GPIO read/write function should be enabled/disabled entirely by setting an input parameter to the SIU.

SIU_BASE + 0xC40 - SIU_BASE + 0xC7C (16)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PGPDI 0	PGPDI 1	PGPDI 2	PGPDI 3	PGPDI 4	PGPDI 5	PGPDI 6	PGPDI 7	PGPDI 8	PGPDI 9	PGPDI 10	PGPDI 11	PGPDI 12	PGPDI 13	PGPDI 14	PGPDI 15
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PGPDI 16	PGPDI 17	PGPDI 18	PGPDI 19	PGPDI 20	PGPDI 21	PGPDI 22	PGPDI 23	PGPDI 24	PGPDI 25	PGPDI 26	PGPDI 27	PGPDI 28	PGPDI 29	PGPDI 30	PGPDI 31
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-39. Parallel GPIO Pin Data Input Register (SIU_PGPDI0 - SIU_PGPDI15)

Table 3-57. SIU_PGPDI0 - SIU_PGPDI15 - SIU_PGPDO15 Field Descriptions

Field	Description
PGPDI*x*	Pin Data In. Stores the value of the pad-interface signals (data in) corresponding to the external GPIO pin associated with the register. 0 The value of the pad-interface signals (data in) for the corresponding GPIO pin is logic low. 1 The value of the pad-interface signals (data in) for the corresponding GPIO pin is logic high.

3.2.1.36 Masked Parallel GPIO Output Register (SIU_MPGPDO0 - SIU_MPGPDO31)

The MPGPDOx registers are written to by software to drive data out on the external GPIO pin.These registers access the same GPIO pins accessed by SIU_GPDO0 - SIU_GPDO511 bit registers. The most significant 16 bits in the SIU_MPGPDO registers should map directly to these registers. For example, SIU_MPGPDO0 bit 31 is SIU_GPDO28_31 bit 31, SIU_MPGPDO0 bit 30 is SIU_MGPDO28_31 bit 23, ..., SIU_MPGPD15 bit 16 is SIU_GPDO240_243, bit 7. The least significant sixteen bits are the corresponding values to be written at GPIO pins defined by MASK field. The masked parallel GPIO read/write should be decode the logical addresses to the same physical address of the normal GPIO. The masked parallel GPIO read/write function should be enabled/disabled entirely by setting an input parameter to the SIU.

SIU_BASE + 0xC80 - SIU_BASE + 0xCFC (32)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MASK 0	MASK 1	MASK 2	MASK 3	MASK 4	MASK 5	MASK 6	MASK 7	MASK 8	MASK 9	MASK 10	MASK 11	MASK 12	MASK 13	MASK 14	MASK 15
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DATA 0	DATA 1	DATA 2	DATA 3	DATA 4	DATA 5	DATA 6	DATA 7	DATA 8	DATA 9	DATA 10	DATA 11	DATA 12	DATA 13	DATA 14	DATA 15
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-40. Masked Parallel GPIO Pin Data Output Register (SIU_MPGPDO0 - SIU_MPGPDO31)

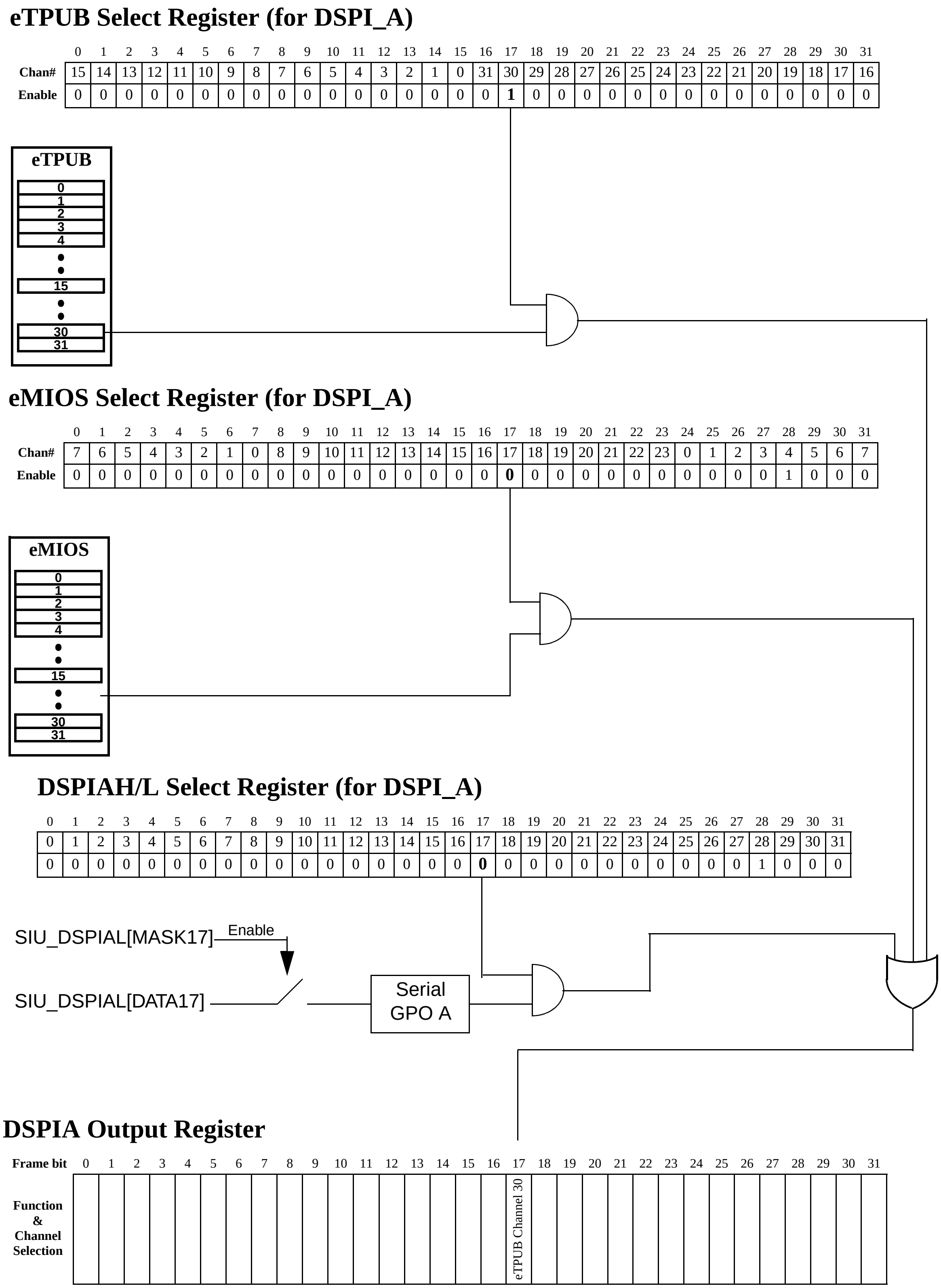
Table 3-58. SIU_MPGPDO0 - SIU_MPGPDO31 Field Descriptions

Field	Description
MASK*x*	Pin Data Out. Controls the write access to the corresponding GPDO. 0 Previous value defined by GPDO is maintain. 1 Corresponding GPDO is written with value defined by DATA field.
DATAx	Pin Data Out. Stores the data to be driven out on the external GPIO pin controlled by this register. 0 Logic low value is driven on the pad interface data out signal for the corresponding GPIO pin when the pin is configured as an output. 1 Logic high value is driven on the pad interface data out signal for the corresponding GPIO pin when the pin is configured as an output.

3.2.1.37 SIU DSPI Serialization Registers

Each bit in the DSPI serialized output frame can contain a signal routed from the output of one of three on-chip sources. The sources on this device are the eTPU module, eMIOS module, or a software-updated GPO data register. The mapping between module or data register and the output frame for a single DSPI channel is performed by independent 32-bit registers, one for each module that is mapped plus one for the data register. An example of how these registers are used to create the DSPI serialized output is shown below.

Example of Pin Multiplexing



eTPUB Select Register (for DSPI_A)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chan#	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Enable	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0

eMIOS Select Register (for DSPI_A)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Chan#	7	6	5	4	3	2	1	0	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	0	1	2	3	4	5	6	7
Enable	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0	0	0	0	0	0	1	0	0	0

DSPIAH/L Select Register (for DSPI_A)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0**	0	0	0	0	0	0	0	0	0	0	1	0	0	0

DSPIA Output Register

Frame bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Function & Channel Selection																		eTPUB Channel 30														

3.2.1.37.1 Masked Serial GPO Registers for DSPI (SIU_DSPIAH, SIU_DSPIAL, SIU_DSPIBH, SIU_DSPIBL, SIU_DSPICH, SIU_DSPICL, SIU_DSPIDH, SIU_DSPIDL)

These registers are written by software to drive data out on the serialization module described in the Figure 3-45, Figure 3-48, Figure 3-51, Figure 3-54. The purpose of these registers is to allow any combination of bits in each half of the 32 bit serialized data frame to be updated with a single 32-bit write operation, while allowing other bits to maintain their previous state. This is accomplished by writing a 16 bit masked value coherently with an update value contained in a 16 bit output field, and only updating those bits in the output register for which the corresponding mask bit is set.

Table 3-59. SIU Address Map for Masked Serial Output and Serial Selection Registers

Address	Use
SIU_BASE + 0x0D00 -SIU_BASE + 0x0D03	DSPIA GP Mask-Output High Register
SIU_BASE + 0x0D04 -SIU_BASE + 0x0D07	DSPIA GP Mask-Output Low Register
SIU_BASE + 0x0D08 -SIU_BASE + 0x0D0B	DSPIB GP Mask-Output High Register
SIU_BASE + 0x0D0C -SIU_BASE + 0x0D0F	DSPIB GP Mask-Output Low Register
SIU_BASE + 0x0D10 -SIU_BASE + 0x0D13	DSPIC GP Mask-Output High Register
SIU_BASE + 0x0D14 -SIU_BASE + 0x0D17	DSPIC GP Mask-Output Low Register
SIU_BASE + 0x0D18 -SIU_BASE + 0x0D1B	DSPID GP Mask-Output High Register
SIU_BASE + 0x0D1C -SIU_BASE + 0x0D1F	DSPID GP Mask-Output Low Register
SIU_BASE + 0x0D20 -SIU_BASE + 0x0D3F	Reserved
SIU_BASE + 0x0D40 -SIU_BASE + 0x0D43	DSPIA eTPUB Select Register
SIU_BASE + 0x0D44 -SIU_BASE + 0x0D47	DSPIA eMIOS Select Register
SIU_BASE + 0x0D48 -SIU_BASE + 0x0D4B	DSPIA GPO Select Register
SIU_BASE + 0x0D4C -SIU_BASE + 0x0D4F	Reserved
SIU_BASE + 0x0D50 -SIU_BASE + 0x0D53	DSPIB eTPUA Select Register
SIU_BASE + 0x0D54 -SIU_BASE + 0x0D57	DSPIB eMIOS Select Register
SIU_BASE + 0x0D58 -SIU_BASE + 0x0D5B	DSPIB GPO Select Register
SIU_BASE + 0x0D5C -SIU_BASE + 0x0D5F	Reserved
SIU_BASE + 0x0D60 -SIU_BASE + 0x0D63	DSPIC eTPUA Select Register
SIU_BASE + 0x0D64 -SIU_BASE + 0x0D67	DSPIC eMIOS Select Register
SIU_BASE + 0x0D68 -SIU_BASE + 0x0D6B	DSPIC GPO Select Register
SIU_BASE + 0x0D6C -SIU_BASE + 0x0D6F	Reserved
SIU_BASE + 0x0D70 -SIU_BASE + 0x0D73	DSPID eTPUB Select Register
SIU_BASE + 0x0D74 -SIU_BASE + 0x0D77	DSPID eMIOS Select Register
SIU_BASE + 0x0D78 -SIU_BASE + 0x0D7B	DSPID GPO Select Register
SIU_BASE + 0x0D7C -SIU_BASE + 0x0D7F	Reserved
SIU_BASE + 0x0D80 -SIU_BASE + 0x0DC0	Reserved

SIU_BASE + 0xD00, SIU_BASE + 0xD08, SIU_BASE + 0xD10, SIU_BASE + 0xD18

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MASK 0	MASK 1	MASK 2	MASK 3	MASK 4	MASK 5	MASK 6	MASK 7	MASK 8	MASK 9	MASK 10	MASK 11	MASK 12	MASK 13	MASK 14	MASK 15
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DATA 0	DATA 1	DATA 2	DATA 3	DATA 4	DATA 5	DATA 6	DATA 7	DATA 8	DATA 9	DATA 10	DATA 11	DATA 12	DATA 13	DATA 14	DATA 15
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-41. Masked Serial GPO Register for DSPI - DSPI_A/B/C/D GPO Mask Output High Register (SIU_DSPIAH/SIU_DSPIBH/SIU_DSPICH/SIU_DSPIDH)

SIU_BASE + 0xD04, SIU_BASE + 0xD0C, SIU_BASE + 0xD14, SIU_BASE + 0xD1C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MASK 16	MASK 17	MASK 18	MASK 19	MASK 20	MASK 21	MASK 22	MASK 23	MASK 24	MASK 25	MASK 26	MASK 27	MASK 28	MASK 29	MASK 30	MASK 31
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DATA 16	DATA 17	DATA 18	DATA 19	DATA 20	DATA 21	DATA 22	DATA 23	DATA 24	DATA 25	DATA 26	DATA 27	DATA 28	DATA 29	DATA 30	DATA 31
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-42. Masked Serial GPO Register for DSPI - DSPI_A/B/C/D GPO Mask Output Low Register (SIU_DSPIAL/SIU_DSPIBL/SIU_DSPICL/SIU_DSPIDL)

Table 3-60. SIU_DSPIAL/SIU_DSPIBL/SIU_DSPICL/SIU_DSPIDL Field Descriptions

Field	Description
MASK*x*	Pin Data Out. Controls the write access to the corresponding GPO for DSPI. 0 Previous value defined by GPDO is maintain. 1 Corresponding GPO is written with value defined by DATA field.
DATAx	Pin Data Out. Stores the data to be driven out on the external GPIO pin controlled by this register. 0 Logic low value is driven on the pad interface data out signal for the corresponding GPO for DSPI when this output is selected in the DSPI serialization module. 1 Logic high value is driven on the pad interface data out signal for the corresponding GPO for DSPI when this output is selected in the DSPI serialization module.

3.2.1.37.2 Serialized Output Signal Selection Registers for DSPI_A

The following three registers are used by DSPI_A to select the sources of the serialized output when running in DSI or CSI configuration.

Each register bit enables a path from the eTPU_B channel, eMIOS channel and data register bit SIU_DSPIAH/SIU_DSPIAL to the equivalent bit position in the DSPI_A serialized output frame. The user must ensure that bit selections from each of these registers do not overlap. Multiple sources are logically ORed, which provides the potential for combining outputs from multiple timer channels and data registers to produce more complex bit behavior.

SIU_BASE + 0xD40

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ETPUB 15	ETPUB 14	ETPUB 13	ETPUB 12	ETPUB 11	ETPUB 10	ETPUB 9	ETPUB 8	ETPUB 7	ETPUB 6	ETPUB 5	ETPUB 4	ETPUB 3	ETPUB 2	ETPUB 1	ETPUB 0
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ETPUB 31	ETPUB 30	ETPUB 29	ETPUB 28	ETPUB 27	ETPUB 26	ETPUB 25	ETPUB 24	ETPUB 23	ETPUB 22	ETPUB 21	ETPUB 20	ETPUB 19	ETPUB 18	ETPUB 17	ETPUB 16
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-43. eTPU_B Select Register for DSPI_A (SIU_ETPUBA)

Table 3-61. SIU_ETPUBA Field Descriptions

Field	Description
ETPUB*x*	ETPUB channel select 0 This bit in the DSPI_A serialized output frame will not use the respective ETPUB channel 1 This bit in the DSPI_A serialized output frame will use the respective ETPUB channel

SIU_BASE + 0xD44

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	EMIOS 7	EMIOS 6	EMIOS 5	EMIOS 4	EMIOS 3	EMIOS 2	EMIOS 1	EMIOS 0	EMIOS 8	EMIOS 9	EMIOS 10	EMIOS 11	EMIOS 12	EMIOS 13	EMIOS 14	EMIOS 15
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EMIOS 16	EMIOS 17	EMIOS 18	EMIOS 19	EMIOS 20	EMIOS 21	EMIOS 22	EMIOS 23	EMIOS 0	EMIOS 1	EMIOS 2	EMIOS 3	EMIOS 4	EMIOS 5	EMIOS 6	EMIOS 7
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-44. eMIOS Select Register for DSPI_A (SIU_EMIOSA)

Table 3-62. SIU_EMIOSA Field Descriptions

Field	Description
EMIOS*x*	EMIOS channel select 0 This bit in the DSPI_A serialized output frame will not use the respective EMIOS channel 1 This bit in the DSPI_A serialized output frame will use the respective EMIOS channel

SIU_BASE + 0xD48

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DSPI AH 0	DSPI AH 1	DSPI AH 2	DSPI AH 3	DSPI AH 4	DSPI AH 5	DSPI AH 6	DSPI AH 7	DSPI AH 8	DSPI AH 9	DSPI AH 10	DSPI AH 11	DSPI AH 12	DSPI AH 13	DSPI AH 14	DSPI AH 15
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DSPI AL 16	DSPI AL 17	DSPI AL 18	DSPI AL 19	DSPI AL 20	DSPI AL 21	DSPI AL 22	DSPI AL 23	DSPI AL 24	DSPI AL 25	DSPI AL 26	DSPI AL 27	DSPI AL 28	DSPI AL 29	DSPI AL 30	DSPI AL 31
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-45. SIU_DSPIAH/L Select Register for DSPI_A (SIU_DSPIAHLA)

Table 3-63. SIU_DSPIAHLA Field Descriptions

Field	Description
DSPIAH/L*x*	DSPI_A Data Register bit 0 The corresponding serial GPO A output (from the SIU_DSPIAH/L register) is disabled 1 The corresponding serial GPO A output (from the SIU_DSPIAH/L register) is enabled

3.2.1.37.3 Serialized Output Signal Selection Registers for DSPI_B

The following three registers are used by DSPI_B to select the sources of the serialized output when running in DSI or CSI configuration.

Each register bit enables a path from the eTPU_A channel, eMIOS channel and data register bit SIU_DSPIBH/SIU_DSPIBL to the equivalent bit position in the DSPI_B serialized output frame. The user must ensure that bit selections from each of these registers do not overlap. Multiple sources are logically ORed, which provides the potential for combining outputs from multiple timer channels and data registers to produce more complex bit behavior.

SIU_BASE + 0xD50

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ETPUA 23	ETPUA 22	ETPUA 21	ETPUA 20	ETPUA 19	ETPUA 18	ETPUA 17	ETPUA 16	ETPUA 29	ETPUA 28	ETPUA 27	ETPUA 26	ETPUA 25	ETPUA 24	ETPUA 31	ETPUA 30
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ETPUA 12	ETPUA 13	ETPUA 14	ETPUA 15	ETPUA 0	ETPUA 1	ETPUA 2	ETPUA 3	ETPUA 4	ETPUA 5	ETPUA 6	ETPUA 7	ETPUA 8	ETPUA 9	ETPUA 10	ETPUA 11
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-46. eTPU_A Select Register for DSPI_B (SIU_ETPUAB)

Table 3-64. SIU_ETPUAB Field Descriptions

Field	Description
ETPUA*x*	ETPUA channel select 0 This bit in the DSPI_B serialized output frame will not use the respective ETPUA channel 1 This bit in the DSPI_B serialized output frame will use the respective ETPUA channel

SIU_BASE + 0xD54

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	EMIOS 11	EMIOS 10	EMIOS 9	EMIOS 8	EMIOS 6	EMIOS 5	EMIOS 4	EMIOS 3	EMIOS 2	EMIOS 1	EMIOS 0	EMIOS 23	EMIOS 15	EMIOS 14	EMIOS 13	EMIOS 12
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EMIOS 23	EMIOS 15	EMIOS 14	EMIOS 13	EMIOS 12	EMIOS 11	EMIOS 10	EMIOS 9	EMIOS 8	EMIOS 6	EMIOS 5	EMIOS 4	EMIOS 3	EMIOS 2	EMIOS 1	EMIOS 0
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-47. eMIOS Select Register for DSPI_B (SIU_EMIOSB)

Table 3-65. SIU_EMIOSB Field Descriptions

Field	Description
EMIOS*x*	EMIOS channel select 0 This bit in the DSPI_B serialized output frame will not use the respective EMIOS channel 1 This bit in the DSPI_B serialized output frame will use the respective EMIOS channel

SIU_BASE + 0xD58

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DSPI BH 0	DSPI BH 1	DSPI BH 2	DSPI BH 3	DSPI BH 4	DSPI BH 5	DSPI BH 6	DSPI BH 7	DSPI BH 8	DSPI BH 9	DSPI BH 10	DSPI BH 11	DSPI BH 12	DSPI BH 13	DSPI BH 14	DSPI BH 15
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DSPI BL 16	DSPI BL 17	DSPI BL 18	DSPI BL 19	DSPI BL 20	DSPI BL 21	DSPI BL 22	DSPI BL 23	DSPI BL 24	DSPI BL 25	DSPI BL 26	DSPI BL 27	DSPI BL 28	DSPI BL 29	DSPI BL 30	DSPI BL 31
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-48. SIU_DSPIBH/L Select Register for DSPI_B (SIU_DSPIBHLB)

Table 3-66. SIU_DSPIBHLB Field Descriptions

Field	Description
DSPIBH/L*x*	DSPI_B Data Register bit 0 The corresponding serial GPO B output (from the SIU_DSPIBH/L register) is disabled 1 The corresponding serial GPO B output (from the SIU_DSPIBH/L register) is enabled

3.2.1.37.4 Serialized Output Signal Selection Registers for DSPI_C

The following three registers are used by DSPI_C to select the sources of the serialized output when running in DSI or CSI configuration.

Each register bit enables a path from the eTPU_A channel, eMIOS channel and data register bit SIU_DSPICH/SIU_DSPICL to the equivalent bit position in the DSPI_C serialized output frame. The user must ensure that bit selections from each of these registers do not overlap. Multiple sources are logically ORed, which provides the potential for combining outputs from multiple timer channels and data registers to produce more complex bit behavior.

SIU_BASE + 0xD60

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ETPUA 12	ETPUA 13	ETPUA 14	ETPUA 15	ETPUA 0	ETPUA 1	ETPUA 2	ETPUA 3	ETPUA 4	ETPUA 5	ETPUA 6	ETPUA 7	ETPUA 8	ETPUA 9	ETPUA 10	ETPUA 11
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ETPUA 23	ETPUA 22	ETPUA 21	ETPUA 20	ETPUA 19	ETPUA 18	ETPUA 17	ETPUA 16	ETPUA 29	ETPUA 28	ETPUA 27	ETPUA 26	ETPUA 25	ETPUA 24	ETPUA 31	ETPUA 30
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-49. eTPU_A Select Register for DSPI_C (SIU_ETPUAC)

Table 3-67. SIU_ETPUAC Field Descriptions

Field	Description
ETPUA*x*	ETPUA channel select 0 This bit in the DSPI_C serialized output frame will not use the respective ETPUA channel 1 This bit in the DSPI_C serialized output frame will use the respective ETPUA channel

SIU_BASE + 0xD64

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	EMIOS 12	EMIOS 13	EMIOS 14	EMIOS 15	EMIOS 23	EMIOS 0	EMIOS 1	EMIOS 2	EMIOS 3	EMIOS 4	EMIOS 5	EMIOS 6	EMIOS 8	EMIOS 9	EMIOS 10	EMIOS 11
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EMIOS 23	EMIOS 22	EMIOS 21	EMIOS 20	EMIOS 19	EMIOS 18	EMIOS 17	EMIOS 16	EMIOS 29	EMIOS 28	EMIOS 27	EMIOS 26	EMIOS 25	EMIOS 24	EMIOS 31	EMIOS 30
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-50. eMIOS Select Register for DSPI_C (SIU_EMIOSC)

Table 3-68. SIU_EMIOSC Field Descriptions

Field	Description
EMIOS*x*	EMIOS channel select 0 This bit in the DSPI_C serialized output frame will not use the respective EMIOS channel 1 This bit in the DSPI_C serialized output frame will use the respective EMIOS channel

SIU_BASE + 0xD68

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DSPI CH 0	DSPI CH 1	DSPI CH 2	DSPI CH 3	DSPI CH 4	DSPI CH 5	DSPI CH 6	DSPI CH 7	DSPI CH 8	DSPI CH 9	DSPI CH 10	DSPI CH 11	DSPI CH 12	DSPI CH 13	DSPI CH 14	DSPI CH 15
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DSPI CL 16	DSPI CL 17	DSPI CL 18	DSPI CL 19	DSPI CL 20	DSPI CL 21	DSPI CL 22	DSPI CL 23	DSPI CL 24	DSPI CL 25	DSPI CL 26	DSPI CL 27	DSPI CL 28	DSPI CL 29	DSPI CL 30	DSPI CL 31
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-51. SIU_DSPICH/L Select Register for DSPI_C (SIU_DSPICHLC)

Table 3-69. SIU_DSPICHLC Field Descriptions

Field	Description
DSPICH/L*x*	DSPI_C Data Register bit 0 The corresponding serial GPO C output (from the SIU_DSPICH/L register) is disabled 1 The corresponding serial GPO C output (from the SIU_DSPICH/L register) is enabled

3.2.1.38 Serialized Output Signal Selection Registers for DSPI_D

The following registers are used by DSPI_D to select the sources of the serialized output when running in DSI or CSI configuration.

The register bit enables a path from the eTPU_B channel or eMIOS channel to the equivalent bit position in the DSPI_D serialized output frame. The user must ensure that bit selections from each of these registers do not overlap. Multiple sources are logically ORed, which provides the potential for combining outputs from multiple timer channels and data registers to produce more complex bit behavior.

SIU_BASE + 0xD70

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ETPUB 21	ETPUB 20	ETPUB 19	ETPUB 18	ETPUB 17	ETPUB 16	0	0	0	0	ETPUB 29	ETPUB 28	ETPUB 27	ETPUB 26	ETPUB 25	ETPUB 24
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-52. eTPU_B Select Register for DSPI_D (SIU_ETPUBD)

Table 3-70. SIU_ETPUAD Field Descriptions

Field	Description
ETPUB*x*	ETPUB channel select 0 This bit in the DSPI_D serialized output frame will not use the respective ETPUB channel 1 This bit in the DSPI_D serialized output frame will use the respective ETPUB channel

SIU_BASE + 0xD74

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	EMIOS 11	EMIOS 10	EMIOS 13	EMIOS 12	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-53. eMIOS Select Register for DSPI_D (SIU_EMIOSD)

Table 3-71. SIU_EMIOSD Register Field Descriptions

Field	Description
EMIOS*x*	EMIOS channel select 0 This bit in the DSPI_D serialized output frame will not use the respective EMIOS channel 1 This bit in the DSPI_D serialized output frame will use the respective EMIOS channel

SIU_BASE + 0xD78

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3-54. DSPIDH/L Select Register for DSPI_D (SIU_DSPIDHLD)

3.2.1.39 GPIO Pin Data Input Registers (SIU_GPDI0_3 - SIU_GPDI508_511)

Previous devices are limited to supporting a maximum of 256 general purpose input registers, and thus a maximum GPIO input assignment of GPIO[0:255]. The reason is that only 256 bytes of SIU address space is allocated for the GPIO Pin Data Input registers, and adjacent memory locations are unavailable.

To accommodate the extended number of the GPIO pads, this device has a new memory map definition of 512 bytes for allocating up to 512 GPIO Pin Data Input registers. For compatibility, the first used 256 GPIO Pin Data Input registers are mapped to both the old and new locations. The memory map is shown in Table 3-72.

Table 3-72. GPIO Pin Data Input Registers Memory Map

Address Offset from SIU_BASE	Number
0x0E00 - 0x0E4A	Reserved
0x0E4B - 0x0E6E	GPIO Pin Data Input Registers 75 - 110
0x0E6F - 0x0E70	Reserved
0x0E71 - 0x0ECC	GPIO Pin Data Input Registers 113- 204

Table 3-72. GPIO Pin Data Input Registers Memory Map

0x0ECD - 0x0ECF	Reserved
0x0ED0 - 0x0ED1	GPIO Pin Data Input Registers 208- 209
	Reserved
0x0ED3	GPIO Pin Data Input Registers 211
	Reserved
0x0ED5	GPIO Pin Data Input Registers 213
	Reserved
0x0EE7 - 0x0EFD	GPIO Pin Data Input Registers 231 - 253
	Reserved
0x0F00 - 0x0F33	GPIO Pin Data Input Registers 256 - 307
	Reserved
0x0FB0 - 0x0FB5	GPIO Pin Data Input Registers 432 - 437
	Reserved
0x0FB8 - 0x0FD8	GPIO Pin Data Input Registers 440 - 472
	Reserved

The GPDI*x*_*x* registers are read-only registers that allow reading of the input state of an external GPIO pin. Each byte of a register represents the input state of a single external GPIO pin. The first 256 GPDI*x*_*x* registers corresponds to the same GPDI inputs described in Section 3.2.1.15, "GPIO Pin Data Input Registers 0–255 (SIU_GPDIn)".

SIU_BASE+0x0E00

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	PDI0
W								
RESET:	0	0	0	0	0	0	0	U

Figure 3-55. GPIO Pin Data In Register 0 (SIU_GPDI0)

SIU_BASE+0xFFC

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	PDI 511
W								
RESET:	0	0	0	0	0	0	0	U

Figure 3-56. GPIO Pin Data In Register 511 (SIU_GPDI511)

Table 3-73. SIU_GPIO Pin Data In Field Descriptions

Field	Description
PDI*x*	This bit reflects the input state on the external GPIO pin for the register. If PCR*n*[IBE] = 1, then: 0 Signal on pin is a logic 0 1 Signal on pin is a logic 1

3.3 SIU_B

SIU_B contains a set of mirrored PCR configuration, GPO and GPI data registers for GPIO[240:243] balls. By default after reset, the mirrored registers are disabled and provide no function. The disabled state is indicated by PA[2:0] = 0b111 in the mirrored PCR register, as shown in Table 3-74. When a mirrored PCR is enabled by writing a different value to its PA[2:0], it has priority over the equivalent “non-mirrored” register, and controls the ball function and characteristics defined by the implemented bit fields of the mirrored PCR, and the mirrored GPO (Table 3-75) and GPI (Table 3-76) data registers control and reflect the logic level. When the mirrored registers are enabled, writes to the non mirrored registers have no effect.

Table 3-74. Mirrored PCR[240:243]

PCRI	PCR Address offset from SIU_B base	PCR bit fields reset values												
		PA2	PA1	PA0	OBE	IBE	DSC1	DSC0	ODE	HYS	SRC1	SRC0	WPE	WPS
PCR240M	0x220	1	1	1	0	0	-	-	0	0	0	0	1	1
PCR241M	0x222	1	1	1	0	0	-	-	0	0	0	0	1	1
PCR242M	0x224	1	1	1	0	0	-	-	0	0	0	0	1	1
PCR243M	0x226	1	1	1	0	0	-	-	0	0	0	0	1	1

Table 3-75. Mirrored GPDO[240:243]

GPDO	GPDO Address offset from SIU_B base	Reset Value
GPDO240M	0x6F0	0x00
GPDO241M	0x6F1	0x00
GPDO242M	0x6F2	0x00
GPDO243M	0x6F3	0x00

Table 3-76. Mirrored GPDI[240:243]

GPDI	GPDI Address offset from SIU_B base	Reset Value
GPDI240M	0x8F0	0x00
GPDI241M	0x8F1	0x00
GPDI242M	0x8F2	0x00
GPDI243M	0x8F3	0x00

3.4 Functional Description

The following sections provide a functional overview of the SIU operation.

3.4.1 External Interrupts

There are 16 external interrupt inputs $\overline{\text{IRQ}}[0]$–$\overline{\text{IRQ}}[15]$ to the SIU. The IRQ inputs can be configured for rising-edge events, falling-edge events, or both.

External interrupt requests are triggered by rising- and/or falling-edge events that are enabled by setting a bit in:

- IRQ rising-edge event enable register (SIU_IREER)
- IRQ falling-edge event enable register (SIU_IFEER)

If the bit is set in both registers, both rising- and falling-edge events trigger an interrupt request. Each IRQ has a counter that tracks the number of system clock cycles that occur between the rising- and falling-edge events. An IRQ counter exists for each IRQ rising- or falling-edge event enable bit.

The digital filter length field in the IRQ digital filter register (SIU_IDFR) specifies the minimum number of system clocks that the IRQ signal must hold a logic value to qualify the edge-triggered event as a valid state change. When the number of system clocks in the IRQ counter equals the value in the digital filter length field, the IRQ state latches and the IRQ counter is cleared.

If the previous filtered state of the IRQ does not match the current state, and the rising- or falling-edge event is enabled, the IRQ flag bit is set to 1. For example, the IRQ flag bit is set if a rising-edge event occurs under the following conditions:

- Previous filtered IRQ state was a logic 0
- Current latched IRQ state is a logic 1
- Rising-edge event is enabled for the IRQ

When the counter for an IRQ is not enabled, the state of the IRQ is held in the current and previous state latches. The IRQ counter operates independently of the IRQ or overrun flag bit. Clearing the IRQ flag or overrun flag bits does not clear or reload the counter.

Refer to the following sections for more information:

Section 3.2.1.4, “External Interrupt Status Register (SIU_EISR)”
Section 3.2.1.9, “IRQ Rising-Edge Event Enable Register (SIU_IREER)”
Section 3.2.1.10, “IRQ Falling-Edge Event Enable Register (SIU_IFEER)”
Section 3.2.1.11, “IRQ Digital Filter Register (SIU_IDFR)”

3.4.1.0.1 External Interrupts

The IRQ signals map to 16 independent interrupt requests output from the SIU. The IRQ flag bit is set when a rising-edge and/or falling-edge event occurs for the IRQ. An external IRQ signal is asserted when all of the following occur:

- Enable bit is set in the IRQ rising- and/or falling-edge event registers (SIU_IREER, SIU_IFEER)

- IRQ flag bit is set in the external interrupt status register (SIU_EISR)
- Enable bit is cleared in the DMA/Interrupt request enable register (SIU_DIRER)
- Select bit is cleared in the DMA/Interrupt select register (SIU_DIRSR)

The NMI pin function or platform SWT can generate either an NMI or a critical interrupt. When WKPCFG_NMI_GPIO213 is enabled as NMI, the pin will override the PCR configuration after reset. SIU_DIRER selects between critical and non maskable interrupt use, SIU_EISR reports status of NMI and SIU_IFEER selects edge sensitivity of NMI input

Refer to the following sections for more information:

Section 3.2.1.5, “DMA/Interrupt Request Enable Register (SIU_DIRER)”

Section 3.2.1.6, “DMA/Interrupt Request Select Register (SIU_DIRSR)”

3.4.1.0.2 DMA Transfers

DMA IRQ signals ($\overline{\text{IRQ}}$[0] through $\overline{\text{IRQ}}$[3]) map to four independent DMA transfer *or* interrupt request outputs configured in the SIU. A DMA transfer or interrupt request asserts when all of the following occur:

- IRQ flag bit is set in the external interrupt status register (SIU_EISR)
- Enable bit is set in the DMA transfer or interrupt request enable register (SIU_DIRER)
- Select bit is set in the DMA transfer or interrupt request select register (SIU_DIRSR)

The SIU receives a ‘DMA transfer done’ signal for each DMA or interrupt request transmitted. When the ‘DMA done’ signal asserts, the IRQ flag bit is cleared.

Refer to the following sections for more information:

Section 3.2.1.5, “DMA/Interrupt Request Enable Register (SIU_DIRER)”

Section 3.2.1.6, “DMA/Interrupt Request Select Register (SIU_DIRSR)”

3.4.1.0.3 Overruns

An overrun IRQ exists for each overrun flag bit in the overrun status register (SIU_OSR).
An overrun IRQ asserts when all of the following occur:

- Enable bit is set in the IRQ rising- and/or falling-edge event registers (SIU_IREER, SIU_IFEER)
- IRQ flag bit is set in the external interrupt status register (SIU_EISR)
- Bit is set in the overrun request enable and overrun status registers (SIU_ORER, SIU_OSR)
- Rising- or falling-edge event triggers an interrupt request

The SIU outputs one overrun IRQ bit that is the logical OR of all of the IRQ overrun bits.

Refer to the following sections for more information:

Section 3.2.1.4, “External Interrupt Status Register (SIU_EISR)”
Section 3.2.1.7, “Overrun Status Register (SIU_OSR)”
Section 3.2.1.8, “Overrun Request Enable Register (SIU_ORER)”

3.4.1.0.4 Edge-Detect Events

An IRQ asserts when an:

- Edge-detect event is enabled
- Edge-detect event occurs

To assert an IRQ when an edge-detect event occurs:

1. Set the enable bit in the IRQ rising- and falling-edge event enable registers (SIU_IREER, SIU_IFEER)
2. Clear the enable bits for the DMA/Interrupt request enable register (SIU_DIRER)

The IRQ bit is set in the external IRQ status register (SIU_EISR) when an edge-detect event occurs for that IRQ.

Refer to the following sections for more information:

Section 3.2.1.4, “External Interrupt Status Register (SIU_EISR)”
Section 3.2.1.9, “IRQ Rising-Edge Event Enable Register (SIU_IREER)”
Section 3.2.1.10, “IRQ Falling-Edge Event Enable Register (SIU_IFEER)”

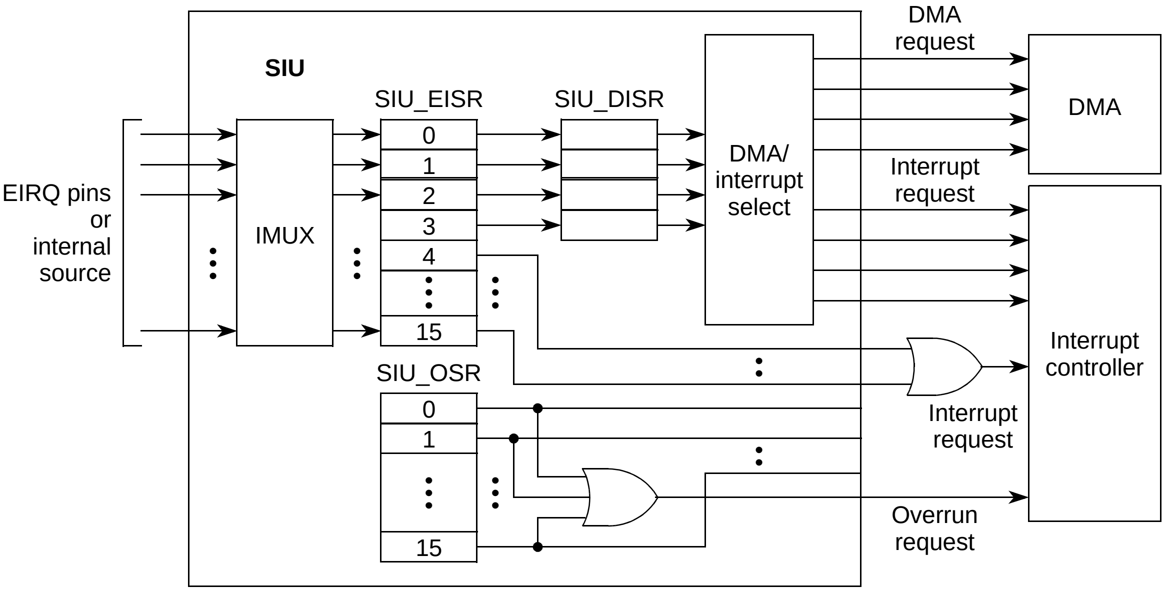


Figure 3-57. SIU DMA/Interrupt Request Diagram

3.4.2 GPIO Operation

All GPIO functions for the device are provided by the SIU. Each device pad that has a GPIO signal has a pin configuration register (PCR) in the SIU where the GPIO function is selected. In addition, each device GPIO signal has an input data register (SIU_GPDI*n*) and an output data register (SIU_GPDO*n*).

The SIU also implements several parallel GPIO registers (SIU_PGPDO*x*_*x* and SIU_PGPDI*x*_*x*) that can be used to access up to 32 GPIO bits in a single- and word-sized accesses. The values read/written to these parallel registers are coherent with the data read/written to the SIU_GPDO*x*_*x* and SIU_GPDI*x*_*x* registers.

3.4.3 Internal Multiplexing

The internal multiplexing select registers (SIU_ISEL4-8, SIU_EIISR, and SIU_DISR) select the input source for the following components:

- eTPUA[24:29] inputs
- eQADC command FIFO trigger sources.
- SIU external interrupt request signals
- DSPI signals used for chaining serial and parallel DSPI modules

A block diagram of the internal multiplexing feature is shown in Figure 3-58. The figure shows the multiplexing of four external signals to an SIU output.

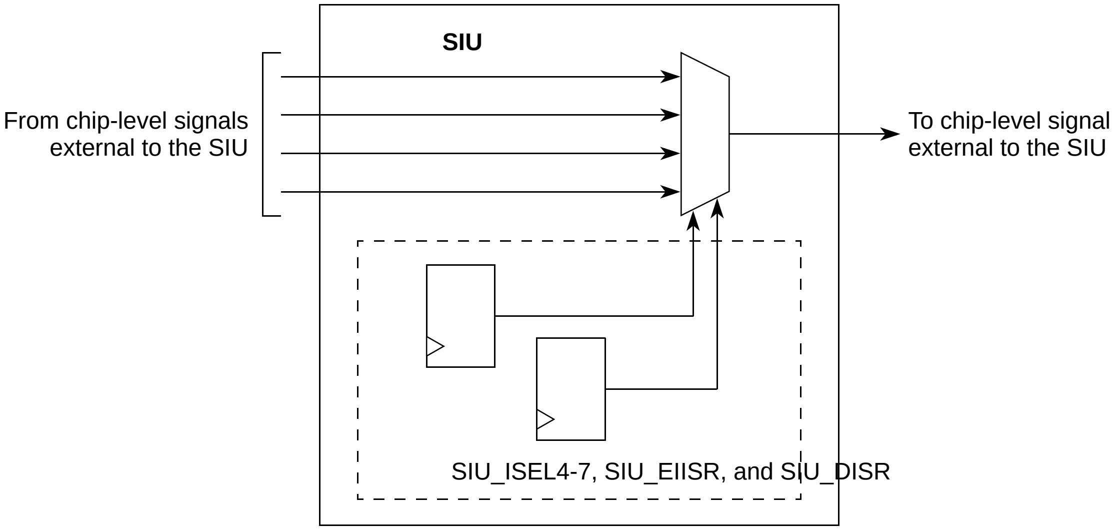


Figure 3-58. Four-to-One Internal Multiplexing Block Diagram

3.4.3.1 eQADC External Trigger Input Multiplexing

The eQADC external trigger inputs can connect to one of the following pins:

- External pin
- eTPU channel
- eMIOS channel
- Periodic interrupt timer

The input source for each eQADC external trigger is configured in the eQADC Command FIFO Trigger Source Select - IMUX Select Registers (SIU_ISEL4-7). For example, you can select one of the following signals to source the ETRIG[0] input trigger for the eQADC:

- TXDC_ETRIG[0]_GPIO[244] pin
- ETPUA[30] output channel
- EMIOS[10] output channel
- Extended trigger source selection (see Table 3-27)

All ETRIG inputs are multiplexed in the same manner. If an eTPU or eMIOS channel is selected as an ETRIG input to the eQADC, you can activate the alternate function of that eTPU or eMIOS signal on the external pin.

An example of the multiplexing of an eQADC (EQADC_A) external trigger input is given below.

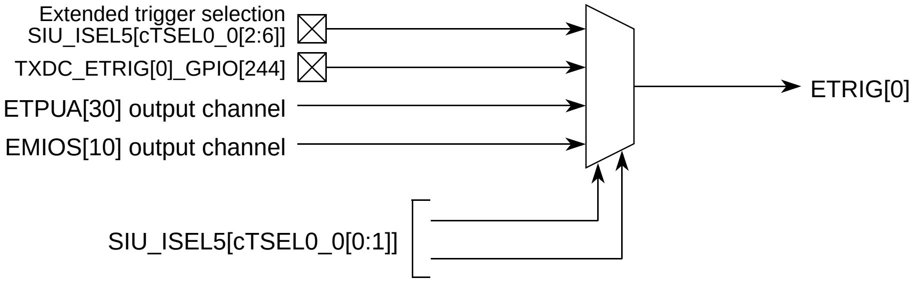


Figure 3-59. eQADC External Trigger Input Multiplexing

SIU_ISEL4-7 always provides one of four choices based on the most-significant two bits of each cTSEL field. When those bits are 0b00, the extended trigger selection is chosen and the remaining bits in cTSEL select among a longer list of trigger sources.

3.4.3.2 Multiplexed Inputs for DSPI Multiple Transfer Operation

Each DSPI module can be combined in a serial or parallel chain (multiple transfer operation). Serial chaining allows SPI operation with an external device that has more bits than one DSPI module.

In a serial chain, one DSPI module operates as a master, the second, third, or fourth DSPI modules operates as aslaves.The slave DSPI and external SPI device use the master peripheral chip select (PCS) and clock (SCK). The trigger input of the master allows a slave DSPI to trigger a transfer when a data change occurs in the slave DSPI and the slave DSPI is operating in change in data mode. The trigger input of the master is connected to the $\overline{\text{MTRIG}}$ output of the slave. If more than two DSPIs are chained in change in data mode, a chain must be connected of $\overline{\text{MTRIG}}$ outputs to trigger inputs through the slaves with the last slave $\overline{\text{MTRIG}}$ output connected to the master trigger input.

An example of a serial chain is shown in Figure 3-60.

Parallel chaining allows the PCS and SCK from one DSPI to be used by more than one external SPI device, thus reducing pin utilization of the device MCU. In this example, the SOUT and SIN of the two DSPIs connect to separate external SPI devices, which share a common PCS and SCK.

To support multiple transfer operation of the DSPIs, an input multiplexor is required for the SIN, $\overline{\text{SS}}$, SCK IN, and trigger signals of each DSPI. The input source for the SIN input of a DSPI can be a pin or the SOUT of any of the other three DSPIs. The input source for the $\overline{\text{SS}}$ input of a DSPI can be a pin or the PCS*X*[0] of any of the other three DSPIs. The input source for the SCK input of a DSPI can be a pin or the SCK output of any of the other three DSPIs. The input source for the trigger input can be the $\overline{\text{PCSS}}$ output of any of the other three DSPIs. The input source for each DSPI SIN, $\overline{\text{SS}}$, SCK, and trigger signal is individually specified in the DSPI input select register (SIU_DISR).

An example of a parallel chain is shown in Figure 3-61.

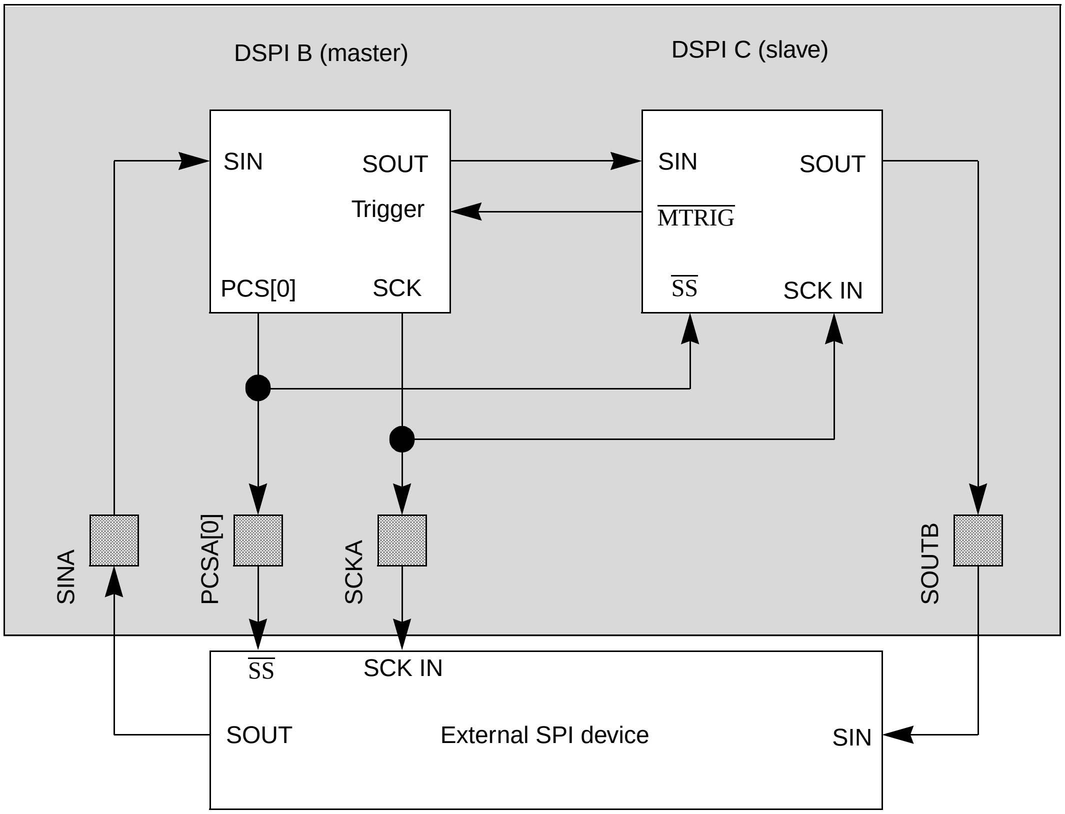


Figure 3-60. DSPI Serial Chaining

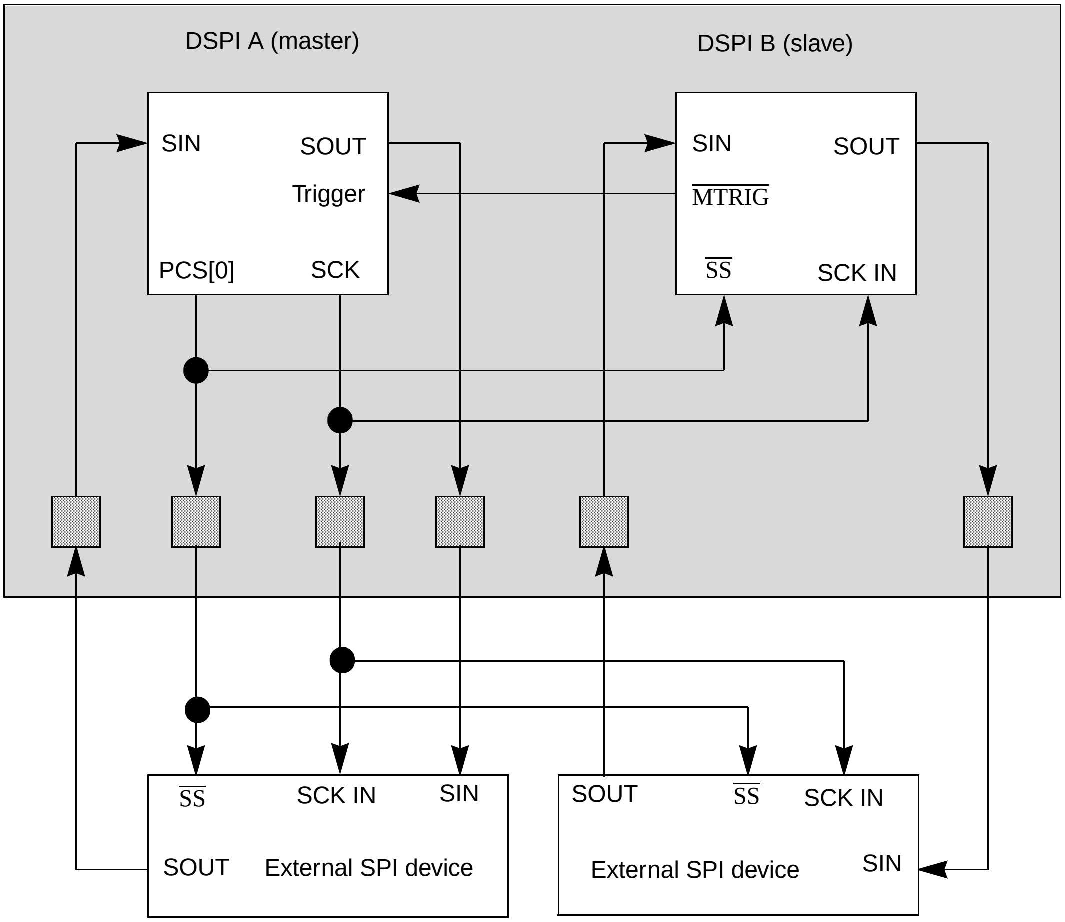


Figure 3-61. DSPI Parallel Chaining

Chapter 4
System Information Module

4.1 SIM Overview

The System Information Module (SIM) contains the calibration constants for the on-chip Temperature Sensor and a 128 bit Device ID constant that is unique to every chip. All values are read-only, and may not be changed by software.

Two 32-bit locations are reserved for temperature sensor calibration constants. A description of their usage is provided in the Temperature Sensor Chapter. The Unique Device ID is encoded and stored in four adjacent memory locations. There is no specific user information provided in the Device ID. Table 4-1 shows the organization of the constants.

4.1.1 SIM Constants

Table 4-1. SIM Constants

Offset from 0xFFFE_C000	0:15	16:31
0x00	Temperature sensor calibration constant	
0x04	Temperature sensor calibration constant	
0x08	Reserved	
0x0C		
0x10	Unique Device ID	
0x14	Unique Device ID	
0x18	Unique Device ID	
0x1C	Unique Device ID	
0x20	Reserved	
0x24		
0x28		
0x2C		
0x30		
0x34		
0x38		
0x3C		

Chapter 5
Resets

NOTE

Throughout this text the phrase “reset configuration pins” is used to refer to WKPCFG, BOOTCFG, and PLLCFG pins.

Not all packages have BOOTCFG[0]. In this case, BOOTCFG[0] is sampled as 0b0.

On reset, the MPC5676R executes from core 0 only. Core 1 is held in a reset state until user software executing on core 0 releases it. Additionally on reset, core 1 software watchdog timer is disabled.

5.1 Reset Sources and Configuration

This device supports the following system reset sources:

- Power-on Reset
- External Reset
- Loss of Lock Reset
- Loss of Clock Reset
- Two Software Watchdog Timer Resets (one per core)
- Two Core Watchdog Timer/Debug Resets (one per core)
- Dual core reset
- JTAG Reset
- Software System Reset

All input reset sources are monitored and processed by the reset controller incorporated in the SIU. A detected reset event resets internal logic and controls the assertion of the $\overline{\text{RSTOUT}}$ pin. Software may set the SIU_SRCR[SER] bit to 1 to assert the $\overline{\text{RSTOUT}}$ pin for a number of clock cycles determined by the source of the system clock (refer to Section 5.3.2, “RSTOUT”). This action is referred to as the Software External Reset, and it does not reset the device.

For all reset sources, the BOOTCFG[0:1] and PLLCFG[0:2] signals are used to determine the boot mode and configuration of the FMPLL, respectively. Table 5-1 shows the options for BOOTCFG[0:1] and Table 5-2 for PLLCFG[0:2].

Table 5-1. BOOTCFG Options

BOOTCFG[0]	BOOTCFT[1]	Meaning
0	0	Boot from internal flash memory
0	1	FlexCAN / eSCI boot
1	0	Boot from external memory (16 bit data, non-mux address)[1]
1	1	Boot from external memory (16 bit data, muxed address)[1]

[1] This mode is only available in packages that have an EBI.

Table 5-2. PLLCFG Options

Package Pins[1]		Clock Mode
PLLCFG[0]	PLLCFG[1]	
0	0	PLL Off mode
0	1	Normal mode with external reference
1	0	Normal mode with crystal reference
1	1	Reserved

[1] The PLLCFG[2] pin configures the crystal oscillator range:
PLLCFG[2] = 0, for 8 MHz to 20 MHz
PLLCFG[2] = 1, for 40 MHz

After reset, the device system clock source is derived from an internal RC oscillator (IRC) that is trimmed to a nominal 16MHz. The system clock operates at half the IRC nominal frequency. The PLL is not the source of the system clock until user software selects the PLL output instead of the default IRC oscillator by updating the SIU_SYSDIV[SYSCLKSEL] bits described in Section 3.2.1.29, or if the boot mode automatically reverts to, or is user-configured for the FlexCAN/eSCI serial boot option listed in Table 5-1. In serial boot mode, the Boot Assist Module (BAM) switches the system clock source to the PLL to provide a stable baud rate. Refer to the BAM Chapter for detailed information on boot modes.

The Reset Status Register (SIU_RSR) gives the source or sources of the last reset and indicates whether a glitch has occurred on the RESET pin. The SIU_RSR register is updated for all reset sources except JTAG reset.

All reset sources initiate execution of the BAM program with the exception of the Software External Reset and the JTAG reset. In internal or external boot mode, the BAM attempts to read from memory and decode the contents of a user defined reset configuration half word (RCHW) which serves several basic functions:

- System reset (including watchdog reset) resets the entire chip, including both cores
- CPU0 exits reset, while CPU1 is still held in reset
- CPU0 utilizes BAM (to determine boot block or serial boot)
- CPU0 performs normal chip initialization. As part of this initialization, CPU0 writes to CPU1 reset vector register
- CPU0 enables CPU1 & CPU1 watchdog

- CPU1 exits reset to CPU1 reset vector
- Allows lower power operating modes, and single core emulation
- determine if Flash memory is programmed or erased
- enable or disable the watchdog timer
- if booting from external memory, set multiplexed or non-multiplexed bus mode

Details of the BAM code options and functionality, including serial boot mode operation and user code start address location are provided in the BAM chapter of this document.

5.2 Reset Vector

The reset vector for the MPC5676R is 0xFFFF_FFFC, fetched from the SIU_RSTVEC0 register.

5.3 Reset Pins

5.3.1 RESET

The RESET pin is an active low input. The RESET pin must be asserted by an external device during a power-on or whenever an external reset is required. The internal reset signal asserts only if the RESET pin asserts for 10 clock cycles. Assertion of the RESET pin while the reset state machine is already processing a reset causes the reset cycle to start over. The RESET pin has a glitch detector which detects spikes greater than 2 clocks in duration that fall below the switch point of its input buffer. The switch point lies between the maximum VIL and minimum VIH specifications for the RESET pin. Figure 5-1 and Figure 5-2 show logic flows of the reset state machine on assertion of RESET.

5.3.2 RSTOUT

The RSTOUT pin is an active low output that uses a push/pull configuration. The RSTOUT pin is driven low by the MCU for all internal and external reset sources.

The RSTOUT pin is asserted for a number of clock cycles that depends on the reset source, as shown in Table 5-3, plus 4 cycles for sampling of the configuration pins. The RSTOUT pin can also be asserted by a write to the SER bit of the System Reset Control Register (SIU_SRCR); however no system reset occurs under this circumstance. The duration of RSTOUT assertion caused by setting the SER bit depends on whether the clock source is the IRC or not, as given in Table 5-3.

Table 5-3. Clock Cycles for Different Reset Sources

Source[1]	Description	Number of clock cycles[2]
PORS	Power On Reset	2400
ERS	External Reset (RESET pin)	2410
LLRS	Loss of Lock Reset	2420
LCRS	Loss of Clock Reset	2430
WDRS0	Watchdog Timer or Debug Reset from Core 0	2440

Table 5-3. Clock Cycles for Different Reset Sources (continued)

Source[1]	Description	Number of clock cycles[2]
SWTRS0	Software Watchdog Timer 0 Reset	2460
SWTRS1	Software Watchdog Timer 1Reset	2470
WDRS1	Watchdog Timer or Debug Reset from Core 1	2480
CPURS	Simultaneous Core 0 and Core 1 Reset	2490
STCURS	Self Test Control Unit Reset	2500
SSRS	Software System Reset	2510
SERF[3]	Software External Reset	2520 (IRC) / 17200 (other)

[1] Bits of the SIU_RSR register.

[2] The value of 2450 is reserved.

[3] $\overline{\text{RSTOUT}}$ duration depends on the current clock source: if IRC, duration is 2520; for all others, duration is 17200.

5.4 Reset Source Descriptions

For the following reset source descriptions refer to the reset flow diagrams in Figure 5-1 and Figure 5-2. Figure 5-1 shows the reset flow for validating the assertion of the $\overline{\text{RESET}}$ pin. Figure 5-2 shows the internal processing of reset for all reset sources.

The source of each reset is indicated by a different bit in the SIU_RSR register. In certain cases, more than one bit may be set simultaneously. Refer to Table 3-8 in Chapter 3, “System Integration Units (SIU, SIU_B)” for details of the conditions that can cause this to happen.

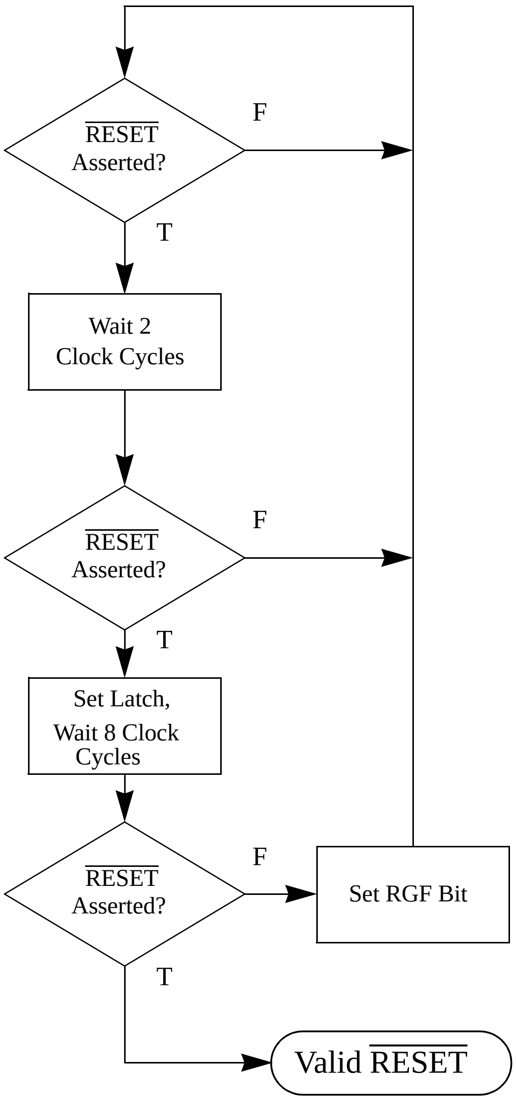
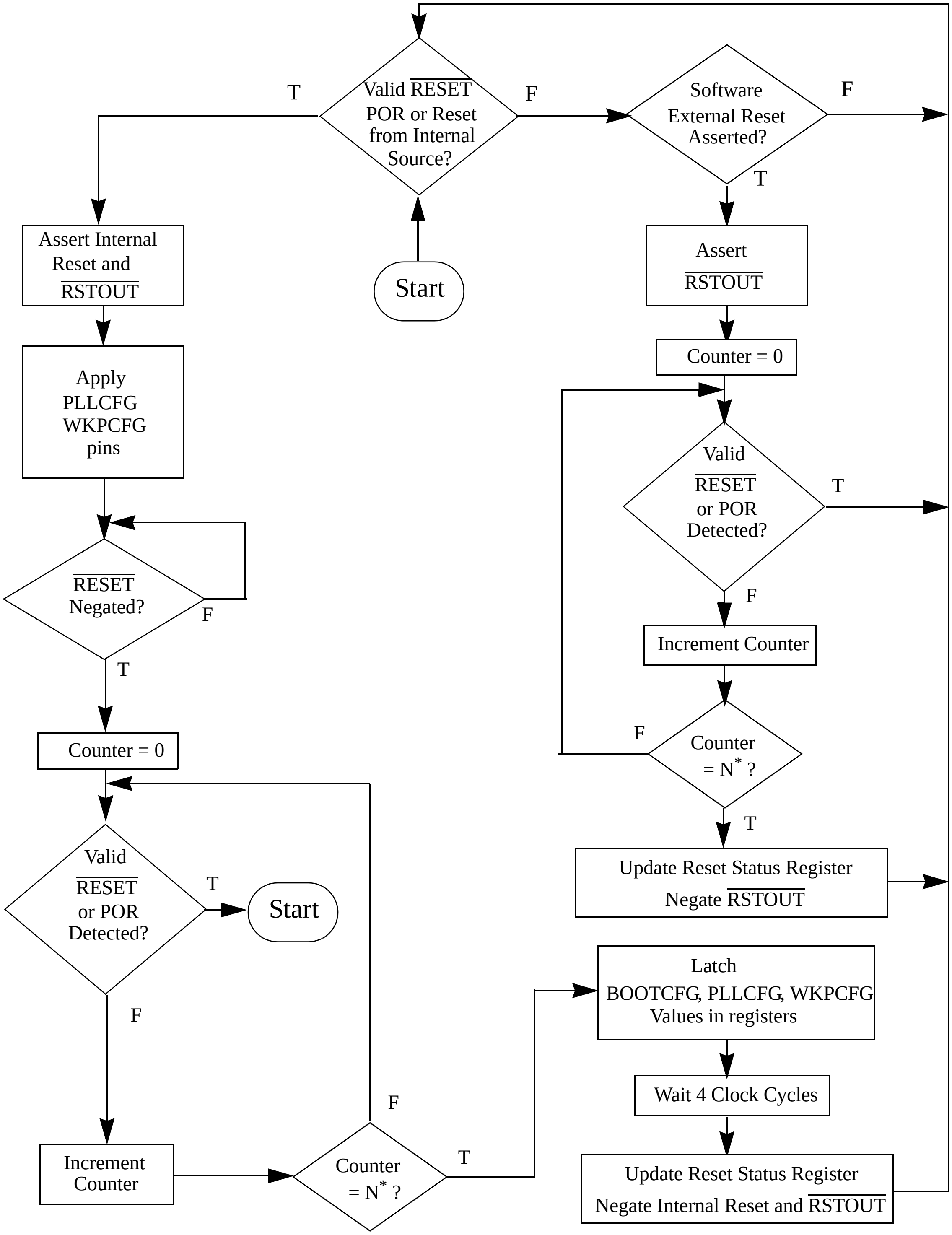


Figure 5-1. External Reset Validation Flow Diagram



NOTES:
* The clock count is dependent on the clock and reset sources (refer to Section 5.3.2, “RSTOUT”).

Figure 5-2. Internal Reset Flow Diagram

5.4.1 Power-on Reset (POR)

The internal power-on reset signal is asserted when either the supply voltages, nominally 3.3V (VDD33) and 1.2V (VDD) or the RESET supply (VDDEH1) fall below defined threshold voltages. See the device data sheet for the specifications of these thresholds. The PMC provides additional software configurable registers for masking of power on reset assertions based on these and additional supplies. See the PMC chapter for details. Although assertion of the power-on reset signal causes reset, the RESET pin must be asserted during a power-on reset to guarantee proper operation of the MCU.

The system clock source is the internal RC oscillator (IRC) on power on reset.

The signal on the WKPCFG pin determines whether weak pull up or pull down devices are enabled after reset on the eTPU and eMIOS pins. The WKPCFG pin is applied on the assertion of the internal reset signal (assertion of RSTOUT). See Section 5.6.3, "WKPCFG operation during reset", for more information.

After the power supplies stabilize and the internal POR signals negate, the reset controller waits for a predetermined number of clock cycles (refer to Section 5.3.2, "RSTOUT") before negating the RSTOUT pin. The reset configuration pins (BOOTCFG, PLLCFG, WKPCFG) are sampled 4 clock cycles before the negation of RSTOUT. The BOOTCFG and WKPCFG values are applied directly to the associated bit fields in the SIU_RSR, while the PLLCFG values are used to update the PLLSEL, PLLREF and MODE bit fields in the FMPLL SYNSR register. The application of the PLLCFG values are necessary to support the BAM serial boot mode.

In addition, the PORS and ERS bits are set in the SIU_RSR register.

5.4.2 External Reset

When the reset controller detects assertion of the RESET pin, the internal reset signal and RSTOUT pin are asserted. The values on the WKPCFG and PLLCFG pins are applied at the assertion of the internal reset signal (assertion of RSTOUT). Once the RESET pin is negated and the system clock switches to the IRC source, the reset controller waits for a predetermined number of clock cycles (refer to Section 5.3.2, "RSTOUT"). Once the clock count finishes, the reset configuration pins are latched. The reset controller then waits 4 clock cycles before it negates RSTOUT and updates the configuration bit-fields in the SIU_RSR register. In addition, the ERS bit is set in the SIU_RSR register.

5.4.3 Loss of Lock

A Loss of Lock Reset occurs when the FMPLL loses lock and the Loss of Lock Reset Enable (LOLRE) bit in the FMPLL Synthesizer Control Register (ESYNCR2) is set. The internal reset signal and RSTOUT pin are asserted. The values on the WKPCFG and PLLCFG pins are applied at the assertion of the internal reset signal (assertion of RSTOUT). The system clock source switches to the IRC source and the reset controller waits for a predetermined number of clock cycles (refer to Section 5.3.2, "RSTOUT"). Once the clock count finishes, all configuration pins are latched. The reset controller then waits 4 clock cycles before it negates RSTOUT and updates the configuration bit-fields in the SIU_RSR register. In addition, the LLRS bit is set in the SIU_RSR register.

Refer to Section 8.4.3.1, “PLL Lock Detection,” in the FMPLL chapter for more information on loss of lock.

5.4.4 Loss of Clock

A Loss of Clock Reset occurs on a failure of either the reference signal or FMPLL output, and the Loss of Clock Reset Enable (LOCRE) bit in the ESYNCR2 is set. The internal reset signal and $\overline{\text{RSTOUT}}$ pin are asserted. The values on the WKPCFG and PLLCFG pins are applied at the assertion of the internal reset signal (assertion of $\overline{\text{RSTOUT}}$). The system clock source switches to the IRC source and the reset controller waits for a predetermined number of clock cycles (refer to Section 5.3.2, “$\overline{\text{RSTOUT}}$”). Once the clock count finishes, all configuration pins are latched. The reset controller then waits 4 clock cycles before it negates $\overline{\text{RSTOUT}}$ and updates the configuration bit-fields in the SIU_RSR register. In addition, the LCRS bit is set in the SIU_RSR register.

Refer to Section 8.4.3.2, “Loss-of-Clock Detection,” in the FMPLL chapter for more information on loss of clock.

5.4.5 Core Watchdog Timer/Debug Reset

Each core has its own integrated Watchdog Timer/Debug Reset function. After reset, only core 0 is enabled, and therefore only core 0 Watchdog Timer/Debug Reset function is operational.

A Core Watchdog Timer Reset occurs when the e200z7 core watchdog timer is enabled and a time-out occurs. Software may enable the watchdog timeout by configuring the WRC field in the core’s Timer Control Register (SPR 340). The WDRSn (n = 0 for core 0, n = 1 for core 1) bit in the SIU_RSR is set when a Core *n* Watchdog Timer Reset occurs. It is also set when a debug reset command is issued from a debug tool.

The effect of a Watchdog Timer or Debug Reset request is the same for the reset controller. The internal reset signal and $\overline{\text{RSTOUT}}$ pin are asserted. The values on the WKPCFG and PLLCFG pins are applied at the assertion of the internal reset signal (assertion of $\overline{\text{RSTOUT}}$). Once the Watchdog Timer/Debug reset request is negated and the system clock switches to the IRC source, the reset controller waits for a predetermined number of clock cycles (refer to Section 5.3.2, “$\overline{\text{RSTOUT}}$”). Once the clock count finishes, the reset configuration pins are latched. The reset controller then waits 4 clock cycles before it negates $\overline{\text{RSTOUT}}$ and updates the configuration bit-fields in the SIU_RSR register. In addition, the WDRS bit is set in the SIU_RSR register.

Refer to the *e200z7 Core Reference Manual* for more information on the core watchdog timer and debug operation.

5.4.6 Software Watchdog Timer Reset

There are two Software Watchdog Timers, SWT0 and SWT1. Their reset functionality is identical, except that each SWT sets a different bit in the SIU_RSR when a SWT reset occurs.

After reset, only SWT0 is enabled, and clocked from the IRC source. SWT0 clock is stopped when debug mode is entered, allowing software debug from reset, without interference from the SWT0.

Application software may enable SWT1 if necessary.

On a SWT0 or SWT1 reset event, the system clock switches to the IRC source. The reset controller then executes the flow for Reset from Internal Source shown in Figure 5-2, and finally sets the corresponding SWTRS0 or SWTRS1 bit in the SIU_RSR register when $\overline{\text{RSTOUT}}$ negates.

Refer to Chapter 33, “Software Watchdog Timers (SWT_A, SWT_B)” for more information on the Software Watchdog Timer.

5.4.7 Dual Core Reset

Core 0 and Core 1 Reset Vector Registers described in Section 3.2.1.32 allow software executing on either core to place either core in reset by setting the RST bit in the appropriate Reset Vector Register.

Any attempt to reset both cores by setting both RST bits will result in an immediate internal reset.

On this reset event, the system clock switches to the IRC source. The reset controller then executes the flow for Reset from Internal Source as shown in Figure 5-2, and finally sets the CPURS bit in the SIU_RSR register when $\overline{\text{RSTOUT}}$ negates.

5.4.8 JTAG Reset

A system reset occurs when JTAG is enabled and either the EXTEST, CLAMP, or HIGHZ instructions are executed by the JTAG controller. The internal reset signal is asserted. The state of the $\overline{\text{RSTOUT}}$ pin is determined by the JTAG instruction. The values on the WKPCFG and PLLCFG pins are applied at the assertion of the internal reset signal. Once the JTAG reset request is negated and the system clock switches to the IRC source, the reset controller waits for a predetermined number of clock cycles (refer to Section 5.3.2, “$\overline{\text{RSTOUT}}$”). Once the clock count finishes the reset configuration pins are latched, and the configuration bit-fields are updated in the SIU_RSR. The reset status bits in the SIU_RSR are unaffected.

5.4.9 Software System Reset

A Software System Reset occurs when software sets the SSR bit to 1 in the System Reset Control Register (SIU_SRCR), causing assertion of the internal reset signal and the $\overline{\text{RSTOUT}}$ pin. At the same time the WKPCFG and PLLCFG values are applied. The SSR bit is automatically cleared, the system clock switches to the IRC source and the reset controller waits for a predetermined number of clock cycles (refer to Section 5.3.2, “$\overline{\text{RSTOUT}}$”). Once the clock count finishes, the reset configuration pins are latched. The reset controller then waits 4 clock cycles before it negates $\overline{\text{RSTOUT}}$ and updates the configuration bit-fields in the SIU_RSR register. In addition, the SSRS bit is set in the SIU_RSR register.

5.4.10 Software External Reset

A write of one to the SER bit in the SIU_SRCR causes the external $\overline{\text{RSTOUT}}$ pin to be asserted for a predetermined number of clock cycles (refer to Section 5.3.2, “$\overline{\text{RSTOUT}}$”). The SER bit automatically clears after the clock counting expires. A Software External Reset does not cause a reset of the MCU, the BAM program is not executed, the reset configuration pins are not sampled. The SERF bit in the SIU_RSR is set, but no other status bits are affected. The SERF bit in the SIU_RSR is not automatically cleared after the $\overline{\text{RSTOUT}}$ clock cycles expire, and remains set until cleared by software or another reset besides the Software External Reset occurs.

For a Software External Reset, the e200z7 core will continue to execute instructions, timers that are enabled will continue to operate, and interrupt requests will continue to be processed. It is the responsibility of the application to ensure devices connected to $\overline{\text{RSTOUT}}$ are not accessed during a Software External Reset, and to determine how to manage MCU resources when using the Software External Reset.

5.5 Reset Registers in the SIU

The System Integration Unit (SIU) of the MPC5676R includes two registers, SIU_RSR and SIU_SRCR, that affect its reset behavior. The BAM executes a boot sequence based on the values latched in the SIU_RSR[BOOTCFG] field. Application code may force a reset by setting SIU_SRCR[SSR] to 1. The SIU contains two additional registers, SIU_RSTVEC0 and SIU_RSTVEC1 to allow software to reset core 0 or core 1.

See Chapter 3, “System Integration Units (SIU, SIU_B)” and the BAM chapter for detailed descriptions of these registers and their usage.

5.6 Reset Configuration

5.6.1 Reset Configuration Half Word (RCHW)

5.6.1.1 RCHW Overview

The Reset Configuration Half Word (RCHW) defines boot options and must be programmed in a choice of predefined locations in internal flash or in a single location at the beginning of the external memory. The word at the word address boundary after the RCHW must be programmed with the user application’s starting address. The BAM passes control to the user application at this starting address.

On every reset except the Software External Reset (SER), in internal or external boot modes, the BAM attempts to read the RCHW from internal or external memory respectively. The locations for the RCHW are given in Table 5-4. For internal boot, the predefined locations are searched in the order given in the table. If a valid RCHW is not found in internal boot mode or in external boot mode, the BAM initiates the serial boot mode, which causes the system clock to switch from the IRC to the FMPLL source. Note that in serial boot mode, a user defined start address must still be supplied as part of the download protocol. Refer to the BAM Chapter for complete details.

Table 5-4. RCHW Location for Internal and External Boot Modes

Boot Mode	Address
External	0x0000_0000
Internal	0x0002_0000 0x0003_0000 0x0000_0000 0x0000_4000 0x0001_0000 0x0001_C000

5.6.1.2 RCHW Structure

In internal or external boot mode, the RCHW must reside in the first 16 bits of memory.

BOOT_BLOCK_ADDRESS + 0x0000_0000

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
				SWT	WTE	PS0	VLE	0	1	0	1	1	0	1	0
								Boot Identifier = 0x5A							

Figure 5-3. Reset Configuration Half Word

The following table describes the fields in the reset configuration half word.

Table 5-5. RCHW Bit Field Descriptions

Field	Description
0–3	Reserved. These bit values are ignored when the halfword is read. Program to 0 for future compatibility.
SWT	Software watchdog timer enable. This bit determines if the software watchdog timer for core 0 is enabled after passing control to the user application code. 0 Disable software watchdog timer 1 Software watchdog timer maintains its default state of enabled after reset. The timeout period is 392400 clock cycles of the IRC oscillator.
WTE	Core 0 watchdog timer enable. This bit determines if the core watchdog timer is enabled.after passing control to the user application code. 0 Disable core watchdog timer 1 Enable core watchdog timer after reset. The BAM programs the timeout period to $2.5*2^{18}$ cycles of core 0 clock[1].
PS0	Port size. Used for external memory boot modes only. Defines the data width of the external memory selected by CAL_CS0. After reset, the BAM sets the 16-bit port size to support fetching the RCHW from either 16- or 32-bit external memories. Then the BAM reconfigures the port size, depending on this bit. 0 32-bit CAL_CS0 port size 1 16-bit CAL_CS0 port size **Note:** Do not clear this bit if the device only has a 16-bit data bus.

Table 5-5. RCHW Bit Field Descriptions (continued)

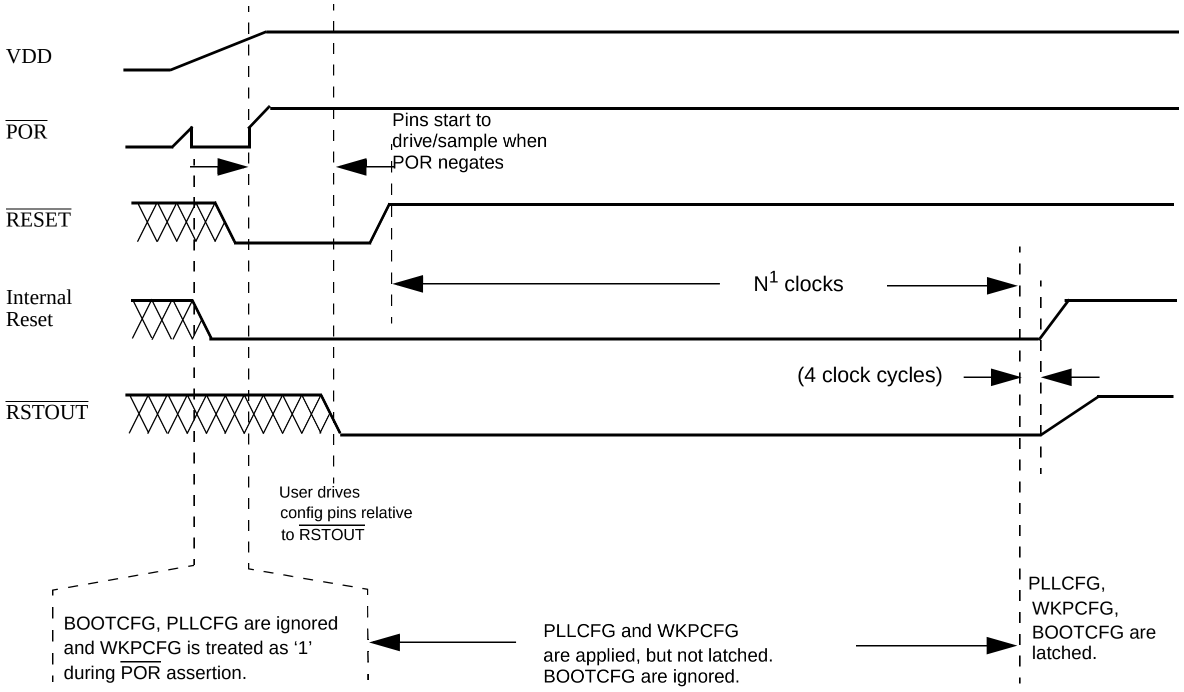
Field	Description
VLE	VLE Code Indicator. This bit configures the MMU entries 1-3 coded as Classic Book E instructions or as VLE instructions. 0 User code executes as classic Book E code 1 User code executes as Freescale VLE code
BOOTID	Boot identifier. • Indicates to the BAM program that this location contains a RCHW

[1] After reset, core 0 clock is connected to the IRC operating at a nominal 16MHz.

Refer to the BAM chapter for details of how the RCHW is used to configure the boot sequence of the MPC5676R.

5.6.2 Reset Configuration Timing

The timing diagram in Figure 5-4 shows the relationship between reset signals and the configuration pins for a power-on reset. The timing diagram is valid for internal and external resets assuming that VDD and VDD33 are within valid operating ranges. The PLLCFG and WKPCFG signals are applied at the assertion of the internal reset signal (assertion of $\overline{\text{RSTOUT}}$). The BOOTCFG pins are ignored during reset and a default value used instead, as described in Section 5.6.4. The values of the configuration pins are latched 4 clock cycles before the negation of the $\overline{\text{RSTOUT}}$ pin and stored in the Reset Status Register and SYNSR register.



NOTE:
1. The clock count is dependent on the clock and reset sources (refer to Section 5.3.2, “RSTOUT”).

Figure 5-4. Reset Configuration Timing

5.6.3 WKPCFG operation during reset

The signal on the WKPCFG pin determines whether specified eTPU and eMIOS pins are connected to weak pull up or weak pull down devices at reset (see Chapter 2, “Signal Descriptions” for the eTPU and eMIOS pins that are affected by WKPCFG). For all reset sources except the Software External Reset, the WKPCFG pin is applied at the assertion of the internal reset signal (assertion of $\overline{\text{RSTOUT}}$). If the WKPCFG signal is logic high at this time, pull up devices will be enabled on the eTPU and eMIOS pins. If the WKPCFG signal is logic low at the assertion of the internal reset signal, pull down devices will be enabled on those pins. The value on WKPCFG must be held constant during reset to avoid oscillations on the eTPU and eMIOS pins caused by switching pull up/down states. The final value of WKPCFG is latched 4 clock cycles before the negation of $\overline{\text{RSTOUT}}$. After reset, software may modify the weak pull up/down selection for all I/O pins through the PCR registers in the SIU.

5.6.4 BOOTCFG operation during reset

During internal reset assertion, the BOOTCFG values on the pins are not used, and instead a value of 0b00 is applied to internal logic that qualifies the censorship word to enable or disable the internal flash memory and Nexus/JTAG interface. Refer to the BAM chapter for details of the values for BOOTCFG and censorship word, and the method to enable the Nexus/JTAG interface on a censored device. The BOOTCFG pin values are latched into the SIU_RSR register 4 clock cycles before $\overline{\text{RSTOUT}}$ negates. After $\overline{\text{RSTOUT}}$ negates, the BOOTCFG pins do not affect the boot mode and application code may reassign the pins to an available alternative function.

5.6.5 PLLCFG operation during reset

During internal reset assertion, the PLLCFG values are directly connected to the FMPLL block. Any change in the state of the PLLCFG pins at this time will cause the FMPLL to change operating mode according to the options defined in Table 5-2. Encoded values of the PLLCFG pins are latched into the SYNSR register 4 clock cycles before $\overline{\text{RSTOUT}}$ negates. After $\overline{\text{RSTOUT}}$ negates, the PLLCFG pins do not affect FMPLL operation and application code may reassign the pins to an available alternative function.

Chapter 6
Boot Assist Module (BAM)

6.1 Overview

The Boot Assist Module (BAM) is a 4-KB block of read-only memory, containing the boot program code for the MCU. The BAM program supports the following boot modes:

- Boot from internal flash
- Serial boot via SCI or CAN interface with optional baud-rate detection
- Boot from a memory connected to the MCU development bus (EBI[1]) with multiplexed or separate address and data lines

On a dual-core MCU like the MPC5676R only Core0 is enabled out of reset, therefore the BAM program is executed by Core0 just after the MCU resets. Depending on the boot mode, the program initializes the appropriate minimum MCU resources to start user application code.

6.2 Features

- Initial MCU core MMU setup with minimum address translation for all internal MCU resources
- MMU configuration to boot user application, compiled as Classic PowerPC Book E code or as Freescale VLE code
- Passes control to user application code in the internal flash or external memory device
- Automatic switch to Serial Boot mode if internal or external flash is blank or invalid
- Serial boot by loading user program via CAN bus or eSCI to the internal SRAM
  - — User programmable 64 bit password protection
  - — Optional automatic detection of the host SCI or CAN speed
- Boot from an external memory device, connected to the EBI
  - — Option to boot from 16 bit memory device with separate data and address lines
  - — Option to boot from 32/16 bit memory device with multiplexed data and address lines
- Controls MCU core Watchdog Timer and/or the Software Watchdog Timer (SWT)

6.3 Modes of Operation

6.3.1 Normal Mode

The BAM program is executed immediately following the negation of $\overline{\text{RESET}}$.

6.3.2 Debug Mode

The BAM program is not executed when the MCU exits reset in OnCE debug mode. Before accessing the MCU resources, use the development tool to initialize the MCU.

---

[1] EBI not available on all packages.

6.3.3 Internal Boot Mode

Use this operating mode to boot from internal flash memory, which holds all of the code and configuration data.

6.3.4 Serial Boot Mode

Use this mode to load a user program into internal SRAM, using the eSCI or CAN serial interface, then to execute that program. The program can then control

- the download of data, and
- erasing and programming of the internal or external flash memory.

6.3.5 Development Bus Boot Mode

This boot mode is intended for packaged versions of the device that have the development bus pinned out. The development bus (EBI) boot mode can be used for application code development.

6.4 Memory Map

The BAM occupies 16 KB of memory space, 0xFFFF_C000 to 0xFFFF_FFFF. The actual code size of the BAM program is less than 4 KB and starts at 0xFFFF_F000, repeating itself every 4 KB in the BAM address space. The CPU starts the BAM program execution at its reset vector from address 0xFFFF_FFFC.

The BAM exits to the user code at 0xFFFF_FFF8. The last instruction the BAM executes is a BLR. The link register is pre-loaded with the user application start address. The value of the start address depends on the boot mode:

- Booting from internal or external flash — 32-bit word following a valid RCHW holds the start address value
- Serial boot — set according to the serial boot protocol

Table 6-1 shows the BAM address map.

Table 6-1. BAM Memory Map

Address	Description
0xFFFF_C000 – 0xFFFF_EFFF	BAM program mirrored
0xFFFF_F000 – 0xFFFF_FFFF	BAM program
0xFFFF_FFFC	MCU reset vector
0xFFFF_FFF8	BAM last executed instruction

6.5 Functional Description

6.5.1 BAM Program Flow Chart

Figure 6-1 shows the BAM program flow.

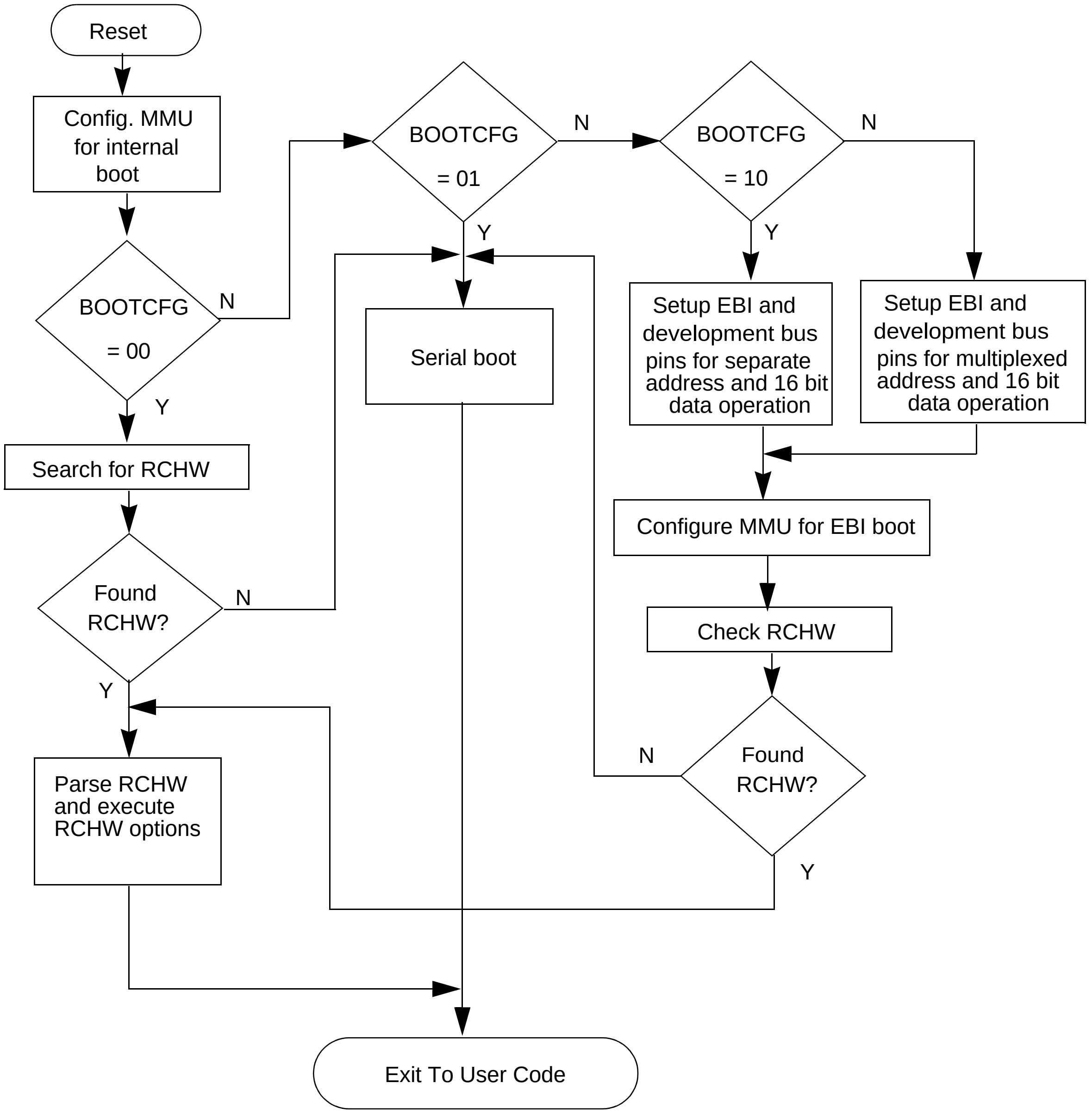


Figure 6-1. BAM program Flow Chart

6.5.2 BAM Program Operation

After negation of $\overline{\text{RESET}}$, the MCU core accesses the BAM before user code starts. First, the BAM program configures the core MMU to allow access to all MCU internal resources, according to Table 6-2. This MMU setup remains untouched through internal flash boot mode.

Table 6-2. MMU Configuration for Internal Flash Boot

TLB Entry	Region	Logical Base Address	Physical Base Address	Size	Attributes
0	Peripheral Bridge B and BAM	0xFFF0_0000	0xFFF0_0000	1 MB	Guarded Big endian Global PID Cache Inhibit
1	Internal Flash	0x0000_0000	0x0000_0000	16 MB	Not guarded Big endian Global PID
2	EBI	0x2000_0000	0x0000_0000	16 MB	Not guarded Big endian Global PID
3	Internal SRAM	0x4000_0000	0x4000_0000	256 KB	Not guarded Big endian Global PID Cache Inhibit
4	Peripheral Bridge A	0xC3F0_0000	0xC3F0_0000	1 MB	Guarded Big endian Global PID Cache Inhibit

The MMU regions are mapped with logical addresses the same as physical addresses except for the external bus interface (EBI). The logical EBI address space is mapped to physical address space of the internal flash memory. This allows code, written to run from external memory, to execute from internal flash.

After the MMU configuration, the BAM program starts the boot sequence depending on the SIU_RSR[BOOTCFG] value, as shown in Table 6-3.

Depending on the values stored in the censorship word and serial boot control word in the shadow row of the internal flash memory, the internal flash memory can be enabled or disabled, the Nexus port can be enabled or disabled, the password received in the serial boot mode is compared with the fixed public password or compared to a user programmable password in the internal flash memory.

Table 6-3. Boot Modes

Boot Mode Name	BOOTCFG	Censorship Control 0x00FF_FDE0	Serial Boot Control 0x00FF_FDE2	Internal Flash State	Nexus State	Serial Password
Internal - Censored	00	any value except 0x55AA	Don't care	Enabled	Disabled	Flash
Internal - Public		0x55AA		Enabled	Enabled	Public

Table 6-3. Boot Modes

Boot Mode Name	BOOTCFG	Censorship Control 0x00FF_FDE0	Serial Boot Control 0x00FF_FDE2	Internal Flash State	Nexus State	Serial Password
Serial - Flash Password	01	Don't care	0x55AA	Enabled	Disabled	Flash
Serial - Public Password			Any value except 0x55AA	Disabled	Enabled	Public
Development Bus separate address and data, flash is censored	10	0x55AA	Don't care	Enabled		Public
Development Bus separate address and data, flash is NOT censored		Any value except 0x55AA		Disabled		
Development Bus multiplexed address and data, flash is censored	11	0x55AA	Don't care	Enabled	Enabled	
Development Bus multiplexed address and data, flash is NOT censored		Any value except 0x55AA		Disabled		

The censorship word is a 32 bit word of data stored in the shadow row of internal flash memory. This memory location is read by hardware as part of the boot process and is used in conjunction with the BOOTCFG pins to enable and disable the internal flash memory and the Nexus interface. The censorship word consists of two fields: censorship control and serial boot control. The censorship word is programmed during manufacturing to be 0x55AA_55AA. This results in a device that is not censored and uses a flash-based password for serial boot mode.

Address: 0x00FF_FDE0 Value: 0x55AA

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binary value	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0
Hex value	5				5				A				A			

Censorship control field–default value configures the device as uncensored.

Address: 0x00FF_FDE2 Value: 0x55AA

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binary value	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0
Hex value	5				5				A				A			

Serial boot control field–default value reads a password from internal flash.

Figure 6-2. Censorship Word

The BAM program uses SIU_CCR[DISNEX], which reflects the Nexus module state, to determine if the serial password received in serial boot mode should be compared to:

- Public password (0xFEED_FACE_CAFE_BEEF) if DISNEX is cleared

- Flash password (64-bit value, stored in the shadow block of internal flash at address 0x00FF_FDD8) if DISNEX is set

NOTE

> Regardless of the boot mode used, program a valid serial password. This allows you to rescue the device using serial boot mode if the flash content is corrupted.

Flash Password@ 0x00FF_FDD8 – 0x00FF_FDDF

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	1	1	1	1	0	1	1	1	0	1	1	0	1
Serial Boot Password (0x00FF_FDD8) - 0xFEED (Factory Default)															

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	1	1	1	1	0	1	0	1	1	0	0	1	1	1	0
Serial Boot Password (0x00FF_FDDA) - 0xFACE (Factory Default)															

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
1	1	0	0	1	0	1	0	1	1	1	1	1	1	1	0
Serial Boot Password (0x00FF_FDDC) - 0xCAFE (Factory Default)															

48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
1	0	1	1	1	1	1	0	1	1	1	0	1	1	1	1
Serial Boot Password (0x00FF_FDDE) - 0xBEEF (Factory Default)															

Figure 6-3. Serial Boot Flash Password

6.5.3 Reset Configuration Half Word (RCHW)

The Reset Configuration Half Word defines boot options and must be programmed to predefined locations in the internal flash or at the beginning of the external flash device. The 32 bit word after the RCHW must be programmed with the user application's starting address. The BAM passes control to the user application at this starting address.

Table 6-4 provides possible RCHW locations in the internal flash. When booting from the external flash device, the RCHW must reside in the first 16 bit half word of the flash.

BOOT_BLOCK_ADDRESS

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
				SWT	WTE	PS0	VLE	0	1	0	1	1	0	1	0
								Boot Identifier = 0x5A							

Figure 6-4. Reset Configuration Half Word

Table 6-4. RCHW Field Descriptions

Field	Description
0–3	Reserved. These bit values are ignored when the halfword is read. Program to 0 for future compatibility.
SWT	Software watchdog timer enable. This bit determines if the software watchdog timer is enabled after passing control to the user application code. 0 Disable software watchdog timer 1 Enable software watchdog timer after reset. The timeout period is 392400 cycles of the Internal RC Oscillator.
WTE	MCU core watchdog timer enable. This bit determines if the core software watchdog timer is enabled.after passing control to the user application code. 0 Disable core software watchdog timer 1 Enable core watchdog timer after reset. The timeout period is $2.5*2^{18}$ cycles of the CPU clock[1].
PS0	Port size. Defines the width of the data bus connected to the memory on D_CS0. After system reset, the BAM changes D_CS0 to a 16-bit port to fetch the RCHW from either 16- or 32-bit external memories. Then the BAM reconfigures the EBI as a 16- or 32-bit port, depending on this bit. 0 32-bit D_CS0 port size 1 16-bit D_CS0 port size **Note:** Used in development bus boot modes only (i.e. 496- and 516-pin packages). Do not clear this bit if the device only has a 16-bit data bus.
VLE	VLE Code Indicator. This bit configures the MMU entries 1-3 coded as Classic Book E instructions or as Freescale VLE instructions. 0 User code executes as classic Book E code 1 User code executes as Freescale VLE code
BOOTID	Boot identifier. This field serves two functions: • Indicates which block in flash memory contains the boot program • Indicates if the flash memory is programmed (BOOTID=0x5A) or invalid

[1] Upon exiting reset, the CPU clock is connected to the Internal RC Oscillator at 16 MHz.

6.5.3.1 Application Start Address Register

Application Start Address Register (Figure 6-5) is the 32-bit word after the RCHW in the application code memory device (internal or external flash). The BAM uses the value from this location as a start address of the user application to pass control to.

BOOT_BLOCK_ADDRESS + 0x0000_0004

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
A16	A17	A18	A19	A20	A21	A22	A23	A24	A25	A26	A27	A28	A29	A30	A31

Figure 6-5. Application Start Address Register

6.5.4 Internal Boot Mode

Upon reset exit, when the BAM program starts running, the system clock is by default sourced by the Internal RC Oscillator (IRC). If internal boot mode was configured in the BOOTCFG pins, the BAM program executes the following tasks:

1. Configures a machine check exception handler, since it will be accessing flash memory locations that may be corrupted and cause a bus error
2. Searches for a valid RCHW in six pre-defined locations, as shown in Table 6-5

Table 6-5. Possible RCHW Locations in the Internal Flash

Priority	Address
0	0x0002_0000
1	0x0003_0000
2	0x0000_0000
3	0x0000_4000
4	0x0001_0000
5	0x0001_C000

3. If a valid RCHW is not found, the BAM program switches to serial boot mode
4. If the BAM program finds a valid RCHW:
   - — BOOT_BLOCK_ADDRESS is the address (from Table 6-5) of the valid RCHW
   - — The BAM program fetches the application start address from BOOT_BLOCK_ADDRESS + 0x4
   - — The BAM branches to this start address (shown in Figure 6-5)
   - — The MMU TLB entries 1,2,3 are programmed with VLE attribute if the RCHW[VLE] = 1
   - — The watchdog timers are programmed as shown in

Table 6-6. Watch Dog Timeout Settings

Core WD		SWT	
Clock Source	**Condition**	**Clock Source**	**Condition**
IRC 16 MHz	Enabled if RCHW[WTE] = 1 Timeout = 2.5 x 2^{18} cycles	IRC 16 MHz	Enabled if RCHW[SWT] = 1 Timeout = 392400 cycles

6.5.5 Serial Boot Mode

When serial boot mode is detected, either because of the BOOTCFG configuration or because a valid RCHW word was not found in any of the expected locations in the Flash, the BAM program does the following:

1. Uses message buffers in CAN_A for program stack and variables
2. Polls the FMPLL_SYNCR[LOCK] bit until it indicates that the PLL has locked
3. Switches the system clock source to the PLL, by setting SIU_SYSDIV[SYSCLKSEL] = 0x2
4. Disables the core watchdog timer and sets the SWT timeout period to 2.5 x 2^{27} cycles
5. Waits for the host to send a 64-bit password by either the CAN or SCI
6. Receives start address, size of download code in bytes, and VLE bit
7. Receives the application code data
8. Disables the SWT and enables the core watchdog timer with a timeout period of 2.5 x 2^{28} CPU clock cycles
9. Branches to the loaded code at the start address

The MMU setup depends on how the BAM entered the serial boot mode. If it attempted to boot from the development bus but switched to serial boot because it could not find a valid RCHW in Flash, the MMU is set up as for that mode (see Table 6-12). Otherwise the MMU setup matches the Table 6-2.

The serial boot mode can run in two modes, depending on the state of the EVTO pin during reset (the SIU_RSR[ABR] bit reflects the inverted state of the EVTO pin):

- Standard serial boot mode — fixed baud rates derived from the MCU system frequency. EVTO pin is kept high during reset.
- Baud Rate Detection serial boot mode — allows communication with adaptable speed, based on measured baud rate of the host transmission. EVTO pin has to be driven low for this mode.

The EVTO pin is pulled up by an internal pull-up. Actively driven low EVTO pin selects the Baud Rate Detection mode.

When the Fixed Baud Rate mode is selected, the BAM program configures the RXD_A pin to be the input of the eSCI_A module, CNRX_A pin as an input and CNTX_A as an output of the CAN_A module. When Baud Rate Detection Mode is selected, the BAM program configures RXD_A and CNRX_A pins as GPI inputs for polling their state by the CPU.

The SCI and CAN controllers pins configuration summary is shown in the Table 6-7.

Table 6-7. CAN/eSCI Pins Configuration for CAN/eSCI Fixed Baud Rate Boot Modes

Pins	Reset Function	Initial Serial Boot Mode		Serial Boot Mode after a valid CAN message received		Serial Boot Mode after a valid eSCI message received	
		Function	Pad Configuration	Function	Pad Configuration	Function	Pad Configuration
CNTX_A	GPIO	CNTX_A	Push/Pull output, with medium slew rate	CNTX_A	Push/Pull output, with medium slew rate	GPIO	—

Table 6-7. CAN/eSCI Pins Configuration for CAN/eSCI Fixed Baud Rate Boot Modes

CNRX_A	GPIO	CNRX_A	Input with pull-up and hysteresis	CNRX_A	Input with pull-up and hysteresis	GPIO	—
TXD_A	GPIO	GPIO	—	GPIO	—	TXD_A	Push/Pull output, with medium slew rate
RXD_A	GPIO	RXD_A	Input with pull-up and hysteresis	GPIO	—	RXD_A	Input with pull-up and hysteresis

The BAM configures the communication modules for reception with fixed baud rates as shown in the Table 6-8 and then waits for data reception. The watchdog timers are configured as shown in Table 6-9.

Table 6-8. Serial Boot Fixed Baud Rates

Crystal Frequency [MHz]	System Clock Frequency [MHz]	Desired SCI Baud Rate [baud]	Actual SCI Baud Rate [baud]	CAN Baud Rate[1] [baud]
f_{xtal}	$f_{sys} = 1.5 * f_{xtal}$	-	f_{sys} / 1248	f_{sys} / 60
8	12	9600	9615	200k
12	18	14400	14423	300k
16	24	19200	19230	400k
20	30[2]	24000	24038	500k
40				

[1] The CAN engine is clocked by the system clock.

[2] With a 40 MHz crystal, PLLCFG[2] must be set to 1 during reset, which has the effect of doubling the PLL input divider, so that the PLL output frequency is the same for both 20 and 40 MHz crystal frequencies.

Table 6-9. Watch Dog Timeout Settings for Serial Boot

Core WD		SWT	
Clock Source	**Condition**	**Clock Source**	**Condition**
PLL 12-30 MHz	Disabled during boot Enabled after boot Timeout = 2.5 x 2^{28} cycles	IRC 16 MHz	Enabled during boot Disabled after boot Timeout = 2.5 x 2^{27} cycles

If a message containing 8 bytes with ID 0x11 is received by the CAN controller first, the BAM program:

- Transitions to serial CAN boot mode
- Disables the eSCI
- Configures RXD_A to its reset state
- Transitions to the CAN serial download protocol routine

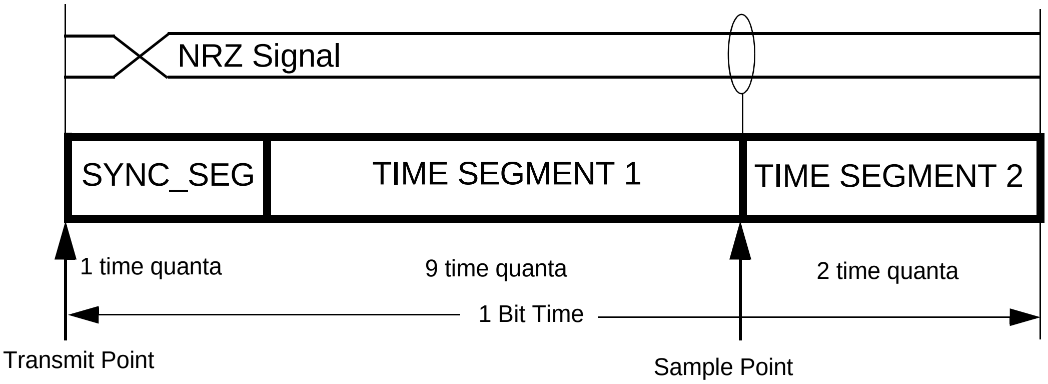
If a byte from eSCI is received first, the BAM program:

- Transitions to the Serial SCI Boot sub-mode
- Disables CAN_A module
- Configures CAN_A’s pins to their reset state
- Transitions to the SCI serial download protocol routine

6.5.5.1 CAN Controller Configuration in the Fixed Baud Rate Mode

The CAN controller is configured to operate at a baud rate equal to system frequency divided by 60, using the standard 11 bit identifier format detailed in CAN 2.0A specification. See Table 6-8for examples of baud rates. Only one message buffer 0 is used for all communications.

The bit timing is configured as shown in Figure 6-6.



Note: 1 Time quanta = 5 System clock periods

Figure 6-6. CAN Bit timing

The BAM program ignores CAN errors and all received messages are assumed to be good. Received messages are transmitted back. The host processor must compare the MCU transmissions with the sent data and restart the process if an error is detected.

6.5.5.2 SCI Controller Configuration in Fixed Baud Rate Mode

The eSCI is configured for 1 start bit, 8 data bits, no parity, 1 stop bit and operates at a baud rate equal to system clock divided by 1248. See Table 6-8 for examples of baud rates.

The BAM program ignores the eSCI errors. All data received is assumed to be good and is sent back through the TXD pin. The host processor must compare the MCU transmitted data with the sent data and restart the process if an error is detected.

6.5.5.3 Serial Boot Mode Download Protocol

The download protocol follows four steps:

1. Host sends 64-bit password.
2. Host sends start address, size of download code in bytes, and VLE bit.
3. Host sends the application code data.
4. The MCU switches to the loaded code at the start address.

The communication is done in half-duplex manner; any transmission from host is followed by the MCU transmission. The host computer should not send data until it receives echo from the MCU. All multi byte data structures must be sent most-significant byte (MSB) first.

When the CAN is used for serial download, the data is packed into standard CAN messages in the following manner:

- A message with 0x11 ID and 8-byte length to send the password. The MCU transmits back the same data, but with ID of 0x1.
- A message with 0x12 ID and 8-byte length to send the start address, length, and the VLE mode bit. The MCU transmits back the same data, but with ID of 0x2.
- Messages with 0x13 ID are used to send the downloaded data. The MCU transmits back the same data with ID of 0x3.

When the SCI is used for serial download, the data must be sent on a byte-by-byte basis. The MCU transmits back the received bytes.

Since the MCU starts up with the system clock derived from the Internal RC Oscillator (IRC), it does not wait for the crystal oscillator to stabilize or the PLL to lock before finishing the reset sequence and negating the $\overline{\text{RTSTOUT}}$ pin. Therefore, the BAM program starts executing with the IRC clock, but it needs to switch to the PLL clock in case of serial boot. Before switching to the PLL clock, the BAM program waits for the PLL lock flag to be asserted, which is an indication that the crystal oscillator has settled and the PLL has locked. As a consequence, the external device that will upload the serial code needs to wait enough time after $\overline{\text{RSTOUT}}$ negates to allow the crystal oscillator to stabilize and the PLL to lock. The exact time depends on the crystal used, but is typically less than 10 ms. On the other hand, the code must be uploaded before the SWT timeout expires. The SWT timeout is 21 seconds.

NOTE

Serial boot is only possible if the PLLCFG pins have been configured to enable the PLL working in normal mode.

6.5.5.4 Download Protocol Execution

The BAM program executes the serial boot as following:

1. Download 64-bit password.
    a) The received 8-byte password is checked for validity. For a password to be valid, none of its four 16-bit half words must equal 0x0000 or 0xFFFF.
    b) The BAM program then checks the censorship status of the MCU. If SIU_CCR[DISNEX] is set, the MCU is considered to be censored and the password is compared with a password stored in the shadow row in internal flash memory.
    c) If SIU_CCR[DISNEX] is cleared, the MCU is not considered to be censored and the password is compared to the fixed value of 0xFEED_FACE_CAFE_BEEF.
    d) If the password check fails, the MCU stops responding. Assert $\overline{\text{RESET}}$ to force the MCU out of this state.
    e) If the password check passes, the BAM transitions to the next step in the protocol.
2. Download start address, size of download, and VLE bit.

    The BAM considers the next 8 bytes to contain a 32-bit start address, the VLE mode bit, and a 31-bit code length (see Figure 6-7).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
START_ADDRESS[0:15]															

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
START_ADDRESS[16:31]															

32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
VLE	CODE_LENGTH[0:14]														

48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
CODE_LENGTH[15:30]															

Figure 6-7. Start Address, VLE Bit and Download Size in Bytes

- START_ADDRESS defines where the received data is stored and where the MCU branches after the download is finished. The two least significant bits of the start address are ignored by the BAM program, thus the loaded code should be 32-bit word aligned.
- CODE_LENGTH defines how many data bytes to be loaded.
- VLE instructs the MCU to program MMU entries 1–3 with VLE attribute. If it is 1, the downloaded code must be compiled to VLE instructions, if it is 0 the code contains classic Power Book E architecture instructions.

3. Download data.

Each byte of data received is stored in the MCU memory, starting at the START_ADDRESS specified in the previous step. The data increments through memory until the number of bytes stored matches CODE_LENGTH specified in the previous step.

The BAM program buffers incoming data, collecting up to eight bytes. The buffered data is written to the RAM with 64-bit writes to prevent ECC errors, which may happen if the MCU RAM is protected by 64-bit ECC code.

Once the buffered data is written to the RAM the BAM program refreshes the SWT.

NOTE

Only system RAM supports 64-bit writes; therefore, attempting to download data to other RAM apart from system RAM causes errors.

If the start address of the downloaded data is not on an 8-byte boundary, the BAM writes 0x0 to the memory locations from the proceeding 8-byte boundary to the start address (maximum 4 bytes). The BAM also writes 0x0 to all memory locations from the last byte of data downloaded to the following 8 byte boundary (maximum 7 bytes). An additional 8 zero bytes are written to prevent possible ECC errors that may be caused by the CPU pre-fetching.

BAM writes additional two zero double words to the system RAM after loaded code to prevent possible ECC errors, which could happen due the CPU speculatively pre-fetches data after last loaded instruction, where the RAM can be not initialized. The last loaded code address must not exceed 0x4003_FFF0 (the upper allowed RAM address by MMU settings minus two zero double words, written by BAM at the end of code download).

4. Switch to the loaded code.

   The BAM program waits for the last echo message transmission to complete, then the active communication controller is disabled. Its pins revert to GPIO inputs.

   To provide compatibility with older devices, the BAM

   - writes the core's time base registers (TBU and TBL) with 0x0
   - enables the core watchdog to cause a reset after a time-out period of 2.5 x 2^{28} CPU clock cycles
   - disables software watchdog (SWT)

   The BAM code passes control to the loaded code at START_ADDRESS, which was received in step 2 of the protocol.

NOTE

The loaded code must periodically refresh the core watchdog timer or change the timeout period to a value that does not cause resets during normal operation.

6.5.5.5 Baud Rate Detection Procedure

When EVTO pin is driven low during reset, the BAM program enters into Baud Rate Detection mode.

Following additional steps are taken:

1. The baud rate detection routine is copied to the beginning of the system RAM from the BAM ROM to improve baud rate detection accuracy
2. The CPU branches to the RAM
3. The MCU configures the CNRX_A and RXD_A pins as general purpose inputs
4. The MCU polls these GPIs until one of them goes low
   a) If the CNRX_A pin transitions first, the BAM program starts CAN baud rate detection routine, ignoring RXD_A. After detecting the CAN baud rate, the BAM program transitions to the CAN download protocol routine described above.
   b) If the RXD_A pin transitions first, the SCI baud rate detection and download protocol routines are called, ignoring any further CAN pins activity.

The host, executing the serial boot, should send a test frame in order the MCU can detect the host communication speed. When booting through the SCI, the host should send a zero byte as the test frame. When booting through the CAN, the host should send zero ID, zero length message as the test frame.

6.5.5.5.1 SCI Baud Rate Detection

The host must send a zero byte to allow the MCU to detect the serial link baud rate. The host transmits one start bit, eight zero data bits and one stop bit (the MCU does not echo it).

The MCU polls the RXD_A pin for high to low transition and starts the e200 core Time Base counter (TB). Then the MCU polls for low to high transition on the pin and when it happens, the MCU turns off the TB counter. The TB content is used to calculate incoming signal baud-rate. The SCI baud rate is equal to the

TB content divided by 288. (Measured over 9 bits with 16 system clocks per bit and the core frequency is two times higher than system frequency).

6.5.5.6 CAN Baud Rate Detection

The host transmits a zero length message with zero 11 bit ID and MCU measures time over 40 bits, polling CNRX_A pin for high and low, according to the sent data. The MCU does not acknowledges this message.

The CAN baud rate depends on the number of quantas per bit and serial clock frequency, which is defined by a prescaler. The CAN baud rate detection routine selects these parameters to maximize number of quantas per bit and achieve minimum difference between measured value and duration of the 40 CAN bits, to be programmed with selected pair of the parameters.

The CAN controller can be programmed with 8 to 25 number of quantas per bit. The bit timing parameters, can be selected by the baud rate detection routine, are shown in the Table 6-10. See the parameters descriptions in the FlexCAN block guide)

Table 6-10. Lookup Table for CAN Bit Timing.

Time quanta per bit	Time segment 1		Time Segment 2	RJW
	PROPSEG	PSEG1	PSEG2	
8	1	3	3	2
9	2	3	3	2
10	3	3	3	2
11	4	3	3	2
12	3	4	4	2
13	4	4	4	3
14	5	4	4	3
15	6	4	4	4
16	7	4	4	3
17	8	4	4	3
18	7	5	5	4
19	8	5	5	4
20	7	6	6	4
21	8	6	6	4
22	7	6	6	4
23	8	6	6	4
24	7	7	7	4
25	8	7	7	4

Maximum and minimum speeds of the serial communication modules are defined by the MCU system frequency and shown in the Table 6-11.

Table 6-11. Maximum and Minimum Detectable Baud Rates[1]

F_{xtal}l (MHz)	F_{sys} = 1.5*F_{xtal} (MHz)	Max Baud rate for CAN (F_{sys}/9)[2](bps)	Min CAN Baud rate (F_{sys}/25/256) bps	Max Baud rate for SCI (F_{sys}/160)(bps)	Min Baud rate for SCI ($F_{sys}/160/2^{13}$)(bps)
8	12	1M	1875	75K	9.2
12	18		2812.5	112.5K	13.7
16	24		3750	150K	18.3
20	30		4687.5	187.5K	22.9
40					

[1] When the MCU operates with the PLL in normal mode with crystal oscillator as a reference clock source.

[2] Limited by 1Mbps by CAN standard

6.5.6 Booting from the Development Bus

If the MCU boots in one of the Development Bus boot modes, the BAM program:

- Reprograms the MMU entries for EBI and internal flash (see Table 6-12)
- Sets up the EBI and development bus pins
- Tries to read RCHW from logical address 0x2000_0000

If the valid RCHW is read from that address, the BAM program:

- Reads the user application code start address from 0x2000_0004 address
- Parses RCHW
- Sets up watchdogs
- Updates EBI, SRAM and internal flash MMU entries(1-3), according to RCHW[VLE]
- Passes control to the user code

If no valid RCHW was read, BAM switches to the serial boot mode.

Table 6-12. MMU Configuration for Development Bus Boot modes

TLB Entry	Region	Logical Base Address	Physical Base Address	Size	Attributes
1	Internal Flash	0x0000_0000	0x2000_0000	16 MB	Not guarded Big endian Global PID
2	EBI	0x2000_0000	0x2000_0000	16 MB	Not guarded Big endian Global PID

6.5.6.1 EBI Configuration for Separate Address and Data Development Bus Boot Mode

The BAM program sets up EBI related registers as shown in the Table 6-13.

Table 6-13. Development Bus/EBI Register Settings for Separate Address Data lines mode

Register	Function	Value	Comments
EBI_CAL_BR0	—	0x2000_0803	Sets the base address to 0x2000_0000,16-bit wide bus, burst Inhibit
EBI_CAL_OR0	—	0xFF80_00F0	Set 15 wait states, 8MB
SIU_PCR259–SIU_PCR293	D_ADD[12:30], D_ADD_DAT[0:15]	0x0440	Set pads to 20pf drive strength
SIU_PCR305–SIU_PCR307	D_ADD[9:11]		
SIU_PCR256	D_CS0	0x443	Set pads to 20pf drive strength and pull-up enable
SIU_PCR295	D_WE0		
SIU_PCR297	D_OE		
SIU_PCR298	D_TS		

RCHW[PS0] must be programmed to “1”, since the development bus does not support 32-bit port size in that sub-mode.

6.5.6.2 EBI Configuration for multiplexed Address and Data Development Bus Boot Mode

The BAM program sets up EBI related registers as shown in the Table 6-14.

Table 6-14. Development Bus/EBI Register Settings multiplexed mode.

Register	Function	Value	Comments
The following registers are programmed when RCHW[PS0] = 1			
EBI_MCR	—	0x0000_0806	AD multiplexed mode, small port size use D16-31
EBI_CAL_BR0	—	0x2000_0883	Sets the base address to 0x2000_0000,16-bit wide bus, AD multiplexed mode, burst Inhibit
EBI_CAL_OR0	—	0xFF80_00F0	Set 15 wait states, 8MB
SIU_PCR259-SIU_PCR262	D_ADD[12:15],	0x440	Set pads to 20pf drive strength
SIU_PCR305-SIU_PCR307	D_ADD[9:11]		
SIU_PCR263–SIU_PCR277	D_ADD_DAT[16:30]	0x840	Set pads to 20pf drive strength
SIU_PCR257	D_ADD_DAT[31]		
SIU_PCR278-SIU_PCR293	D_ADD_DAT[0:15]	0x440	Set pads to 20pf drive strength
SIU_PCR256	D_CS0	0x443	Set pads to 20pf drive strength and pull-up enable
SIU_PCR295	D_WE0		
SIU_PCR297	D_OE		
SIU_PCR298	D_TS		
SIU_PCR299	D_ALE		
The following register is reprogrammed when RCHW[PS0] = 0			
EBI_CAL_BR0	—	0x2000_0083	Sets the base address to 0x2000_0000,32-bit wide bus, AD multiplexed mode, burst Inhibit

If the boot memory device is 32 bit wide, the RCHW must be programmed in two first half-words of the memory because when the MCU tries to read RCHW, the EBI is configured to use data lines 16–31 for 16-bit port accesses.

6.5.7 Enabling Debug of a Censored Device

When a device is in a censored state, the debug port (JTAG/Nexus) is disabled and only JTAG BSDL commands can be used. Access to the Nexus/JTAG clients on a censored device requires inputting the proper password into the JTAG Censorship Control Register during reset.

NOTE

> When the debug port is enabled on a censored device, it is enabled only until the next reset.

Figure 6-8 shows the logic that enables access to Nexus clients in a censored device using the JTAG port.

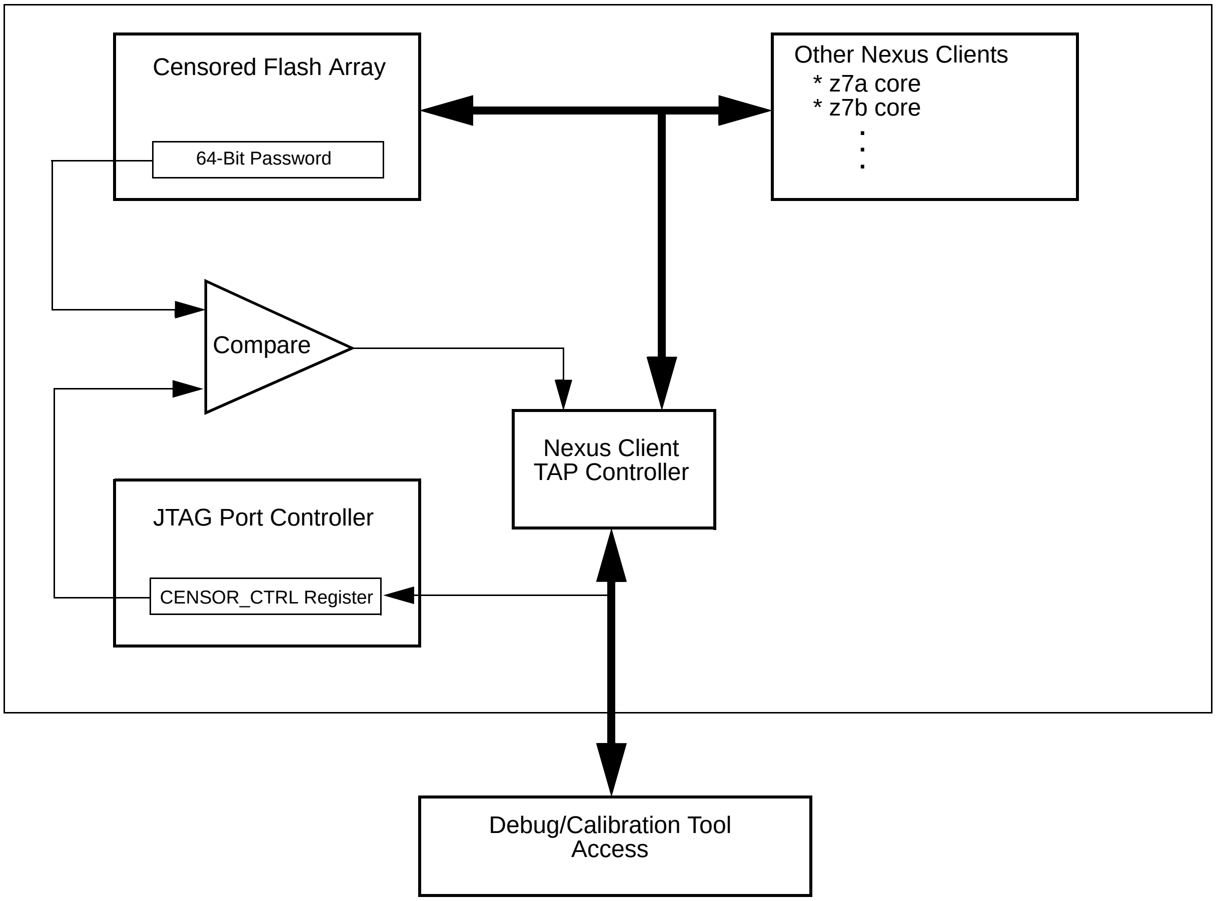


Figure 6-8. Enabling JTAG/Nexus Access on a Censored Device

The steps to enable the debug port on a censored device are as follows:

1. Hold the device in system reset state using a debugger or other tool.

2. While the device is being held in system reset state, shift the 64-bit password into the CENSOR_CTRL register via the JTAG port using the JTAG ENABLE_CENSOR_CTRL instruction. The JTAG serial password is compared against the serial boot flash password from the flash shadow block.
3. If there is a match, the Nexus client TAP controller enters normal operation mode and the DISNEX flag in the SIU_CCR register is negated, indicating Nexus is enabled. Upon negation of reset, the debug / calibration tool is able to access the device via NEXUS port and JTAG. If the JTAG serial password does not match the serial boot flash password or the serial boot flash password is an illegal password then the debug / calibration tool is not able to access the device. After the debug port is enabled, the tool can access the censored device and can erase and reprogram the shadow flash block in order to uncensor the device.

NOTE

> If the shadow flash block is erased without reprogramming a new valid password before a reset, it will contain an illegal password and the debug port will be inaccessible.

4. Subsequent resets will clear the JTAG censor password register and the Nexus client TAP controller will be held in reset again. Therefore, the tool must resend the JTAG serial password, as described above, in order to enable the Nexus client TAP controller again.

Chapter 7
Clocking

This chapter presents the clock architecture for the device. It describes the SoC level clock generation and distribution.

7.1 Overview

This SoC contains the following clock related modules:

- External Oscillator (XOSC)
- Internal RC Oscillator (IRC)
- FM PLL
- System Clock Divider
- FlexCAN Clock Selector
- FlexRay Clock Selector
- Nexus MCKO Divider

7.2 Clock Distribution

This device has separate clock trees for each of the cores to eliminate, where possible, common mode clock failures. In addition, MPC5676R includes an internal 16MHz RC oscillator (IRC), which will be used to drive the system clock at power on or system reset. MPC5676R also must have an option to drive the system clock from the external oscillator (XOSC).

7.3 Architecture

Figure 7-1 shows the clock distribution paths for all on-chip modules.

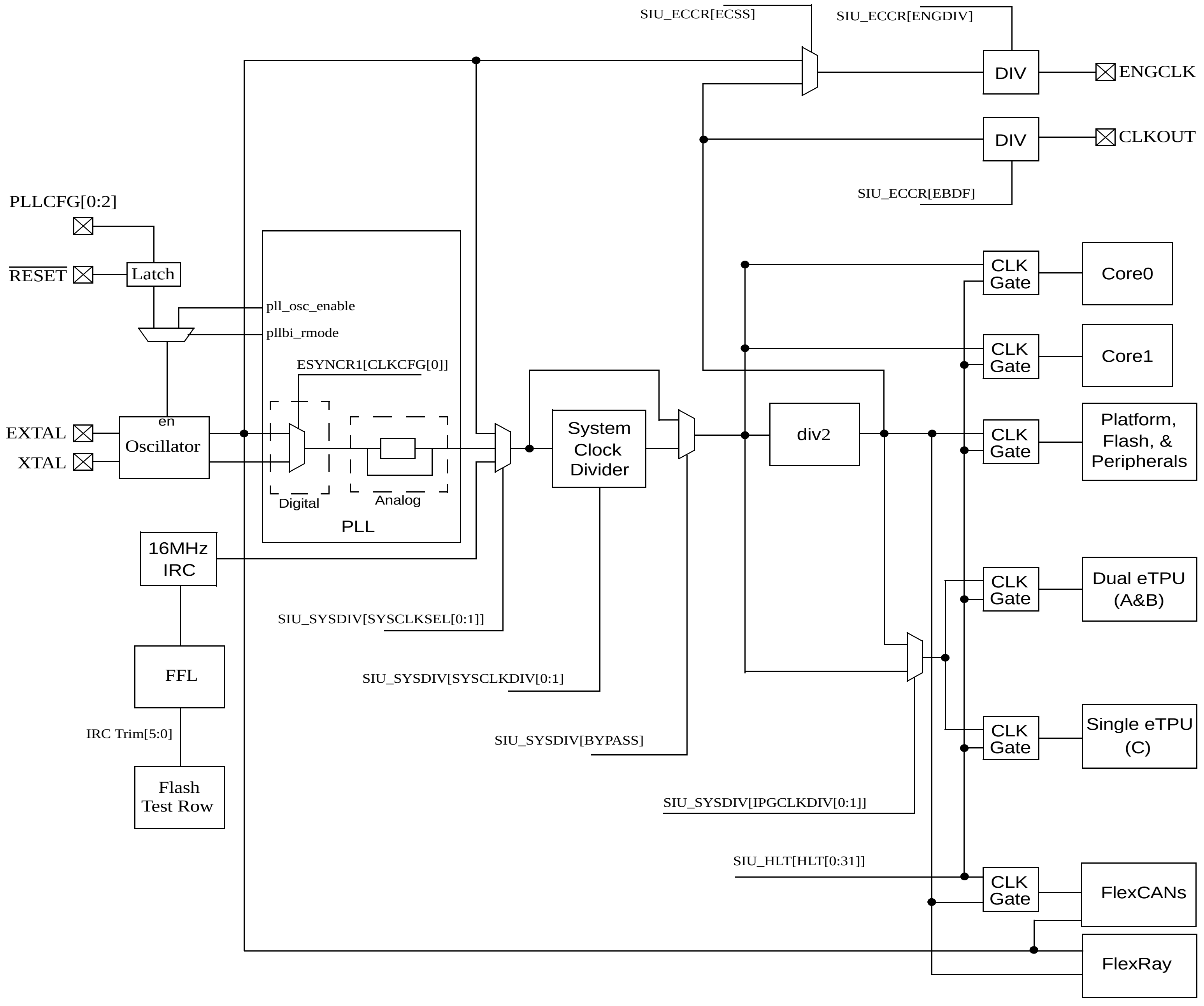


Figure 7-1. MPC5676R System Level Clock Diagram

7.4 Internal Clocking Requirements

7.4.1 Operating Frequencies

The device supports the operating frequencies given in Table 7-1. The maximum frequency for the cores and eTPU is 184 MHz (180 MHz with 2% FM).

Table 7-1. MPC5676R Operating Frequencies

Mode	Cores	Platform	eTPU
Double	184MHz	92MHz	92MHz
eTPU	184MHz	92MHz	184MHz

7.4.2 16MHz Internal RC Oscillator (IRC)

7.4.2.1 Overview

The benefits of providing and booting the device from the IRC are:

- Fast start up (IRC starts up in tens of IRC clock cycles).
- The IRC is very reliable (extremely low PPM failure rate).
- Removes historical issues seen with the PLL trying to lock during power up transients. The PLL will be enabled by the user, after reset, and after the supplies are within specified regulation.
- The IRC can provide a limp mode clock in the event of a crystal oscillator failure.

7.4.2.2 Functional Requirements

The IRC provides the default system clock at power-up and reset for the device. In the event of PLL or oscillator failures, the user software will be required to switch clocks or enable a reset. There is no automatic switching to an IRC system clock for any failure condition. The IRC oscillator cannot be disabled.

After reset, device hardware applies a trim value to the IRC to optimize its accuracy. During reset, device hardware applies a default centered trim value to the IRC.

7.4.3 External Oscillator (XOSC)

The external oscillator supports input frequencies of 8-20, and 40MHz. The oscillator is designed for optimal startup margin with typical crystals. Oscillator power is supplied from its own 3.3 V PLL supply voltage generated by the voltage regulator to minimize noise. The oscillator may be used to drive the system clock directly (when the PLL is bypassed), or as the input reference clock for the PLL. A square-wave input can also be supplied to the device through the oscillator by connecting the external clock source to the EXTAL pin with the oscillator operating in external clock mode.

7.4.4 Default Clock Configuration

7.4.4.1 IRC

The device starts up from reset with the 16MHz internal RC oscillator (IRC) enabled and driving the system clock.

7.4.4.2 XOSC

The external oscillator (XOSC) is enabled or disabled based on PLLCFG encodings to support the BAM serial boot mode.

7.4.4.3 PLL

In order to support BAM serial boot mode, all PLLCFG pins, connections and configurations are effective from reset.

7.4.4.4 Default Core Clock Configuration

The clock for Core0 is enabled, and the clock gate for Core1 is disabled.

7.4.4.5 Default External Clock Configuration

The external clock pins CLKOUT and ENGCLK are enabled during and immediately after reset. CLKOUT defaults to the system clock frequency divided by two, and ENGCLK defaults to the system clock frequency divided by 32.

7.4.5 Clock Configuration Selection

After reset, system clock selection is done via a read/write field (SYSCLKSEL[0:1]) in the SIU_SYSDIV register. The default value for the field selects the IRC. The other selections are the PLL and XOSC clocks. A transition from one system clock source to another does not cause a glitch on the system clock.

7.4.6 Halt Clock Gating

Software controlled clock gating is implemented using the SIU_HLT and SIU_HLTACK registers. See the SIU chapter of this document for more information on the SIU_HLT and SIU_HLTACK registers.

7.4.7 Serial Boot Clocking

Since the IRC is not accurate enough for the serial protocols, prior to entering serial boot mode the BAM will wait for PLL lock and then switch the system clock to the PLL output. Note that no direct signal is available to an external host to indicate serial boot has been initiated by the BAM.

7.4.8 FlexRay Clocking

The FlexRay block has two distinct software-controlled clock domains. One of the clock domains is always derived from the system clock. The source for the second clock domain can be the system clock or the Oscillator output. The logic in the second clock domain controls the FlexRay interface pins. The CLKSEL bit in the FlexRay FR_MCR register selects between the system clock and the oscillator clock as the clock source for the second domain. Selecting the oscillator as the clock source ensures low jitter on the FlexRay bus. If the oscillator clock source is selected for the FlexRay interface, then a divided down oscillator clock must not be selected as the source for the system clock.

7.4.9 FlexCAN Clocking

The FlexCAN block has two distinct software-controlled clock domains. One of the clock domains is always derived from the system clock. This clock domain includes the message buffer logic. The source for the second clock domain can be the system clock or the oscillator output. The logic in the second clock domain controls the CAN interface pins. The CLK_SRC bit in the FlexCAN CTRL register selects between the system clock and the oscillator clock as the clock source for the second domain. Selecting the oscillator as the clock source ensures low jitter on the CAN bus. System software can gate both clocks by writing to the MDIS bit in the FlexCAN MCR register.

7.4.10 MCKO

MCKO is an output clock from the Nexus Port Controller (NPC) to the development tools used for the timing of MSEO and MDO pin functions. MCKO is derived from the system clock, and its frequency is determined by the value of the MCKO_DIV field in the port configuration register (PCR) located in the NPC.

7.4.11 D_CLKOUT/ENGCLK

D_CLKOUT is the device system clock output designed for, but not restricted to, use with the EBI.
ENGCLK is a 50% duty cycle output clock derived from the device system clock or the oscillator frequency. ENGCLK maximum frequency is the system clock or oscillator frequency divided by two.
ENGCLK is not synchronous to CLKOUT.

Chapter 8
Frequency Modulated Phase-Locked Loop (FMPLL)

8.1 Introduction

The frequency modulated phase-locked loop (FMPLL) module is a frequency modulated phase-locked loop that has been optimized to generate voltage controlled oscillator (VCO) frequencies from 192 MHz – 600 MHz based on an input clock range of 8 MHz to 40 MHz. The frequency multiplication, output dividers and the frequency modulation waveform are register-programmable through a peripheral bus interface.

For register and bit descriptions, see:

- Section 3.2.1.29, “System Clock Register (SIU_SYSDIV)”
- Section 3.2.1.26, “External Clock Control Register (SIU_ECCR)

8.1.1 Block Diagram

A simplified block diagram of the FMPLL illustrates the functionality and interdependence of major blocks (see Figure 8-1). Shaded blocks represent analog circuit components that make up the core analog portion of the FMPLL. The complete FMPLL closed-loop system contains the feedback divider (EMFD) and output divider (ERFD). Refer to Section 8.4.3.3, “PLL Normal Mode Without FM,” for details on each sub-block.

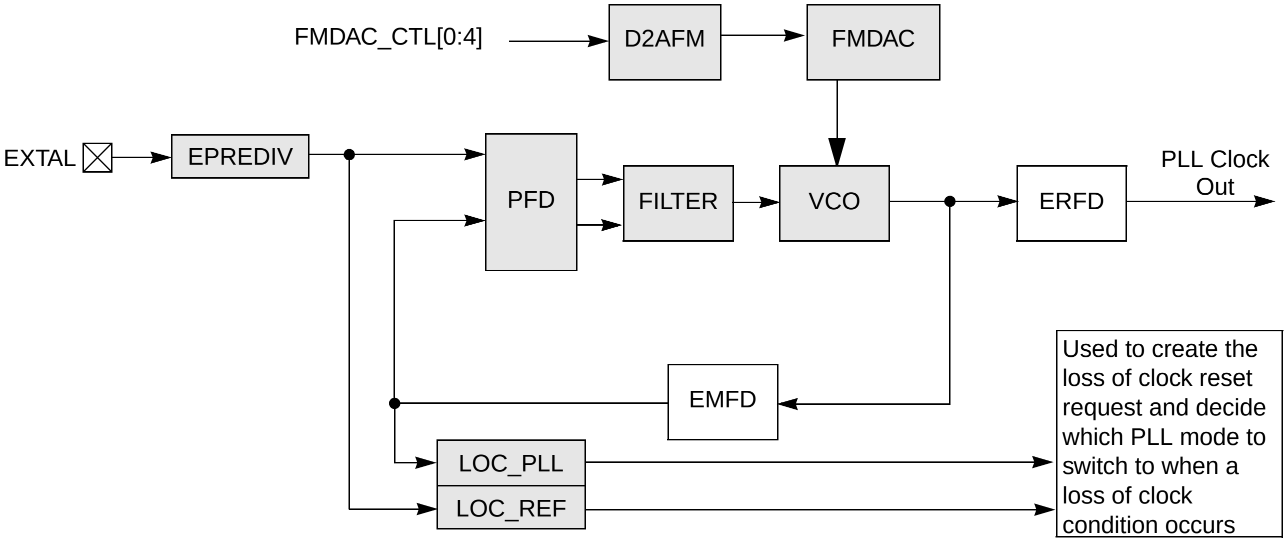


Figure 8-1. FMPLL Block Diagram

8.1.2 Features

The FMPLL has these major features (refer to *MPC5676R Microcontroller Data Sheet* for performance data and restrictions):

- Input clock frequency range: 8 MHz to 40 MHz (EXTAL)
- Programmable frequency multiplication factor settings which specify VCO frequencies of 192 MHz – 600 MHz
- PLL Off mode (low-power mode)
- Register programmable output clock divider (ERFD)
- Programmable frequency modulation
  - Modulation applied as a triangle waveform
  - Peak-to-peak register programmable modulation depths
  - Register programmable modulation rates of $F_{extal}/80$, $F_{extal}/40$, and $F_{extal}/20$ (modulation rate must be between 400 kHz and 1 MHz).
- Lock detect circuitry provides a signal indicating the FMPLL has acquired lock and continuously monitors the FMPLL output for any loss of lock
- Loss-of-clock circuitry monitors input reference and FMPLL output clocks with programmable ability to select a backup clock source as well as generate a reset or interrupt in the event of a failure

8.1.3 Modes of Operation

There are two main modes of FMPLL: PLL Off mode and normal mode. These modes are briefly described in this section.

When PLL Off mode is selected, the FMPLL is turned off; the clock source must come from somewhere else or the device will not function. The lock detector is not functional and does not indicate that the FMPLL is in a locked state. Frequency modulation is not available and the FMPLL is put into a low-power, idle state. A full swing square wave clock input for the entire system must be supplied on the EXTAL pin (Refer to *MPC5676R Microcontroller Data Sheet* for external clock input requirements). This operating mode is described in Section 8.4.2, “PLL Off Mode.”

When normal mode is selected, the FMPLL is fully programmable. The FMPLL reference clock source can be a crystal oscillator or an external clock generator. The lock detector indicates the lock status of the FMPLL, and frequency modulation of the output clock can be enabled. This operating mode is described in Section 8.4.3, “Normal Mode.”

8.2 External Signal Description

Refer to Chapter 2, “Signal Descriptions” for detailed signal descriptions.

8.3 Memory Map and Registers

This section provides a detailed description of the FMPLL registers.

8.3.1 Module Memory Map

Table 8-1 shows the FMPLL memory map. The address of each register is given as an offset to the FMPLL base address. The table lists registers in the order of their addresses, identified by complete name and mnemonic, and the type of their accesses.

Table 8-1. FMPLL Memory Map

Offset from FMPLL_BASE_ADDR (0xC3F8_0000)	Register	Bits	Access	Reset Value	Section/Page
0x0000	Reserved				
0x0004	SYNSR—FMPLL synthesizer status register	32	R/W	—[1]	8.3.2.1/8-3
0x0008	ESYNCR1—FMPLL enhanced synthesizer control register 1	32	R/W	0x8001_0053	8.3.2.2/8-6
0x000C	ESYNCR2—FMPLL enhanced synthesizer control register 2	32	R/W	0x0000_0005	8.3.2.3/8-8
0x0010–0x001C	Reserved				
0x20	SYNFMCR—FMPLL synthesizer FM control register	32	R/W	—[1]	8.3.2.4/8-11

[1] See specific register description.

8.3.2 Register Descriptions

This section lists the FMPLL registers in address order and describes the registers and their bit fields.

8.3.2.1 FMPLL Synthesizer Status Register (SYNSR)

Offset: FMPLL_BASE_ADDR + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	LOLF	LOC	MODE	PLL SEL	PLL REF	LOCKS	LOCK	LOCF	U[1]	U[1]
W							w1c							w1c		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-2. FMPLL Synthesizer Status Register (SYNSR)

[1] These bits may read 0 or 1, depending on current state of the PLL, however they do not provide any useful user information.

Table 8-2. SYNSR Bit Field Descriptions

Field	Description
0–21	Reserved
22 LOLF	Loss- of-Lock Flag. This bit provides the interrupt request flag. To clear the flag, write a 1 to the bit. Writing 0 has no effect. The LOLF is not set in response to the following conditions: • Loss of lock due to a system reset, or • Loss of lock due to changing any of the following: Multiplication Factor Divider (ESYNCR1_EMFD), Pre-Divider (ESYNCR1_EPREDIV), or Modulation Depth (EYSNCR2_EDEPTH) prior to the LOLF being set and cleared (for example, in response to an unexpected loss of lock condition). The LOLF will be set in response to the following conditions: • Changing any of the following: EMFD, EPREDIV, or EDEPTH subsequent to the LOLF being set and cleared (for example, in response to an unexpected loss of lock condition), or • Changing the Modulation Rate (ESYNCR2_ERATE), regardless of previous conditions. If the LOLIRQ bit is set, these conditions will trigger an interrupt request. To avoid unintentional interrupt requests, the following steps are recommended: • Clear the LOLIRQ bit • Change EMFD, EPREDIV, EDEPTH, and/or ERATE • Ensure the PLL is locked • Clear the LOLF bit • Set the LOLIRQ bit, if desired If the flag is set due to a system failure, writing the ESYNCR1[EMFD] bits or enabling FM does not clear the flag. Assert reset to clear the flag. If lock is reacquired, the bit remains set until either a write 1 or reset is asserted. 0 Interrupt service not requested. 1 Interrupt service requested.
23 LOC	Loss-Of-Clock Status. The LOC bit is an indication of whether a loss-of-clock condition is present when operating in normal PLL mode. If LOC = 0, the system clocks are operating normally. If LOC = 1, the system clocks have failed due to a reference failure or a PLL failure. If the read of the LOC bit and the loss-of-clock condition occur simultaneously, the bit does not reflect the current loss-of-clock condition. If a loss-of-clock condition occurs that sets this bit and the clocks later return to normal, this bit is cleared. LOC is always zero in PLL Off mode. 0 Clocks are operating normally. 1 Clocks are not operating normally.
24 MODE	Clock Mode. The state of this bit, along with PLLSEL and PLLREF, indicates which clock mode the PLL is operating in (see Table 8-13). The value of ESYNCR1[CLKCFG2] is reflected in this location. 0 PLL Off mode. 1 PLL clock mode.
25 PLLSEL	PLL Mode Select. The state of this bit, along with MODE and PLLREF, indicates which mode the PLL operates in. This bit is cleared in PLL Off mode. The value of ESYNCR1[CLKCFG1] is reflected in this location. 0 PLL Off mode. 1 Normal PLL mode.
26 PLLREF	PLL Clock Reference Source. The state of this bit, along with MODE and PLLSEL, indicates which reference source has been chosen for normal PLL mode. This bit is cleared in PLL Off mode. The value of ESYNCR1[CLKCFG0] is reflected in this location. 0 External clock reference chosen 1 Crystal clock reference chosen **Note:** The PLL controls the oscillator.

Table 8-2. SYNSR Bit Field Descriptions (continued)

Field	Description
27 LOCKS	Sticky PLL Lock Status Bit. The LOCKS bit is a sticky indication of PLL lock status. LOCKS is set by the lock detect circuitry when the PLL acquires lock after: 1) a system reset, or 2) a write to the ESYNCR1 which modifies the ESYNCR1[EMFD] bits, or 3) frequency modulation is enabled. Whenever the PLL loses lock, LOCKS is cleared. LOCKS remains cleared after the PLL re-locks, until one of the three conditions occurs. Furthermore, if the LOCKS bit is read when the PLL simultaneously loses lock, the bit does not reflect the current loss-of-lock condition. If operating in PLL Off mode, LOCKS remains cleared after reset. 0 PLL has lost lock since last system reset, a write to ESYNCR1 to modify the ESYNCR1[EMFD] and ESYNCR1[EPREDIV] bit fields, or frequency modulation enabled 1 PLL has not lost lock since last system reset, a write to ESYNCR1 to modify the ESYNCR1[EMFD] and ESYNCR1[EPREDIV] bit fields, or frequency modulation enabled
28 LOCK	PLL Lock Status Bit. The LOCK bit indicates whether the PLL has acquired lock. Refer to *MPC5676R Microcontroller Data Sheet* for tolerances. If the LOCK bit is read when the PLL simultaneously loses lock or acquires lock, the bit does not reflect the current condition of the PLL. If operating in PLL Off mode, LOCK remains cleared after reset. 0 PLL is unlocked 1 PLL is locked
29 LOCF	Loss-of-Clock Flag. This bit provides the interrupt request flag. To clear the flag, write a 1 to the bit. Writing 0 has no effect. Asserting reset clears the flag. If clocks return to normal after the flag has been set, the bit remains set until cleared by either writing 1 or asserting reset. A loss-of-clock condition can only be detected if LOCEN = 1. 0 Interrupt service not requested. 1 Interrupt service requested.
30–31	Reserved

8.3.2.2 FMPLL Enhanced Synthesizer Control Register 1 (ESYNCR1)

This is one of two FMPLL synthesizer control registers that are used to access enhanced features in the FMPLL. The bit fields in the ESYNCR1 behave as described in Figure 8-3.

Offset: FMPLL_BASE_ADDR + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	1	CLKCFG[2:0]			0	0	0	0	0	0	0	0	EPREDIV			
W																
Reset	1	—[1]	—[1]	—[1]	0	0	0	0	0	0	0	0	0	0	—[2]	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	EMFD							
W	0[3]															
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

[1] Reset value determined by PLLCFG pins during reset.

[2] Resets to value of PLLCFG[2] (0 if PLLCFG[2]=0; 1 if PLLCFG[2]= 1)

[3] Do not set this bit to 1.

Figure 8-3. FMPLL Enhanced Synthesizer Control Register 1 (ESYNCR1)

Table 8-3. ESYNCR1 Bit Field Descriptions

Field	Description
0	Reserved. **Note:** This bit is set to 1 on reset and always reads as 1. Writes to this bit have no effect.
1–3 CLKCFG[2:0]	Clock Configuration. The CLKCFG[2:0] bits are writable versions of the MODE, PLLSEL, and PLLREF bits in the SYNSR. These change the clock mode, after reset has negated, via software. CLKCFG[2:0] map directly to MODE, PLLSEL, and PLLREF to control the system clock mode. **Note:** CLKCFG = 0b101 (or any reserved/invalid value) can produce an unpredictable clock output. **Note:** The ESYNCR2[LOLRE] and ESYNCR2[LOCRE] should be set to 0 before changing the PLL mode, so that a reset is not immediately generated when CLKCFG is written.
4–11	Reserved
12–15 EPREDIV	Enhanced Pre-Divider. The EPREDIV bits control the value of the divider on the input clock. The output of the pre-divider circuit generates the reference clock to the PLL analog loop. The decimal equivalent of the EPREDIV binary number is substituted into the equation from Table 8-8. **Note:** Setting EPREDIV to any of the invalid states in Table 8-4 causes the PLL to produce an unpredictable output clock. The output frequency of the divider must equal f_{pllref} (see the *MPC5676R Microcontroller Data Sheet*). When the EPREDIV bits are changed, the PLL immediately loses lock. Do not change the EPREDIV bits during FM operation. Before changing EPREDIV, FM must be disabled and then reconfigured after the PLL re-locks to the new EPREDIV value. To prevent an immediate reset, clear the LOLRE bit before writing the EPREDIV bits. In PLL Off mode, the EPREDIV bits have no effect.

Table 8-3. ESYNCR1 Bit Field Descriptions (continued)

Field	Description
16–23	Reserved. **Note:** Do not set this bit to 1.
24–31 EMFD	Enhanced Multiplication Factor Divider. The EMFD bits control the value of the divider in the PLL feedback loop. The value specified by the EMFD bits establish the multiplication factor applied to the reference frequency. The decimal equivalent of the EMFD binary number is substituted into the equation from Table 8-10 for F_{sys} to determine the equivalent multiplication factor. The range of settings is $32 \leq \text{EMFD} \leq 132$. **Note:** EMFD values less than 32 and greater than 132 are invalid and cause the PLL to produce an unpredictable clock output. The VCO frequency must be within the f_{VCO} specification (see the *MPC5676R Microcontroller Data Sheet*). When the EMFD bits are changed, the PLL loses lock. Do not change the EMFD bits during FM operation. Before changing EMFD, FM must be disabled and then reconfigured after the PLL re-locks to the new EMFD value.To prevent an immediate reset, clear the LOLRE bit before writing the EMFD bits. In PLL Off mode, the EMFD bits have no effect. Table 8-5 shows the available divide ratios.

Table 8-4. Pre-divider Ratios

EPREDIV	Input Divide Ratio (EPREDIV+1)
0000	1
0001	2 (default if PLLCFG[2]=0)
0010	3
0011	4 (default if PLLCFG[2]=1)
0100	5
0101	6
0110	Invalid
0111	8
1000	Invalid
1001	10
1010–1111	Invalid

Table 8-5. Feedback Divide Ratios

EMFD	Feedback Divide Ratio (EMFD+16)
0000_0000–0001_1111	Invalid
0010_0000	48 (default for MPC5676R)
0010_0001	49
0010_0010	50
0010_0011	51
0010_0100	52

Table 8-5. Feedback Divide Ratios

EMFD	Feedback Divide Ratio (EMFD+16)
0010_0101	53
. . 0101_0011 . .	. . 99 . .
1000_0100	132
1000_0101–1111_1111	Invalid

8.3.2.3 FMPLL Enhanced Synthesizer Control Register 2 (ESYNCR2)

This is the second of two enhanced versions of the FMPLL synthesizer control register used to access enhanced features in the FMPLL. The bit fields in the ESYNCR2 behave as described in Figure 8-4.

Offset: FMPLL_BASE_ADDR + 0x000C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	LOCEN	LOLRE	LOCRE	LOL IRQ	LOC IRQ	0	ERATE	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CLK CFG _DIS	0	0	0	0	EDEPTH			0	0	ERFD					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Figure 8-4. FMPLL Enhanced Synthesizer Control Register 2 (ESYNCR2)

Table 8-6. ESYNCR2 Bit Field Descriptions

Field	Description
0–7	Reserved
8 LOCEN	Loss-of-Clock Enable. The LOCEN bit determines whether the loss-of-clock function is operational along with backup clock modes, and interrupt and reset functions. See Section 8.4.3.2, “Loss-of-Clock Detection,” for more information. In PLL Off mode, this bit has no effect. LOCEN does not affect the loss-of-lock circuitry. 0 Loss-of-clock disabled. 1 Loss-of-clock enabled.
9 LOLRE	Loss-of-Lock Reset Enable. The LOLRE bit determines how the integration module handles a loss-of-lock indication. See Section 8.4.3.1, “PLL Lock Detection,” for more information. When operating in normal PLL mode, the PLL must be locked before setting the LOLRE bit. Otherwise reset is immediately asserted. The LOLRE bit has no effect in PLL Off mode. 0 Assert reset on loss of lock is disabled. 1 Assert reset on loss of lock.

Table 8-6. ESYNCR2 Bit Field Descriptions (continued)

Field	Description
10 LOCRE	Loss-of-Clock Reset Enable. The LOCRE bit determines how the integration module handles a loss-of-clock condition when LOCEN is equal to 1. LOCRE has no effect when LOCEN is equal to 0. If the LOCF bit in the SYNSR indicates a loss-of-clock condition, setting the LOCRE bit causes an immediate reset. The LOCRE bit has no effect in PLL Off mode. 0 Assert reset on loss of clock is disabled. 1 Assert reset on loss of clock.
11 LOLIRQ	Loss-of-Lock Interrupt Request. The LOLIRQ bit determines how the integration module handles a loss-of-lock indication. See Section 8.6.1, “Loss-of-Lock Interrupt Request,” for more information. When operating in normal mode, the PLL must be locked before setting the LOLIRQ bit. Otherwise an interrupt is immediately requested. The LOLIRQ bit has no effect in PLL Off mode. 0 Request interrupt is disabled. 1 Request interrupt.
12 LOCIRQ	Loss- of-Clock Interrupt Request. The LOCIRQ bit determines how the integration module handles a loss-of-clock condition when LOCEN = 1. LOCIRQ has no effect when LOCEN = 0. If the LOCF bit in the SYNSR indicates a loss-of-clock condition, setting (or having previously set) the LOCIRQ bit causes an interrupt request. The LOCIRQ bit has no effect in PLL Off mode. 0 Request interrupt on loss of clock is disabled. 1 Request interrupt on loss of clock.
13	Reserved
14–15 ERATE[1]	Enhanced Modulation Rate. The ERATE bits control the rate of frequency modulation applied to the system frequency. Table 8-7 shows the allowable modulation rates.
16 CLKCFG_DIS	The CLKCFG_DIS bit is used to disable the ability to change the PLL mode using the CLKCFG bits. This protects the system from errant software writes and/or bit flips on the CLKCFG[2:0] bits that could change the PLL clock mode. **Note:** If the PLL is configured for PLL Off mode when the CLKCFG_DIS bit is set, the PLL will automatically enter normal mode. For this reason, it is advisable to set the PLL for the desired mode (normal mode with crystal reference or normal mode with external reference) before setting the CLKCFG_DIS bit to protect from inadvertent mode changes. 0 Writes to CLKCFG[2:0] enabled. 1 Writes to CLKCFG[2:0] disabled.
17–20	Reserved
21–23 EDEPTH	Enhanced Modulation Depth. The EDEPTH bit field controls the frequency modulation depth, and in conjunction with the SYNFMCR[FMDAC_EN] bit enables frequency modulation. The EDEPTH bit must be set to a non-zero value for FM operation. The sequence for enabling and configuring FM operation is described in Section 8.4.3.4.2, “Programming System Clock Frequency With Frequency Modulation””. This program sequence must be followed exactly to insure proper operation of the FM.
24–25	Reserved

Table 8-6. ESYNCR2 Bit Field Descriptions (continued)

Field	Description
26–31 ERFD	Enhanced Reduced Frequency Divider. The ERFD bits control a divider at the output of the PLL. The value specified by the ERFD bits establish the divisor applied to the PLL frequency. The ERFD divides the output clock by the quantity (ERFD + 1). Even-numbered ERFD settings, which would result in odd divide ratios, are not allowed. The decimal equivalent of the ERFD binary number is substituted into the equation from Table 8-10. **Note:** The ERFD divides the output clock by the quantity (ERFD + 1). Even numbered ERFD settings, which would result in odd divide ratios, are invalid and cause the PLL to produce an unpredictable output clock. The PLL output clock must be within the f_{PLL} specification (see the *MPC5676R Microcontroller Data Sheet*). Changing the ERFD bits does not affect the PLL, hence, no re-lock delay is incurred. Resulting changes in clock frequency are synchronized to the next falling edge of the current system clock. These bits should be written only when the lock bit (LOCK) is set, to avoid surpassing the allowable system operating frequency. In PLL Off mode, the ERFD bits have no effect. The available output divider ratios are given in Table 8-8.

[1] ERATE and EDEPTH must be enabled simultaneously to avoid unintentional assertion of the LOLF. Program the desired modulation rate and depth to the ERATE and EDEPTH bit fields simultaneously with a single 32 bit write to the ESYNCR2 register.

Table 8-7. Programmable Modulation Rates

ERATE	Modulation Rate (Hz)
00	$F_{mod} = F_{extal}/80$
01	$F_{mod} = F_{extal}/40$
10	$F_{mod} = F_{extal}/20$
11	Invalid

Table 8-8. Output Divide Ratios

ERFD	Output Divide Ratio (ERFD+1)
00_0000	Divide-by-1
00_0001	Divide-by-2
00_0010	Invalid
00_0011	Divide-by-4
00_0100	Invalid
00_0101	Divide-by-6
00_0110	Invalid
00_0111	Divide-by-8 (default value for MPC5676R)
. . .	. . .
11_1100	Invalid

Table 8-8. Output Divide Ratios

ERFD	Output Divide Ratio (ERFD+1)
11_1101	Divide-by-62
11_1110	Invalid
11_1111	Divide-by-64

8.3.2.4 FMPLL Synthesizer FM Control Register(SYNFMCR)

The synthesizer FM control register (SYNFMCR) contains bits for enabling and configuring PLL frequency modulation.

Offset: FMPLL_BASE_ADDR + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W		FMDAC_EN										FMDAC_CTL				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-5. FMPLL Synthesizer FM Control Register (SYNFMCR)

Table 8-9. ESYNFMCR Bit Field Descriptions

Field	Description
0	Reserved
1 FMDAC_EN	Frequency Modulation Register Enable. When this bit is set, the FMDAC_CTL field is enabled and the FM depth can be controlled directly by the value in FMDAC_CTL. The ESYNCR2[EDEPTH] field must also be set to a non-zero value to enable FM. 0 FMDAC_CTL disabled. 1 FMDAC_CTL enabled. DAC is controlled by the value in FMDAC_CTL.
2–10	Reserved
11–15 FMDAC_CTL	Digital-to-Analog Converter Control. This bit-field value is written to the DAC to control the FM depth by percentage during FM operation. 00100 – 1% 01000 – 2% 01100 – 3% 10000 – 4% These values have been shown in characterization data to produce the specified FM percentage within the device specification. However, the user may program intermediate values to trim the FM percentage for a specific application if desired. Do not program FMDAC_CTL to any value that will cause the system frequency to exceed the maximum specification.
16–0	Reserved

8.4 Functional Description

The FMPLL module contains the frequency modulated phase lock loop (FMPLL), enhanced frequency divider (ERFD), enhanced synthesizer control registers (ESYNCR1 and ESYNCR2), synthesizer status register (SYNSR), synthesizer FM control register (SYNFMCR) and clock/PLL control logic. The block also contains a reference frequency pre-divider controlled by the EPREDIV bits in the ESYNCR1. This enables the user to use a high frequency crystal or external clock generator and obtain finer frequency synthesis resolution than would be available if the raw input clock were used directly by the analog loop. For the remainder of this chapter, the term “reference frequency” and the symbol F_{ref} indicate the output of the pre-divider circuit. This is the clock on which frequency multiplication is performed.

8.4.1 General

The system clock source is determined during reset as shown in Table 8-13. The value of the PLLCFG[0:1] pins are latched during reset. If PLLCFG[0:1] are changed during a reset other than power-on reset, the internal clocks may glitch as the clock source is changed between PLL Off mode and PLL clock mode or from one PLL clock mode to another. Whenever PLLCFG[0:1] are changed in reset to a value other than what it was before the reset, an immediate loss of lock condition is declared. This only applies if the PLL was running in a locked state prior to the assertion of reset and change of PLLCFG[0:1].

Table 8-10 shows the PLL clock to input clock frequency relationships for the available clock modes.

Table 8-10. Clock-Out vs. Clock-In Relationships

Clock Mode	Frequency Equation
PLL Off Mode	$F_{PLL} = F_{extal}$
Normal PLL Mode[1]	$F_{PLL} = \frac{F_{extal} \bullet (EMFD + 16)}{(EPREDIV + 1)(ERFD + 1)}$

[1] Equation to be used when programming enhanced control registers (ESYNCR1 and ESYNCR2). See EPREDIV, EMFD, and ERFD bit-field descriptions for valid ranges for these fields.

8.4.2 PLL Off Mode

When PLL Off mode is selected, the PLL is turned off. The user must supply an external clock on the EXTAL pin and select that clock source before entering PLL Off mode. The selected clock is directly used to produce the various system clocks. Refer to *MPC5676R Microcontroller Data Sheet* for external clock input requirements. In PLL Off mode, the analog portion of the PLL is disabled, the frequency modulation capability is not available, and no clocks are generated at the PLL output. The pre-divider is bypassed and has no effect on the system clock frequency in PLL Off mode.

8.4.3 Normal Mode

When normal PLL mode is selected, the PLL is fully programmable. The PLL can synthesize frequencies ranging from 48x to 148x the reference frequency of the output of the predivider, with or without frequency modulation enabled. The post-divider is capable of reducing the PLL clock frequency without forcing a re-lock. The PLL reference can be a crystal oscillator reference or an external clock reference. This clock is divided by the pre-divider circuit to create the PLL reference clock.

8.4.3.1 PLL Lock Detection

The lock detect logic monitors the reference frequency and the PLL feedback frequency to determine when frequency lock has been achieved. Phase lock is inferred by the frequency relationship, but is not guaranteed. The PLL lock status is reflected in the LOCK status bit in the SYNSR. A sticky lock status indication, LOCKS, is also provided.

The lock detect function uses two counters, which are clocked by the reference and PLL feedback respectively. When the reference counter has counted N cycles, the feedback counter’s count is compared. If the feedback counter has also counted N cycles, the process is repeated for N + K counts. Then if the two counters’ counts match, the lock criteria is relaxed by one count and the system is notified that the PLL has achieved frequency lock. After three successful compares, the tolerance is relaxed.

After lock has been detected, the lock circuitry continues to monitor the reference and feedback frequencies using the alternate count and compare process. If the counters do not match at any comparison time, then the LOCK status bit is cleared to indicate that the PLL has lost lock. At this point, the lock criteria is tightened and the lock detect process is repeated.

The alternate count sequences prevent false lock detects due to frequency aliasing while the PLL tries to lock. Alternating between a tight and relaxed lock criteria prevents the lock detect function from randomly toggling between locked and not locked status due to phase sensitivities. Figure 8-6 illustrates the sequence for detecting locked and not-locked conditions.

When the frequency modulation is enabled, the loss of lock continues to function as described but with the lock and loss of lock criteria reduced to ensure that false loss of lock conditions are not detected.

In PLL Off mode, the PLL cannot lock because the PLL is disabled.

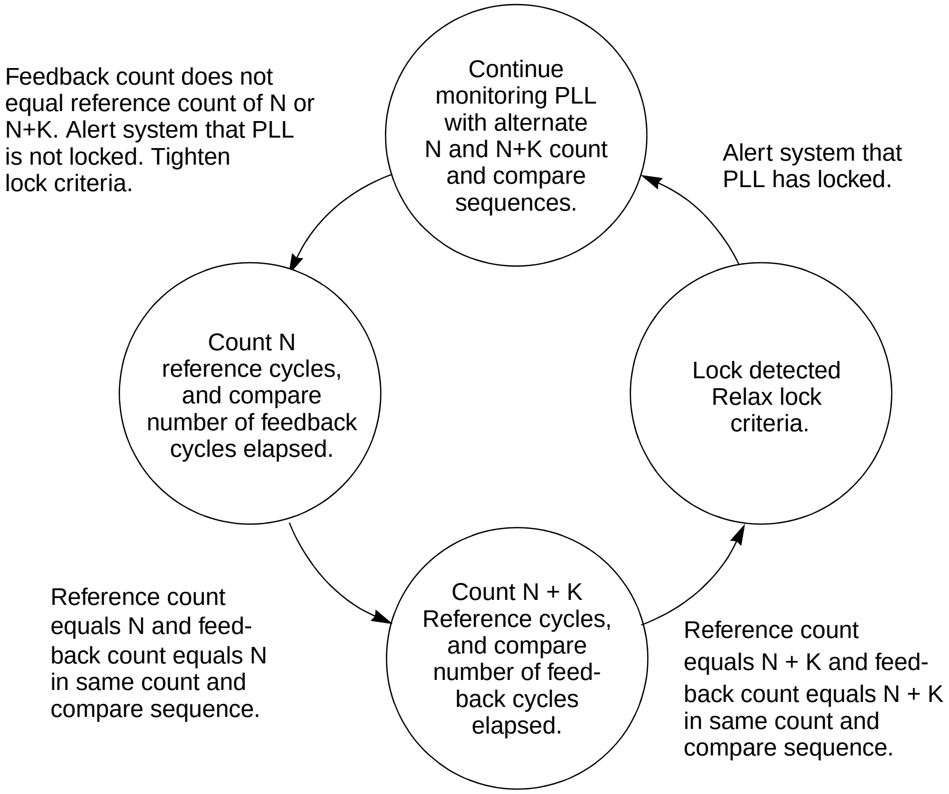


Figure 8-6. Lock Detect Sequence

After the PLL acquires lock after reset, the LOCK and LOCKS status bits are set. If the EPREDIV or EMFD are changed, or if an unexpected loss-of-lock condition occurs, the LOCK and LOCKS status bits are negated. While the PLL is in an unlocked condition, the system clocks continue to be sourced from the PLL as the PLL attempts to re-lock. Consequently, during the re-locking process, the system clock frequency is not well defined and may exceed the maximum system frequency violating the system clock timing specifications. Because of this condition, use of the loss-of-lock reset function is recommended.

After the PLL has re-locked, the LOCK bit is set. The LOCKS bit remains cleared if the loss of lock was unexpected. The LOCKS bit is set to one when the loss of lock was caused by changing the EPREDIV or EMFD fields.

8.4.3.2 Loss-of-Clock Detection

When enabled by the LOCEN bit in the ESYNCR2, the loss-of-clock (LOC) detection circuit monitors the input clocks to the phase/frequency detector (PFD) (see Figure 8-1). When the reference or feedback clock frequency falls below a minimum frequency, the LOC circuitry considers the clock to have failed and a loss-of-clock status is reflected by the sticky LOCF bit, and non-sticky LOC bit in the SYNSR. See *MPC5676R Microcontroller Data Sheet* for the minimum clock frequency. In PLL Off mode, the loss-of-clock circuitry is disabled.

Depending which clock source has failed, the LOC circuitry switches the PLL output clock’s source to the remaining operational clock if enabled by LOCEN. The PLL output clocks are derived from the alternate clock source until reset is asserted. The alternate clock source used is dependent on whether the LOC is caused by a reference clock failure or a PLL failure. If the reference fails, the PLL goes out of lock and into self-clocked mode (SCM) (see Table 8-11). The PLL remains in SCM until the next reset. When the PLL is operating in SCM, the PLL runs open loop at a default VCO frequency. The RFD will set to divide-by-6 to ensure the clock presented to the system is well below the maximum allowable frequency for the device. If the loss-of-clock condition is due to a PLL failure (i.e., loss of feedback clock), the PLL reference becomes the system clock source until the next reset, even if the PLL regains itself and re-locks.

Table 8-11. Loss-of-Clock Summary

Clock Mode	System Clock Source before Failure	REFERENCE FAILURE Alternate Clock Selected by LOC Circuitry until Reset	PLL FAILURE Alternate Clock Selected by LOC Circuitry until Reset
PLL	PLL	PLL self-clocked mode	PLL reference
PLL Off	Ext. Clock(s)	None	NA

Note: The LOC circuit monitors the inputs to the PFD: reference and feedback clocks (see Figure 8-1).

A special loss-of-clock condition occurs when both the reference and the PLL fail. The failures may be simultaneous or the PLL may fail first. In either case, the reference clock failure takes priority and the PLL attempts to operate in SCM. If successful, the PLL remains in SCM until the next reset. During SCM, modulation is always disabled. If the PLL cannot operate in SCM, the system remains static until the next reset. Both the reference and the PLL must be functioning properly to exit reset.

8.4.3.3 PLL Normal Mode Without FM

In PLL mode, the system clocks are synthesized by the FMPLL by multiplying up the reference clock frequency. It is critical that the system clock frequency remain within the range for the device (see *MPC5676R Microcontroller Data Sheet*). The output of the FMPLL can be divided down in powers of 2 up to 64 to reduce the system frequency with the ERFD. The ERFD is not contained in the feedback loop of the PLL, so changing the ERFD bits does not affect FMPLL operation. Finally, the PLL can be frequency modulated to reduce electromagnetic interference often associated with clock circuitry. Figure 8-1 shows the overall block diagram for the PLL. Each of the major blocks is discussed briefly in the following sections.

8.4.3.3.1 Phase/Frequency Detector

The phase/frequency detector (PFD) is a dual-latch phase-frequency detector. It compares both the phase and frequency of the reference clock and the feedback clock. The reference clock comes from the crystal oscillator or an external clock source. The feedback clock comes from the VCO output divided down by the EMFD in normal PLL mode.

When the frequency of the feedback clock equals the frequency of the reference clock (i.e., the PLL is frequency locked), the PFD pulses the UP or DOWN signals depending on the relative phase of the two clocks. If the falling edge of the reference clock leads the falling edge of the feedback clock, then the UP signal is pulsed. If the falling edge of the feedback clock leads the falling edge of the reference clock, then the DOWN signal is pulsed. The width of these pulses relative to the reference clock is dependent on how much the two clocks lead or lag each other. After phase lock is achieved, the PFD continues to pulse the

UP and DOWN signals for a very short duration during each reference clock cycle. These short pulses force the PLL to continually update and prevent a frequency drift phenomena referred to as "dead-banding." Dead-band describes the minimum amount of phase error between the reference and feedback clocks that a phase detector cannot correct.

8.4.3.3.2 Charge Pump/Loop Filter

Operation of the charge pump is controlled by the UP and DOWN signals from the PFD. They control whether the charge pumps apply or remove charge, respectively, from the loop filter.

8.4.3.3.3 VCO

The voltage into the VCO controls the frequency of its output. The frequency-to-voltage relationship (VCO gain) is positive.

8.4.3.3.4 EMFD

The MFD divides down the output of the VCO and feeds it back to the PFD. The PFD controls the VCO frequency (via the charge pump and loop filter) such that the reference and feedback clocks have the same frequency and phase. Thus, the input to the MFD, which is also the output of the VCO, is at a frequency that is the reference frequency multiplied by the same amount the MFD divides by. For example, if the MFD divides the VCO frequency by 48, then the PLL is frequency locked when the VCO frequency is 48 times the reference frequency. The presence of the MFD in the loop allows the PLL to perform frequency multiplication, or synthesis.

8.4.3.3.5 Programming System Clock Frequency

In normal PLL clock mode, the default system frequency is determined by the default EPREDIV, EMFD, and ERFD values.

When programming the PLL, do not violate the maximum system clock frequency or max/min VCO frequency specifications. Based on the desired system clock frequency, EPREDIV, EMFD, and ERFD must be calculated for the given crystal or external reference frequency. See *MPC5676R Microcontroller Data Sheet* for the max/min VCO frequency range and the maximum allowable system frequency.

Frequency modulation should be disabled prior to changing the EPREDIV, EMFD, or ERATE bit fields. A change to EPREDIV, EMFD, EDEPTH, or ERATE while modulation is enabled invalidates the previous FM depth configuration.

Use these directions to accommodate the frequency overshoot that occurs when the EPREDIV or EMFD bits are changed. If frequency modulation is going to be enabled the maximum allowable frequency must be reduced by the programmed ΔF_m.

1. Determine the appropriate value for the EPREDIV, EMFD, and ERFD fields in the synthesizer control register(s), remember to include the ΔF_m if frequency modulation is to be enabled. The amount of jitter in the system clocks can be minimized by selecting the maximum EMFD factor that can be paired with an ERFD factor to provide the desired frequency. The maximum EMFD value that can be used is determined by the VCO and EMFD range.

2. Write a value of ERFD = ERFD (from step 1) + 2 to the ERFD field of the ESYNCR2. Not increasing the ERFD when changing the EPREDIV or EMFD could subject the device to clock frequencies beyond the range specified for the device due to the PLL's unlocked state.
3. If frequency modulation is currently enabled, disable it by writing 00 to the EDEPTH field of the ESYNCR2.
4. If programming the EPREDIV and/or EMFD, write the value(s) determined in step 1 to the appropriate field(s) in the ESYNCR1.
5. Monitor the synthesizer lock bit (LOCK) in the synthesizer status register (SYNSR). When the PLL achieves lock, write the ERFD value determined in step 1 to the ERFD field of the ESYNCR2. This changes the system clocks frequency to the desired frequency. If frequency modulation is desired, leave ERFD programmed to ERFD + 2 until after completing the steps in Section 8.4.3.4.2, "Programming System Clock Frequency With Frequency Modulation."
6. If frequency modulation was enabled initially, it can be re-enabled following the steps listed in Section 8.4.3.4.2, "Programming System Clock Frequency With Frequency Modulation."

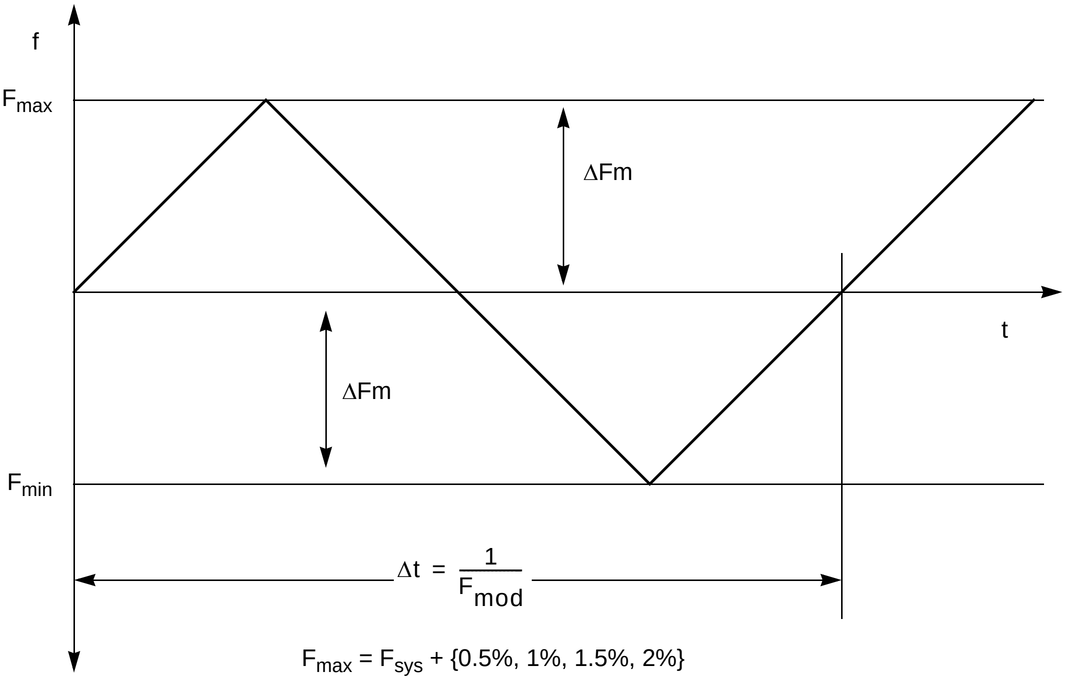
During startup, current transients on the VDD supply are related to the system frequency. A technique can be used to reduce these current transients when the system frequency is changed from its default value to your desired frequency.

Follow the above procedure for step 1. In step 2, rather than set ERFD to ERFD (from step 1) + 2, set this to a value which will produce a low system frequency (close to the default system frequency), e.g. ERFD = ERFD (from step 1) + 4. Once set, follow steps 3 and 4 as above. In step 5, wait for the LOCK bit to set, then set the EFRD bit to ERFD (from step 2) – 2. Wait for a small duration of time for the current to stabilize, then repeat this procedure until the ERFD value is equal to the value determined in step 1.

Using this technique you should observe the system frequency increasing in steps to the desired system frequency. This results in the VDD current increasing to its equivalent final value in smaller current steps which, therefore, produce smaller current transients, making it easier for the power supply to handle.

8.4.3.4 PLL Normal Mode With Frequency Modulation

In normal PLL clock mode, frequency modulation is not enabled in the default synthesis mode. When frequency modulation is enabled several parameters must be set to generate the desired level of modulation. The parameters to be programmed are the ERATE and EDEPTH bit fields of the ESYNCR2 register and the FMDAC_EN and FMDAC_CTL bits in the SYNFMCR register. The ERATE bit controls the frequency of modulation, F_{mod}. The EDEPTH bits work in conjunction with the FMDAC_CTL bits in the SYNFMCR to enable and control the modulation depth, F_m. The available modulation rates and depths are given in Table 8-7 and Table 8-8, respectively. The modulation waveform is always a triangle wave and its shape is not programmable. An example of one period of the modulation waveform is shown in Figure 8-7.



$F_{max} = F_{sys}$ + {0.5%, 1%, 1.5%, 2%}
$F_{min} = F_{sys}$ – {0.5%, 1%,1.5%, 2%}
$F_{mod} = F_{extal}/Q$ where Q = {20, 40, 80}

Figure 8-7. Frequency Modulation Waveform

8.4.3.4.1 Frequency Modulation Control

The frequency modulation control consists of programming a reference current into the modulation D/A so that the modulation depth (F_{max} and F_{min}) remains within specification. Disable frequency modulation prior to changing the EPREDIV, EMFD, or ERATE bit fields. Upon enabling frequency modulation a new configuration sequence is required. Do not change EPREDIV, EMFD or ERATE while modulation is active or unpredictable results may occur.

8.4.3.4.2 Programming System Clock Frequency With Frequency Modulation

The following steps illustrate proper programming of the frequency modulation mode. These steps ensure proper operation of the FM configuration routine and prevent frequency overshoot from the programming sequence. The PLL should be programmed and allowed to lock in non-FM mode at the desired frequency as outlined in Section 8.4.3.3.5, “Programming System Clock Frequency.”

1. Write a value of ERFD = ERFD + 2 to the ERFD field of the ESYNCR2 to ensure the maximum system frequency is not exceeded during the FM configuration. This is done when allowing the PLL to lock in non-FM mode.

NOTE

Even numbered ERFD settings, which would result in odd divide ratios, are invalid and cause the PLL to produce an unpredictable output clock. The PLL output clock must be within the f_{PLL} specification

2. Write the SYNFMCR[FMDAC_EN] bit to logical 1 (to enable FMDAC_CTL) and the SYNFMCR[FMDAC_CTL] bit field to the appropriate value shown in Table 8-12.

Table 8-12. FMDAC_CTL settings for FM Configuration

Peak-to-Peak FM depth (EDEPTH)	FMDAC_CTL
1% (center frequency ±0.5%)	0x04
2% (center frequency ±1.0%)	0x08
3% (center frequency ±1.5%)	0x0C
4% (center frequency ±2.0%)	0x10

3. Program the desired modulation rate into the ERATE field of the ESYNCR2 register and set the EDEPTH field to a non-zero value. The absolute value in the EDEPTH field is non-critical as this value is not used for actual FM depth, however EDEPTH must be non-zero to enable FM. Make sure not to change ERFD from step 2 when setting ERATE and EDEPTH as they share the same register space.
4. Wait for the PLL to lock. When the PLL achieves lock, write the desired ERFD value. Make sure not to modify ERATE/EDEPTH as they share the same register space.

The frequency modulation system is dependent on several factors, including the accuracies of the V_{DDSYN}/V_{SSSYN} voltage, of the crystal oscillator frequency, and of the manufacturing variation.

For example, if a 5% accurate supply voltage is used, then a 5% modulation depth error results. If the crystal oscillator frequency is skewed from the nominal operating frequency, the resulting modulation frequency is proportionally skewed. Finally, the error due to the manufacturing and environment variation alone can cause the frequency modulation depth error to be greater than 20%.

8.5 Resets

This section describes the reset operation of the PLL, including power-on reset and normal resets. The reset values of registers and signals are provided in other sections.

8.5.1 Clock Mode Selection

The initial clock mode is reflected in the MODE, PLLSEL, and PLLREF bits of the synthesizer status register (SYNSR) as well as the ESYNCR1[CLKCFG[2:0]] bit field. The clock mode can be modified by writing to the CLKCFG[2:0] bit field. The synthesizer status register then reflects the newly-selected PLL clock mode.

Table 8-13 summarizes clock mode selection.

Table 8-13. Clock Mode Selection

Package Pins[1]		Clock Mode	Synthesizer Status Register (SYNSR) MODE, PLLSEL, and PLLREF Bits		
PLLCFG[0]	PLLCFG[1]		MODE/ CLKCFG[2]	PLLSEL/ CLKCFG[1]	PLLREF/ CLKCFG[0]
0	0	PLL Off mode	0	X	X
0	1	Normal mode with external reference	1	1	0
1	0	Normal mode with crystal reference	1	1	1
1	1	Reserved	1	0	0

[1] The PLLCFG[2] pin configures the crystal oscillator range:
PLLCFG[2] = 0, for 8 MHz to 20 MHz
PLLCFG[2] = 1, for 40 MHz

8.5.1.1 Power-On Reset (POR)

The PLL will not operate until the POR state has ended. Refer to *MPC5676R Microcontroller Data Sheet* for these thresholds. At this point, the PLL operates in self-clocked mode (SCM) until a valid reference clock is detected by the internal clock monitor circuit.

Internal to the PLL, the VCO is held in reset until the negation of the POR signal. This prevents the PLL from attempting to lock before its supplies are within specification, which can cause VCO/loop gain to be lower than what the analog loop is designed for.

8.5.1.2 External Reset

After POR has negated, the PLL will begin its lock detect algorithm if Normal Mode is selected. However, if a valid reference is not present, the PLL will continue to operate in Self Clocked Mode until one is present. PLL configuration at POR may be selected by external pins PLLCFG[0:1] or reset state values of configuration registers.

After the initial lock with the default divider settings (assuming Normal Mode was selected), you may write to the SYNCR/ESYNCR(s) to modify the dividers for the desired operating frequency. The PLL may not be able to lock due to an (E)MFD and crystal frequency combination that attempts to force the VCO outside of its operating range

CAUTION

> When running in an unlocked state, the clocks generated by the PLL are not guaranteed stable and may exceed the maximum specified operating frequency of the device. The RFD should always be used as described in Section 8.4.3.3.5, “Programming System Clock Frequency,” to insulate the system from any potential frequency overshoot of the PLL clocks.

8.5.2 PLL Loss-of-Lock Reset

By programming the LOLRE bit in the ESYNCR2, the PLL can assert reset when a loss-of-lock condition occurs. Because the LOCK and LOCKS bits in the SYNSR are re-initialized after reset, the SIU reset status register described in Section 3.2.1.2, “Reset Status Register (SIU_RSR),” must be read to determine a loss-of-lock condition occurred.

In PLL Off mode, the PLL cannot lock; therefore a loss-of-lock condition cannot occur and LOLRE has no effect.

8.5.3 PLL Loss-of-Clock Reset

When a loss-of-clock condition is recognized, $\overline{\text{RESET}}$ is asserted if the LOCRE bit in the SYNCR is set. The LOCF and LOC bits in the SYNSR are cleared after reset, therefore, the LOC bit must be read in the SIU_RSR to determine that a loss-of-clock condition occurred. LOCRE has no effect in PLL Off mode.

8.6 Interrupts

This section describes the interrupt requests that the PLL can generate.

8.6.1 Loss-of-Lock Interrupt Request

By setting the LOLIRQ bit in the ESYNCR2, the PLL can request an interrupt when a loss-of-lock condition occurs.

In PLL Off mode, the PLL cannot lock; therefore a loss-of-lock condition cannot occur and the LOLIRQ has no effect.

8.6.2 Loss-of-Clock Interrupt Request

When a loss-of-clock condition is recognized, the PLL requests an interrupt if the LOCIRQ bit in the SYNCR is set. The LOCIRQ bit has no effect in PLL Off mode or if LOCEN is equal to 0.

Chapter 9
Power Management Controller (PMC)

9.1 Introduction

The power management controller (PMC) is compatible with 3.3V or 5V operation. It provides voltage regulation to the microcontroller and includes advanced power on reset (POR) and precision low voltage detector (LVD) monitors that guarantee safe device operation.

9.1.1 Features

The PMC contains the following features (see Section 9.1.1.1, “Features of the Analog Portion of PMC_SMPS”, and Section 9.1.1.2, “Features of the Digital Portion of PMC_SMPS”, for detailed analog and digital features of this device):

- Compatible for both 5V and 3.3V operations.
- A Switched Mode Power Supply (SMPS) Buck regulator and a Low Drop Out (LDO) linear regulator, sharing the same control pin (REGSEL).
- SMPS regulator is selected when the REGSEL pin is connected to VDDREG. A 5V nominal supply voltage is recommended to operate and defines the LVD level to 5V.
- LDO regulator is selected when the REGSEL pin is connected to VSS. A 3V nominal supply voltage is recommended to operate and defines the VDDREG LVD level to 3V.
- High precision Low Voltage Detector (LVD) monitor for PMC supply voltage VDDREG, VDD core voltage supply, and VDDSYN.
- A low voltage band gap generates the reference voltages and currents for voltage regulators and LVDs.
- A Power On Reset (POR) monitor is used to check main regulator supply VDDREG and core supply VDD and guarantees proper system function even at very low voltage supply levels.
- No power sequencing constraint required.
- An independent internal regulator generates 1.2V supply voltage for the low voltage digital portion of the PMC.
- A 5V to 3.3V LDO regulator is enabled when the PMC is in SMPS5V mode and when the PMC is in LDO mode with nominal supply voltage of 5V (LDO5V mode). The LDO 5V to 3V is disabled when VDDREG is 3.3V nominal and in this case VDDREG and VDDSYN must be connected together (LDO3V mode).
- A loose precision temperature sensor detects over-temperature conditions in the PMC and adjacent area. Its low accuracy requires that temperature is checked with the included precision temperature sensor before taking any corrective action.
- Direct measurement of PMC internal voltages is available at predefined ADC channels.

9.1.1.1 Features of the Analog Portion of PMC_SMPS

The analog portion of PMC_SMPS has the following features:

- Embedded voltage regulator (2.7 V up to 5.5 V regulator supply (VDDREG)) controller for generating core VDD voltage. For the correct allowed voltage range refer to *MPC5676R Microcontroller Data Sheet*.
- The PMC supports two regulation modes (linear and switch mode), which are controlled by REGSEL pin.
- When VDDREG is above 4.0 V roughly, the 3.3 V internal linear regulator is enabled and regulates the VDDSYN ball to a nominal 3.3 V, sense point is taken on VDD33. VDDSYN and VDD33 must be connected/shorted together with minimum impedance (no resistors or inductors allowed).
- Monitors the following supplies:
  - core voltage pin that connects in the package to core voltage
  - 3.3V via the VDD33 pin (input to the 3.3V network)
  - 5V on VDDREG
- Disabling 3.3 V internal regulator
  - See Section 9.5.4, “3.3V Internal Voltage Regulator”. If VDDREG is 5 V, you need to drive VDD33/VDDSYN at 3.5 V +/- 3% and program the PMC to regulate the 3.3 V at minimum voltage by programming the correspondent control nibble to eliminate contention between the two regulators.

9.1.1.2 Features of the Digital Portion of PMC_SMPS

The digital portion of the PMC_SMPS has the following features:

Software interfaces for:

- low voltage detects and POR circuits from the PMC analog blocks plus LVD from VDDSYN segment as 3.3V LVD
- low voltage detects for reset segment LVD (VDDEH1)
- POR level of reset pin segment (VDDEH1)
- low voltage detects for other IO segment LVD (VDDEH3, VDDEH4, VDDEH5, VDDEH6, VDDEH7)
- low voltage detect on VDDA (from ADC band gap reference)
- standby regulator brownout circuit

9.1.2 Block Diagram

The Figure 9-1 is the block diagram for the PMC

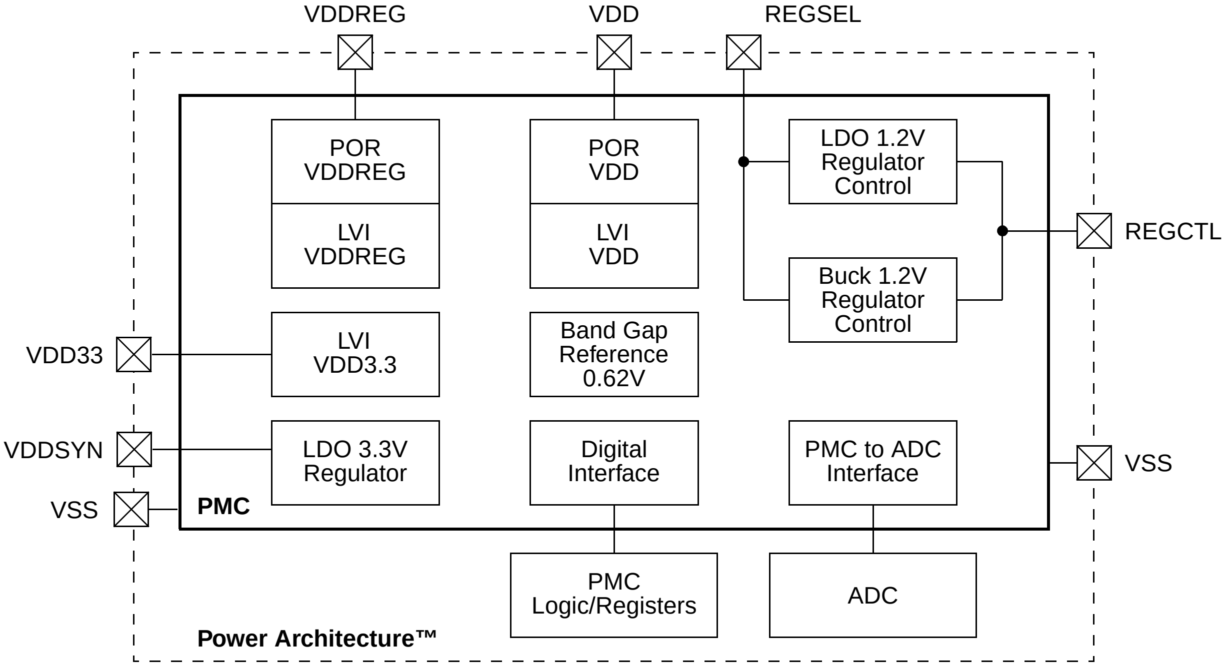


Figure 9-1. Power Management Controller Block Diagram

9.1.3 PMC Operation Modes

There are three modes of operation and relative configuration used at board level to choose which regulators are active from start up on and which levels for LVD should be enforced during operation. PMC operation modes are resumed in Table 9-1.

Table 9-1. PMC Operation Modes

Short Mode	Full Name	Description
LDO3V	3.3V Voltage Supply, LDO regulator enabled	When external pin REGSEL is connected to VSS -it has a weak internal resistive pull down- and regulator supply voltage is 3.3V nominal, the Linear 1.2V VRC is enabled and the internal 3.3V regulator is disabled with tri-stated output. VDDSYN must be connected to VDDREG. An external ballast transistor is expected on REGCTL as described in 1.2V LDO regulator section. VDDREG LVD selected to 3V nominal.

Table 9-1. PMC Operation Modes

Short Mode	Full Name	Description
LDO5V	5V Voltage Supply, LDO regulator enabled	When external pin REGSEL is connected to VSS -it has a weak internal resistive pull down- and regulator supply voltage is 5V nominal, the Linear 1.2V VRC is enabled, as the internal 3.3V regulator. An external ballast transistor is expected on REGCTL as described in 1.2V LDO regulator section. An external decoupling capacitor is expected on VDDSYN (details in the 3.3V regulator section). VDDREG LVD selected to 3V nominal.
SMPS5V	5V Voltage Supply, SMPS regulator enabled	When external pin REGSEL is connected to VDDREG and regulator supply voltage is 5V nominal, the Switched Mode VRC is enabled, and the internal 3.3V regulator is enabled. An external MOS - Schottky device is expected on *REGCTL* as described in SMPS regulator section. A decoupling capacitor must be used on VDDSYN (details in the 3.3V regulator section). VDDREG LVD selected to 5V nominal.

9.2 External Signals Description

9.2.1 Signals Information

The following table describes the PMC signals and their properties (also refer to the Signals chapter).

Table 9-2. PMC Signals

Pin Name	Type	Nominal Voltage (V)
VDDREG	supply	3.3 or 5
VSS	supply	ground
VDD	supply	1.2
VDDSYN	supply	3.3
REGSEL	input	ground or VDDREG
REGCTL	output	0V – 5V PWM signal in SMPS mode About VDD+0.7V in LDO mode.
VSS	supply	ground
VDD33	input	3.3

9.3 Signals Details

The following sections detail the signals used by the PMC.

9.3.1 VDDREG

Positive analog power supply for PMC and voltage regulators. It can be nominal 5V or nominal 3.3V. It supplies internal regulators and LVDs.

Voltage range VDDR for 5V operation and VDD33 for 3V operation can be found in the *MPC5676R Microcontroller Data Sheet*.

Requires a decoupling cap of 5 – 20 uF between VDDREG and VSS as close as possible to the pins to minimize board parasitcs.

9.3.2 VDD

Positive digital power supply for core voltage. Nominal value is 1.2V, voltage range is VDD12. Decoupling capacitance configuration depends on selected regulator (LDO or SMPS). More details available in the correspondent regulator description and in Section 9.7.2, “Hardware Design Recommendations”.

9.3.3 VDDSYN

Positive 3.3V regulator output - power supply, voltage range VDD33, usually tied flash. Internal regulator is ON in SMPS5V mode or in LDO5V mode. The 3.3V regulator is OFF in LDO3V mode with VDDREG in the 3.3V range. When enabled, the VDDSYN regulator can source up to 80 mA keeping the regulation in the 3.3V range. See the *MPC5676R Microcontroller Data Sheet* for more details. Sense voltage is taken at VDD33, hence it is recommended to short VDDSYN and VDD33 with low impedance short track.

When the internal regulator is disabled the pin has a weak 35 kΩ pull down resistor and VDDSYN must be connected to VDDREG. Requires an external capacitor of 200 nF - 4 uF (depending on the application), on the pad or as closest as possible with negative connection to VSS.

9.3.4 VSS

Negative digital power supply.

PMC substrate connection.

9.3.5 REGCTL

VRC 1.2V output that connects to the base of ballast NPN in LDO3V or LDO5V mode, or to the gate of the n-MOS in SMPS5V mode.

9.3.6 REGSEL

Analog input that selects 1.2 VRC operation. It can be connected only to VDDREG or VSS.

When connected to VDDREG it enables the SMPS regulator, and requires a nominal supply voltage of 5V. Else the LDO regulator is enabled and both 3.3V and 5V operation are allowed. A weak 200kΩ pull down resistor keeps the default level at 0V when pin is floating.

9.3.7 VDD33

Sense point for LVD and regulator feedback of the 3.3V VDDSYN analog supply.

Input that produces 3.3V regulator reference and LVD 3.3V reference.

Must be shorted to VDDSYN with low impedance connection.

9.4 Memory Map/Register Definition

Table 9-3 shows the PMC memory map. The PMC memory maps 3 registers for configuring, monitoring, and trimming the LVD monitors.

Table 9-3. Power Management Controller Memory Map

Address[1]	Register	Bits	Access	Reset Value	Section/Page
PMC_BASE + 0x0000	PMC_MCR — Configuration register	32	R/W	0x9800_0000	9.4.1/9-6
PMC_BASE + 0x0004	PMC_TRIMR — Trimming register	32	R/W	0x0000_0006	9.4.2/9-8
PMC_BASE + 0x0008	PMC_SR — Status register	32	R/W	0x020U[2]_0000 or 0x060U[2]_0000[3]	9.4.3/9-12

[1] PMC_BASE = 0xC3FB_C000

[2] Undefined

[3] Reset value depends on whether RAM standby regulator switch reported a brownout condition.

9.4.1 Configuration Register (PMC_MCR)

The configuration register contains configuration and interrupt enable bits for the LVD monitors.

Refer to Section 9.1.1, “Features”, for a listing of which VDDEH powers are monitored.

Offset: PMC_BASE + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LVRER	LVREH	LVRE50	LVRE33	LVREC	LVREA	0	0	LVIER	LVIEH	LVIE50	LVIE33	LVIEC	LVIEA	0	TLK
W																
Reset	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-2. Configuration and Status Register (PMC_MCR)

Table 9-4. PMC_MCR Field Descriptions

Field	Description
0 LVRER	Reset-pin-supply low-voltage reset enable. This bit defines whether an LVD assertion on the supply of the I/O segment that contains the reset pin will generate system reset or not. 0 Disabled. LVD assertion on the supply of the I/O segment that contains the reset pin does not cause system reset. 1 Enabled. LVD assertion on the supply of the I/O segment that contains the reset pin causes system reset.
1 LVREH	VDDEH low-voltage reset enable. This bit defines whether an LVD assertion on any monitored VDDEH supply will generate system reset or not. 0 Disabled. LVD assertion on any monitored VDDEH supply does not cause system reset. 1 Enabled. LVD assertion on any monitored VDDEH supply causes system reset.

Table 9-4. PMC_MCR Field Descriptions (continued)

Field	Description
2 LVRE50	VDDREG low-voltage reset enable. This bit defines whether an LVD assertion on the VDDREG supply of the voltage regulator will generate system reset or not. 0 Disabled. LVD assertion on the VDDREG supply of the voltage regulator does not cause system reset. 1 Enabled. LVD assertion on the VDDREG supply of the voltage regulator causes system reset.
3 LVRE33	VDDSYN low-voltage reset enable. This bit defines whether an LVD assertion on the VDDSYN supply will generate system reset or not. 0 Disabled. LVD assertion on the VDDSYN supply does not cause system reset. 1 Enabled. LVD assertion on the VDDSYN supply causes system reset.
4 LVREC	Core-voltage-supply VDD low-voltage reset enable. This bit defines whether an LVD assertion on the core voltage VDD supply will generate system reset or not. 0 Disabled. LVD assertion on the core voltage supply does not cause system reset. 1 Enabled. LVD assertion on the core voltage supply causes system reset.
5 LVREA	VDDA low-voltage reset enable. This bit defines whether an LVD assertion on the analog power input VDDA1 will generate system reset or not. 0 Disabled. LVD assertion on the analog power input VDDA1 does not cause system reset. 1 Enabled. LVD assertion on the analog power input VDDA1 causes system reset.
6–7	Reserved
8 LVIER	Reset-pin-supply low-voltage interrupt enable. This bit enables the generation of the low-voltage interrupt request when the supply of the I/O segment that contains the reset pin falls below the corresponding LVD threshold. The low-voltage interrupt is independent from low-voltage reset. If both, interrupt and reset, are enabled, then reset and interrupt will be generated, but reset will then clear the interrupt. 0 Disabled. Low-voltage interrupt request is disabled. 1 Enabled. Low-voltage interrupt request is enabled.
9 LVIEH	VDDEH low-voltage interrupt enable. This bit enables the generation of the low-voltage interrupt request when any monitored VDDEH supply falls below the corresponding LVD threshold. The low-voltage interrupt is independent from low-voltage reset. If both, interrupt and reset, are enabled, then reset and interrupt will be generated, but reset will then clear the interrupt. 0 Disabled. Low-voltage interrupt request is disabled. 1 Enabled. Low-voltage interrupt request is enabled.
10 LVIE50	VDDREG low-voltage interrupt enable. This bit enables the generation of the low-voltage interrupt request when the VDDREG supply of the voltage regulator falls below the corresponding LVD threshold. The low-voltage interrupt is independent from low-voltage reset. If both, interrupt and reset, are enabled, then reset and interrupt will be generated, but reset will then clear the interrupt. 0 Disabled. Low-voltage interrupt request is disabled. 1 Enabled. Low-voltage interrupt request is enabled.
11 LVIE33	VDDSYN low-voltage interrupt enable. This bit enables the generation of the low-voltage interrupt request when the VDDSYN power supply gets below the corresponding LVD threshold. The low-voltage interrupt is independent from low-voltage reset. If both, interrupt and reset, are enabled, then reset and interrupt will be generated, but reset will then clear the interrupt. 0 Disabled. Low-voltage interrupt request is disabled. 1 Enabled. Low-voltage interrupt request is enabled.
12 LVIEC	Core-voltage-supply low-voltage VDD interrupt enable. This bit enables the generation of the low-voltage interrupt request when the core voltage supply gets below the corresponding LVD threshold. The low-voltage interrupt is independent from low-voltage reset. If both, interrupt and reset, are enabled, then reset and interrupt will be generated, but reset will then clear the interrupt. 0 Disabled. Low-voltage interrupt request is disabled. 1 Enabled. Low-voltage interrupt request is enabled.

Table 9-4. PMC_MCR Field Descriptions (continued)

Field	Description
13 LVIEA	VDDA low-voltage interrupt enable. This bit enables the generation of the low-voltage interrupt request when the analog power input VDDA1 falls below the corresponding LVD threshold. The low-voltage interrupt is independent from low-voltage reset. If both, interrupt and reset, are enabled, then reset and interrupt will be generated, but reset will then clear the interrupt. 0 Disabled. Low-voltage interrupt request is disabled. 1 Enabled. Low-voltage interrupt request is enabled.
14	Reserved
15 TLK	Trimming lock. This is a set-only bit that comes out of reset negated, and can be asserted one time after reset to lock the trimming register. Once asserted, it cannot be negated anymore. When TLK is asserted, the Trimming Register becomes read-only and cannot be changed until the next reset. 0 Trimming register can be written. 1 Trimming register is read-only.
16–31	Reserved

9.4.2 Trimming Register (PMC_TRIMR)

The trimming register allows the user to fine tune the voltage of the regulators and the LVD thresholds. It can only be written when the TLK bit of the PMC_MCR is negated. Once TLK has been asserted, this register becomes read-only until the next system reset.

Offset: PMC_BASE + 0x0004

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	LVDATRIM				LVDREGTRIM			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	VDD33TRIM				LVD33TRIM				VDDCTRIM				LVDCTRIM			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Figure 9-3. Trimming Register (PMC_TRIMR)

Table 9-5. PMC_TRIMR Field Descriptions

Field	Description
0–7	Reserved

Table 9-5. PMC_TRIMR Field Descriptions (continued)

Field	Description
8–11 LVDATRIM	LVD VDDA trimming. This field is used to fine tune the voltage threshold of the VDDA1 rising LVD, used to monitor the analog power input VDDA1. See the *MPC5676R Microcontroller Data Sheet* for details.0111LVDA + 140 mV. 0110 LVDA + 120 mV 0101 LVDA + 100 mV 0100 LVDA + 80 mV 0011 LVDA + 60 mV 0010 LVDA + 40 mV 0001 LVDA + 20 mV 0000 Nominal, start-up and default value LVDA 1111 LVDA – 20 mV 1110 LVDA – 40 mV 1101 LVDA – 60 mV 1100 LVDA – 80 mV 1011 LVDA – 100 mV 1010 LVDA – 120 mV 1001 LVDA – 140 mV 1000LVDA – 160 mV
12–15 LVDREGTRIM	Description: This field is used to fine tune the voltage threshold of LvdReg the rising LVD, used to monitor the VDDREG supply - rising edge. See the *MPC5676R Microcontroller Data Sheet* for details. Bit Values: 0111 LvdReg + 7 × LVDSTEPREG 0110 LvdReg + 6 × LVDSTEPREG 0101 LvdReg + 5 × LVDSTEPREG 0100 LvdReg + 4 × LVDSTEPREG 0011 LvdReg + 3 × LVDSTEPREG 0010 LvdReg + 2 × LVDSTEPREG 0001 LvdReg + 1 × LVDSTEPREG 0000 Nominal, start-up and default value LvdReg 1111 LvdReg – 1 × LVDSTEPREG 1110 LvdReg – 2 × LVDSTEPREG 1101 LvdReg – 3 × LVDSTEPREG 1100 LvdReg – 4 × LVDSTEPREG 1011 LvdReg – 5 × LVDSTEPREG 1010 LvdReg – 6 × LVDSTEPREG 1001 LvdReg – 7 × LVDSTEPREG 1000 LvdReg – 8 × LVDSTEPREG

Table 9-5. PMC_TRIMR Field Descriptions (continued)

Field	Description
16–19 VDD33TRIM	Description: This field is used to fine tune VDD33 the output voltage of the 3.3V regulator, the VDDSYN supply. See the *MPC5676R Microcontroller Data Sheet* for details. Bit Values: 0111 VDD33 + 7 × STEPV33 0110 VDD33 + 6 × STEPV33 0101 VDD33 + 5 × STEPV33 0100 VDD33 + 4 × STEPV33 0011 VDD33 + 3 × STEPV33 0010 VDD33 + 2 × STEPV33 0001 VDD33 + 1 × STEPV33 0000 Nominal, start-up and default value VDD33 1111 VDD33 – 1 × STEPV33 1110 VDD33 – 2 × STEPV33 1101 VDD33 – 3 × STEPV33 1100 VDD33 – 4 × STEPV33 1011 VDD33 – 5 × STEPV33 1010 VDD33 – 6 × STEPV33 1001 VDD33 – 7 × STEPV33 1000 VDD33 – 8 × STEPV33
20–23 LVD33TRIM	Description: LVD 3.3V trimming. This field is used to fine tune the rising voltage threshold of the VDDSYN supply, which can be internally regulated by the 3.3V regulator in LDO5V and SMPS5V modes or can be provided externally in LDO3V mode. See the *MPC5676R Microcontroller Data Sheet* for details. Bit Values 0111 LVD33 + 7 × LVDSTEP33 0110 LVD33 + 6 × LVDSTEP33 0101 LVD33 + 5 × LVDSTEP33 0100 LVD33 + 4 × LVDSTEP33 0011 LVD33 + 3 × LVDSTEP33 0010 LVD33 + 2 × LVDSTEP33 0001 LVD33 + 1 × LVDSTEP33 0000 Nominal, start-up and default value of LVD33 1111 LVD33 – 1 × LVDSTEP33 1101 LVD33 – 2 × LVDSTEP33 1110 LVD33 – 3 × LVDSTEP33 1100 LVD33 – 4 × LVDSTEP33 1011 LVD33 – 5 × LVDSTEP33 1001 LVD33 – 6 × LVDSTEP33 1010 LVD33 – 7 × LVDSTEP33 1000 LVD33 – 8 × LVDSTEP33

Table 9-5. PMC_TRIMR Field Descriptions (continued)

Field	Description
24–27 VDDCTRIM	Description: This field is used to fine tune VDD12OUT the output voltage of the 1.2V regulator, correspondent to the VDD supply. See the *MPC5676R Microcontroller Data Sheet* for details. Bit Values 0111 VDD12OUT + 7 × STEPV12 0110 VDD12OUT + 6 × STEPV12 0101 VDD12OUT + 5 × STEPV12 0100 VDD12OUT + 4 × STEPV12 0011 VDD12OUT + 3 × STEPV12 0010 VDD12OUT + 2 × STEPV12 0001 VDD12OUT + 1 × STEPV12 0000 Nominal, start-up and default value of VDD12OUT 1111 VDD12OUT – 1 × STEPV12 1110 VDD12OUT – 2 × STEPV12 1101 VDD12OUT – 3 × STEPV12 1100 VDD12OUT – 4 × STEPV12 Program VDD12OUT to this code when using Internal LDO generator 1011 VDD12OUT – 5 × STEPV12 1010 VDD12OUT – 6 × STEPV12 1001 VDD12OUT – 7 × STEPV12 1000 VDD12OUT – 8 × STEPV12
28–31 LVDCTRIM	Description: LVD 1.2V trimming. This field is used to fine tune the rising voltage threshold of the VDD supply. See the *MPC5676R Microcontroller Data Sheet* for details. Bit Values 0111 LVD12 + 13 × LVDSTEP12 0110 LVD12 + 12 × LVDSTEP12 Default LVD12 value to be programmed immediately after reset if core voltage internal regulator is used 0101 LVD12 + 11 × LVDSTEP12 0100 LVD12 + 10 × LVDSTEP12 0011 LVD12 + 9 × LVDSTEP12 0010 LVD12 + 8 × LVDSTEP12 0001 LVD12 + 7 × LVDSTEP12 0000 LVD12 + 6 × LVDSTEP12 Default value before reset 1111 LVD12 + 5 × LVDSTEP12 1101 LVD12 + 4 × LVDSTEP12 1110 LVD12 + 3 × LVDSTEP12 1100 LVD12 + 2 × LVDSTEP12 1011 LVD12 + 1 × LVDSTEP12 1010 LVD12 Default value at reset 1001 LVD12 – 1 × LVDSTEP12 1000 LVD12 – 2 × LVDSTEP12

9.4.3 Status Register (PMC_SR)

The status register contains interrupt flag bits for the LVD monitors.

Offset: PMC_BASE + 0x0008

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	LVF STBY	BGRDY	U[1]	0	0	0	0	0	LVFC STBY	0	0
W														w1c		
Reset	0	0	0	0	0	u[2]	1	U[1]	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LVFCR	LVFCH	LVFC 50	LVFC 33	LVFCC	LVFCA	0	0	LVFR	LVFH	LVF 50	LVF 33	LVFC	LVFA	0	0
W	w1c	w1c	w1c	w1c	w1c	w1c										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 9-4. Status Register (PMC_SR)

[1] This bit is not defined.

[2] Reset value depends on whether RAM standby regulator switch reported a brownout condition.

Table 9-6. PMC_SR Field Descriptions

Field	Description
0–4	Reserved
5 LVFSTBY	Standby-RAM-supply low-voltage flag. This read-only bit indicates that a brownout condition was reported by the RAM standby regulator switch. Software can clear this bit by writing ‘1’ to the LVFCSTBY bit. 0 No occurrence. 1 LVD occurrence, or brownout, reported by the RAM standby regulator switch.
6 BGRDY	Bandgap ready. This read-only bit gets asserted when the PMC bandgap circuit has finished its startup procedure during power-up. The PMC LVDs are disabled (output negated) while BGRDY is negated. 0 Bandgap not ready. PMC LVDs disabled. 1 Bandgap ready. PMC LVDs enabled.
7 U	This bit is not defined.
8–12	Reserved
13 LVFCSTBY	Standby-RAM-supply LVF clear. This write-only bit is used to clear the low-voltage flag reported by the RAM standby regulator switch. Writing 1 to this bit informs the RAM standby regulator switch to clear LVFSTBY. Writing 0 has no effect. Reading this bit always returns 0. 0 No effect. 1 Clears LVFSTBY.
14–15	Reserved
16 LVFCR	Reset-pin-supply LVF clear. This write-only bit is used to clear the low-voltage flag associated with the supply of the I/O segment that contains the reset pin. Writing 1 to this bit clears LVFR. Writing 0 has no effect. Reading this bit always returns 0. 0 No effect. 1 Clears LVFR.

Table 9-6. PMC_SR Field Descriptions (continued)

Field	Description
17 LVFCH	VDDEH LVF clear. This write-only bit is used to clear the low-voltage flag associated with the monitored VDDEH supplies. Writing 1 to this bit clears LVFH. Writing 0 has no effect. Reading this bit always return 0. 0 No effect. 1 Clears LVFH.
18 LVFC50	VDDREG LVF clear. This write-only bit is used to clear the low-voltage flag associated with the VDDREG voltage regulator supply. Writing 1 to this bit clears LVF50. Writing 0 has no effect. Reading this bit always returns 0. 0 No effect. 1 Clears LVF50.
19 LVFC33	VDDSYN LVF clear. This write-only bit is used to clear the low-voltage flag associated with the VDDSYN 3.3 V supply. Writing 1 to this bit clears LVF33. Writing 0 has no effect. Reading this bit always returns 0. 0 No effect. 1 Clears LVF33.
20 LVFCC	Core-voltage-supply LVF clear. This write-only bit is used to clear the low-voltage flag associated with the core voltage supply. Writing 1 to this bit clears LVFC. Writing 0 has no effect. Reading this bit always returns 0. 0 No effect. 1 Clears LVFC.
21 LVFCA	VDDA LVF clear. This write-only bit is used to clear the low-voltage flag associated with the analog power input VDDA1. Writing 1 to this bit clears LVFA. Writing 0 has no effect. Reading this bit always return 0. 0 No effect. 1 Clears LVFA.
22–23	Reserved
24 LVFR	Reset-pin-supply low-voltage flag. This read-only bit is the low-voltage flag associated with the supply of the I/O segment that contains the reset pin. It is asserted when the supply falls below the corresponding LVD threshold, and can be cleared by the CPU by writing 1 to the LVFCR bit. If the LVIER bit is also asserted, a low-voltage interrupt is sent to the CPU. If LVRER is also asserted, a system reset will be generated, which will clear LVFR and negate the interrupt request. 0 No occurrence. 1 LVD occurrence detected on the supply of the I/O segment that contains the reset pin.
25 LVFH	VDDEH low-voltage flag. This read-only bit is the low-voltage flag associated with the monitored VDDEH supplies. It is asserted when any monitored VDDEH supply falls below its corresponding LVD threshold, and can be cleared by the CPU by writing 1 to the LVFCH bit. If the LVIEH bit is also asserted, a low-voltage interrupt is sent to the CPU. If LVREH is also asserted, a system reset will be generated, which will clear LVFH and negate the interrupt request. 0 No occurrence. 1 LVD occurrence detected on a monitored VDDEH supply.
26 LVF50	VDDREG low-voltage flag. This read-only bit is the low-voltage flag associated with the VDDREG supply of the voltage regulator. It can be cleared by the CPU by writing 1 to the LVFC50 bit. If the LVIE5 bit is also asserted, a low-voltage interrupt is sent to the CPU. If LVRE50 is also asserted, a system reset will be generated, which will clear LVF50 and negate the interrupt request. 0 No occurrence. 1 LVD occurrence detected on the VDDREG supply of the voltage regulator.

Table 9-6. PMC_SR Field Descriptions (continued)

Field	Description
27 LVF33	VDDSYN low-voltage flag. This read-only bit is the low-voltage flag associated with the VDDSYN 3.3 V supply. It is asserted when the 3.3 V supply falls below the corresponding LVD threshold, and can be cleared by the CPU by writing 1 to the LVFC33 bit. If the LVIE33 bit is also asserted, a low-voltage interrupt is sent to the CPU. If LVRE33 is also asserted, a system reset will be generated, which will clear LVF33 and negate the interrupt request. 0 No occurrence. 1 LVD occurrence detected on the 3.3V supply.
28 LVFC	Core-voltage-supply low-voltage flag. This read-only bit is the low-voltage flag associated with the core voltage supply. It is asserted when the core voltage supply falls below the corresponding LVD threshold, and can be cleared by the CPU by writing 1 to the LVFCC bit. If the LVIEC bit is also asserted, a low-voltage interrupt is sent to the CPU. If LVREC is also asserted, a system reset will be generated, which will clear LVFC and negate the interrupt request. 0 No occurrence. 1 LVD occurrence detected on the core voltage supply.
29 LVFA	VDDA low-voltage flag. This read-only bit is the low-voltage flag associated with the analog power input VDDA1. It is asserted when the VDDA1 supply falls below its corresponding LVD threshold, and can be cleared by the CPU by writing 1 to the LVFCA bit. If the LVIEA bit is also asserted, a low-voltage interrupt is sent to the CPU. If LVREA is also asserted, a system reset will be generated, which will clear LVFA and negate the interrupt request. 0 No occurrence. 1 LVD occurrence detected on the VDDA1 supply.
30–31	Reserved

9.5 Functional Description

See Figure 9-1 for a block diagram of the PMC block. Its main building blocks are

- a precision bandgap voltage
- Power On Reset and Low Voltage Detector on VDDREG regulator supply
- a voltage regulator controller with a linear Low Drop Out and a Switched Mode Power Supply options selectable via the REGSEL control
- POR and LVD on VDD digital core supply
- a 3.3V LDO regulator, with its relative 3.3V LVD
- a digital interface to core logic
- an interface between measurable PMC internal signals to the ADC

A start-up sequence has been implemented aiming at improved predictability of PMC behavior. A loose tolerance POR keeps the device in reset until the VDDREG rises above the minimum required voltage for the internal bandgap to come up. When POR clears and the band gap reference is stable, the LVDs are enabled. A dedicated circuit is used to keep LVDs set until current and voltage references are stable and the real LVDs’ values are valid.

When the references are stable, the voltage regulator (selected by the pin REGSEL) enters in soft start mode and rises in a controlled fashion the 1.2V regulated voltage supply VDD. As both target regulated voltage VDD12OUT and LVD level LVD12 rely on bandgap voltage, an equivalent variation is to be expected in absolute LVD trip points and voltage regulator output, so the system shall come up correctly in any condition.

If an external regulator is used to generate the supply voltage, the allowed nominal range is VDD12, but the start up value must be higher than the maximum LVD threshold LVD12 to clear POR and LVD flags and exit the reset state.

All LVDs force reset at start-up. As soon as all of them are cleared, the trimming register can be loaded from flash into the band gap. The low variation of band gap reference voltage after trim, reported in the *MPC5676R Microcontroller Data Sheet*, allows the system to achieve high precision regulator target voltage output (not considering transient effects on regulator loads) and LVD trip points.

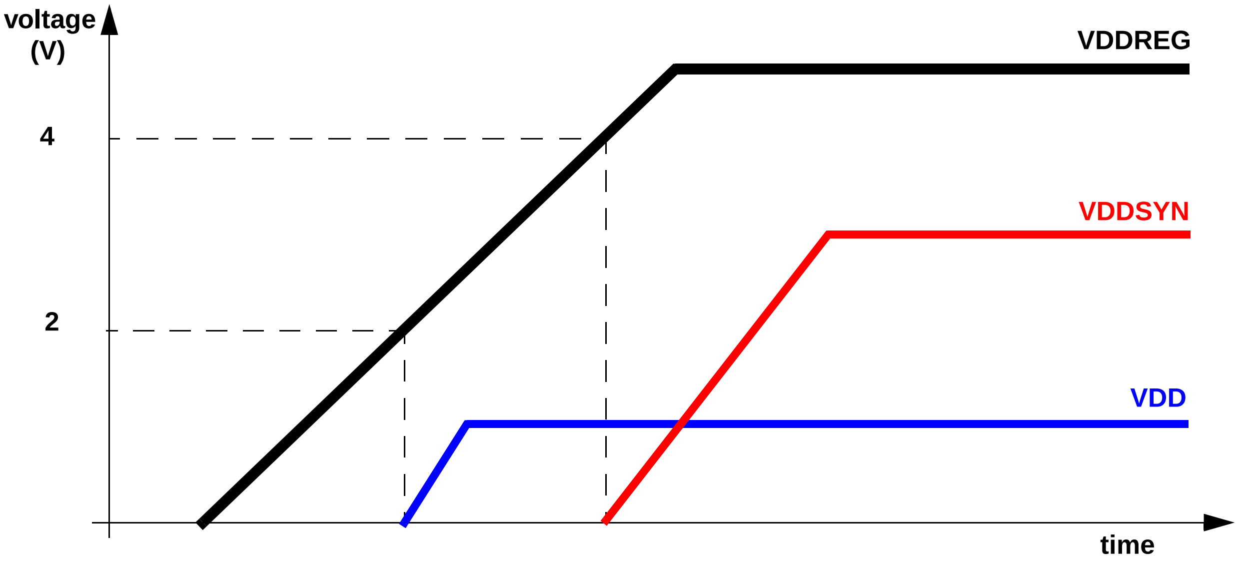


Figure 9-5. PMC internal regulators nominal start-up sequence, rising slope may vary

9.5.1 PMC Internal 1.2V Voltage Regulator Selection

The PMC features two main regulators for core supply voltage VDD. The selection is done at board level by connecting the REGSEL to VDDREG or VSS.

A linear Low Drop Out regulator controller is selected when the pin REGSEL is connected to VSS, otherwise an asynchronous buck switched mode regulator controller is selected when the pin is connected to VDDREG. Do not change REGSEL pin voltage when the device is powered by any supply.

The allowed voltage range of the PMC supply VDDREG is reported in the *MPC5676R Microcontroller Data Sheet* under the VDDR symbol. Specifically, when the switched mode supply is chosen (SMPS5V mode) the nominal supply voltage expected is 5 V. Nominal voltage values of 3V and 5V are allowed in LDO3V and LDO5V, respectively. This is because in the LDO3V mode, VDDSYN/VDD33 are supplied to the SOC from external source; whereas in the LDO5V mode, VDDSYN is connected back to VDD33.

When the LDO regulator is selected in LDO3V or LDO5V mode both nominal 5.0 V and 3.3 V are allowed.

A weak pull down of about 100 kΩ is added to ensure the selection of the LDO regulator even when the signal REGSEL is not connected at board level.

9.5.2 PMC Bandgap

An accurate band gap voltage is used in the PMC as reference for regulators and LVDs. Target band gap voltage VBG is 0.62V with variation defined in *MPC5676R Microcontroller Data Sheet*.

During factory test, the bandgap is calibrated in curvature and absolute value, in order to achieve a good accuracy over temperature range and supply voltage variation.

9.5.3 VDDREG LVD

A user programmable low voltage detector (LVD) monitors the PMC main supply voltage VDDREG.

When LDO regulator is selected, with respective selection pin REGSEL, the LVD threshold is for a nominal 3.3V supply (both LDO3V and LDO5V modes), else when the SMPS regulator is selected the LVD threshold is nominal 5.0V (SMPS5V mode).

Rising LVD threshold voltage is documented under LvdReg symbol in the *MPC5676R Microcontroller Data Sheet*.

The assertion and negation voltages are adjustable via software by writing to the LVDREGTRIM field of the PMC_TRIMR register, which selects one of the 16 voltages available through the appropriate tapped output. The reset and default value of the 4-bit register is “0000”, corresponding to the nominal LvdReg voltage.

LVD scaled voltage can be measured via ADC by selecting the respective channel reported in Table 9-8. During this measurement, the output of the LVD is temporarily forced to low level so that false events, which may be caused by ADC reading, are discarded.

9.5.4 3.3V Internal Voltage Regulator

A 3.3V internal voltage regulator is available and it can supply a total DC current of IDD33 with a maximum load frequency of 10 kHz. The board should be designed to dump all current overshoots and undershoots, by having the VDDSYN pin connected to decoupling capacitors. The recommended external capacitor range may vary between 220nF and 2.2 uF with ESR < 100 mΩ.

This regulator is always enabled when supply voltage VDDREG is in the VDDR 5V nominal range. When the supply voltage is in the VDDR 3V nominal range the regulator is automatically turned off, so that an external supply can be applied. In this case VDDREG and VDDSYN must be connected to the same supply voltage with a maximum voltage difference of 200mV.

The regulator can be disabled, so that an external 3.3 V supply can be used (see Section 9.1.1, “Features”). In this case it is recommended that the supplied 3.3V is nominal 3.5V +/- 3% during start up when both regulators may be enabled. For the correct operation of the device the externally supplied VDD33 voltage must be higher than the maximum correspondent LVD rising voltage LVD33.

The bond diagram and over voltage protection is shown in Figure 9-6. Core LVD and flash supply are provided by pad VDD33, which is also the default feedback signal for the 3.3V regulator loop. A high voltage detect comparator switches the feedback point from VDD33 to the regulator output VDDSYN in the event that this node raises above its maximum rating value, forcing the regulator to correct its DC point. This feature protects the circuitry connected to the internal 3.3V supply in any event which disconnects VDD33 and VDDSYN (e.g. a board failure or a bond wire failure).

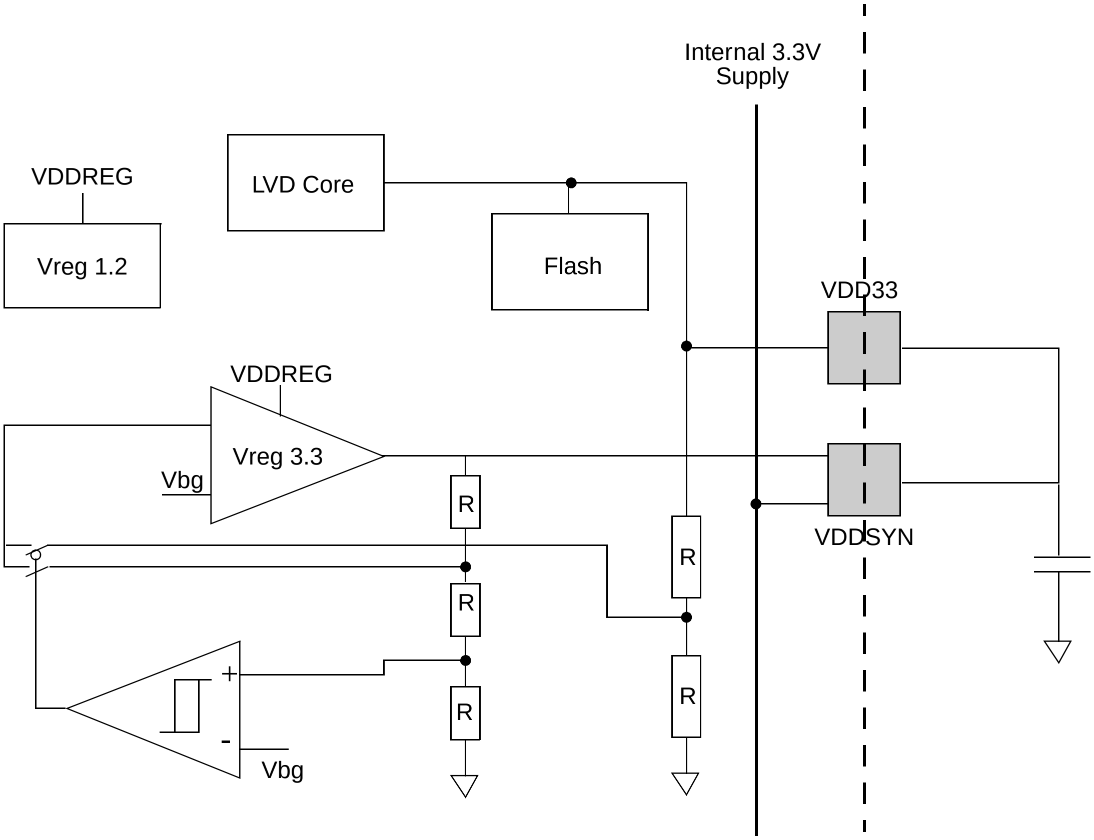


Figure 9-6. 3.3V Regulator power connection

Tolerance of the 3.3V regulator, reported in the *MPC5676R Microcontroller Data Sheet*, assumes appropriate decoupling capacitance on VDDSYN pin, maximum current load less than or equal to IDD33, and a correct board layout with reduced parasitics. It excludes line and load variation above 10 kHz.

The regulator output voltage, VDD33OUT, is adjustable via software by writing to the field VDD33TRIM of the correspondent trimming register PMC_TRIMR, which selects one of the 16 voltages spaced at STEPV33 step and available through the relative resistor chain. The reset value of the 4-bit register is “0000”.

9.5.5 3.3V VDDSYN LVD

A user programmable low voltage detector (LVD) monitors the voltage VDDSYN.

Rising LVD threshold voltage is documented under LVD33 symbol in the *MPC5676R Microcontroller Data Sheet*.

The assertion and negation voltages are adjustable via software by writing to the LVD33TRIM field of the PMC_TRIMR register, which selects one of the 16 voltages available through the appropriate tapped output. The reset and default value of the 4-bit register is “0000”, corresponding to the nominal LVD33 voltage.

LVD scaled voltage can be measured via ADC by selecting the respective channel reported in Table 9-8. During this measurement, the output of the LVD is temporarily forced to low level so that false events, which may be caused by ADC reading, are discarded.

9.5.6 1.2V Voltage Regulator Controller

A double Voltage Regulator Controller (VRC) is implemented in this power management system. It is composed of a linear LDO VRC, and an SMPS VRC. A soft startup block slews the output voltage of the regulator output smoothly in order to avoid ringing or over voltage condition and steep voltage and current slopes. A block diagram of the regulator is shown in Figure 9-7.

The LDO Voltage Regulator Controller is designed to drive an external bipolar transistor and relative decoupling capacitance at bipolar emitter. A smaller compensation capacitor might be required on REGCTL, depending on the external bipolar selected.

The switched controller is a full analog asynchronous regulator, with ramp compensation and equalized error integration. It is used to drive an external high side n-MOS driver / Schottky diode.The regulator output voltage is adjustable using software, to permit the device to center the supply for maximum transient margin.

The feedback of the regulation loop comes from the VDD core voltage through a trimmable resistive divider. By adjusting the trimming control 4-bit word VDDCTRIM it is possible to adjust the regulator target DC output voltage VDD12OUT during device operation, with 16 voltage steps of size STEPV12, around the typical regulator target voltage. The reset value, start up and default condition of the 4-bit register is “0000”.

Tolerance of the 1.2V regulators, reported in the *MPC5676R Microcontroller Data Sheet*, assumes appropriate external active and passive devices as reported in bill of materials Section 9.7.2, “Hardware Design Recommendations”, maximum current load less than or equal to **Idd12**, and a correct board layout with reduced parasitics. It excludes line and load variation above 10 kHz.

The 1.2V supply is internally connected to an ADC channel so that the actual voltage may be read by the microcontroller.

The suitable external driver has an automotive range temperature profile and

- in case of NPN device a minimum beta of 50, maximum current rating >1.5A;
- in case of high side driver maximum threshold voltage of 1.5V, gate capacitance less than 5nF, maximum current rating >2A.

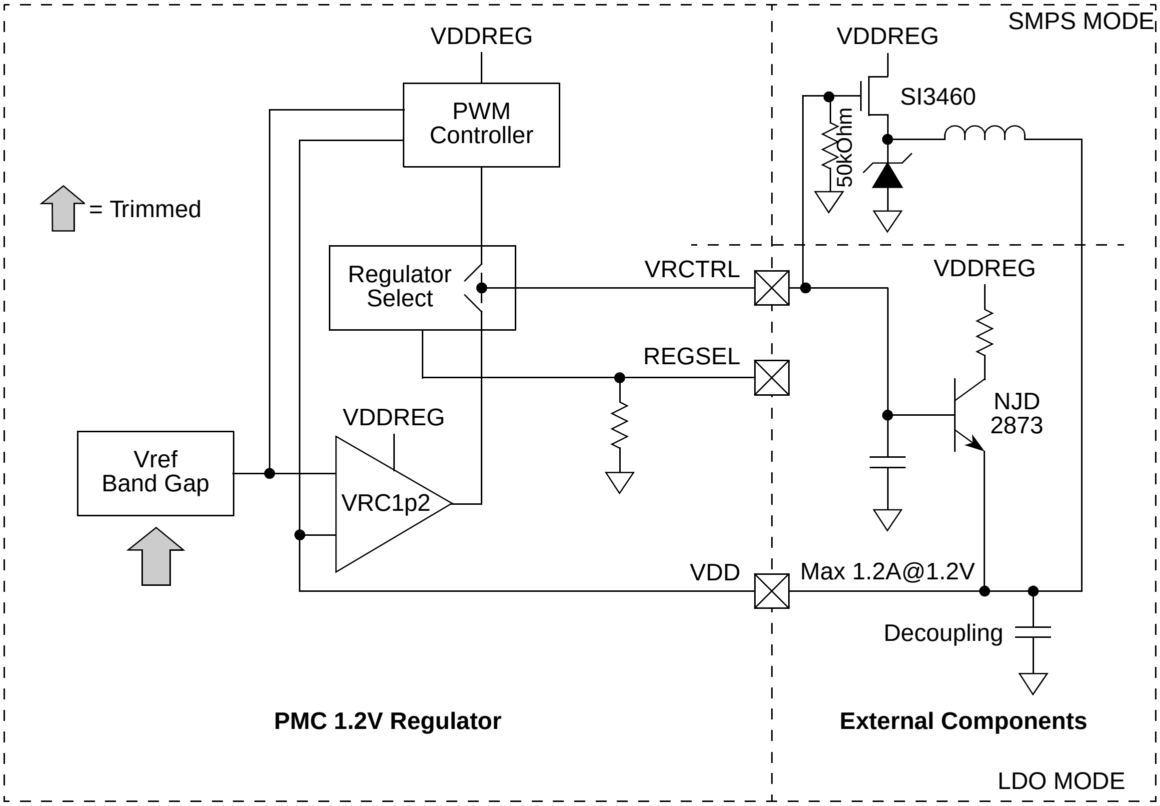


Figure 9-7. Internal Regulator 1.2V LDO and SMPS diagram

9.5.7 1.2V VDD LVD

A user programmable low voltage detector (LVD) monitors the core supply voltage VDD.

Rising LVD threshold voltage is documented under LVD12 symbol in the *MPC5676R Microcontroller Data Sheet*.

The assertion and negation voltages are adjustable via software by writing to the LVDCTRIM field of the PMC_TRIMR register, which selects one of the 16 voltages available through the appropriate tapped output. The reset value of the 4-bit register is “0110”, corresponding to the nominal LVD12 voltage.

When an internal regulator (SMPS or LDO) is used to generate the core voltage supply, it is required to change the field of the register to “1100” before increasing core logic clock frequency.

LVD scaled voltage can be measured via ADC by selecting the respective channel reported in Table 9-8. During this measurement, the output of the LVD is temporarily forced to low level so that false events, which may be caused by ADC reading, are discarded.

9.5.8 Trimming

During Power Up and Reset, the BandGap is untrimmed. This allows the MCU to come out of reset.

At the end of the Reset sequence, the BandGap is trimmed by the PMC. This trimming controls the BandGap voltage, which is used as the reference for the Internal Regulators and LVD.

The BandGap Trim values are not visible and they cannot be altered or overwritten by the user.

The levels of the internal regulators and LVD, though, can be adjusted by the user by programming the Trim Registers. This allows the user to control the internal regulator or LVD level within a range of +/- 8 steps from the default value. This is described in Section 9.4.2, “Trimming Register (PMC_TRIMR).

9.5.9 Interrupts

The PMC generates one interrupt request signal for each LVD source: VDDREG LVD, VDDSYN LVD, VDD LVD, and VDDA1 LVD. The module also generates combined interrupt request signal which is asserted whenever any of the individual interrupt request signals becomes asserted.

9.5.10 PMC Power-on Reset

A Power-on reset (POR) circuit monitors its supply voltage, providing a logic reset in a guaranteed low voltage range.

Power-on reset will assert as soon as possible after the voltage levels of the POR power supplies begin to rise. Each POR will negate before its power supply rises into the specified range. The behavior for each POR during power supply ramping is shown in Figure 9-8.

Power-on reset (POR) circuits are present at the following power supplies:

- 5V supply of the PMC block and bandgap (VDDREG)
- 1.2V core supply VDD

The POR can be used to prevent critical circuit-like band gap reference or LVDs to operate when the supply voltage is too low (output signals are out of specification). POR trip voltage tracks with technology variations and it should be such that all circuits, that are disabled by its output, can at least work with supplies down to POR level.

The dependence between POR and LVD on VDDREG is summarized in Figure 9-9. As shown, the LVD will reach a consistent state before the POR actually releases the reset, avoiding false startup condition.

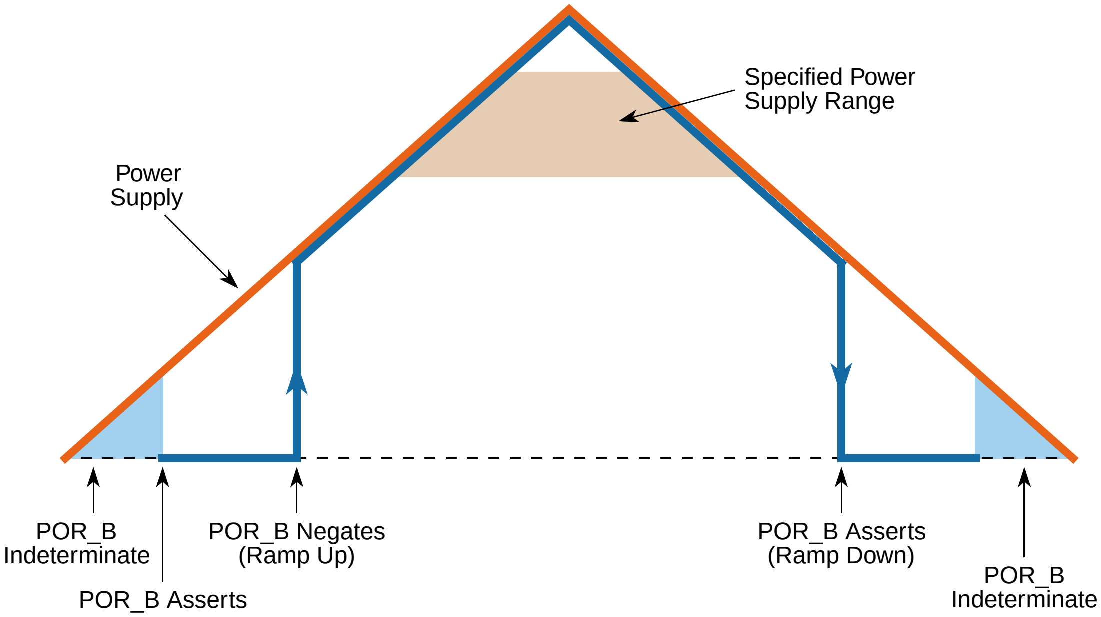


Figure 9-8. POR rising and falling edges

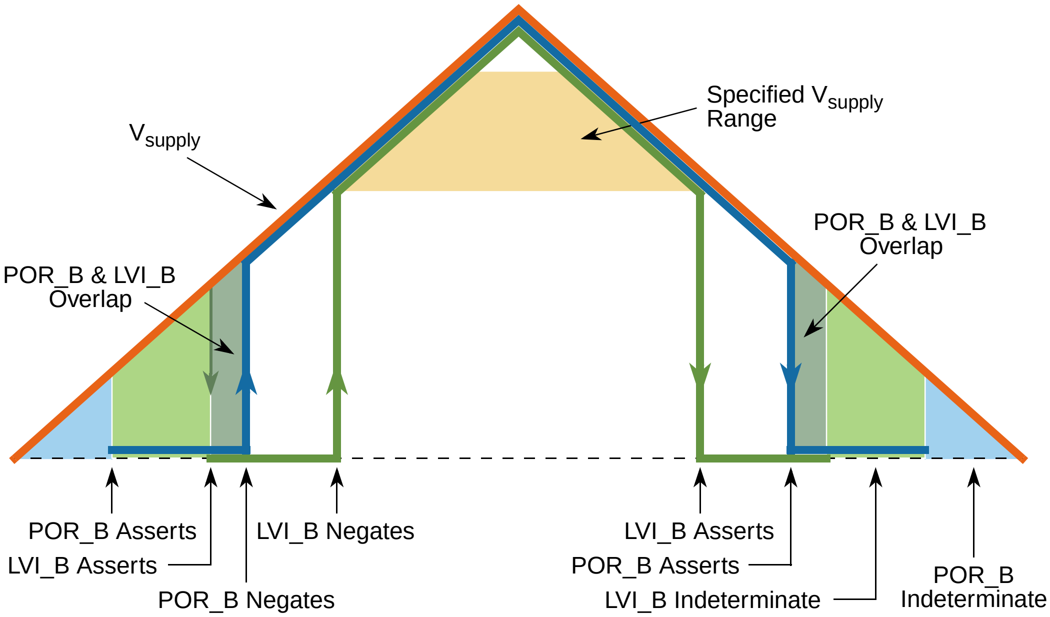


Figure 9-9. POR and LVD rising and falling edges

Table 9-7. POR and LVDs that Gate MCU Reset

Type	Supply	Description
POR	VDDREG	5V supply of the PMC block and bandgap (VDDREG)
POR	VDD	1.2V core supply VDD
LVD	VDD LVD	Core-voltage-supply VDD low-voltage
LVD	VDDSYN LVD	VDDSYN low voltage
LVD	Reset LVD	Reset-pin-supply low-voltage

9.5.11 ADC Test Mux

During PMC functional mode it is possible to perform direct measurements through the ADC. PMC internal voltages are routed to the ADC. Each signal can be measured with ADC running at full speed.

Table 9-8. ADC test mux channel for internal PMC signals

eQADC_A channel number	Signal propagated to ADC	Typical value (approximated)
ADC0 CH145	Band gap 0.62V	0.62V (trimmed)
ADC0 CH146	Analog supply 1.2V	1.22V
ADC0 CH147	VDD12OUT	Equal to VDD12OUT / 2.045
ADC0 CH180	LVD 1.2V	Equal to VLVD12 / 1.774
ADC0 CH181	VDD33OUT	Equal to VDD33OUT / 5.460
ADC0 CH182	LVD 3.3V	Equal to VLVD33 / 4.758
ADC0 CH183	LVD 5.0V	Equal to VLVDREG / 4.758 in LDO mode; (VLVDREG / 7.032 in SMPS mode)
ADC1 CH196	LVD VDDA	Equal to VDDA LVD / 3.8934

Regulator feedback and LVD threshold signals propagated to ADC are a scaled down version of the correspondent supply, by means of a resistive divider with programmable steps. The scaled signal read by the ADC Vscaled will be approximately equal to the supply voltage Vsupply divided by the target voltage (LVD or regulator output) Vtarget and multiplied by the band gap reference Vbg:

$$\text{Vscaled}=\text{Vsupply}*\text{Vbg}/\text{Vtarget}. \qquad \textit{Eqn. 9-1}$$

By measuring with ADC scaled supply voltage, supply voltage and bandgap voltage, it is possible to calculate the approximate target LVD or regulator value. Vtarget is equal to: Vtarget=Vsupply*Vbg/Vadc.

During LVD measurement, the continuous time monitoring is temporarily disabled as the multiplexer toggling could induce a false detection.

9.6 Initialization

The PMC module requires that its main supply voltage, VDDREG, rises above the POR level, so that the band gap reference voltage can be enabled.

After the band gap has come up and stabilized, the 1.2V regulator soft start and optionally the 3.3V regulator soft start begin.

When the internal regulator is used to generate 1.2V core supply, it is required to write “1100” to the LVDCTRIM field before clock frequency is increased.

9.7 Application Information

9.7.1 Regulator Example

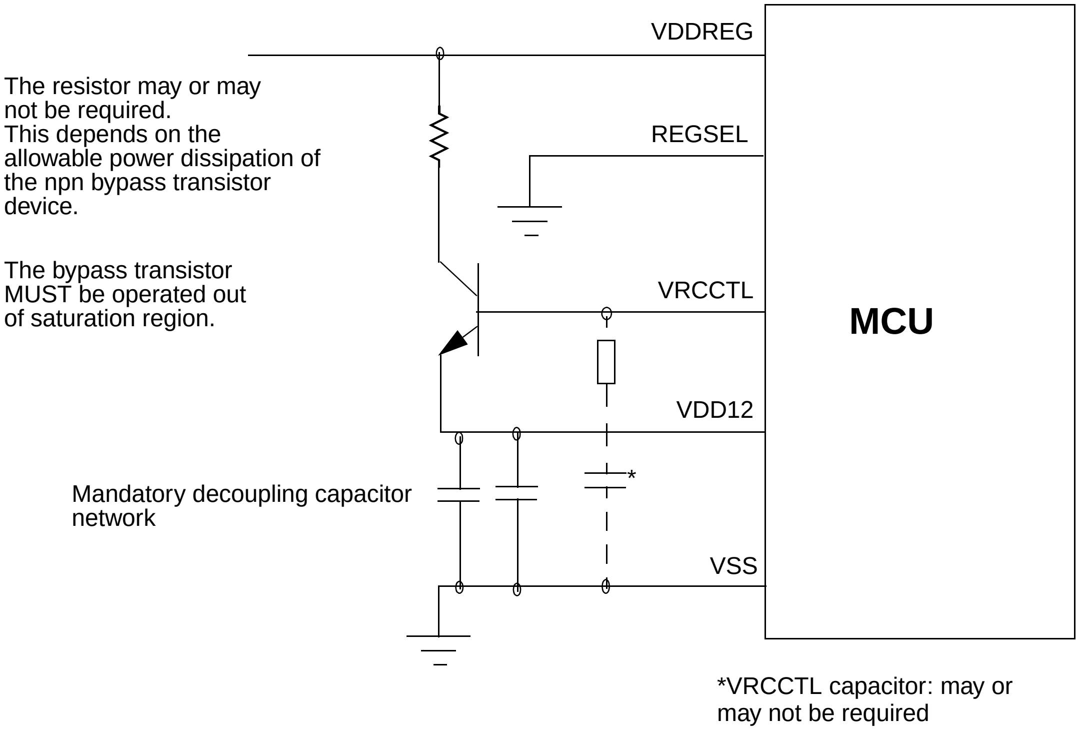
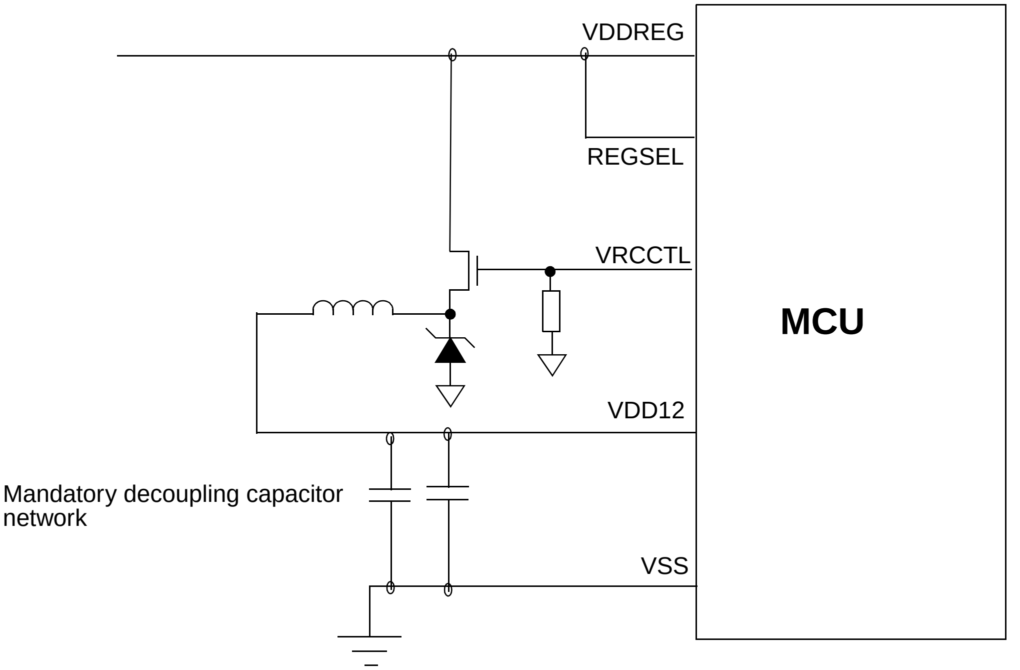


Figure 9-10. VRC 1.2V LDO configuration with external bipolar



No VRCCTL capacitor is allowed

Figure 9-11. VRC 1.2V buck SMPS LDO configuration with external MOS - Schottky diode

9.7.2 Hardware Design Recommendations

Table 9-10. VRC LDO recommended external devices

Part Name	Part Type	Nominal	Description
BCP68T1	npn		ON Semiconductor™
NJD2874	npn		ON Semiconductor™
3BCP68	npn		NXP Semiconductors™
	capacitor	6 x 4.7µF–20V	Ceramic low ESR -One for each VDD pin
	capacitor	6 x 0.1µF–20V	Ceramic -One capacitor for each VDD pin
	capacitor	20µF	Supply decoupling cap (close to bipolar collector)
	capacitor	2.2uF	Snubber cap, required with NJD2874 (on bipolar base)
	resistor	12 Ohm	ESR for snubber cap

Table 9-11. VRC SMPS recommended external devices

Part Name	Part Type	Nominal	Description
IR7353	HS nMOS + Schottky		Low threshold n-MOS / Low Vf Schottky diode
SS8P3L	Schottky		Low Vf Schottky diode
SI3460f	nMOS		Low threshold n-MOS
LQH66SN2R2M03	inductor	2.2 µH–3.2A	muRata™ unshielded or shielded coil
C3225X7R1E106M	capacitor	2X10uF - 25V	TDK high capacitance ceramic SMD (on VDD close to coil)
C3225X7R1E225K	capacitor	2x2.2uF - 25V	TDK ceramic SMD (on VDD close to MCU)
	capacitor	6 x 0.1uF - 20V	Ceramic -One capacitor for each VDD pin
C3225X7R1E106M	capacitor	2X10uF - 25V	Supply decoupling cap - close to n-MOS drain
	resistor	20 kOhm	Pull down for power n-MOS gate

Chapter 10
Core

10.1 Overview

The MPC5676R core is a dual-issue, 32-bit PowerPC Book E compliant design with 64-bit general purpose registers (GPRs). The MPC5676R has two symmetrical cores. All features in this section are in both cores, they are dedicated resources (not shared between the cores). Each core has a dedicated cache with hardware support for cache coherency to ensure the two cores always have the latest data and no coherency issues occur. All other MPC5676R resources are shared and accessible by both cores.

PowerPC Book E floating-point instructions are not supported by MPC5676R in hardware, but are trapped and may be emulated by software.

An Embedded Floating-point (EFPU2) APU is provided to support real-time single-precision embedded numerics operations using the general-purpose registers.

A Signal Processing Extension (SPE) APU is provided to support real-time SIMD fixed point and single-precision, embedded numerics operations using the general-purpose registers. All arithmetic instructions that execute in the core operate on data in the general purpose registers (GPRs). The GPRs have been extended to 64-bits in order to support vector instructions defined by the SPE APU. These instructions operate on a vector pair of 16-bit or 32-bit data types, and deliver vector and scalar results.

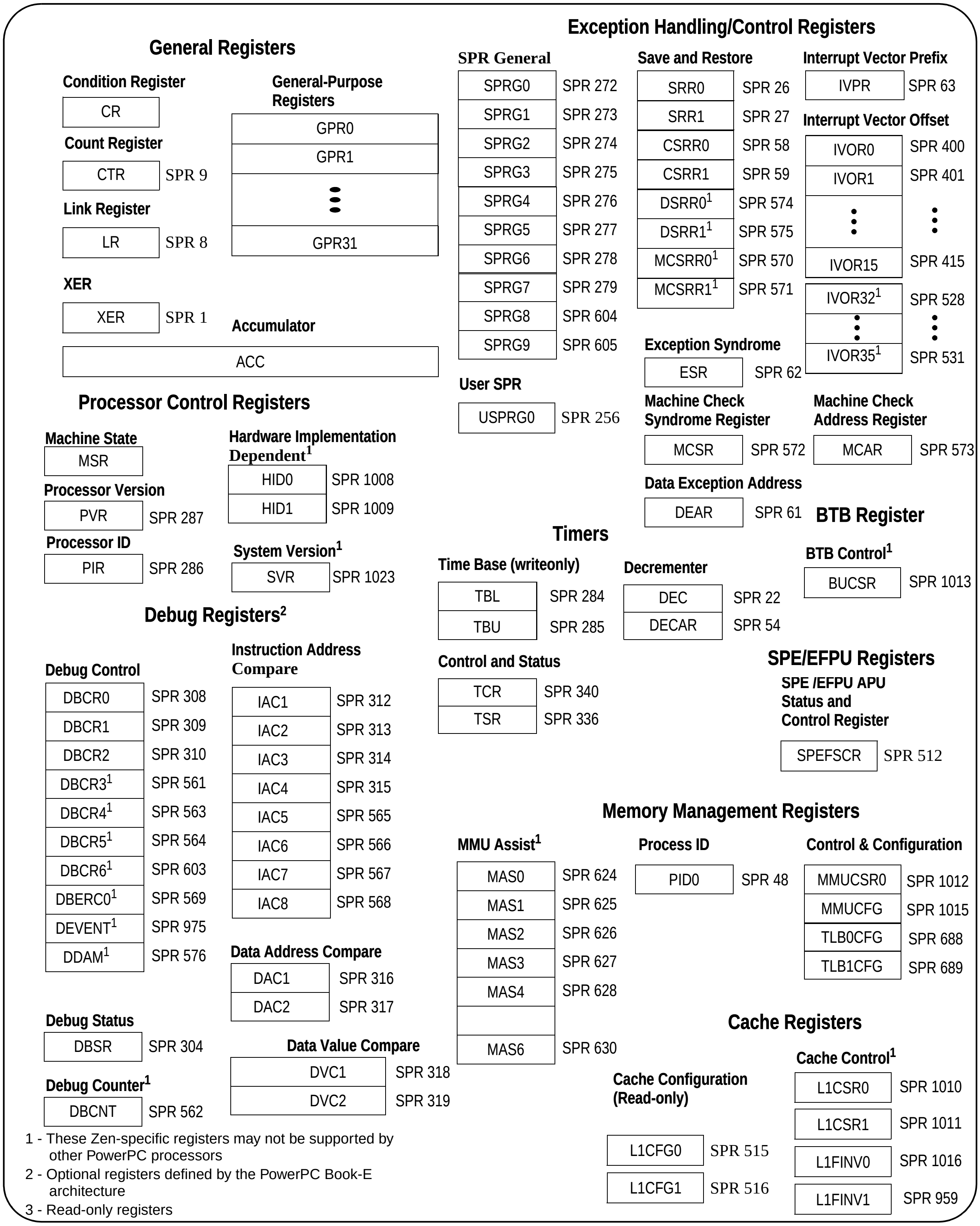
In addition to the base PowerPC Book E instruction set support, the MPC5676R core also implements the VLE (variable-length encoding) technology, providing improved code density. The VLE technology is further documented in “PowerPC VLE Definition, Version 1.03”, a separate document.

The MPC5676R processor integrates a pair of integer execution units, a branch control unit, instruction fetch unit and load/store unit, and a multi-ported register file capable of sustaining six read and three write operations per clock. Most integer instructions execute in a single clock cycle. Branch target prefetching is performed by the branch unit to allow single-cycle branches in many cases.

The MPC5676R contains a 16KB Instruction Cache, a 16KB Data Cache, as well as a Memory Management Unit. A Nexus Class 3+ module is also integrated.

10.2 Register Model

The figures below show the complete register set for Supervisor and User Modes. The number to the right of the special-purpose registers (SPRs) is the decimal number used in the instruction syntax to access the register (for example, the integer exception register (XER) is SPR 1).



1 - These Zen-specific registers may not be supported by other PowerPC processors
2 - Optional registers defined by the PowerPC Book-E architecture
3 - Read-only registers

Figure 10-1. SUPERVISOR Mode Programmer’s Model SPRs and GPRs

Software may distinguish which core it is executing on by reading the PIR (SPR 286) in Supervisor mode. Table 10-1 lists the PIR reset values for each core. Note that software may change the value after reset.

Table 10-1. PIR reset values for each core

Core	PIR value
Core 0	0
Core 1	1

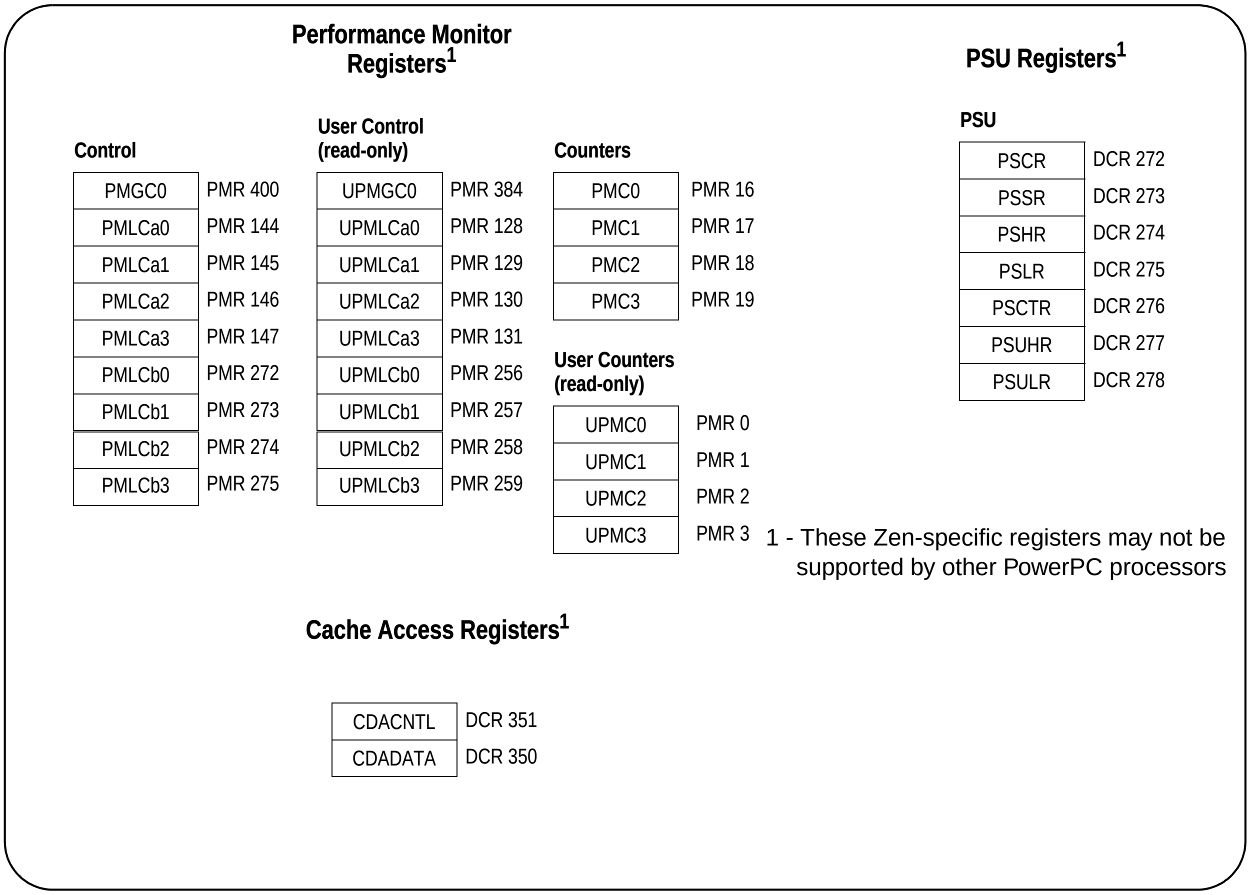


Figure 10-2. Supervisor Mode Programmer’s Model DCRs and PMRs

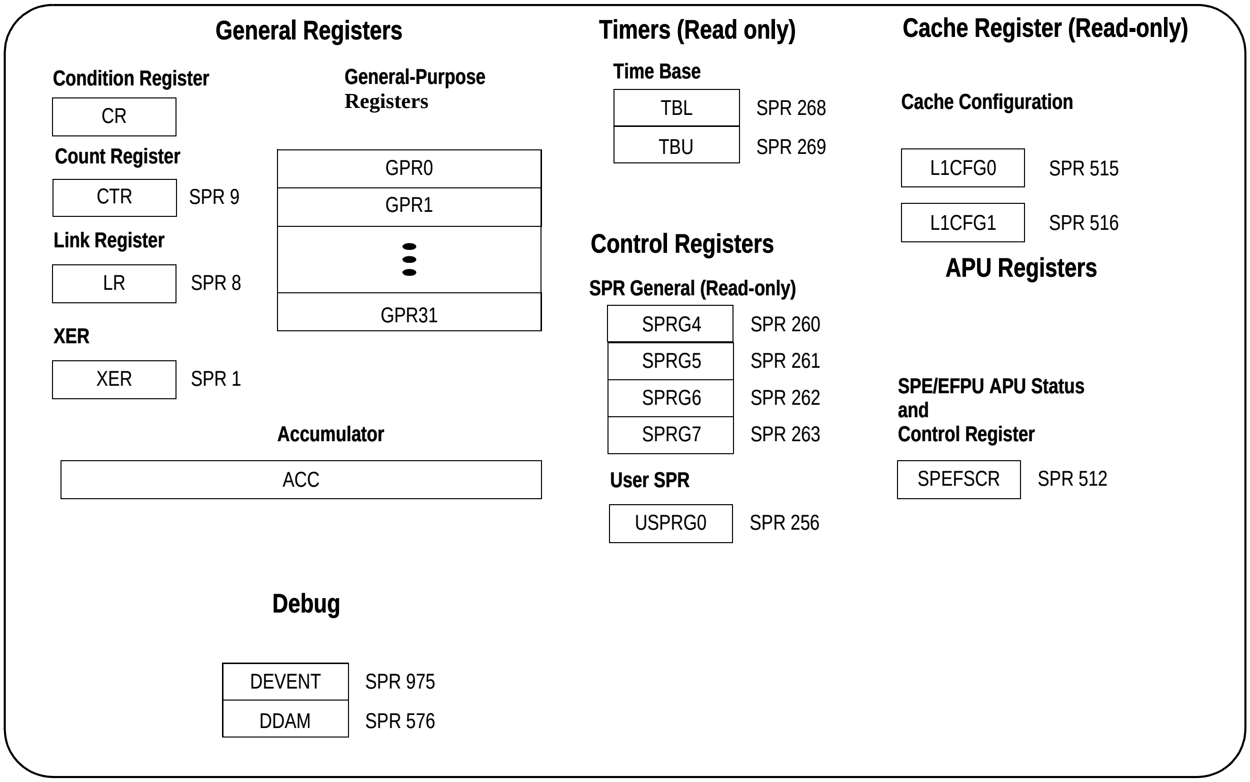


Figure 10-3. USER Mode Programmer's Model Registers

10.3 Cache

This section lists the most commonly used registers, instructions, and features of Cache. For a complete listing of all registers and features refer to *z759n3 Core Reference Manual*.

10.3.1 Cache Overview

The MPC5676R supports a pair of 16KB, 4-way set-associative, split instruction and data caches with a 32-byte line size. The caches improve system performance by providing low-latency data to the MPC5676R instruction and data pipelines, which decouples processor performance from system memory performance. The caches are virtually indexed and physically tagged.

Instruction and data addresses from the processor to the caches are virtual addresses used to index the cache array. The MMU provides the virtual to physical translation for use in performing the cache tag compare. If the physical address matches a valid cache tag entry, the access hits in the cache. For a read operation, the cache supplies the data to the processor, and for a write operation, the data from the processor updates the cache. If the access does not match a valid cache tag entry (misses in the cache) or a write access must be written through to memory, the cache performs a bus cycle on the system bus.

10.3.2 Cache Registers

10.3.2.1 L1 Cache Control and Status Register 0 (L1CSR0)

The L1 Cache Control and Status Register 0 (L1CSR0) is a 32-bit register used for general control of the data cache as well as providing general control over disabling ways in both caches. The L1CSR0 register is accessed using a **mfspr** or **mtspr** instruction. The SPR number for L1CSR0 is 1010 in decimal. The L1CSR0 register is shown in Figure 10-4.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
WID				WDD				0			DCWM	DCWA		0	DCECE	DCEI	0	DCEDT		DCSLC	DCUL	DCLO	DCLFC	DCLOA	DCEA		0	DCBZ32	DCABT	DCINV	DCE

SPR - 1010; Read/Write; Reset - 0x0

Figure 10-4. L1 Cache Control and Status Register 0 (L1CSR0)

The L1CSR0 bits are described below.

Table 10-2. L1CSR0 Field Descriptions

Bits	Name	Description
0:3	WID	Way Instruction Disable. 0 The corresponding way in the instruction cache is available for replacement by instruction miss line fills. 1 The corresponding way instruction cache is not available for replacement by instruction miss line fills. Bit 0 corresponds to way 0. Bit 1 corresponds to way 1. Bit 2 corresponds to way 2. Bit 3 corresponds to way 3. The WID bits may be used for locking ways of the instruction cache, and also are used in determining the replacement policy of the instruction cache.
4:7	WDD	Way Data Disable. 0 The corresponding way in the data cache is available for replacement by data miss line fills. 1 The corresponding way in the data cache is not available for replacement by data miss line fills. Bit 4 corresponds to way 0. Bit 5 corresponds to way 1. Bit 6 corresponds to way 2. Bit 7 corresponds to way 3. The WDD bits may be used for locking ways of the data cache, and also are used in determining the replacement policy of the data cache.
8:10	—	Reserved[1]

Table 10-2. L1CSR0 Field Descriptions (continued)

Bits	Name	Description
11	DCWM	Data Cache Write Mode 0 Data Cache operates in writethrough mode 1 Data Cache operates in copyback mode When set to writethrough mode, the “W” page attribute from the MMU is ignored and all writes are treated as writethrough required. When set, write accesses are performed in copyback mode unless the “W” page attribute from the MMU is set.
12:13	DCWA	Data Cache Write Allocation Policy 00 Cache line allocation on a cacheable write miss is disabled 01 Cache line allocation on a cacheable copyback write miss is enabled 10 Cache line allocation on a cacheable copyback or writethrough write miss is enabled 11 Reserved This field also controls merging of store data into the linefill buffer while a cache linefill is in progress. Store data will not be merged when write allocation is disabled. If DCWA is non-zero, store data merging is enabled regardless of the type (writethrough/copyback) of write.
14	—	Reserved[1]
15	DCECE	Data Cache Error Checking Enable 0 Error Checking is disabled 1 Error Checking is enabled
16	DCEI	Data Cache Error Injection 0 Cache Error Injection is disabled 1 Parity errors will be purposefully injected into every byte subsequently written into the cache. The parity bit of each 8-bit data element written will be inverted. This includes writes due to store hits as well as writes due to cache line refills. DCEI will cause injection of errors regardless of the setting of DCECE, although reporting of errors will be masked while DCECE = 0.
17	—	Reserved[1]
18:19	DCEDT	Data Cache Error Detection Type 00 Reserved (defaults to DCEDT=01(EDC) actions) 01 EDC Error Detection is selected for the tag array and parity is selected for the data arrays 1x Reserved
20	DCSLC	Data Cache Snoop Lock Clear 0 Snoop has not invalidated a locked line 1 Snoop has invalidated a locked line Indicates a cache line lock was cleared by a snoop operation which caused an invalidation. This bit is set by hardware and will remain set until cleared by software writing 0 to this bit location.
21	DCUL	Data Cache Unable to Lock Indicates a lock set instruction was not effective in locking a cache line. This bit is set by hardware on an “unable to lock” condition (other than lock overflows), and will remain set until cleared by software writing 0 to this bit location.

Table 10-2. L1CSR0 Field Descriptions (continued)

Bits	Name	Description
22	DCLO	Data Cache Lock Overflow Indicates a lock overflow (overlocking) condition occurred. This bit is set by hardware on an “overlocking” condition, and will remain set until cleared by software writing 0 to this bit location.
23	DCLFC	Data Cache Lock Bits Flash Clear When written to a ‘1’, a cache lock bits flash clear operation is initiated by hardware. Once complete, this bit is reset to ‘0’. Writing a ‘1’ while a flash clear operation is in progress will result in an undefined operation. Writing a ‘0’ to this bit while a flash clear operation is in progress will be ignored. Cache Lock Bits Flash Clear operations require approximately 134 cycles to complete. Clearing occurs regardless of the enable (DCE) value.
24	DCLOA	Data Cache Lock Overflow Allocate Set by software to allow a lock request to replace a locked line when a lock overflow situation exists. 0 Indicates a lock overflow condition will not replace an existing locked line with the requested line 1 Indicates a lock overflow condition will replace an existing locked line with the requested line
25:26	DCEA	Data Cache Error Action 00 Error Detection causes Machine Check exception. 01 Error Detection causes Correction/Auto-invalidation. No machine check is generated for uncorrectable errors unless the cache line was locked and invalidated or is dirty. Dirty lines are not auto-invalidated. In EDC mode, correction is performed for single-bit tag errors, single-bit lock errors, and single or multi-bit dirty errors. Correction is performed for data errors by reloading of the line. 1x Reserved
27	—	Reserved[1]
28	DCBZ32	Data Cache **dcba**, **dcbz** operation length 0 **dcba**, **dcbz** operations operate on an entire cache line 1 **dcba**, **dcbz** operations operate on 32bytes of a cache line **Note:** This bit is implemented for forward compatibility. Since cache lines are 32 bytes, this bit is ignored for **dcba**, **dcbz** operations
29	DCABT	Data Cache Operation Aborted Indicates a Cache Invalidate or a Cache Lock Bits Flash Clear operation was aborted prior to completion. This bit is set by hardware on an aborted condition, and will remain set until cleared by software writing 0 to this bit location.
30	DCINV	Data Cache Invalidate 0 No cache invalidate 1 Cache invalidation operation When written to a ‘1’, a cache invalidation operation is initiated by hardware. Once complete, this bit is reset to ‘0’. Writing a ‘1’ while an invalidation operation is in progress will result in an undefined operation. Writing a ‘0’ to this bit while an invalidation operation is in progress will be ignored. Cache invalidation operations require approximately 134 cycles to complete. Invalidation occurs regardless of the enable (DCE) value. During cache invalidations, the parity check bits are written with a value dependent on the DCEDT selection. DCEDT should be written with the desired value for subsequent cache operation when DCINV is set to ‘1’ for proper operation of the cache.

Table 10-2. L1CSR0 Field Descriptions (continued)

Bits	Name	Description
31	DCE	Data Cache Enable 0 Cache is disabled 1 Cache is enabled When disabled, cache lookups are not performed for normal load or store accesses, or for snoop requests. Other L1CSR0 cache control operations are still available. Also, operation of the store buffer is not affected by DCE.

[1] These bits are not implemented and should be written with zero for future compatibility.

10.3.2.2 L1 Cache Control and Status Register 1 (L1CSR1)

The L1 Cache Control and Status Register 1 (L1CSR1) is a 32-bit register used for general control of the instruction cache. The L1CSR1 register is accessed using a **mfspr** or **mtspr** instruction. The SPR number for L1CSR1 is 1011 in decimal. The L1CSR1 register is shown in Figure 10-5.

0–14	15	16	17	18–19	20	21	22	23	24	25–26	27–28	29	30	31
0	ICECE	ICEI	0	ICEDT	0	ICUL	ICLO	ICLFC	ICLOA	ICEA	0	ICABT	ICINV	ICE

SPR - 1011; Read/Write; Reset - 0x0

Figure 10-5. L1 Cache Control and Status Register 1 (L1CSR1)

The L1CSR1 bits are described in Table 10-3.

Table 10-3. L1CSR1 Field Descriptions

Bits	Name	Description
0:14	—	Reserved[1]
15	ICECE	Instruction Cache Error Checking Enable 0 Error Checking is disabled 1 Error Checking is enabled
16	ICEI	Instruction Cache Error Injection Enable 0 Cache Error Injection is disabled 1 When ICEDT=01, a double-bit error will be injected into each doubleword written into the cache by inverting the two uppermost parity check bits (p_chk[0:1]). ICEI will cause injection of errors regardless of the setting of ICECE, although reporting of errors will be masked when ICECE=0.
17	—	Reserved[1]
18:19	ICEDT	Instruction Cache Error Detection Type 00 Reserved (defaults to ICEDT=01(EDC) actions) 01 EDC Error Detection is selected 1x Reserved
20	—	Reserved[1]

Table 10-3. L1CSR1 Field Descriptions (continued)

Bits	Name	Description
21	ICUL	Instruction Cache Unable to Lock Indicates a lock set instruction was not effective in locking a cache line. This bit is set by hardware on an “unable to lock” condition (other than lock overflows), and will remain set until cleared by software writing 0 to this bit location.
22	ICLO	Instruction Cache Lock Overflow Indicates a lock overflow (overlocking) condition occurred. This bit is set by hardware on an “overlocking” condition, and will remain set until cleared by software writing 0 to this bit location.
23	ICLFC	Instruction Cache Lock Bits Flash Clear When written to a ‘1’, a cache lock bits flash clear operation is initiated by hardware. Once complete, this bit is reset to ‘0’. Writing a ‘1’ while a flash clear operation is in progress will result in an undefined operation. Writing a ‘0’ to this bit while a flash clear operation is in progress will be ignored. Cache Lock Bits Flash Clear operations require approximately 134 cycles to complete. Clearing occurs regardless of the enable (ICE) value.
24	ICLOA	Instruction Cache Lock Overflow Allocate Set by software to allow a lock request to replace a locked line when a lock overflow situation exists. 0 Indicates a lock overflow condition will not replace an existing locked line with the requested line 1 Indicates a lock overflow condition will replace an existing locked line with the requested line
25:26	ICEA	Instruction Cache Error Action 00 Error Detection causes Machine Check exception. 01 Error Detection causes Correction/Auto-invalidation. No machine check is generated unless a locked line is invalidated. Correction is performed for single-bit tag and lock errors, and lines with multi-bit tag or lock errors are invalidated. In parity mode, tag or lock errors will result in invalidation of lines. Correction is performed for single or multi-bit data errors by reloading of the line. 1x Reserved
27:28	—	Reserved[1]
29	ICABT	Instruction Cache Operation Aborted Indicates a Cache Invalidate or a Cache Lock Bits Flash Clear operation was aborted prior to completion. This bit is set by hardware on an aborted condition, and will remain set until cleared by software writing 0 to this bit location.
30	ICINV	Instruction Cache Invalidate 0 No cache invalidate 1 Cache invalidation operation When written to a ‘1’, a cache invalidation operation is initiated by hardware. Once complete, this bit is reset to ‘0’. Writing a ‘1’ while an invalidation operation is in progress will result in an undefined operation. Writing a ‘0’ to this bit while an invalidation operation is in progress will be ignored. Cache invalidation operations require approximately 134 cycles to complete. Invalidation occurs regardless of the enable (ICE) value. During cache invalidations, the parity check bits are written with a value dependent on the ICEDT selection. ICEDT should be written with the desired value for subsequent cache operation when ICINV is set to ‘1’ for proper operation of the cache.

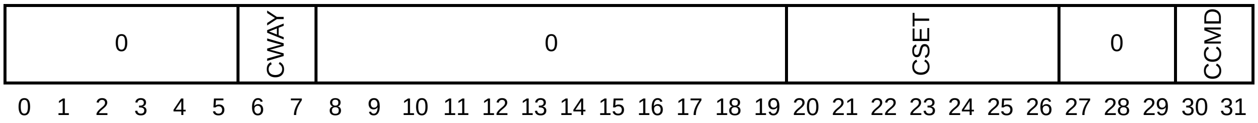
Table 10-3. L1CSR1 Field Descriptions (continued)

Bits	Name	Description
31	ICE	Instruction Cache Enable 0 Cache is disabled 1 Cache is enabled When disabled, cache lookups are not performed for instruction accesses. Other L1CSR1 cache control operations are still available and are not affected by ICE.

[1] These bits are not implemented and should be written with zero for future compatibility.

10.3.2.3 L1FINV0

The SPR number for L1FINV0 is 1016 in decimal. The L1FINV0 register is shown in Figure 10-6.



SPR - 1016; Read/Write; Reset - 0x0

Figure 10-6. L1 Flush/Invalidate Register 0 (L1FINV0)

Table 10-4. L1FINV0 Field Descriptions

Bits	Name	Description
0:5	—	Reserved[1] for way extension
6:7	CWAY	Cache Way Specifies the data cache way to be selected
8:19	—	Reserved[1] for set extension
20:26	CSET	Cache Set Specifies the cache set to be selected
27:29	—	Reserved[1] for set/command extension
30:31	CCMD	Cache Command 00 The data contained in this entry is invalidated without flushing 01 The data contained in this entry is flushed if dirty and valid without invalidation 10 The data contained in this entry is flushed if dirty and valid and then is invalidated 11 Reset way replacement pointer to the way indicated by CWAY

[1] These bits are not implemented and should be written with zero for future compatibility.

10.3.2.4 L1FINV1

The SPR number for L1FINV1 is 959 in decimal. The L1FINV1 register is shown in Figure 10-7.

Figure 10-7. L1 Flush/Invalidate Register 1 (L1FINV1)

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

SPR - 959; Read/Write; Reset - 0x0

Figure 10-7. L1 Flush/Invalidate Register 1 (L1FINV1)

Table 10-5. L1FINV1 Field Descriptions

Bits	Name	Description
0:5	—	Reserved[1] for way extension
6:7	CWAY	Cache Way Specifies the instruction cache way to be selected
8:19	—	Reserved[1] for set extension
20:26	CSET	Cache Set Specifies the instruction cache set to be selected
27:29	—	Reserved[1] for set/command extension
30:31	CCMD	Cache Command 00 The data contained in this entry is invalidated 01 Reserved 10 Reserved 11 Reset way replacement pointer to the way indicated by CWAY

[1] These bits are not implemented and should be written with zero for future compatibility.

10.3.3 Cache Coherency Unit (CCU)

The Cache Coherency Unit is a hardware mechanism for maintaining memory coherency between multiple bus masters. The implementation on the MPC5676R supports cache coherency between the two cores, including writes via the Nexus read-write access block. In addition DMA writes to system memory may be configured to provide cache coherency.

For writes originating from either core, the MMU for the core must be configured with Memory Coherence ('M' bit) and Write Through ('W' bit) set for the address spaces to be accessed. For writes originating from either DMA, use the DMA Global Write Registers to enable coherency on a channel-by-channel basis. Finally, set the Master Write Monitor Enable bits for each master requiring coherency for its writes as well as the global enable bit (ENB) in the CCU's configuration register.

10.3.3.1 Memory Map

The programming model contains control and status registers and can only be referenced using 32-bit (word) accesses. Writes to the configuration fields can only be performed in supervisor mode. Attempted accesses using different data sizes, accesses to undefined or reserved addresses, or accesses with a non-supported access type will result in a bus error termination.

Table 10-6. CCU Memory Map

CCU_BASE= 0xFFF50000	Register	Size	Access	Mode
CCU_BASE+ 0x000	CCU_CESR	32b	R/W	R = All W = Spv
CCU_BASE + 0x010	CCU_EAR0	32b	R	All
CCU_BASE + 0x014	CCU_EDR0	32b	R	All
CCU_BASE + 0x020	CCU_EAR1	32b	R	All
CCU_BASE + 0x024	CCU_EDR1	32b	R	All
CCU_BASE + 0x030	CCU_IR0	32b	R/W	R = All W = Spv
CCU_BASE + 0x34	CCU_IR1	32b	R/W	R = All W = Spv

10.3.3.2 Register Descriptions

10.3.3.2.1 CCU Configuration and Error Status Register

The CCU_CESR provides controls to allow processor and non-processor masters writes to be snooped, hardware error interrupt enablement, snoop queue status and reset of the Cache Coherency Unit.

Offset CCU_Base + 0x00 (CCU_CESR) Access: Read/Write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CP1 ERR		CP0 ERR		SRST FLAG		CP1 IDLE	CP0 IDLE			CP1 IEN	CP0 IEN	Reserved				M3 WMEN	M2 WMEN	M1 WMEN	M0 WMEN	Reserved										SRST EN	ENB
W																																
Reset	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-8. CCU Configuration Register (CCU_CESR)

Table 10-7. CCU_CESR Field Descriptions

Field	Description
31-30 CP1ERR	Core Processor 1 Error. This field reperesents a flag maintained by the CCU for signaling the presence of a captured snoop error or the occurence of a queue overflow. The error details are contained in CCU_EAR1 and CCU_EDR1. CCU_CESR[31] is set when the snoop queue for Core1 has overflowed; the address and attributes of the global write which caused the queue overflow are recorded in CCU_EAR1 and CCU_EDR1. CCU_CESR[30] is set when the hardware detects a snoop error and records the faulting address and attributes. This field is cleared when the corresponding bits are written as a logical one. If another Core1 error occurs before CP1ERR bit is cleared, CP1ERR remains set while CCU_EAR1 and CCU_EDR1 capture address and attributes for the most recent error event. 00 CCU_EAR1/CCU_EDR1 does not contain a captured snoop error 01 CCU_EAR1/CCU_EDR1 contains a captured snoop error. 10 CCU_EAR1/CCU_EDR1 contains a queue overflow occurence. 11 An encoding of ‘11’ for this field indicates a snoop error event occured followed by a queue overflow event. Since the CCU_EAR1/CCU_EDR1 registers capture the most recent error event, the CCU_EAR1/CCU_EDR1 contains the queue overflow occurence.

Table 10-7. CCU_CESR Field Descriptions

Field	Description
29-28 CP0ERR	Core Processor 0 Error. This field reperesents a flag maintained by the CCU for signaling the presence of a captured snoop error or the occurence of a queue overflow. The error details are contained in CCU_EAR0 and CCU_EDR0. CCU_CESR[29] is set when the snoop queue for Core0 has overflowed; the address and attributes of the global write which caused the queue overflow are recorded in CCU_EAR0 and CCU_EDR0. CCU_CESR[28] is set when the hardware detects a snoop error and records the faulting address and attributes. This field is cleared when the corresponding bits are written as a logical one. If another Core0 error occurs before CP0ERR bit is cleared, CP0ERR remains set while CCU_EAR0 and CCU_EDR0 capture address and attributes for the most recent error event. 00 CCU_EAR0/CCU_EDR0 does not contain a captured snoop error 01 CCU_EAR0/CCU_EDR0 contains a captured snoop error. 10 CCU_EAR0/CCU_EDR0 contains a queue overflow occurence. 11 An encoding of ‘11’ for this field indicates a snoop error event occured followed by a queue overflow event. Since the CCU_EAR0/CCU_EDR0 registers capture the most recent error event, the CCU_EAR0/CCU_EDR0 contains the queue overflow occurence.
27 SRST FLAG	Software reset flag. This field represents a flag maintained by the CCU to indicate a software reset event has taken place. A software reset of the CCU is invoked by programming the CCU_CESR[SRST_EN] field. 0 The last CCU reset event was a hardware reset event. 1 The last CCU reset event was a software-invoked reset event.
25 CP1IDLE	Core Processor 1 Idle. This read-only staus flag indicates when Core1 snoop queue is empty. 0 Core1 snoop queue not empty; core 1 snoop queue is busy processing valid snoop requests to core 1. 1 Core1 snoop queue is empty.
24 CP0IDLE	Core Processor 0 Idle. This read-only staus flag indicates when Core0 snoop queue is empty. 0 Core0 snoop queue not empty; core 0 snoop queue is busy processing valid snoop requests to core 0. 1 Core0 snoop queue is empty.
21 CP1IEN	Core Processor 1 Interrupt Enable. This bit enables interrupt generation upon detection of a Core1 snoop requests which results in an error. 0 CCU generates an interrupt request upon detection of a Core1 snoop that terminated with error. 1 CCU does not generate an interrupt request upon detection of a Core1 snoop request that terminated with error.
20 CP0IEN	Core Processor 0 Interrupt Enable. This bit enables interrupt generation upon detection of a Core0 snoop requests which results in an error. 0 CCU generates an interrupt request upon detection of a Core0 snoop that terminated with error. 1 CCU does not generate an interrupt request upon detection of a Core0 snoop request that terminated with error.
15 M3WMEN	Master 3 Write Monitor Enable. This bit enables monitoring of eDMA_B for global writes 0 Disable eDMA_B monitoring. Global writes from eDMA_B will not initiate any snoop requests. 1 Enable eDMA_B monitoring for global writes.
14 M2WMEN	Master 2 Write Monitor Enable. This bit enables monitoring of eDMA_A for global writes 0 Disable eDMA_A monitoring. Global writes from Master2 will not initiate any snoop requests. 1 Enable eDMA_A monitoring for global writes.
13 M1WMEN	Master 1 Write Monitor Enable. This bit enables monitoring of Core 1 for global writes 0 Disable Core 1 monitoring. Global writes from Master1 will not initiate any snoop requests. 1 Enable Core 1 monitoring for global writes.

Table 10-7. CCU_CESR Field Descriptions

Field	Description
12 M0WMEN	Master 0 Snoop Enable. This bit enables monitoring of Core 0 for global writes 0 Disable Core 0 monitoring. Global writes from Master0 will not initiate any snoop requests. 1 Enable Core 0 monitoring for global writes.
11-2	Reserved
1 SRST_EN	Software reset enable. Writing a ‘1’ to this bit invokes a software-driven reset of the CCU. On the next cycle following the programming of CCU_CESR[SRST_EN], all programming model registers in the CCU will be reset, and all snoop queue entries will be invalidated. CCU_CESR[SRST_FLAG] will be set to ‘1] to indicate a sofware-driven reset of the CCU has occured. Writing a ‘0’ to this bit has no effect. Reading this bit will return ‘0’. 0 No software-driven reset is requested. Writing a ‘0’ to this field has no effect. Reading this bit will always return ‘0’. 1 Invoke software-driven reset.
0 ENB	ENB. This bit provides a global enable/disable for the CCU. 0 The CCU is disabled. 1 The CCUis enabled. While the CCU is disabled, hardware coherency checking is not performed.

10.3.3.2.2 CCU Error Address Register, Core n (CCU_EARn)

CCU_EAR0 contains the address of the core 0 global write which caused a CCU error. CCU_EAR1 contains the address of the core 1 global write which caused a CCU error.

Offset CCU_Base + 0x10 (CCU_EAR0)
CCU_Base + 0x20 (CCU_EAR1)

Access: Read
Read

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	EADDR																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 10-9. CCU Address Error Register, Core n (CCU_EARn)

Table 10-8. CCU_EARn Field Description

Field	Description
31-0 EADDR	Error Address. This read-only field is the reference address associated with the global write which resulted in a snoop error or overflow event.

10.3.3.2.3 CCU Error Detail Register, Core n (CCU_EDRn)

CCU_EDR0 contains the details of an error caused by a core 0 global write. CCU_EDR1 contains the details of an error caused by a core 1 global write.

Offset CCU_Base + 0x14 (CCU_EDR0)
CCU_Base + 0x24 (CCU_EDR1)

Access: Read

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	EMSTR				Reserved				ESTAMP										EPROT						ERESP							
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-	-	0	0	0	0	0	0	0	0	0	0	0	-	-	-

Figure 10-10. CCU Error Detail Register, Core n (CCU_EDRn)

Table 10-9. CCU_EDRn Field Description

Field	Description
31-28 EMSTR	Error Master. This read-only field holds the AHB master ID associated with the global write which resulted in a snoop error or overflow event.
27-24	Reserved
23-16 ESTAMP	Error Time Stamp. This read-only fields holds the time stamp value associated with the snoop error or overflow event. The time stamp value is based on a free-running 8-bit timer which begins counting upon assertion of CCU_CESR[ENB]. Upon a snoop error event, this field captures the value of the time stamp at the time the snoop request was loaded into the queue. Upon a queue overlow event, this field captures the value of the time stamp at the time the overflow condition is detected.
13-8 EPROT	Error Protection Attribute. This read-only field holds thes AHB protection attributes which resulted in a snoop error or overflow event.
7-3 ERESP	Error snoop response. This read-only field holds the snoop response associated with the snoop which terminated with error. This field is cleared when CCU_EDRn is loaded on an overflow event. Upon a queue overflow event, this field reads as all zeroes. 000 cc NULL - no operation performed or no matching cache entry 001cc AutoInv - AutoInvalidation performed on clean unlocked lines with tag parity errors 010cc ERROR - Error in processing a snoop request due to tag parity error. 01100 SYNC - Sync completed, snoop queue synchronized 100cc Hit Clean - matching unlocked clean cache entry found 101cc Hit Dirty - matching unlocked dirty cache entry found 110cc Hit Locked - matching locked clean cache entry found 111cc Hit Dirty Locked - matching locked clean cache entry found cc - # of collapsed reqests; 00-no collapsin, 01-two requests combined, 10-three requests combined, 11-four requests combined.

10.3.3.2.4 CCU Interrupt Register, Core n (CCU_IRn)

Offset CCU_Base + 0x30 (CCU_IR0) Access: Read
CCU_Base + 0x34 (CCU_IR1) Read

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R																															OIF	EIF
W																																
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	0	0

Figure 10-11. CCU Interrupt Register, Core n (CCU_IRn)

Table 10-10. CCU_IRn Field Description

Field	Description
1 OIF	Overflow Interrupt Flag. This field indicates an outstanding interrupt request as a result of a a queue overflow. The flag and interrupt are cleared by writing a ‘1’ to this bit. Writing a ‘0’ to this bit has no effect. 0 No interrupt request. 1 Interrupt request due to snoop termination with error.
1 EIF	Error Interrupt Flag. This field indicates an outstanding interrupt request as a result of a snoop which terminated with error if CCU_CESR[M*n*IEN]. The flag and interrupt are cleared by writing a ‘1’ to this bit. Writing a ‘0’ to this bit has no effect. 0 No interrupt request. 1 Interrupt request due to snoop termination with error.

10.4 MMU

This section lists the most commonly used registers, instructions, and features of MMU. For a complete listing of all registers and features refer to *z759n3 Core Reference Manual*.

10.4.1 Overview

The MPC5676R Memory Management Unit is a 32-bit PowerPC Book E compliant implementation, with the following feature set:

- Translates from 32-bit effective to 32-bit real addresses
- 32-entry fully associative TLB with support for twenty-three page sizes (1K, 2K, 4K, 8K, 16K, 32K, 64K, 128K, 256K, 512K, 1M, 2M, 4M, 8M, 16M, 32M, 64M, 128M, 256M, 512M, 1G, 2G, 4G)
- Hardware assist for TLB miss exceptions
- Software managed by **tlbre**, **tlbwe**, **tlbsx**, **tlbsync**, and **tlbivax** instructions

10.4.2 MMU Instructions

10.4.3 TLB Read Entry Instruction (tlbre)

The TLB read entry instruction causes the content of a single TLB entry to be placed in the MMU assist registers. The entry is specified by the TLBSEL and ESEL fields of the MAS0 register. The entry contents are placed in the MAS1, MAS2, and MAS3 registers.

tlbre tlbre

tlb read entry

31	0	1 1 1 0 1 1 0 0 1 0	0
0 5	6 20	21 30	31

```
tlb_entry_id = MAS0(TLBSEL, ESEL)
result = MMU(tlb_entry_id)
MAS1, MAS2, MAS3 = result
```

10.4.4 TLB Write Entry Instruction (tlbwe)

The TLB write entry instruction causes the contents of certain fields within the MMU assist registers MAS1, MAS2, and MAS3 to be written into a single TLB entry in the MMU. The entry written is specified by the TLBSEL, and ESEL fields of the MAS0 register.

tlbwe tlbwe

tlb write entry

31	0	1 1 1 1 0 1 0 0 1 0	0
0 5	6 20	21 30	31

```
tlb_entry_id = MAS0(TLBSEL, ESEL)
MMU(tlb_entry_id) = MAS1, MAS2, MAS3
```

10.4.5 MMU Registers

10.4.5.1 DEAR Register

The Data Exception Address register is loaded with the effective address of the data access which results in an Alignment, Data TLB Miss, or DSI exception.

Effective Page Address
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

SPR - 61; Read/ Write; Reset - Unaffected

Figure 10-12. DEAR

The DEAR register can be read or written using the **mfspr** and **mtspr** instructions.

10.4.5.2 MMU Control and Status Register 0 (MMUCSR0)

The MMU Control and Status Register 0 (MMUCSR0) is a 32-bit register. The SPR number for MMUCSR0 is 1012 in decimal. MMUCSR0 controls the state of the MMU. The MMUCSR0 register is shown in Figure 10-13.

0	TLB1_FI	0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29	30	31

SPR - 1012; Read/ Write; Reset - 0x0

Figure 10-13. MMU Control and Status Register 0 (MMUCSR0)

The MMUCSR0 bits are described in Table 10-11.

Table 10-11. MMUCSR0 - MMU Control and Status Register 0

Bits	Name	Description
0:29 [32:61]	—	Reserved[1]
30 [62]	TLB1_FI	TLB1 flash invalidate 0 - No flash invalidate 1 - TLB1 invalidation operation When written to a ‘1’, a TLB1 invalidation operation is initiated by hardware. Once complete, this bit is reset to ‘0’. Writing a ‘1’ while an invalidation operation is in progress will result in an undefined operation. Writing a ‘0’ to this bit while an invalidation operation is in progress will be ignored. TLB1 invalidation operations require 3 cycles to complete.
31 [63]	—	Reserved[1]

[1] These bits are not implemented, will be read as zero, and writes are ignored.

10.4.5.3 MMU Assist Registers (MAS)

The MPC5676R uses special purpose registers (MAS0, MAS1, MAS2, MAS3, MAS4 and MAS6) to facilitate reading and writing the TLBs. The MAS registers can be read or written using the **mfspr** and **mtspr** instructions.

The MAS0 register is shown in Figure 10-14. Fields are defined in Table 10-12.

0	TLBSEL (01)	0	ESEL	0	NV
0 1	2 3	4 5 6 7 8 9 10	11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26	27 28 29 30 31

SPR - 624; Read/ Write; Reset - Unaffected

Figure 10-14. MMU Assist Register 0 (MAS0)

Table 10-12. MAS0 —MMU Read/Write and Replacement Control

Bit	Name	Comments, or Function when Set
0:1 [32:33]	—	Reserved[1]
2:3 [34:35]	TLBSEL	selects TLB for access: 00=TLB0, 01=TLB1 (ignored by Zen, should be written to 01 for future compatibility)
4:10 [36:42]	—	Reserved[1]
11:15 [43:47]	ESEL	Entry select for TLB.
16:25 [48:57]	—	Reserved[1]
27:31 [59:63]	NV	Next replacement victim for TLB1 (software managed) Software updates this field; it is copied to the ESEL field on a TLB Error.

[1] These bits are not implemented, will be read as zero, and writes are ignored.

The MAS1 register is shown in Figure 10-15. Fields are defined in Table 10-13.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
VALID	IPROT	0						TID								0			TS	TSIZ					0						

SPR - 625; Read/ Write; Reset - Unaffected

Figure 10-15. MMU Assist Register 1 (MAS1)

Table 10-13. MAS1 —Descriptor Context and Configuration Control

Bit	Name	Comments, or Function when Set
0 [32]	VALID	TLB Entry Valid 0 This TLB entry is invalid 1 This TLB entry is valid
1 [33]	IPROT	Invalidation Protect 0 Entry is not protected from invalidation 1 Entry is protected from invalidation as described in as described in the e200z759n3 reference manual’s IPROT Invalidation Protection section. Protects TLB entry from invalidation by **tlbivax** (TLB1 only), or flash invalidates through MMUSCR0[TLB1_FI].
2:7 [34:39]	—	Reserved[1]
8:15 [40:47]	TID	Translation ID bits This field is compared with the current process IDs of the effective address to be translated. A TID value of 0 defines an entry as global and matches with all process IDs.

Table 10-13. MAS1 —Descriptor Context and Configuration Control

Bit	Name	Comments, or Function when Set
16:18 [48:50]	—	Reserved[1]
19 [51]	TS	Translation address space This bit is compared with the IS or DS fields of the MSR (depending on the type of access) to determine if this TLB entry may be used for translation.
20:24 [52:56]	TSIZE	Entry's page size Supported page sizes are: 0b00000 - 1KB 0b00001 - 2KB 0b00010 - 4KB 0b00011 - 8KB 0b00100 - 16KB 0b00101 - 32KB 0b00110 - 64KB 0b00111 - 128KB 0b01000 - 256KB 0b01001 - 512KB 0b01010 - 1MB 0b01011 - 2MB 0b01100 - 4MB 0b01101 - 8MB 0b01110 - 16MB 0b01111 - 32MB 0b10000 - 64MB 0b10001 - 128MB 0b10010 - 256MB 0b10011 - 512MB 0b10100 - 1GB 0b10101 - 2GB 0b10110 - 4GB All other values are undefined
25:31 [57:63]	—	Reserved[1]

[1] These bits are not implemented, will be read as zero, and writes are ignored.

The MAS2 register is shown in Figure 10-16. Fields are defined in Table 10-14.

EPN	0	VLE	W	I	M	G	E
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21	22 23 24 25	26	27	28	29	30	31

SPR - 626; Read/ Write; Reset - Unaffected

Figure 10-16. MMU Assist Register 2 (MAS2)

Table 10-14. MAS2 - EPN and Page Attributes

Bit	Name	Comments, or Function when Set
0:21 [32:53]	EPN	Effective page number [0:21]
22:25 [54:57]	—	Reserved[1]
26 [58]	VLE	PowerPC VLE 0 This page is a standard BookE page 1 This page is a PowerPC VLE page This bit will always read as zero and writes will be ignored if **p_vle_present** is negated.
27 [59]	W	Write-through Required 0 This page is considered write-back with respect to the caches in the system 1 All stores performed to this page are written through to main memory
28 [60]	I	Cache Inhibited 0 This page is considered cacheable 1 This page is considered cache-inhibited
29 [61]	M	Memory Coherence Required 0 Memory Coherence is not required 1 Memory Coherence is required
30 [62]	G	Guarded 0 Access to this page are not guarded, and can be performed before it is known if they are required by the sequential execution model 1 All loads and stores to this page are performed without speculation (i.e. they are known to be required) MPC5676R uses the guarded attribute as described in the e200z759n3 reference manual’s Page Table Control Bits section.
31 [63]	E	Endianness 0 The page is accessed in big-endian byte order. 1 The page is accessed in true little-endian byte order. Determines endianness for the corresponding page. Refer to the e200z759n3 reference manual’s Byte Lane Specification section for more information.

[1] These bits are not implemented, will be read as zero, and writes are ignored.

The MAS3 register is shown in Figure 10-17. Fields are defined in Table 10-15.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
RPN																						U0	U1	U2	U3	UX	SX	UW	SW	UR	SR

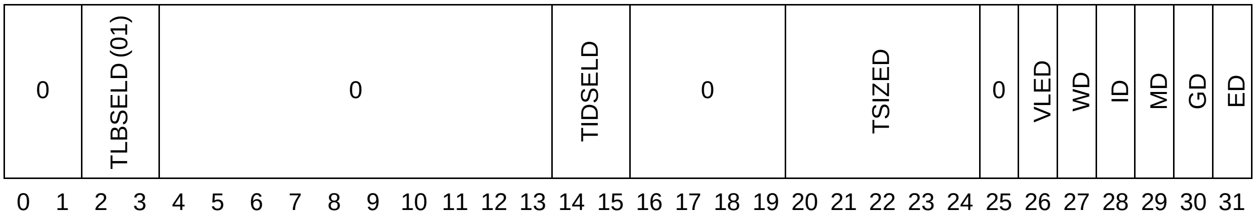
SPR - 627; Read/ Write; Reset - Unaffected

Figure 10-17. MMU Assist Register 3 (MAS3)

Table 10-15. MAS3 - RPN and Access Control

Bit	Name	Comments, or Function when Set
0:21 [32:53]	RPN	Real page number [0:21] Only bits that correspond to a page number are valid. Bits that represent offsets within a page are ignored and should be zero.
22:25 [54:57]	U0-U3	User bits [0-3] for use by system software
26:31 [58:63]	PERMIS	Permission bits (UX, SX, UW, SW, UR, SR)

The MAS4 register is shown in Figure 10-18. Fields are defined in Table 10-16.



SPR - 628; Read/ Write; Reset - Unaffected

Figure 10-18. MMU Assist Register 4 (MAS4)

Table 10-16. MAS4 - Hardware Replacement Assist Configuration Register

Bit	Name	Comments, or Function when Set
0:1 [32:33]	—	Reserved[1]
2:3 [34:35]	TLBSELD	Default TLB selected 00 TLB0 01 TLB1
4:13 [36:45]	—	Reserved[1]
14:15 [46:47]	TIDSELD	Default PID# to load TID from 00 PID0 01 Reserved, do not use 10 Reserved, do not use 11 TIDZ (8'h00)) (Use all zeros, the globally shared value)
16:19 [48:51]	—	Reserved[1]
20:24 [52:56]	TSIZED	Default TSIZE value

Table 10-16. MAS4 - Hardware Replacement Assist Configuration Register

Bit	Name	Comments, or Function when Set
25 [57]	—	Reserved[1]
26 [58]	VLED	Default VLE value
27:31 [59:63]	DWIMGE	Default WIMGE values

[1] These bits are not implemented, will be read as zero, and writes are ignored.

The MAS6 register is shown in Figure 10-19. Fields are defined in Table 10-17.

0								SPID								0															SAS
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

SPR - 630; Read/ Write; Reset - Unaffected

Figure 10-19. MMU Assist Register 6 (MAS6)

Table 10-17. MAS6 - TLB Search Context Register 0

Bit	Name	Comments, or Function when Set
0:7 [32:39]	—	Reserved[1]
8:15 [40:47]	SPID	PID value for searches
16:30 [48:62]	—	Reserved[1]
31 [63]	SAS	AS value for searches

[1] These bits are not implemented, will be read as zero, and writes are ignored.

10.4.6 External TLB Selection

In order to support realtime systems in which dynamic mapping of calibration or other data types is needed, the MMU provides special capabilities on a subset of TLB entries. For a complete listing of the capabilities refer to the *z759n3 Core Reference Manual*. These capabilities allow external hardware to dynamically select one of multiple mappings to one or more physical pages by the same logical address. This capability provides an inexpensive way of dynamically overlaying selected RAM pages on top of read-only memory during runtime. The particular physical page that a given logical page maps to can be dynamically altered by replacing PID0[6:7] with values provided by the SIU module registers SIU_C0PID and SIU_C1PID.

10.5 Exceptions

This section provides an overview of core exceptions. For detailed explanation see *z759n3 Core Reference Manual*.

10.5.1 Exception Syndrome Register

The Exception Syndrome Register (ESR) provides a *syndrome* to differentiate between exceptions that can generate the same interrupt type.

0				PIL	PPR	PTR	FP	ST	0	DLK	ILK	AP	PUO	BO	PIE	0								SPE	0	VLEMI	0			MIF	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

SPR - 62; Read/Write; Reset - 0x0

Figure 10-20. Exception Syndrome Register (ESR)

The ESR bits are defined in Table 10-18.

Table 10-18. ESR Bit Settings

Bit(s)	Name	Description	Associated Interrupt Type
0:3 (32:35)	—	Reserved	—
4 (36)	PIL	Illegal Instruction exception	Program
5 (37)	PPR	Privileged Instruction exception	Program
6 (38)	PTR	Trap exception	Program
7 (39)	FP	Floating-point operation	Program
8 (40)	ST	Store operation	Alignment Data Storage Data TLB
9 (41)	—	Reserved	—
10 (42)	DLK	Data Cache Locking	Data Storage
11 (43)	ILK	Instruction Cache Locking	Data Storage
12 (44)	AP	Reserved	—
13 (45)	PUO	Unimplemented Operation exception	Program

Table 10-18. ESR Bit Settings (continued)

Bit(s)	Name	Description	Associated Interrupt Type
14 (46)	BO	Byte Ordering exception Mismatched Instruction Storage exception	Data Storage Instruction Storage
15 (47)	PIE	Reserved	
16:23 (48:55)	—	Reserved	—
24 (56)	SPE	SPE/EFPU APU Operation	SPE/EFPU Unavailable EFPU Floating-point Data Exception EFPU Floating-point Round Exception Alignment Data Storage Data TLB
25 (57)	—	Reserved	—
26 (58)	VLEMI	VLE Mode Instruction	SPE/EFPU Unavailable EFPU Floating-point Data Exception EFPU Floating-point Round Exception Data Storage Data TLB Instruction Storage Alignment Program System Call
27:29 (59:61)	—	Reserved	—
30 (62)	MIF	Misaligned Instruction Fetch	Instruction Storage Instruction TLB
31 (63)	—	Reserved	—

10.6 Machine State Register

The Machine State Register defines the state of the processor. The Zen MSR is shown in Figure 10-21.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0					UCLE	SPE	0							CE	0	EE	PR	FP	ME	FE0	0	DE	FE1	0		IS	DS	0	PMM	RI	0

Read/ Write; Reset - 0x0

Figure 10-21. Machine State Register (MSR)

The MSR bits are defined in Table 10-19.

Table 10-19. MSR Bit Settings

Bit(s)	Name	Description
0:4 (32:36)	—	Reserved
5 (37)	UCLE	User Cache Lock Enable 0 Execution of the cache locking instructions in user mode (MSR_{PR}=1) disabled; DSI exception taken instead, and ILK or DLK set in ESR. 1 Execution of the cache lock instructions in user mode enabled.
6 (38)	SPE	SPE/EFPU Available 0 Execution of SPE and EFPU APU vector instructions is disabled; SPE/EFPU Unavailable exception taken instead, and SPE bit is set in ESR. 1 Execution of SPE and EFPU APU vector instructions is enabled.
7:12 (39:44)	—	Reserved
13 (45)	—	Reserved. Write a 0 for future compatibility.
14 (46)	CE	Critical Interrupt Enable 0 Critical Input and Watchdog Timer interrupts are disabled. 1 Critical Input and Watchdog Timer interrupts are enabled.
15 (47)	—	Reserved
16 (48)	EE	External Interrupt Enable 0 External Input, Decrementer, and Fixed-Interval Timer interrupts are disabled. 1 External Input, Decrementer, and Fixed-Interval Timer interrupts are enabled.
17 (49)	PR	Problem State 0 The processor is in supervisor mode, can execute any instruction, and can access any resource (e.g. GPRs, SPRs, MSR, etc.). 1 The processor is in user mode, cannot execute any privileged instruction, and cannot access any privileged resource.
18 (50)	FP	Floating-Point Available 0 Floating point unit is unavailable. The processor cannot execute floating-point instructions, including floating-point loads, stores, and moves. 1 Floating-point unit is available. The processor can execute floating-point instructions.

Table 10-19. MSR Bit Settings (continued)

Bit(s)	Name	Description
19 (51)	ME	Machine Check Enable 0 Asynchronous Machine Check interrupts are disabled. 1 Asynchronous Machine Check interrupts are enabled.
20 (52)	FE0	Reserved
21 (53)	—	Reserved
22 (54)	DE	Debug Interrupt Enable 0 Debug interrupts are disabled. 1 Debug interrupts are enabled.
23 (55)	FE1	Reserved
24 (56)	—	Reserved
25 (57)	—	Reserved
26 (58)	IS	Instruction Address Space 0 The processor directs all instruction fetches to address space 0 (TS=0 in the relevant TLB entry). 1 The processor directs all instruction fetches to address space 1 (TS=1 in the relevant TLB entry).
27 (59)	DS	Data Address Space 0 The processor directs all data storage accesses to address space 0 (TS=0 in the relevant TLB entry). 1 The processor directs all data storage accesses to address space 1 (TS=1 in the relevant TLB entry).
28 (60)	—	Reserved
29 (61)	PMM	PMM Performance monitor mark System software can set PMM when a marked process is running to enable statistics to be gathered only during the execution of the marked process. MSR_{PR} and MSR_{PMM} together define a state that the processor (supervisor or user) and the process (marked or unmarked) may be in at any time. If this state matches an individual state specified in the Performance Monitor registers PMLCa n, the state for which monitoring is enabled, counting is enabled.
30 (62)	RI	Recoverable Interrupt This bit is provided for software use to detect nested exception conditions. This bit is cleared by hardware when a Machine Check interrupt is taken
31 (63)	—	Reserved

10.6.1 Machine Check Syndrome Register (MCSR)

When the processor takes a machine check interrupt, it updates the Machine Check Syndrome register (MCSR) to differentiate between machine check conditions. The MCSR is shown in Figure 10-22.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MCP	IC_DPERR	CP_PERR	DC_DPERR	EXCP_ERR	IC_TPERR	DC_TPERR	IC_LKERR	DC_LKERR	0		NMI	MAV	MEA	0	IF	LD	ST	G	0							SNPERR	BUS_IRERR	BUS_DRERR	BUS_WRERR	0	

SPR - 572; Read/Clear; Reset - 0x0

Figure 10-22. Machine Check Syndrome Register (MCSR)

Table 10-20 describes MCSR fields. The MCSR indicates the source of a machine check condition.

All bits in the MCSR are implemented as “write ‘1’ to clear”. Software in the machine check handler is expected to clear the MCSR bits it has sampled prior to re-enabling MSR_{ME} to avoid a redundant machine check exception and to prepare for updated status bit information on the next machine check interrupt. Hardware will not clear a bit in the MCSR other than at reset.

Note that any set bit in the MCSR other than status-type bits will cause a subsequent machine check interrupt once MSR_{ME}=1.

Table 10-20. Machine Check Syndrome Register (MCSR)

Bit	Name	Description	Exception Type[1]	Recoverable
0 (32)	MCP	Machine check input pin	Async Mchk	Maybe
1 (33)	IC_DPERR	Instruction Cache data array parity error	Async Mchk	Precise
2 (34)	CP_PERR	Data Cache push parity error	Async Mchk	Unlikely
3 (35)	DC_DPERR	Data Cache data array parity error	Async Mchk	Maybe
4 (36)	EXCP_ERR	ISI, ITLB, or Bus Error on first instruction fetch for an exception handler	Async Mchk	Precise
5 (37)	IC_TPERR	Instruction Cache Tag parity error	Async Mchk	Precise
6 (38)	DC_TPERR	Data Cache Tag parity error	Async Mchk	Maybe

Table 10-20. Machine Check Syndrome Register (MCSR) (continued)

Bit	Name	Description	Exception Type[1]	Recoverable
7 (39)	IC_LKERR	Instruction Cache Lock error Indicates a cache control operation or invalidation operation invalidated one or more locked lines in the Icache or encountered an uncorrectable lock error, or that an Icache miss with an uncorrectable lock error occurred. May also be set on locked line refill error.	Status	—
8 (40)	DC_LKERR	Data Cache Lock error Indicates a cache control operation or invalidation operation invalidated one or more locked lines in the Dcache or encountered an uncorrectable lock error, or that an Icache miss with an uncorrectable lock error occurred. May also be set on locked line refill error.	Status	—
9:10 (41:42)	—	Reserved, should be cleared.		—
11 (43)	NMI	NMI input pin	NMI	—
12 (44)	MAV	MCAR Address Valid Indicates that the address contained in the MCAR was updated by hardware to correspond to the first detected Async Mchk error condition	Status	—
13 (45)	MEA	MCAR holds Effective Address If MAV=1,MEA=1 indicates that the MCAR contains an effective address and MEA=0 indicates that the MCAR contains a physical address	Status	—
14 (46)	—	Reserved, should be cleared.		—
15 (47)	IF	Instruction Fetch Error Report An error occurred during the attempt to fetch an instruction. This could be due to a parity error, or an external bus error. MCSRR0 contains the instruction address.	Error Report	Precise
16 (48)	LD	Load type instruction Error Report An error occurred during the attempt to execute the load type instruction located at the address stored in MCSRR0. This could be due to a parity error or an external bus error.	Error Report	Precise
17 (49)	ST	Store type instruction Error Report An error occurred during the attempt to execute the store type instruction located at the address stored in MCSRR0. This could be due to a parity error, or on certain external bus errors.	Error Report	Precise
18 (50)	G	Guarded instruction Error Report An error occurred during the attempt to execute the load or store type instruction located at the address stored in MCSRR0 and the access was guarded and encountered an error on the external bus.	Error Report	Precise

Table 10-20. Machine Check Syndrome Register (MCSR) (continued)

Bit	Name	Description	Exception Type[1]	Recoverable
19:25 (51:57)	—	Reserved, should be cleared.		—
26 (58)	SNPERR	Snoop Lookup Error An error occurred during certain snoop operations. This is typically due to a data cache tag parity error, in which case DC_TPERR will also be set.	Async Mchk	Unlikely?
27 (59)	BUS_IRERR	Read bus error on Instruction fetch or linefill	Async Mchk	Precise if data used
28 (60)	BUS_DRERR	Read bus error on data load or linefill	Async Mchk	Precise if data used
29 (61)	BUS_WRERR	Write bus error on store or cache line push	Async Mchk	Unlikely
30:31 (62:63)	—	Reserved, should be cleared.		—

[1] The Exception Type indicates the exception type associated with a given syndrome bit

- “Error Report” indicates that this bit is only set for error report exceptions which cause machine check interrupts. These bits are only updated when the machine check interrupt is actually taken. Error report exceptions are not gated by MSR_{ME}. These are synchronous exceptions. These bits will remain set until cleared by software writing a “1” to the bit position(s) to be cleared.
- “Status” indicates that this bit is provides additional status information regarding the logging of a machine check exception. These bits will remain set until cleared by software writing a “1” to the bit position(s) to be cleared.
- “NMI” indicates that this bit is only set for the non-maskable interrupt type exception which causes a machine check interrupt. This bit is only updated when the machine check interrupt is actually taken. NMI exceptions are not gated by MSR_{ME}. This is an asynchronous exception. This bit will remain set until cleared by software writing a “1” to the bit position.
- “Async Mchk” indicates that this bit is set for an asynchronous machine check exception. These bits are set immediately upon detection of the error. Once any “Async Mchk” bit is set in the MCSR, a machine check interrupt will occur if MSR_{ME}=1. If MSR_{ME}=0, the machine check exception will remain pending. These bits will remain set until cleared by software writing a “1” to the bit position(s) to be cleared.

10.7 Interrupt Vector Prefix Registers (IVPR)

The Interrupt Vector Prefix Register is used during interrupt processing for determining the starting address of a software handler used to handle an interrupt. The value contained in the Vector Offset field of the IVOR selected for a particular interrupt type is concatenated with the value held in the Interrupt Vector Prefix register (IVPR) to form an instruction address from which execution is to begin. The format of IVPR is shown in Figure 10-23.

Vector Base	0

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

SPR - 63; Read/Write

Figure 10-23. Zen Interrupt Vector Prefix Register (IVPR)

The IVPR fields are defined in Table 10-21.

Table 10-21. IVPR Register Fields

Bit(s)	Name	Description
0:15 (32:47)	Vec Base	Vector Base This field is used to define the base location of the vector table, aligned to a 64KB boundary. This field provides the high-order 16 bits of the location of all interrupt handlers. The contents of the IVORxx register appropriate for the type of exception being processed are concatenated with the IVPR Vector Base to form the address of the handler in memory.
16:31 (48:63)	—	Reserved

10.8 Interrupt Vector Offset Registers (IVORxx)

The Interrupt Vector Offset Registers are used during interrupt processing for determining the starting address of a software handler used to handle an interrupt. The value contained in the Vector Offset field of the IVOR selected for a particular interrupt type is concatenated with the value held in the Interrupt Vector Prefix register (IVPR) to form an instruction address from which execution is to begin. The format of a Zen IVOR is shown in Figure 10-24.

0	Vector Offset	0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27	28 29 30 31

SPR - 400-415, 528-530; Read/Write

Figure 10-24. Zen Interrupt Vector Offset Register (IVOR)

The IVOR fields are defined in Table 10-22.

Table 10-22. IVOR Register Fields

Bit(s)	Name	Description
0:15 (32:47)	—	Reserved
16:27 (48:59)	Vector Offset	Vector Offset This field is used to provide a quadword index from the base address provided by the IVPR to locate an interrupt handler.
28:31 (60:63)	—	Reserved

10.9 Interrupt Definitions

10.9.1 Critical Input Interrupt (IVOR0)

A Critical Input exception is signalled to the processor by the assertion of the critical interrupt pin. When Zen detects the exception, if the exception is enabled by MSR$_{CE}$, Zen takes the Critical Input interrupt. The critical input is a level-sensitive signal expected to remain asserted until Zen acknowledges the interrupt. If critical input is negated early, recognition of the interrupt request is not guaranteed. After Zen begins execution of the critical interrupt handler, the system can safely negate critical input.

Table 10-23 lists register settings when a Critical Input interrupt is taken.

Table 10-23. Critical Input Interrupt—Register Settings

<table>
<tr><th>Register</th><th colspan="3">Setting Description</th></tr>
<tr><td>CSRR0</td><td colspan="3">Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present.</td></tr>
<tr><td>CSRR1</td><td colspan="3">Set to the contents of the MSR at the time of the interrupt</td></tr>
<tr><td>MSR</td><td>UCLE 0
SPE 0
WE 0
CE 0
EE 0
PR 0</td><td>FP 0
ME —
FE0 0
DE —/0[1]</td><td>FE1 0
IS 0
DS 0
PMM 0
RI —</td></tr>
<tr><td>ESR</td><td colspan="3">Unchanged</td></tr>
<tr><td>MCSR</td><td colspan="3">Unchanged</td></tr>
<tr><td>DEAR</td><td colspan="3">Unchanged</td></tr>
<tr><td>Vector</td><td colspan="3">IVPR$_{0:15}$ || IVOR0$_{16:27}$ || 4b0000 (autovectored)
IVPR$_{0:15}$ || p_voffset[0:11] || 4b0000 (non-autovectored)</td></tr>
</table>

[1] DE is cleared when the Debug APU is disabled. Clearing of DE is optionally supported by control in HID0 when the Debug APU is enabled.

10.9.2 Machine Check Interrupt (IVOR1)

The Machine Check APU defines a separate set of save/restore registers (MCSRR0/1), a Machine Check Syndrome register (MCSR) to record the source(s) of machine checks, and a Machine Check Address register (MCAR) to hold an address associated with a machine check for certain classes of machine checks. Return from Machine Check instructions (**rfmci**, **se_rfmci**) are also provided to support returns using MCSRR0/1.

The MSR$_{RI}$ status bit is provided for software use in determining if multiple nested machine check exceptions have occurred. Software may interrogate the MCSRR1$_{RI}$ bit to determine if a machine check occurred during the initial portion of a machine check handler prior to handler code which sets MSR$_{RI}$ to ‘1’ to indicate that the handler can now tolerate another machine check condition without losing state necessary for recovery.

10.9.2.1 Machine Check Causes

Machine check causes are divided into different types:

- Error Report Machine Check conditions
- Non-Maskable Interrupt (NMI) machine check exceptions
- Asynchronous machine check exceptions

10.9.2.2 Machine Check Interrupt Actions

Machine Check interrupts for “error report” conditions and NMI are enabled and taken regardless of the state of MSR$_{ME}$. Machine check interrupts due to an “async mchk” syndrome bit being set in MCSR are only taken when MSR$_{ME}$ =1. When a Machine Check interrupt is taken, registers are updated as shown in Table 10-24.

Table 10-24. Machine Check Interrupt - Register Settings

<table>
<tr><th>Register</th><th colspan="3">Setting Description</th></tr>
<tr><td>MCSRR0</td><td colspan="3">On a best-effort basis Zen sets this to the address of some instruction that was executing or about to be executing when the machine check condition occurred.</td></tr>
<tr><td>MCSRR1</td><td colspan="3">Set to the contents of the MSR at the time of the interrupt</td></tr>
<tr><td>MSR</td><td>UCLE 0
SPE 0
WE 0
CE 0
EE 0
PR 0</td><td>FP 0
ME 0
FE0 0
DE 0/—[1]</td><td>FE1 0
IS 0
DS 0
PMM 0
RI 0</td></tr>
<tr><td>ESR</td><td colspan="3">Unchanged</td></tr>
<tr><td>MCSR</td><td colspan="3">Updated to reflect the source(s) of a machine check. Hardware only sets appropriate bits, no previously set bits are cleared by hardware.</td></tr>
<tr><td>MCAR</td><td colspan="3">See</td></tr>
<tr><td>Vector</td><td colspan="3">IVPR$_{0:15}$ || IVOR1$_{16:27}$ || 4b0000</td></tr>
</table>

[1] DE is cleared when the Debug APU is disabled. Clearing of DE is optionally supported by control in HID0 when the Debug APU is enabled.

The Machine Check Syndrome register is provided to identify the source(s) of a machine check, and in conjunction with MCSRR1$_{RI}$, may be used to identify recoverable events.

10.9.3 Data Storage Interrupt (IVOR2)

A Data Storage interrupt (DSI) may occur if no higher priority exception exists and one of the following exception conditions exists:

- Read or Write Access Control exception condition
- Byte Ordering exception condition
- Cache Locking exception condition

Table 10-25 lists register settings when a DSI is taken.

Table 10-25. Data Storage Interrupt—Register Settings

Register	Setting Description		
SRR0	Set to the effective address of the excepting load/store instruction.		
SRR1	Set to the contents of the MSR at the time of the interrupt		
MSR	UCLE 0 SPE 0 WE 0 CE — EE 0 PR 0	FP 0 ME — FE0 0 DE —	FE1 0 IS 0 DS 0 PMM 0 RI —
ESR	Access: Byte ordering: Cache locking:	[ST], [SPE], [VLEMI]. All other bits cleared. [ST], [SPE], [VLEMI], BO. All other bits cleared. (DLK, ILK), [VLEMI], [ST]. All other bits cleared.	
MCSR	Unchanged		
DEAR	For Access and Byte ordering exceptions, set to the effective address of a byte within the page whose access caused the violation. Undefined on Cache locking exceptions (DEAR is not updated on a cache locking exception)		
Vector	$\text{IVPR}_{0:15}$ || $\text{IVOR2}_{16:27}$ || 4b0000		

10.9.4 Instruction Storage Interrupt (IVOR3)

An Instruction Storage interrupt (ISI) occurs when no higher priority exception exists and an Execute Access Control exception occurs.

Table 10-26. ISI Exceptions and Conditions

Interrupt Type	Interrupt Vector Offset Register	Causing Conditions
Instruction Storage	IVOR 3	1. Access control. 2. Byte ordering due to misaligned instruction across page boundary to pages with mismatched VLE bits, or access to page with VLE set, and E indicating little-endian. 3. Misaligned Instruction fetch due to a change of flow to an odd halfword instruction boundary on a BookE (non-VLE) instruction page.

Table 10-27 lists register settings when an ISI is taken.

Table 10-27. Instruction Storage Interrupt—Register Settings

Register	Setting Description
SRR0	Set to the effective address of the excepting instruction.
SRR1	Set to the contents of the MSR at the time of the interrupt

Table 10-27. Instruction Storage Interrupt—Register Settings (continued)

MSR	UCLE 0 SPE 0 WE 0 CE — EE 0 PR 0	FP 0 ME — FE0 0 DE —	FE1 0 IS 0 DS 0 PMM 0 RI —
ESR	[BO, MIF, VLEMI]. All other bits cleared.		
MCSR	Unchanged		
DEAR	Unchanged		
Vector	IVPR$_{0:15}$ || IVOR3$_{16:27}$ || 4b0000		

10.9.5 External Input Interrupt (IVOR4)

An External Input exception is signalled to the processor by the assertion an interrupt from the interrupt controller. The input is a level-sensitive signal expected to remain asserted until core acknowledges the external interrupt. If input is negated early, recognition of the interrupt request is not guaranteed. When core detects the exception, if the exception is enabled by MSR$_{EE}$, it takes the External Input interrupt.

Table 10-28 lists register settings when an External Input interrupt is taken.

Table 10-28. External Input Interrupt—Register Settings

Register	Setting Description		
SRR0	Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present.		
SRR1	Set to the contents of the MSR at the time of the interrupt		
MSR	UCLE 0 SPE 0 WE 0 CE — EE 0 PR 0	FP 0 ME — FE0 0 DE —	FE1 0 IS 0 DS 0 PMM 0 RI —
ESR	Unchanged		
MCSR	Unchanged		
DEAR	Unchanged		
Vector	IVPR$_{0:15}$ || IVOR4$_{16:27}$ || 4b0000 IVPR$_{0:15}$ || **p_voffset[0:11]** || 4b0000 (non-autovectored)		

IVOR4 is the vector offset register used by auto-vectored External Input interrupts to determine the interrupt handler location. Zen also provides the capability to directly vector External Input interrupts to multiple handlers by allowing a External Input interrupt request to be accompanied by a vector offset.

10.9.6 Alignment Interrupt (IVOR5)

An Alignment exception is generated when any of the following occurs:

- The operand of **lmw** or **stmw** not word aligned.
- The operand of **lwarx** or **stwcx.** not word aligned.
- The operand of **lharx** or **sthcx.** not halfword aligned.
- Execution of a **dcbz** instruction is attempted with a disabled cache.
- Execution of a **dcbz** instruction with an enabled cache and W or I =1.
- Execution of a SPE APU load or store instruction which is not properly aligned.

Table 10-29 lists register settings when an alignment interrupt is taken.

Table 10-29. Alignment Interrupt—Register Settings

Register	Setting Description		
SRR0	Set to the effective address of the excepting load/store instruction.		
SRR1	Set to the contents of the MSR at the time of the interrupt		
MSR	UCLE 0 SPE 0 WE 0 CE — EE 0 PR 0	FP 0 ME — FE0 0 DE —	FE1 0 IS 0 DS 0 PMM 0 RI —
ESR	[ST], [SPE], [VLEMI]. All other bits cleared.		
MCSR	Unchanged		
DEAR	Set to the effective address of a byte of the load or store whose access caused the violation.		
Vector	IVPR$_{0:15}$ || IVOR5$_{16:27}$ || 4b0000		

10.9.7 Program Interrupt (IVOR6)

A program interrupt occurs when no higher priority exception exists and one or more of the following exception conditions occur:

- Illegal Instruction exception
- Privileged Instruction exception
- Trap exception

Zen will invoke an Illegal Instruction program exception on attempted execution of the following instructions:

- Unimplemented instructions
- Instruction from the illegal instruction class
- **mtspr** and **mfspr** instructions with an undefined SPR specified
- **mtdcr** and **mfdcr** instructions with an undefined DCR specified

Zen will invoke a Privileged Instruction program exception on attempted execution of the following instructions when MSR$_{PR}$=1 (user mode):

- A privileged instruction
- **mtspr** and **mfspr** instructions which specify a SPRN value with SPRN$_{5}$=1 (even if the SPR is undefined).

Zen will invoke an Trap exception on execution of the **tw** and **twi** instructions if the trap conditions are met and the exception is not also enabled as a Debug interrupt.

Table 10-30 lists register settings when a Program interrupt is taken.

Table 10-30. Program Interrupt—Register Settings

<table>
<tr><th>Register</th><th colspan="3">Setting Description</th></tr>
<tr><td>SRR0</td><td colspan="3">Set to the effective address of the excepting instruction.</td></tr>
<tr><td>SRR1</td><td colspan="3">Set to the contents of the MSR at the time of the interrupt</td></tr>
<tr><td>MSR</td><td>UCLE 0
SPE 0
WE 0
CE —
EE 0
PR 0</td><td>FP 0
ME —
FE0 0
DE —</td><td>FE1 0
IS 0
DS 0
PMM 0
RI —</td></tr>
<tr><td>ESR</td><td>Illegal:
Privileged:
Trap:</td><td colspan="2">PIL, [VLEMI]. All other bits cleared.
PPR, [VLEMI]. All other bits cleared.
PTR, [VLEMI]. All other bits cleared.</td></tr>
<tr><td>MCSR</td><td colspan="3">Unchanged</td></tr>
<tr><td>DEAR</td><td colspan="3">Unchanged</td></tr>
<tr><td>Vector</td><td colspan="3">IVPR$_{0:15}$ || IVOR6$_{16:27}$ || 4b0000</td></tr>
</table>

10.9.8 Floating-Point Unavailable Interrupt (IVOR7)

The Floating-point Unavailable exception is not used by MPC5676R.

10.9.9 System Call Interrupt (IVOR8)

A System Call interrupt occurs when a System Call (**sc**, **se_sc**) instruction is executed and no higher priority exception exists.

Table 10-31 lists register settings when a System Call interrupt is taken.

Table 10-31. System Call Interrupt—Register Settings

Register	Setting Description
SRR0	Set to the effective address of the instruction *following* the **sc** instruction.
SRR1	Set to the contents of the MSR at the time of the interrupt

Table 10-31. System Call Interrupt—Register Settings (continued)

MSR	UCLE 0 SPE 0 WE 0 CE — EE 0 PR 0	FP 0 ME — FE0 0 DE —	FE1 0 IS 0 DS 0 PMM 0 RI —
ESR	[VLEMI] All other bits cleared.		
MCSR	Unchanged		
DEAR	Unchanged		
Vector	IVPR$_{0:15}$ || IVOR8$_{16:27}$ || 4b0000		

10.9.10 Auxiliary Processor Unavailable Interrupt (IVOR9)

MPC5676R does not utilize this interrupt.

10.9.11 Decrementer Interrupt (IVOR10)

A Decrementer interrupt occurs when no higher priority exception exists, a Decrementer exception condition exists (TSR$_{DIS}$=1), and the interrupt is enabled (both TCR$_{DIE}$ and MSR$_{EE}$=1).

The Timer Status Register (TSR) holds the Decrementer interrupt bit set by the Timer facility when an exception is detected. Software must clear this bit in the interrupt handler to avoid repeated Decrementer interrupts.

Table 10-32 lists register settings when a Decrementer interrupt is taken.

Table 10-32. Decrementer Interrupt—Register Settings

Register	**Setting Description**		
SRR0	Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present.		
SRR1	Set to the contents of the MSR at the time of the interrupt		
MSR	UCLE 0 SPE 0 WE 0 CE — EE 0 PR 0	FP 0 ME — FE0 0 DE —	FE1 0 IS 0 DS 0 PMM 0 RI —
ESR	Unchanged		
MCSR	Unchanged		
DEAR	Unchanged		
Vector	IVPR$_{0:15}$ || IVOR10$_{16:27}$ || 4b0000		

10.9.12 Fixed-Interval Timer Interrupt (IVOR11)

The triggering of the exception is caused by selected bits in the Time Base register changing from 0 to 1.

A Fixed-Interval Timer interrupt occurs when no higher priority exception exists, a FIT exception exists (TSR_{FIS}=1), and the interrupt is enabled (both TCR_{FIE} and MSR_{EE}=1).

The Timer Status Register (TSR) holds the FIT interrupt bit set by the Timer facility when an exception is detected. Software must clear this bit in the interrupt handler to avoid repeated FIT interrupts.

Table 10-33 lists register settings when a FIT interrupt is taken.

Table 10-33. Fixed-Interval Timer Interrupt—Register Settings

Register	Setting Description		
SRR0	Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present.		
SRR1	Set to the contents of the MSR at the time of the interrupt		
MSR	UCLE 0 SPE 0 WE 0 CE — EE 0 PR 0	FP 0 ME — FE0 0 DE —	FE1 0 IS 0 DS 0 PMM 0 RI —
ESR	Unchanged		
MCSR	Unchanged		
DEAR	Unchanged		
Vector	$IVPR_{0:15}$ || $IVOR11_{16:27}$ || 4b0000		

10.9.13 Watchdog Timer Interrupt (IVOR12)

The triggering of the exception is caused by the first enabled watchdog time-out.

A Watchdog Timer interrupt occurs when no higher priority exception exists, a Watchdog Timer exception exists (TSR_{WIS}=1), and the interrupt is enabled (both TCR_{WIE} and MSR_{CE}=1).

The Timer Status Register (TSR) holds the Watchdog interrupt bit set by the Timer facility when an exception is detected. Software must clear this bit in the interrupt handler to avoid repeated Watchdog interrupts.

Table 10-34 lists register settings when a Watchdog Timer interrupt is taken.

Table 10-34. Watchdog Timer Interrupt—Register Settings

Register	Setting Description
CSRR0	Set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present.
CSRR1	Set to the contents of the MSR at the time of the interrupt

Table 10-34. Watchdog Timer Interrupt—Register Settings

MSR	UCLE 0 SPE 0 WE 0 CE 0 EE 0 PR 0	FP 0 ME — FE0 0 DE 0/—[1]	FE1 0 IS 0 DS 0 PMM 0 RI —
ESR	Unchanged		
MCSR	Unchanged		
DEAR	Unchanged		
Vector	IVPR$_{0:15}$ || IVOR12$_{16:27}$ || 4b0000		

[1] DE is cleared when the Debug APU is disabled. Clearing of DE is optionally supported by control in HID0 when the Debug APU is enabled.

10.9.14 Data TLB Error Interrupt (IVOR13)

A Data TLB Error interrupt occurs when no higher priority exception exists and a Data TLB Error exception exists due to a data translation lookup miss in the TLB.

Table 10-35 lists register settings when a DTLB interrupt is taken.

Table 10-35. Data TLB Error Interrupt—Register Settings

Register	Setting Description		
SRR0	Set to the effective address of the excepting load/store instruction.		
SRR1	Set to the contents of the MSR at the time of the interrupt		
MSR	UCLE 0 SPE 0 WE 0 CE — EE 0 PR 0	FP 0 ME — FE0 0 DE —	FE1 0 IS 0 DS 0 PMM 0 RI —
ESR	[ST], [SPE], [VLEMI]. All other bits cleared.		
MCSR	Unchanged		
DEAR	Set to the effective address of a byte of the load or store whose access caused the violation.		
Vector	IVPR$_{0:15}$ || IVOR13$_{16:27}$ || 4b0000		

10.9.15 Instruction TLB Error Interrupt (IVOR14)

A Instruction TLB Error interrupt occurs when no higher priority exception exists and an Instruction TLB Error exception exists due to an instruction translation lookup miss in the TLB.

Exception extensions implemented in Zen for PowerPC VLE involve extending the definition of the Instruction TLB Error Interrupt to include updating the ESR.

Table 10-36 lists register settings when an ITLB interrupt is taken.

Table 10-36. Instruction TLB Error Interrupt—Register Settings

<table>
<tr><th>Register</th><th colspan="3">Setting Description</th></tr>
<tr><td>SRR0</td><td colspan="3">Set to the effective address of the excepting instruction.</td></tr>
<tr><td>SRR1</td><td colspan="3">Set to the contents of the MSR at the time of the interrupt</td></tr>
<tr><td>MSR</td><td>UCLE 0
SPE 0
WE 0
CE —
EE 0
PR 0</td><td>FP 0
ME —
FE0 0
DE —</td><td>FE1 0
IS 0
DS 0
PMM 0
RI —</td></tr>
<tr><td>ESR</td><td colspan="3">[MIF] All other bits cleared.</td></tr>
<tr><td>MCSR</td><td colspan="3">Unchanged</td></tr>
<tr><td>DEAR</td><td colspan="3">Unchanged</td></tr>
<tr><td>Vector</td><td colspan="3">IVPR$_{0:15}$ || IVOR14$_{16:27}$ || 4b0000</td></tr>
</table>

10.9.16 Debug Interrupt (IVOR15)

There are multiple sources that can signal a Debug exception. A Debug interrupt occurs when no higher priority exception exists, a Debug exception exists in the Debug Status Register, and Debug interrupts are enabled (both DBCR0$_{IDM}$=1 (internal debug mode) and MSR$_{DE}$=1).

Table 10-37 lists register settings when a Debug interrupt is taken.

Table 10-37. Debug Interrupt—Register Settings

Register	Setting Description
CSRR0/ DSRR0[1]	Set to the effective address of the excepting instruction for IAC, BRT, RET, CRET, and TRAP. Set to the effective address of the next instruction to be executed *following* the excepting instruction for DAC and ICMP. For a UDE, IRPT, CIRPT, DCNT, or DEVT type exception, set to the effective address of the instruction that the processor would have attempted to execute next if no exception conditions were present.
CSRR1/ DSRR1	Set to the contents of the MSR at the time of the interrupt

Table 10-37. Debug Interrupt—Register Settings (continued)

<table>
<tr><td>MSR</td><td>UCLE 0
SPE 0
WE 0
CE —/0[2]
EE —/0[2]
PR 0</td><td>FP 0
ME —
FE0 0
DE 0</td><td>FE1 0
IS 0
DS 0
PMM 0
RI —</td></tr>
<tr><td>DBSR[3]</td><td>Unconditional Debug Event:
Instr. Complete Debug Event:
Branch Taken Debug Event:
Interrupt Taken Debug Event:
Critical Interrupt Taken Debug Event:
Trap Instruction Debug Event:
Instruction Address Compare:
Data Address Compare:
Return Debug Event:
Critical Return Debug Event:
Debug Counter Event:
External Debug Event:
and optionally, an
Imprecise Debug Event flag</td><td colspan="2">UDE
ICMP
BRT
IRPT
CIRPT
TRAP
{IAC1, IAC2, IAC3, IAC4}
{DAC1R, DAC1W, DAC2R, DAC2W}
RET
CRET
{DCNT1, DCNT2}
{DEVT1, DEVT2}

{IDE}</td></tr>
<tr><td>ESR</td><td colspan="3">Unchanged</td></tr>
<tr><td>MCSR</td><td colspan="3">Unchanged</td></tr>
<tr><td>DEAR</td><td colspan="3">Unchanged</td></tr>
<tr><td>Vector</td><td colspan="3">IVPR$_{0:15}$ || IVOR15$_{16:27}$ || 4b0000</td></tr>
</table>

---

[1] assumes that the Debug interrupt is precise

[2] conditional based on control bits in HID0

[3] Note that multiple DBSR bits may be set

10.9.17 SPE/EFPU APU Unavailable Interrupt (IVOR32)

The SPE APU Unavailable exception is taken if MSR_{SPE} is cleared and execution of a SPE or EFPU APU instruction other than the scalar floating-point instructions or **brinc** is attempted. When the SPE/EFPU APU Unavailable exception occurs, the processor suppresses execution of the instruction causing the exception. Table 10-38 lists register settings when a SPE/EFPU Unavailable interrupt is taken.

Table 10-38. SPE/EFPU Unavailable Interrupt—Register Settings

Register	Setting Description		
SRR0	Set to the effective address of the excepting SPE/EFPU instruction.		
SRR1	Set to the contents of the MSR at the time of the interrupt		
MSR	UCLE 0 SPE 0 WE 0 CE — EE 0 PR 0	FP 0 ME — FE0 0 DE —	FE1 0 IS 0 DS 0 PMM 0 RI —
ESR	SPE, [VLEMI]. All other bits cleared.		
MCSR	Unchanged		
DEAR	Unchanged		
Vector	$IVPR_{0:15}$ || $IVOR32_{16:27}$ || 4b0000		

10.9.18 Embedded Floating-point Data Interrupt (IVOR33)

The Embedded Floating-point Data interrupt is taken if no higher priority exception exists and a EFPU Floating-point Data exception is generated. When a Floating-point Data exception occurs, the processor suppresses execution of the instruction causing the exception.

Table 10-39 lists register settings when a EFPU Floating-point Data interrupt is taken.

Table 10-39. Embedded Floating-point Data Interrupt—Register Settings

Register	Setting Description		
SRR0	Set to the effective address of the excepting EFPU instruction.		
SRR1	Set to the contents of the MSR at the time of the interrupt		
MSR	UCLE 0 SPE 0 WE 0 CE — EE 0 PR 0	FP 0 ME — FE0 0 DE —	FE1 0 IS 0 DS 0 PMM 0 RI —
ESR	SPE, [VLEMI]. All other bits cleared.		
MCSR	Unchanged		
DEAR	Unchanged		
Vector	$IVPR_{0:15}$ || $IVOR33_{16:27}$ || 4b0000		

10.9.19 Embedded Floating-point Round Interrupt (IVOR34)

The Embedded Floating-point Round interrupt is taken when a EFPU floating-point instruction generates an inexact result and inexact exceptions are enabled.

Table 10-40 lists register settings when a EFPU Floating-point Round interrupt is taken.

Table 10-40. Embedded Floating-point Round Interrupt—Register Settings

Register	Setting Description
SRR0	Set to the effective address of the instruction following the excepting EFPU instruction.
SRR1	Set to the contents of the MSR at the time of the interrupt
MSR	UCLE 0 SPE 0 WE 0 CE — EE 0 PR 0 FP 0 ME — FE0 0 DE — FE1 0 IS 0 DS 0 PMM 0 RI —
ESR	SPE, [VLEMI]. All other bits cleared.
MCSR	Unchanged
DEAR	Unchanged
Vector	$IVPR_{0:15}$ || $IVOR34_{16:27}$ || 4b0000

10.9.20 Performance Monitor Interrupt (IVOR35)

MPC5676R provides a performance monitor interrupt that may be generated by an enabled condition or event. An enabled condition or event is as follows:

A PMC*x* register overflow condition occurs with the following settings:

- $PMLCax_{CE} = 1$; that is, for the given counter the overflow condition is enabled.
- $PMCx_{OV} = 1$; that is, the given counter indicates an overflow.

For a performance monitor interrupt to be signaled on an enabled condition or event, $PMGC0_{PMIE}$ must be set.

Although an exception condition may occur with $MSR_{EE} = 0$, the interrupt cannot be taken until $MSR_{EE} = 1$.

Table 10-41 lists register settings when an performance monitor interrupt is taken.

Table 10-41. Performance Monitor Interrupt—Register Settings

Register	Setting Description
SRR0	Set to the effective address of the next instruction to be executed.
SRR1	Set to the contents of the MSR at the time of the interrupt

Table 10-41. Performance Monitor Interrupt—Register Settings (continued)

<table>
<tr><th>Register</th><th colspan="3">Setting Description</th></tr>
<tr><td>MSR</td><td>UCLE 0
SPE 0
WE 0
CE —
EE 0
PR 0</td><td>FP 0
ME —
FE0 0
DE —</td><td>FE1 0
IS 0
DS 0
PMM 0
RI —</td></tr>
<tr><td>ESR</td><td colspan="3">Unchanged</td></tr>
<tr><td>MCSR</td><td colspan="3">Unchanged</td></tr>
<tr><td>DEAR</td><td colspan="3">Unchanged</td></tr>
<tr><td>Vector</td><td colspan="3">IVPR$_{0:15}$ || IVOR35$_{16:27}$ || 4b0000</td></tr>
</table>

10.10 Special Features

This section describes the WAIT instruction, Cache error detection, performance monitor and dual core reservations.

10.10.1 WAIT APU

The **wait** instruction allows software to shutdown the core and wait for an asynchronous interrupt or debug interrupt to occur. The instruction can be used to cease processor activity in both user and supervisor modes. Asynchronous interrupts which will cause the waiting state to be exited if enabled are critical input, external input, machine check pin, and Non-maskable interrupts (NMI).

Executing a **wait** instruction ensures that all instructions have completed before the **wait** instruction completes, causes processor instruction fetching to cease, and ensures that no subsequent instructions are initiated until an asynchronous interrupt or a debug interrupt occurs.

Once the **wait** instruction has completed, the program counter will point to the next sequential instruction.

Wait instruction can be used in conjunction with the SIU_HALT mechanism and SIU_HLTACK registers (Section 3.2.1.30, “Halt Register (SIU_HLT) and Section 3.2.1.31, “Halt Acknowledge Register (SIU_HLTACK)”) to enter a low power state while waiting.

Software must ensure that interrupts responsible for exiting the waiting state are enabled before executing a wait instruction.

wait

Wait for Interrupt

wait

0					5	6–20	21										31
0	1	1	1	1	1	///	0	0	0	0	1	1	1	1	1	0	/

10.10.2 Volatile Context Save/Restore APU

MPC5676R implements Volatile Context Save/Restore APU to support the capability to quickly save and restore volatile register context on entry into an interrupt handler. To support this functionality, a new set of instructions is defined as part of the APU.

- e_lmvgprw, e_stmvgprw - load/store multiple volatile gprs (r0, r3:r12)
- e_lmvsprw, e_stmvsprw - load/store multiple volatile sprs (CR, LR, CTR, and XER)
- e_lmvsrrw, e_stmvsrrw - load/store multiple volatile srrs (SRR0, SRR1)
- e_lmvcsrrw, e_stmvcsrrw - load/store multiple volatile csrrs (CSRR0, CSRR1)
- e_lmvdsrrw, e_stmvdsrrw - load/store multiple volatile dsrrs (DSRR0, DSRR1)
- e_lmvmcsrrw, e_stmvmcsrrw - load/store multiple volatile mcsrrs (MCSRR0, MCSRR1)

These instructions are available in VLE instruction pages to perform a multiple register load or store to a word aligned memory address.

10.10.3 Cache EDC (Error Detection Code) Checking

The CDACNTL and CDADATA registers can be used to test EDC in cache. Using these registers you can insert bit errors and then use test code to ensure errors are detected.

10.10.3.1 Cache Debug Access Control Register (CDACNTL)

The Cache Debug Access Control Register (CDACNTL) contains location information (T/D, CWAY, CSET, and WORD), and control (R/W and GO) needed to access the Cache Tag or Data SRAM arrays. Also included here are the SRAM parity bit values which must be supplied by the user for write accesses, and which will be supplied by the cache for read accesses. The CDACNTL register is shown in Figure 10-25.

T/D	0	CWAY	0	CSET	WORD	PARITY	0	CACHE	R/W	GO
0	1	2 3	4 5	6 7 8 9 10 11 12	13 14 15	16 17 18 19 20 21 22 23	24 25 26 27	28	29	30 31

DCR - 351; Read/Write; Reset - 0x0

Figure 10-25. CDACNTL Register

Table 10-42 provides bit definitions for the Cache Debug Access Control Register.

Table 10-42. Cache Debug Access Control Register Definition

Bit(s)	Name	Description
0	T/D	Tag / Data 0 Data array selected 1 Tag array selected
1	-	Reserved[1]
2:3	CWAY	Cache Way Specifies the cache way to be selected
4:5	-	Reserved[1]
6:12	CSET	Cache Set: Specifies the cache set to be selected
13:15	WORD	Word (Data array access only, I or D cache) Specifies one of eight words of selected set
16:23	PARITY / EDC CHECK BITS	Parity check bits[2] (I or D cache) EDC Mode (L1CSR[0,1][D,I]CEDT = 01): Dcache Data array: Byte parity bits. One bit per data byte. bit 16: Parity for byte 0, bit 17: Parity for byte 1.... bit 23: Parity for byte 7. Icache Data Array: parity check bits for data. Bits 16:23 correspond to p_dchk[0:7] (See the Data Checkbit Generation table in the *z759n3 Core Reference Manual*). Tag Array: parity check bits for tag. Bits 16:21 correspond to p_tchk[0:5] (See the Tag Checkbit Generation table in the *z759n3 Core Reference Manual*). bits 22:23 reserved.
24:27	-	Reserved[1]
28	CACHE	Cache Select Specifies the cache to be selected 0 Selects the data cache for the operation. 1 Selects the instruction cache for the operation.
29	R/W	Read / Write: 0 Selects write operation. Write the data in the CDADATA register to the location specified by this CDACNTL register. 1 Selects read operation. Read the cache memory location specified by this CDACNTL register and store the resulting data in the CDADATA register and store the parity bits in this CDACNTL register.
30:31	GO	GO command bits 00 Inactive or complete (no action taken) hardware sets GO=00 when an operation is complete 01 Read or write cache memory location specified by this CDACNTL register. 1x Reserved

[1] These bits are not implemented and should be written zero for future compatibility.

[2] Cache parity checkers assume odd parity when using parity protection. EDC coding is used otherwise.

10.10.3.2 Cache Debug Access Data Register (CDADATA)

The Cache Debug Access Data Register (CDADATA) contains the SRAM data for a debug access. The same register is used for Tag and Data SRAM read and write operations for both caches. Note that a single 32-bit word is accessed. Accessing an entire 64-bit doubleword requires two passes. The CDADATA register is shown in Figure 10-26.

TAG or DATA																															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

DCR - 350; Read/Write; Reset - Undefined/Unaffected

Figure 10-26. CDADATA Register

Table 10-43 provides bit definitions for the Cache Debug Access Data Register.

Table 10-43. Cache Debug Access Data Register Bit Definitions

Bit(s)	Name	Description
0:31	TAG	TAG Array Access Data - when accessing the tag array of either cache: 0:21 - Tag compare bits 22 - Reserved 23 - Valid bit 24:27 - Lock bits. These four bits should have the same value, 1-Locked, 0-Unlocked. 28:30 - Dirty bits - (data cache only). These three bits should have the same value, 1-Dirty, 0-Clean.
	DATA	DATA Array Access Data (Bytes 0:3 of the selected word) - when accessing the data array of either cache: 0:7 - byte 0 8:15 - byte 1 16:23 - byte 2 24:31 - byte 3

10.10.4 Performance Monitor

The performance monitor provides the ability to count predefined events and processor clocks associated with particular operations, such as cache misses, mispredicted branches, or the number of cycles an execution unit stalls. The count of such events can be used to trigger the performance monitor interrupt.

The performance monitor can be used to do the following:

- Improve system performance by monitoring software execution and then recoding algorithms for more efficiency. For example, memory hierarchy behavior can be monitored and analyzed to optimize task scheduling or data distribution algorithms.
- Characterize processors in environments not easily characterized by benchmarking.
- Help system developers bring up and debug their systems.

For a complete description of Performance Monitor see *z759n3 Core Reference Manual*.

10.10.5 Dual Core Reservations

A reservation is established when either core issues a load-type reservation instruction (lwarx, lharx, lbarx). Reservations are tagged with the load address, data size and core number.

A memory reservation is awarded on a most-recent basis. A reservation established by one core can be cancelled by a subsequent l{w,h,b}arx from the same core, or an l{w,h,b}arx to the same address by the other core. Only one reservation per core at any time is supported.

Once a reservation is established by a core, a ‘normal’ (non-reserved) write request by either core to the reserved location is allowed to proceed unimpeded, but the reservation is cancelled. The granularity of the current ‘normal’ (non-reserved) write request must be sufficiently large enough to overlap any part of the reservation granule in order to cancel the reservation.

Once a reservation is established by a core, a st{w,h,b}cx write request by the other core to the reserved location is aborted and the reservation is cancelled. The granularity of the current st{w,h,b}cx write request must be sufficiently large enough to overlap any part of the reservation granule in order to clear the reservation.

Any st{w,h,b}cx write request by either core after a reservation has been cancelled will cause the write to abort.

The reservation support mechanism can be independently enabled/disabled on a per-slave basis in the MPU.

Chapter 11
AMBA Crossbar Switch (XBAR)

11.1 Introduction

This chapter describes the multi-port crossbar switch (XBAR), which supports simultaneous connections between master ports and slave ports. XBAR supports a 32-bit address bus width and a 64-bit data bus width at all master and slave ports and runs at half the system frequency.

11.1.1 Overview

The XBAR allows concurrent transactions to occur from any master port to any slave port. It is possible for all master ports and slave ports to be in use at the same time as a result of independent master requests. If a slave port is simultaneously requested by more than one master port, arbitration logic selects the higher priority master and grant it ownership of the slave port. All other masters requesting that slave port are stalled until the higher priority master completes its transactions.

By default, requesting masters are granted access based on a fixed priority. A round-robin priority mode also is available. In this mode, requesting masters are treated with equal priority and are granted access to a slave port in round-robin fashion, based upon the ID of the last master to be granted access. A block diagram of the XBAR is shown in Figure 11-1.

The XBAR can place a slave port in a low-power park mode to avoid dissipating any power transitional address, control or data signals when the master port is not actively accessing the slave port. There is a one-cycle arbitration overhead for exiting low-power park mode.

11.1.2 Block Diagram

Figure 11-1 shows a block diagram of the crossbar switch.

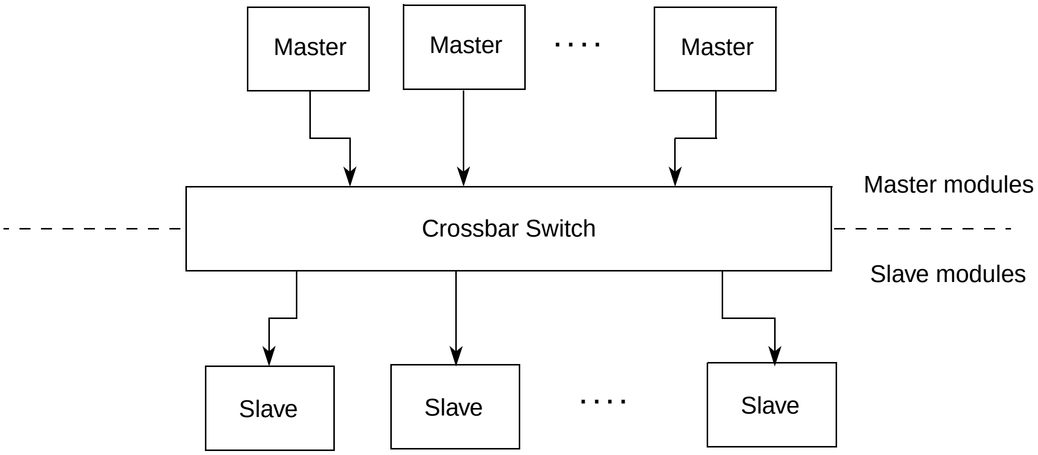


Figure 11-1. XBAR Block Diagram

Table 11-1 gives the crossbar switch port for each master and slave, and the assigned and fixed ID number for each master. The following table shows the master ID numbers as they relate to the master port numbers:

Table 11-1. XBAR Switch Ports

Module	Port		Master ID
	Type	Number	
e200z7 core0 — CPU instruction	Master	0	0
e200z7 core0 — Data	Master	1	0
Nexus 3			8
e200z7 core1 — CPU instruction	Master	2	1
e200z7 core1 — Data	Master	3	1
Nexus 3			9
eDMA_A	Master	4	4
eDMA_B	Master	5	5
FlexRay	Master	6	6
Reserved[1]	Master	7	3
On-chip Flash (accessed by core 0 only)	Slave	0	—
EBI (development bus)	Slave	1	—
On-chip SRAM	Slave	2	—
On chip Flash (accessed by all masters except core 0)	Slave	3	—
Peripheral bridge A (PBRIDGE_A)	Slave	6	—
Peripheral bridge B (PBRIDGE_B)	Slave	7	—

[1] This port is not usable by customers, but exists in the device. Therefore, the priority must be set to a unique value in the Master Priority Registers.

11.1.3 Features

- Multiple master and slave ports with programmable priorities and attributes.
- 32-bit address, 64-bit data paths
- Fully concurrent transfers between independent master and slave ports

11.1.4 Modes of Operation

11.1.4.1 Normal Mode

In normal mode, the XBAR provides the register interface and logic that controls crossbar switch configuration.

11.1.4.2 Debug Mode

The XBAR operation in debug mode is identical to operation in normal mode.

11.2 Memory Map and Register Definition

The memory map for the XBAR program-visible registers is shown in Table 11-2.

Table 11-2. XBAR Register Memory Map

Address	Register	Bits	Access	Reset Value	Section/Page
Base = 0xFFF0_4000	XBAR_MPR0—Master priority register for slave port 0	32	R/W	0x54320010	11.2.1.1/11-4
Base + (0x0004–0x000F)	Reserved				
Base + 0x0010	XBAR_SGPCR0—General-purpose control register for slave port 0	32	R/W	0x0000_0000	11.2.1.2/11-6
Base + (0x0014–0x00FF)	Reserved				
Base + 0x0100	XBAR_MPR1—Master priority register for slave port 1	32	R/W	0x54320010	11.2.1.1/11-4
Base +(0x0104–0x010F)	Reserved				
Base + 0x0110	XBAR_SGPCR1—General-purpose control register for slave port 1	32	R/W	0x0000_0000	11.2.1.2/11-6
Base + (0x0114–0x01FF)	Reserved				
Base + 0x0200	XBAR_MPR2—Master priority register for slave port 2	32	R/W	0x54320010	11.2.1.1/11-4
Base +(0x0204–0x020F)	Reserved				
Base + 0x0210	XBAR_SGPCR2—General-purpose control register for slave port 2	32	R/W	0x0000_0000	11.2.1.2/11-6
Base + (0x0214–0x02FF)	Reserved				

Table 11-2. XBAR Register Memory Map (continued)

Address	Register	Bits	Access	Reset Value	Section/Page
Base + 0x0300	XBAR_MPR3—Master priority register for slave port 3	32	R/W	0x54320010	11.2.1.1/11-4
Base + (0x0304–0x030F)	Reserved				
Base + 0x0310	XBAR_SGPCR3—General-purpose control register for slave port 3	32	R/W	0x0000_0000	11.2.1.2/11-6
Base + (0x0314–0x05FF)	Reserved				
Base + 0x0600	XBAR_MPR6—Master priority register for slave port 6	32	R/W	0x54320010	11.2.1.1/11-4
Base + (0x0604–0x060F)	Reserved				
Base + 0x0610	XBAR_SGPCR6—General-purpose control register for slave port 6	32	R/W	0x0000_0000	11.2.1.2/11-6
Base + (0x0614–0x06FF)	Reserved				
Base + 0x0700	XBAR_MPR7—Master priority register for slave port 7	32	R/W	0x54320010	11.2.1.1/11-4
Base + (0x0704–0x070F)	Reserved				
Base + 0x0710	XBAR_SGPCR7—General-purpose control register for slave port 7	32	R/W	0x0000_0000	11.2.1.2/11-6
Base + **(**0x0714–0x07FF)	Reserved				

11.2.1 Register Descriptions

There are two registers for each slave port of the XBAR. The registers can only be accessed in supervisor mode using 32-bit accesses.

The slave SGPCR also features a bit (RO), which when written with a 1, prevents all slave registers for that port from being written to again until a reset occurs. The registers remain readable, but future write attempts have no effect on the registers and are terminated with an error response.

11.2.1.1 Master Priority Registers (XBAR_MPR*n*)

The XBAR_MPR for a slave port sets the priority of each master port when operating in fixed priority mode. They are ignored in round-robin priority mode unless more than one master has been assigned high priority by a slave.

NOTE

Masters must be assigned unique priority levels.

The master priority register can only be accessed in supervisor mode with 32-bit accesses. After the read only (RO) bit is set in the slave general-purpose control register, the master priority register can only be read. Attempts to write to it have no effect on the MPR and result in an error.

NOTE

XBAR_MPR must be written with a read/modify/write for code compatibility.

Address: Base + 0x0000 (XBAR_MPR0)
Base + 0x0100 (XBAR_MPR1)
Base + 0x0200 (XBAR_MPR2)
Base + 0x0300 (XBAR_MPR3)
Base + 0x0600 (XBAR_MPR6)
Base + 0x0700 (XBAR_MPR7)

Access: Supervisor R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	1	1	1	0	MSTR6			0	MSTR5			0	MSTR4		
W	0	1	1	1	0				0				0			
Reset	0	1	1	1	0	1	1	0	0	1	0	1	0	1	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	MSTR3			0	MSTR2			0	MSTR1			0	MSTR0		
W	0				0				0				0			
Reset	0	0	1	1	0	0	1	0	0	0	0	1	0	0	0	0

Figure 11-2. Master Priority Registers (XBAR_MPR*n*)

Table 11-3. XBAR_MPR*n* Descriptions

Field	Description
0–3	Reserved, must be set to 0b0111. Undefined operation if not.
4	Reserved, must be cleared.
5–7 MSTR6	Master 6 priority. Set the arbitration priority for master port 6 on the associated slave port. 000 Master 6 has the highest priority when accessing slave port *n*. ⋮ 101 Master 6 has the lowest priority when accessing slave port *n*. 110–111 Invalid values
8	Reserved, must be cleared.
9–11 MSTR5	Master 5 priority. Set the arbitration priority for master port 5 on the associated slave port. 000 Master 5 has the highest priority when accessing slave port *n*. ⋮ 101 Master 5 has the lowest priority when accessing slave port *n*. 110–111 Invalid values
12	Reserved, must be cleared.

Table 11-3. XBAR_MPR*n* Descriptions (continued)

Field	Description
13–15 MSTR4	Master 4 priority. Set the arbitration priority for master port 4 on the associated slave port. 000 Master 4 has the highest priority when accessing slave port *n*. ⋮ 101 Master 4 has the lowest priority when accessing slave port *n*. 110–111 Invalid values
16	Reserved, must be cleared.
17–19 MSTR3	Master 3 priority. Set the arbitration priority for master port 3 on the associated slave port. 000 Master 3 has the highest priority when accessing slave port *n*. ⋮ 101 Master 3 has the lowest priority when accessing slave port *n*. 110–111 Invalid values
20	Reserved, must be cleared.
21–23 MSTR2	Master 2 priority. Set the arbitration priority for master port 2 on the associated slave port. 000 Master 2 has the highest priority when accessing slave port *n*. ⋮ 101 Master 2 has the lowest priority when accessing slave port *n*. 110–111 Invalid values
24	Reserved, must be cleared.
25–27 MSTR1	Master 1 priority. Set the arbitration priority for master port 1 on the associated slave port. 000 Master 1 has the highest priority when accessing slave port *n*. ⋮ 101 Master 1 has the lowest priority when accessing slave port *n*. 110–111 Invalid values
28	Reserved, must be cleared.
29–31 MSTR0	Master 0 priority. Set the arbitration priority for master port 0 on the associated slave port. 000 Master 0 has the highest priority when accessing slave port *n*. ⋮ 101 Master 0 has the lowest priority when accessing slave port *n*. 110–111 Invalid values

11.2.1.2 Slave General-Purpose Control Registers (XBAR_SGPCR*n*)

The XBAR_SGPCR*n* of a slave port controls several features of the slave port, including the following:

- Round-robin or fixed arbitration policy for a particular slave port
- Write protection of any slave port registers
- Parking algorithm used for a slave port

The PARK field indicates which master port this slave port parks on when no active access attempts are being made to the slave and the parking control field is set to park on a specific master. XBAR_SGPCR*n*[PARK] must only be programmed to select master ports that are actually available on the device, otherwise undefined behavior results. The low-power park feature can result in an overall power savings if the slave port is not saturated; however, an extra clock of latency results whenever any master tries to access a slave (not being accessed by another master) because it is not parked on any master.

The XBAR_SGPCR can only be accessed in supervisor mode with 32-bit accesses. After the RO (read only) bit is set in the XBAR_SGPCR, the XBAR_SGPCR and the SBAR_MPR can only be read. Attempts to write to them have no effect and results in an error.

NOTE

> Some of the unused bits in the SGPCR*n* registers are writeable and readable, but they serve no function. Setting any of these bits has no effect on the operation of this module.

Address: Base + 0x0010 (XBAR_SGPCR0)
Base + 0x0110 (XBAR_SGPCR1)
Base + 0x0210 (XBAR_SGPCR2)
Base + 0x0310 (XBAR_SGPCR3)
Base + 0x0610 (XBAR_SGPCR6)
Base + 0x0710 (XBAR_SGPCR7)

Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RO[1]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	ARB		0	0	PCTL		0	PARK		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] After this bit is set, only a hardware reset clears it.

Figure 11-3. Slave General-Purpose Control Registers (XBAR_SGPCR*n*)

Table 11-5. XBAR_SGPCR*n* Field Descriptions

Field	Description
0 RO	Read only. Used to force all of a slave port's registers to be read only. After written to 1, it can only be cleared by hardware reset. 0 All this slave port's registers can be written. 1 All this slave port's registers are read only and cannot be written (attempted writes have no effect and result in an error response).
1–21	Reserved, must be cleared.

Table 11-5. XBAR_SGPCR*n* Field Descriptions (continued)

Field	Description
22–23 ARB	Arbitration mode. Used to select the arbitration policy for the slave port. This field is initialized by hardware reset. 00 Fixed priority using MPR 01 Round-robin priority 10 Invalid value 11 Invalid value
24–25	Reserved, must be cleared.
26–27 PCTL	Parking control. Used to select the parking algorithm used by the slave port. This field is initialized by hardware reset. 00 When no master is making a request, the arbiter parks the slave port on the master port defined by the PARK control field. 01 POL—Park on last. When no master is making a request, the arbiter parks the slave port on the last master to own the slave port. 10 LPP—Low-power park. When no master is making a request, the arbiter parks the slave port on no master and drives all slave port outputs to a safe state. 11 Invalid value
28	Reserved, must be cleared.
29–31 PARK	Park. Used to determine which master port this slave port parks on when no masters are actively making requests. PCTL must be set to 0b00. 000 Park on master port 0 001 Park on master port 1 010 Park on master port 2 011 Park on master port 3 100 Park on master port 4 101 Park on master port 5 110 Park on master port 6 111 Invalid value Valid parking options vary by slave port number, as given in Table 11-6

Table 11-6. Master Port Parking Options

Slave Port Number	Slave Module	Valid Master Ports
0	Flash	0, 1
1	EBI	0, 1, 2, 3, 4, 5, 6
2	SRAM	0, 1, 2, 3, 4, 5, 6
3	Flash	2, 3, 4, 5, 6
6	PBRIDGE_A	0, 1, 2, 3, 4, 5, 6
7	PBRIDGE_B	0, 1, 2, 3, 4, 5, 6

11.3 Functional Description

This section describes the functionality of the XBAR in more detail.

11.3.1 Overview

The main goal of the XBAR is to increase overall system performance by allowing multiple masters to communicate concurrently with multiple slaves. To maximize data throughput, it is essential to keep arbitration delays to a minimum.

This section examines data throughput from the point of view of masters and slaves, detailing when the XBAR stalls masters, or inserts bubbles on the slave side.

11.3.2 General Operation

When a master makes an access to the XBAR from an idle master state, the access is taken immediately by the XBAR. If the targeted slave port of the access is available (that is, the requesting master is currently granted ownership of the slave port), the access is immediately presented on the slave port. It is possible to make single clock (zero wait state) accesses through the XBAR by a granted master. If the targeted slave port of the access is busy or parked on a different master port, the requesting master receives wait states until the targeted slave port can service the master request. The latency in servicing the request depends on each master’s priority level and the responding slave’s access time.

Because the XBAR appears to be simply another slave to the master device, the master device has no indication that it owns the slave port it is targeting. While the master does not have control of the slave port it is targeting, it is wait-stated.

A master is given control of a targeted slave port only after a previous access to a different slave port has completed, regardless of its priority on the newly targeted slave port. This prevents deadlock from occurring when a master has the following conditions:

- Outstanding request to slave port A that has a long response time
- Pending access to a different slave port B
- Lower priority master also makes a request to the different slave port B.

In this case, the lower priority master is granted bus ownership of slave port B after a cycle of arbitration, assuming the higher priority master slave port A access is not terminated.

After a master has control of the slave port it is targeting, the master remains in control of that slave port until it gives up the slave port by running an IDLE cycle, leaves that slave port for its next access, or loses control of the slave port to a higher priority master with a request to the same slave port. However, because all masters run a fixed-length burst transfer to a slave port, it retains control of the slave port until that transfer sequence is completed. In round-robin arbitration mode, the current master is forced to hand off bus ownership to an alternately requesting master at the end of its current transfer sequence.

When a slave bus is idled by the XBAR, it can be parked on the master port using the PARK bits in the XBAR_SGPCR (slave general-purpose control register), or on the last master to have control of the slave port. This can avoid the initial clock of the arbitration delay if the master must arbitrate to gain control of the slave port. The slave port can also be put into low-power park mode to save power.

11.3.3 Master Ports

The XBAR terminates an access and it is not allowed to pass through the XBAR unless the master currently is granted access to the slave port to which the access is targeted. A master access is taken if the slave port to which the access decodes is either currently servicing the master or is parked on the master. In this case, the XBAR is completely transparent and the master access is immediately transmitted on the slave bus and no arbitration delays are incurred. A master access stall if the access decodes to a slave port that is busy serving another master, parked on another master or is in low-power park mode.

If the slave port is currently parked on another master or is in low-power park mode, and no other master is requesting access to the slave port, then only one clock of arbitration is incurred. If the slave port is currently serving another master of a lower priority and the master has a higher priority than all other requesting masters, then the master gains control over the slave port as soon as the data phase of the current access is completed. If the slave port is currently servicing another master of a higher priority, then the master gains control of the slave port after the other master releases control of the slave port if no other higher priority master is also waiting for the slave port.

A master access is responded to with an error if the access decodes to a location not occupied by a slave port. This is the only time the XBAR directly responds with an error response. All other error responses received by the master are the result of error responses on the slave ports being passed through the XBAR.

11.3.4 Slave Ports

The goal of the XBAR with respect to the slave ports is to keep them 100% saturated when masters are actively making requests. To do this the XBAR must not insert any bubbles onto the slave bus unless absolutely necessary.

There is only one instance when the XBAR forces a bubble onto the slave bus when a master is actively making a request. This occurs when a hand-off of bus ownership occurs and there are no wait states from the slave port. A requesting master which does not own the slave port is granted access after a one clock delay.

The only other time the XBAR has control of the slave port is when no masters are making access requests to the slave port and the XBAR is forced to either park the slave port on a specific master, or place the slave port into low-power park mode. In these cases, the XBAR forces IDLE for the transfer type.

11.3.5 Priority Assignment

Each master port must be assigned a unique 2-bit priority level in fixed priority mode. If multiple master ports are assigned the same priority level within a register (XBAR_MPR) undefined behavior results.

11.3.6 Arbitration

XBAR supports two arbitration schemes; a simple fixed-priority comparison algorithm, and a round-robin fairness algorithm. The arbitration scheme is independently programmable for each slave port.

11.3.6.1 Fixed Priority Operation

When operating in fixed-priority arbitration mode, each master is assigned a unique priority level in the XBAR_MPR. If two masters both request access to a slave port, the master with the highest priority in the selected priority register gains control over the slave port.

Any time a master makes a request to a slave port, the slave port checks to see if the new requesting master's priority level is higher than that of the master that currently has control over the slave port (if any). The slave port does an arbitration check at every clock edge to ensure that the proper master (if any) has control of the slave port.

If the new requesting master's priority level is higher than that of the master that currently has control of the slave port, the higher priority master is granted control at the termination of any currently pending access, assuming the pending transfer is not part of a burst transfer.

A new requesting master must wait until the end of the fixed-length burst transfer, before it is granted control of the slave port. But if the new requesting master's priority level is lower than that of the master that currently has control of the slave port, the new requesting master is forced to wait until the master that currently has control of the slave port is finished accessing the current slave port.

11.3.6.2 Round-Robin Priority Operation

When operating in round-robin mode, each master is assigned a relative priority based on the master port number. This relative priority is compared to the port number of the last master to perform a transfer on the slave bus. The highest priority requesting master becomes the owner of the slave bus at the next transfer boundary (accounting for fixed-length burst transfers). Priority is based on how far ahead the port number of the requesting master is to the port number of the last master.

After granted access to a slave port, a master may perform as many transfers as desired to that port until another master makes a request to the same slave port. The next master in line is granted access to the slave port when the current transfer is completed, or possibly on the next clock cycle if the current master has no pending access request.

As an example of arbitration in round-robin mode, assume the three masters have ID's 0, 1, and 2. If the last master of the slave port was master 1, and masters 0 and 2 make simultaneous requests, they are serviced in the order 2 and then 0 assuming no further requests are made.

As another example, if master 1 is waiting on a response from a slow slave and has no further pending access to that slave, no other masters are requesting, and master 0 then makes a request, master 0's request is granted on the next clock (assuming that master 1's transfer is not a burst transfer), and the request information for master 0 is driven to the slave as a pending access. If master 2 were to make a request after master 0 has been granted access, but prior to master 0's access being accepted by the slave, master 0 maintains the grant on the slave port, and master 2 is delayed until the next arbitration boundary, which occurs after the transfer is complete. The round-robin pointer is reset to 0, so if master 1 has another request that occurs before master 0's transfer completes, master 1 is the granted the bus. This implies a worst case latency of *N* transfers for a system with *N* masters.

Parking may continue to be used in round-robin mode, but affects the round-robin pointer unless the parked master actually performs a transfer. Hand-off to the next master in line occurs after one cycle of arbitration.

The slave port does an arbitration check at every clock edge to ensure that the proper master (if any) has control of the slave port.

A new requesting master must wait until the end of the fixed-length burst transfer, before it is granted control of the slave port. If the new requesting master's priority level is lower than that of the master that currently has control of the slave port, the new requesting master is forced to wait until the master that currently has control of the slave port completes its access.

11.3.6.2.1 Parking

If no master is currently requesting the slave port, the slave port is parked. The slave port parks in one of three places, indicated by the value of the PCTL field in the XBAR_SGPCR.

- If park-on-specific master mode is selected, the slave port parks on the master designated by the PARK field. When the master accesses the slave port again, a one clock arbitration penalty is incurred only for an access request made by another master port to the slave port. No other arbitration penalties are incurred. All other masters pay a one clock penalty.
- If park-on-last (POL) mode is selected, then the slave port parks on the last master to access it, passing that master's signals through to the slave bus. When the master accesses the slave port again, no other arbitration penalties are incurred except that a one clock arbitration penalty is incurred for each access request to the slave port made by another master port. All other masters pay a one clock penalty.
- If the low-power-park (LPP) mode is selected, then the slave port enters low-power park mode. It is not under control by any master and does not transmit any master signals to the slave bus. All slave bus activity halts because all slave bus signals are not toggling. This saves power if the slave port is not used for some time. However, when a master does make a request to a slave port parked in low-power-park, a one clock arbitration delay is incurred to get ownership of the slave port.

Chapter 12
Cyclic Redundancy Checker (CRC) Unit

12.1 Overview

The CRC module provides a fast on-chip capability for verifying code and data integrity. This capability is particularly important in safety applications. Examples include:

- Verifying memory integrity by setting it to a known value, calculating a checksum and comparing the calculated checksum against a stored checksum value
- Verifying code integrity by comparing its calculated checksum to its stored checksum value
- Verifying the integrity of data received from a network by comparing its received checksum to its calculated checksum

CRC functionality can be implemented in software but there are significant speed advantages to be gained by offloading CRC computation tasks from the processor core to the CRC module. Further gains are made when data is written to the CRC module via DMA.

NOTE

This chapter does not discuss the details of computing CRC checksums but there are many articles to be found via internet searches.

12.2 Features

The CRC module on the MPC5676R includes the following features:

- Three “contexts”. A context is a CRC engine with its own independent set of configuration and data registers. The MPC5676R CRC module can process up to three separate data streams concurrently.
- Each context supports CRC-16-CCITT and CRC-32 ethernet polynomials
- Bit-swap and bit-inversion operations can be applied on the final CRC signature
- Support for byte/half-word/word width of the input data stream
- Computation is performed with zero wait states

12.2.1 Access and Performance

All CRC registers are accessible (read/write) in each access mode: user and supervisor.

The following bus operations (contiguous byte enables) are supported:

- 32-bit data read/write operations to any register

- Low and high half-word read/write operations to any register[1]
- Byte data write/read operations to any register[2]

Bus performance of the operations is as follows:

- Zero wait state (single bus cycle) for each read/write to the CRC_CFG and CRC_INP registers
- Zero wait state (single bus cycle) for each write operation to the CRC_ CSTAT register
- Double wait state (3 bus cycles) for each read operation to the CRC_ CSTAT or CRC_OUTP registers immediately following (next clock cycle) a write operation to the CRC_CSTAT, CRC_INP or CRC_CFG registers belonging to the same context. In all the other cases no wait states are inserted.

The following will result in transfer errors:

- Unaligned reads or writes
- Any attempt to read or write an address that is assigned to the CRC module but not actually mapped to a register.

12.3 Calculating a CRC Checksum

The MPC5676R CRC module has three independent sets of CRC engines and registers, each set is called a context. Each context supports a single data stream, structured as a sequence of bytes, half-words or words, written to its input register. Since the context operate independently, the CRC module can process up to three data streams concurrently.

Figure 12-1 illustrates the steps to calculating a CRC checksum (also called a signature) for a data stream:

1. Configure the context to be used
2. Write a seed value into CRC_CSTAT register
3. Write the data to the CRC input register (CRC_INP), until the end of the data to be checked.
4. Retrieve the calculated checksum from the CRC_OUTP register and verify the checksum against a stored value.

---

[1] 16-bit operations must be aligned to 16-bit boundaries, i.e., bits 0-15 or bits 16-31. Any unaligned operation results in a bus error.

[2] Byte operations must be aligned to 8-bit boundaries, i.e., bits 0-7, bits 8-15, bits 16-23, or bits 24-31. Any unaligned operation results in a bus error.

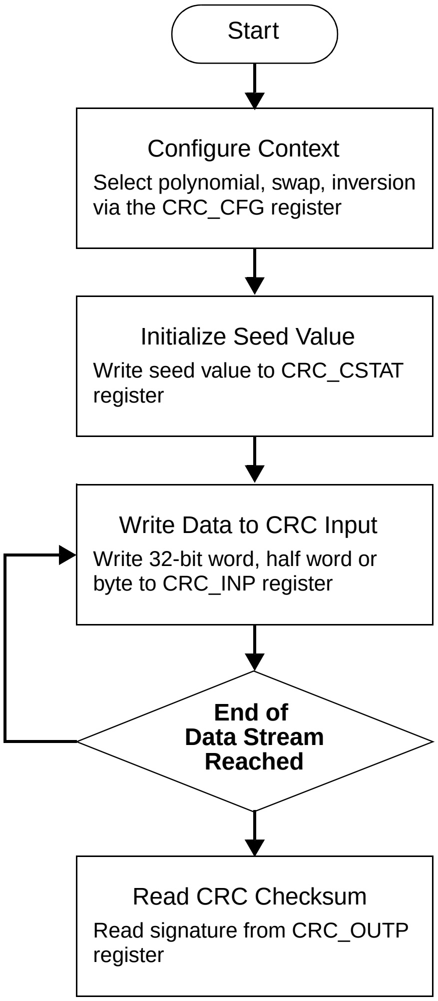


Figure 12-1. CRC Checksum Processing Flow

The following sections describe each step in the process.

12.3.1 Configuring the Context

A context consists of a CRC engine and a dedicated set of registers. The MPC5676R CRC module includes three contexts.

The configuration step consists of:

- Selecting the polynomial
- Specifying whether a swap operation is to be performed on the output
- Specifying whether a bit inversion is to be performed on the output

Selections are made by writing the appropriate values to fields in the CRC_CFG register.

Two standard polynomials are provided by the CRC module: CRC-16-CCITT (x25 protocol) and CRC-32 (ethernet protocol). They are illustrated in Equation 12-1 and Equation 12-2.

$$X^{16} + X^{12} + X^{5} + 1$$

: CRC-CCITT (x25 protocol) ***Eqn. 12-1***

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X + 1$$

: CRC-32 (ethernet protocol) ***Eqn. 12-2***

The polynomial to be used is based on system requirements.

In case of usage of the CRC signature for encapsulation in the data frame of a communication protocol (e.g., SPI) a bit swap (MSB -> LSB, LSB -> MSB) and/or bit inversion of the final CRC signature can be applied (CRC_OUTP register).

12.3.2 Initializing the Context Seed Value

A CRC checksum can be thought of as the remainder of a division of a long, arbitrary number (the data stream) by a known fixed value. The known fixed value is known as the seed value. The same seed value must be used to generate the checksums that are to be compared to each other.

The seed value is specified in the CRC current status register (CRC_CSTAT), which as a dual purpose. Before CRC checksum calculation is performed, i.e., during the configuration phase, the CRC_CSTAT register is used to program the seed value. During CRC checksum calculation, the register contains the current checksum value.

The seed value can be any arbitrary 32-bit value.

NOTE

As with the CRC configuration register (CRC_CFG) register the CRC_CSTAT register can only be written during the configuration phase. A write protection error generated by a write operation to this register indicates it is currently in use.

12.3.3 Writing the Data Stream to the Context Input

After the context is configured and a seed is written, the data stream is written to the context's input register (CRC_INP). The CRC_INP register can be written at byte, half-word (high and low) or word in any sequence. In case of half-word write operation, the bytes must be contiguous.

NOTE

The CRC_INP register only supports aligned writes. Half-word (16-bit) writes must be either to bits 0-15 or 16-31. Byte writes must be to bits 0-7, bits 8-15, bits 16-23 or bits 24-31.

The writes can be by the processor core or by DMA transfer. The writes continue until the end of the data stream is reached.

12.3.4 Reading the Checksum

After writing of the data stream to the input register has been completed, the checksum is read from the output register (CRC_OUTP). The CRC_OUTP register includes the final checksum (signature) corresponding to the CRC_CSTAT register value with swap and inversion operations applied, if selected via the CRC_CFG register.

In case of CRC-16-CCITT polynomial only the16 least significant bits have meaning. The 16 most significant bits are tied 0b0 during the computation.

12.4 Register Descriptions

Table 12-1. CRC Register Map

Context	Address[1]	Register	Section/Page
1	CRC_BASE + 0x0000	CRC Configuration Register (CRC_CFG)	See section 12.4.1/12-6
	CRC_BASE + 0x0004	CRC Input Register (CRC_INP)	See section 12.4.2/12-7
	CRC_BASE + 0x0008	CRC Current Status Register (CRC_CSTAT)	See section 12.4.3/12-7
	CRC_BASE + 0x000C	CRC Output Register (CRC_OUTP)	See section 12.4.4/12-8
2	CRC_BASE + 0x0010	CRC Configuration Register (CRC_CFG)	See section 12.4.1/12-6
	CRC_BASE + 0x0014	CRC Input Register (CRC_INP)	See section 12.4.2/12-7
	CRC_BASE + 0x0018	CRC Current Status Register (CRC_CSTAT)	See section 12.4.3/12-7
	CRC_BASE + 0x001C	CRC Output Register (CRC_OUTP)	See section 12.4.4/12-8
3	CRC_BASE + 0x0020	CRC Configuration Register (CRC_CFG)	See section 12.4.1/12-6
	CRC_BASE + 0x0024	CRC Input Register (CRC_INP)	See section 12.4.2/12-7
	CRC_BASE + 0x0028	CRC Current Status Register (CRC_CSTAT)	See section 12.4.3/12-7
	CRC_BASE + 0x002C	CRC Output Register (CRC_OUTP)	See section 12.4.4/12-8

[1] CRC_BASE for the MPC5676R is 0xFFE6_8000

12.4.1 CRC Configuration Register (CRC_CFG)

Offset: CRC_BASE + 0x0000
CRC_BASE + 0x0010
CRC_BASE + 0x0020

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	POLYG	SWAP	INV
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0/1[1]	0	0

Figure 12-2. CRC Configuration Register (CRC_CFG)

[1] Reset value is 1 for Context 2 and 0 for Context 1 and Context 3.

Field	Description
0-28	Reserved
29 POLYG	POLYG: *Polynomial selection* 0: CRC-CCITT polynomial. 1: CRC-32 polynomial. This bit can be read and written by software. This bit can be written only during the configuration phase.
30 SWAP	SWAP: *SWAP selection* 0: No swap selection applied on the CRC_OUTP content 1: Swap selection (MSB -> LSB, LSB -> MSB) applied on the CRC_OUTP content. In case of CRC-CCITT polynomial the swap operation is applied on the 16 LSB bits. This bit can be read and written by software. This bit can be written only during the configuration phase.
31 INV	INV: *INV selection* 0: No inversion selection applied on the CRC_OUTP content 1: Inversion selection (bit x bit) applied on the CRC_OUTP content. In case of CRC-CCITT polynomial the inversion operation is applied on the 16 LSB bits. This bit can be read and written by software. This bit can be written only during the configuration phase.

12.4.2 CRC Input Register (CRC_INP)

Offset: CRC_BASE + 0x0004
CRC_BASE + 0x0014
CRC_BASE + 0x0024

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	INP															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	INP															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 12-3. CRC Input Register (CRC_INP)

Field	Description
0-31	INP: *Input data for the CRC computation* The INP register can be written at byte, half-word (high and low) or word in any sequence. In case of half-word write operation, the bytes must be contiguous. This register can be read and written by software.

12.4.3 CRC Current Status Register (CRC_CSTAT)

Offset: CRC_BASE + 0x0008
CRC_BASE + 0x0018
CRC_BASE + 0x0028

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CSTAT															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CSTAT															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 12-4. CRC Current Status Register (CRC_CSTAT)

Field	Description
0-31	CSTAT: *Status of the CRC signature* The CSTAT register includes the current status of the CRC signature. No bit swap and inversion are applied to this register. In case of CRC-CCITT polynomial only the16 least significant bits have meaning. The 16 most significant bits are tied to 0 during the computation. The CSTAT register can be written at byte, half-word or word. This register can be read and written by software. This register can be written only during the configuration phase.

12.4.4 CRC Output Register (CRC_OUTP)

Offset: CRC_BASE + 0x000C
CRC_BASE + 0x001C
CRC_BASE + 0x002C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OUTP															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OUTP															
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 12-5. CRC Output Register (CRC_OUTP)

Field	Description
0-31	OUTP: *Final CRC signature* The OUTP register includes the final signature corresponding to the CRC_CSTAT register value after swap/inversion operations, if specified. In case of CRC-CCITT polynomial only the16least significant bits have meaning. The 16 most significant bits are tied to 0 during the computation. This register can be read by software.

12.5 Use cases and limitations

Two main use cases are considered:

- Calculation of the CRC of the configuration registers during the process safety time
- Calculation of the CRC on the incoming and outgoing frames for the communication protocols (not protected with CRC by definition of the protocol itself) used as a safety-relevant peripheral.

12.5.1 Checksums for Configuration Registers

The checksum (signature) of configuration registers is computed in a correct way only if these registers do not contain any status bits, i.e., configuration register contents must not dynamically change during, or as a result of, a CRC checksum calculation.

12.5.2 Calculations on Incoming/Outgoing Protocol Frames

The following sections show the sequence for managing CRC checksums as part of a communication external to the device.

12.5.2.1 Calculating Checksums on Data to be Transmitted

Figure 12-6 illustrates the sequence used to calculate a CRC checksum on a data stream, append the checksum and transfer it to the peripheral to be used for transmission. The sequence is as follows:

1. Software configures the DMA channel and CRC context to be used.
2. DMA copies the data to be transmitted to the CRC context's input register (CRC_INP) to calculate the CRC signature (Phase 1)
3. Software copies the CRC checksum (signature) from the CRC module (CRC_OUTP register) to the memory location immediately following the transmission data. (Phase 2)
4. DMA transfers the data block (payload + CRC checksum from memory to the peripheral module (e.g., SPI Tx FIFO) (Phase 3)

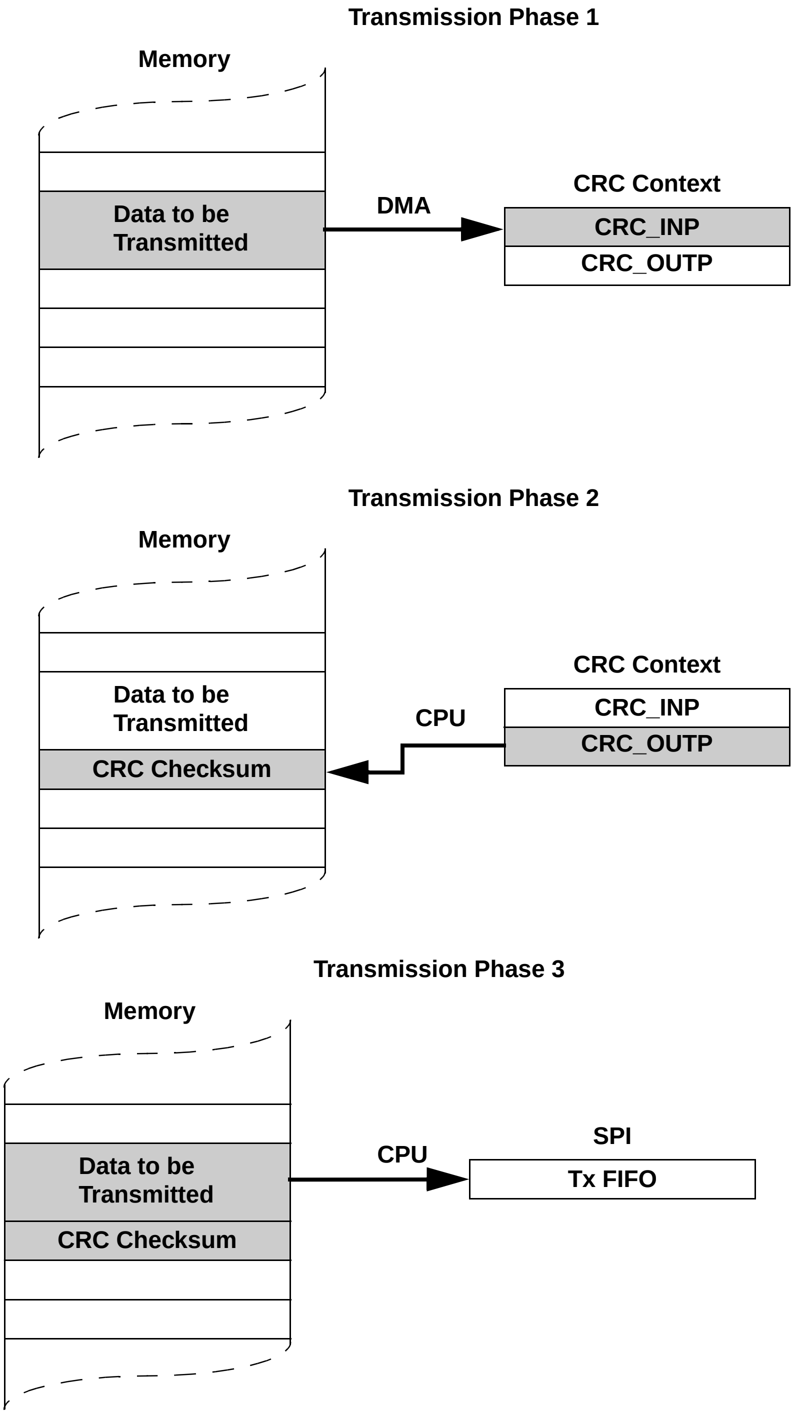


Figure 12-6. Transmission Sequence

12.5.2.2 Calculating Checksums on Received Data

Figure 12-7 illustrates the sequence used to calculate a CRC checksum on a received data stream. The sequence is as follows:

1. Software configures the DMA channel and /CRC context to be used.
2. DMA copies the received data block (payload + CRC) from the peripheral (e.g., SPI Rx FIFO) module to memory (Phase 1)
3. DMA copies the received data block transfer (payload + CRC) from memory to the CRC context (CRC_INP register) to calculate the CRC signature (phase 2)
4. The CRC signature is read from the CRC context (CRC_OUTP register) by software (phase 3)

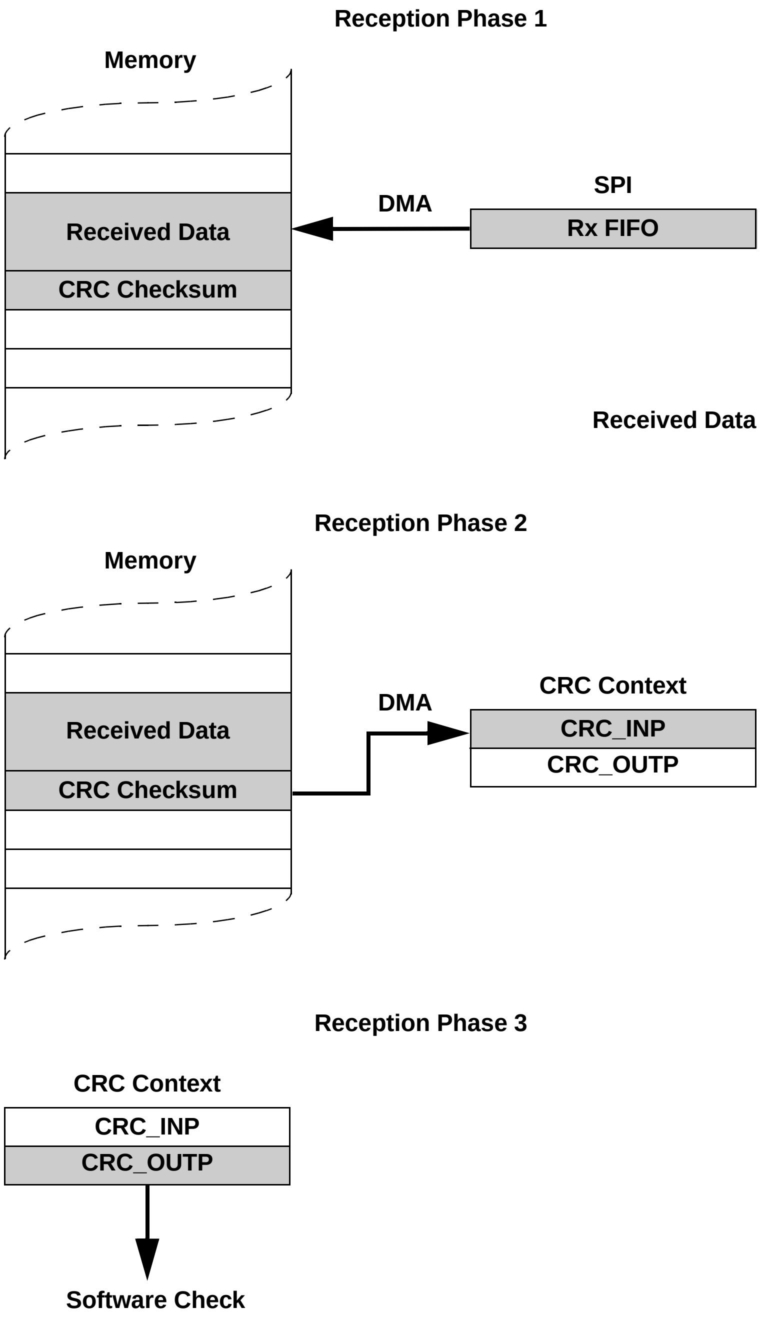


Figure 12-7. Reception Sequence

Chapter 13
Debug

This chapter covers the following:

- Section 13.1, “IEEE 1149.1 Test Access Port Controller (JTAGC)
- Section 13.2, “Nexus Development Interface (NDI)

13.1 IEEE 1149.1 Test Access Port Controller (JTAGC)

13.1.1 Introduction

The JTAG port of the device consists of four inputs and one output. These pins include JTAG compliance select (JCOMP), test data input (TDI), test data output (TDO), test mode select (TMS), and test clock input (TCK). TDI, TDO, TMS, and TCK are compliant with the IEEE 1149.1-2001 standard and are shared with the NDI through the test access port (TAP) interface.

13.1.1.1 Block Diagram

Figure 13-1 is a block diagram of the JTAG Controller (JTAGC).

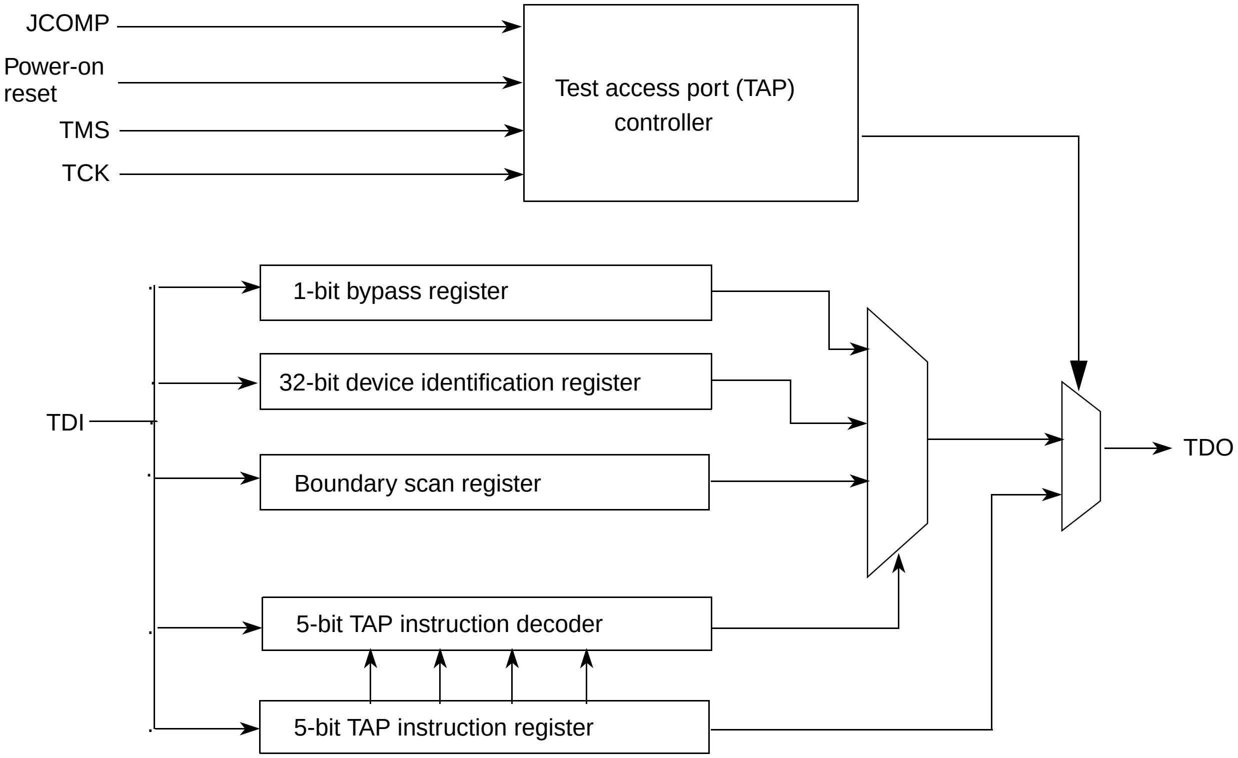


Figure 13-1. JTAG Controller Block Diagram

13.1.1.2 Overview

The JTAGC provides the means to test chip functionality and connectivity while remaining transparent to system logic when not in test mode. Testing is performed via a boundary scan technique, as defined in the IEEE 1149.1-2001 standard. In addition, instructions can be executed that allow the Test Access Port (TAP) to be shared with other modules on the MCU. All data input to and output from the JTAGC is communicated in serial format.

13.1.1.3 Features

The JTAGC is compliant with the IEEE 1149.1-2001 standard, and supports the following features:

- IEEE 1149.1-2001 Test Access Port (TAP) interface.
- 4 pins (TDI, TMS, TCK, and TDO), Refer to Section 13.1.2, “External Signal Description.”
- A JCOMP input that provides the ability to share the TAP.
- A 5-bit instruction register that supports several IEEE 1149.1-2001 defined instructions, as well as several public and private MCU specific instructions.
- Four test data registers: a bypass register, a boundary scan register, and a device identification register. The size of the boundary scan register is 480 bits.

- A TAP controller state machine that controls the operation of the data registers, instruction register and associated circuitry.

13.1.1.4 Modes of Operation

The JTAGC uses JCOMP and a power-on reset indication as its primary reset signals. Several IEEE 1149.1-2001 defined test modes are supported, as well as a bypass mode.

13.1.1.4.1 Reset

The JTAGC is placed in reset when the TAP controller state machine is in the TEST-LOGIC-RESET state. The TEST-LOGIC-RESET state is entered upon the assertion of the power-on reset signal, negation of JCOMP, or through TAP controller state machine transitions controlled by TMS. Asserting power-on reset or negating JCOMP results in asynchronous entry into the reset state. While in reset, the following actions occur:

- The TAP controller is forced into the test-logic-reset state, thereby disabling the test logic and allowing normal operation of the on-chip system logic to continue unhindered.
- The instruction register is loaded with the IDCODE instruction.

In addition, execution of certain instructions can result in assertion of the internal system reset. These instructions include EXTEST, CLAMP, and HIGHZ.

13.1.1.4.2 IEEE 1149.1-2001 Defined Test Modes

The JTAGC supports several IEEE 1149.1-2001 defined test modes. The test mode is selected by loading the appropriate instruction into the instruction register while the JTAGC is enabled. Supported test instructions include EXTEST, HIGHZ, CLAMP, SAMPLE and SAMPLE/PRELOAD. Each instruction defines the set of data registers that can operate and interact with the on-chip system logic while the instruction is current. Only one test data register path is enabled to shift data between TDI and TDO for each instruction.

The boundary scan register is enabled for serial access between TDI and TDO when the EXTEST, SAMPLE or SAMPLE/PRELOAD instructions are active. The single-bit bypass register shift stage is enabled for serial access between TDI and TDO when the HIGHZ, CLAMP or reserved instructions are active. The functionality of each test mode is explained in more detail in Section 13.1.4.4, “JTAGC Instructions.”

13.1.1.4.3 Bypass Mode

When no test operation is required, the BYPASS instruction can be loaded to place the JTAGC into bypass mode. While in bypass mode, the single-bit bypass shift register is used to provide a minimum-length serial path to shift data between TDI and TDO.

13.1.1.4.4 TAP Sharing Mode

The selectable auxiliary TAP controllers that share the TAP with the JTAGC are:

- Nexus port controller (NPC)
- e200 OnCE
- eTPU Nexus
- eDMA A Nexus
- eDMA B Nexus
- FlexRay

The instructions required to grant ownership of the TAP to the auxiliary TAP controllers are:

- ACCESS_AUX_TAP_NPC
- ACCESS_AUX_TAP_ONCE
- ACCESS_AUX_TAP_ONCE_B
- ACCESS_AUX_TAP_eTPU
- ACCESS_AUX_TAP_eTPU_SECONDARY
- ACCESS_AUX_TAP_NXDM
- ACCESS_AUX_TAP_NXDM_B
- ACCESS_AUX_TAP_NXFR
- ACCESS_AUX_TAP_MULTI

Instruction opcodes for each instruction are shown in Table 13-3.

When the access instruction for an auxiliary TAP is loaded, control of the JTAG pins is transferred to the selected TAP controller. Any data input via TDI and TMS is passed to the selected TAP controller, and any TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive.

For more information on the TAP controllers refer to Section 13.2, “Nexus Development Interface (NDI).”

13.1.2 External Signal Description

The JTAGC consists of five signals that connect to off-chip development tools and allow access to test support functions. The JTAGC signals are outlined in the following table:

Table 13-1. JTAG Signal Properties

Name	I/O	Function	Reset State	Pull[1]
TCK	I	Test clock	—	Down
TDI	I	Test data in	—	Up
TDO	O	Test data out	High Z [2]	Down [2]
TMS	I	Test mode select	—	Up
JCOMP	I	JTAG compliancy	—	Down

[1] The pull is not implemented in this module. Pullup/down devices are implemented in the pads.

[2] TDO output buffer enable is negated when JTAGC is not in the Shift-IR or Shift-DR states. A weak pulldown can be implemented on TDO.

13.1.3 Memory Map/Register Definition

This section provides a detailed description of the JTAGC registers accessible through the TAP interface, including data registers and the instruction register. Individual bit-level descriptions and reset states of each register are included. These registers are not memory-mapped and can only be accessed through the TAP.

13.1.3.1 Instruction Register

The JTAGC uses a 5-bit instruction register as shown in Figure 13-2. The instruction register allows instructions to be loaded into the module to select the test to be performed or the test data register to be accessed or both. Instructions are shifted in through TDI while the TAP controller is in the Shift-IR state, and latched on the falling edge of TCK in the Update-IR state. The latched instruction value can only be changed in the update-IR and test-logic-reset TAP controller states. Synchronous entry into the test-logic-reset state results in the IDCODE instruction being loaded on the falling edge of TCK. Asynchronous entry into the test-logic-reset state results in asynchronous loading of the IDCODE instruction. During the capture-IR TAP controller state, the instruction shift register is loaded with the value 0b10101, making this value the register’s read value when the TAP controller is sequenced into the Shift-IR state.

	4	3	2	1	0
R	1	0	1	0	1
W	Instruction Code				
Reset	0	0	0	0	1

Figure 13-2. 5-Bit Instruction Register

13.1.3.2 Bypass Register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS, CLAMP, HIGHZ or reserve instructions are active. After entry into the capture-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

13.1.3.3 Device Identification Register

The device identification register, shown in Figure 13-3, allows the part revision number, design center, part identification number, and manufacturer identity code to be determined through the TAP. The device identification register is selected for serial data transfer between TDI and TDO when the IDCODE instruction is active. Entry into the capture-DR state while the device identification register is selected loads the IDCODE into the shift register to be shifted out on TDO in the Shift-DR state. No action occurs in the update-DR state.

IR[4:0]: 0_0001 (IDCODE) Access: R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PRN				DC						PIN										MIC											ID
W																																
Reset	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	1	1	1	0	1

Figure 13-3. Device Identification Register

Table 13-2. Device Identification Register Field Descriptions

Field	Description
0–3 PRN	Part revision number. Contains the revision number of the device. This field changes with each revision of the device or module.
4–9 DC	Design center. Indicates the Freescale design center. For the MPC5676R this value is 0x20.
10–19 PIN	Part identification number. Contains the part number of the device. For the MPC5676R, this value is 0x276.
20–30 MIC	Manufacturer identity code. Contains the reduced Joint Electron Device Engineering Council (JEDEC) ID for Freescale, 0xE.
31 ID	IDCODE register ID. Identifies this register as the device identification register and not the bypass register. Always set to 1.

13.1.3.4 CENSOR_CTRL Register

The CENSOR_CTRL register is a 64-bit shift register path from TDI to TDO selected when the ENABLE_CENSOR_CTRL instruction is active. The default reset value of the CENSOR_CTRL register is 64’b0. The CENSOR_CTRL register transfers its value to a parallel hold register on the rising edge of TCK when the TAP controller state machine is in the Update-DR state. Once the ENABLE_CENSOR_CTRL instruction is executed, the register value will remain valid until a JTAG reset occurs.

	0	1	2	...	63
R W	CENSOR_CTRL				
Reset:	*[1]	*	*	*	*

[1] The reset value of CENSOR_CTRL is 64’b0.

Figure 1. CENSOR_CTRL Register

CENSOR_CTRL - Censorship Control

The CENSOR_CTRL bits are used to control chiptop censorship functions.

13.1.3.5 Boundary Scan Register

The boundary scan register is connected between TDI and TDO when the EXTEST, SAMPLE or SAMPLE/PRELOAD instructions are active. It is used to capture input pin data, force fixed values on output pins, and select a logic value and direction for bidirectional pins. Each bit of the boundary scan register represents a separate boundary scan register cell, as described in the IEEE 1149.1-2001 standard and discussed in Section 13.1.4.5, “Boundary Scan.”

13.1.4 Functional Description

13.1.4.1 JTAGC Reset Configuration

While in reset, the TAP controller is forced into the test-logic-reset state, thus disabling the test logic and allowing normal operation of the on-chip system logic. In addition, the instruction register is loaded with the IDCODE instruction.

13.1.4.2 IEEE 1149.1-2001 (JTAG) Test Access Port

The JTAGC uses the IEEE 1149.1-2001 TAP for accessing registers. This port can be shared with other TAP controllers on the MCU. Ownership of the port is determined by the value of the JCOMP signal and the currently loaded instruction. For more detail on TAP sharing via JTAGC instructions refer to Section 13.1.4.4.2, “ACCESS_AUX_TAP_x Instructions.”

Data is shifted between TDI and TDO though the selected register starting with the least significant bit, as illustrated in Figure 13-4. This applies for the instruction register, test data registers, and the bypass register.

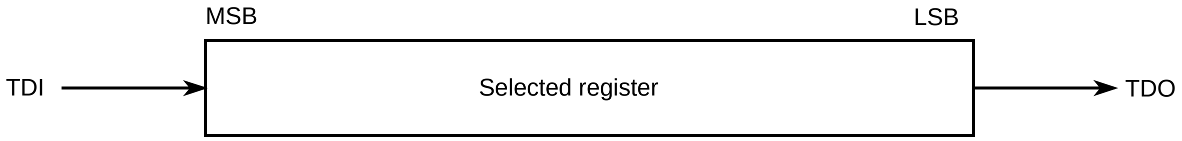
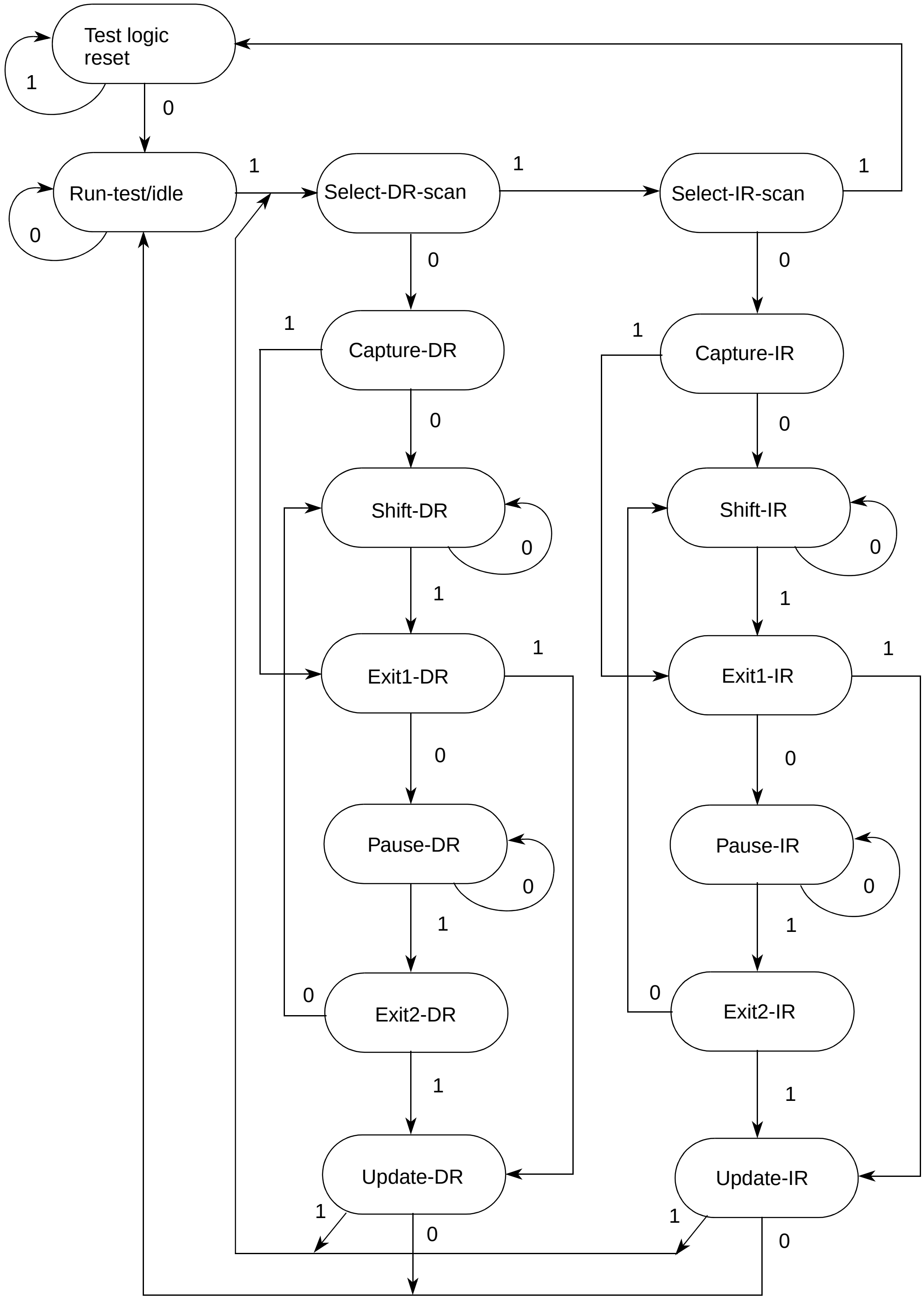


Figure 13-4. Shifting Data Through a Register

13.1.4.3 TAP Controller State Machine

The TAP controller is a synchronous state machine that interprets the sequence of logical values on the TMS pin. Figure 13-5 shows the machine’s states. The value shown next to each state is the value of the TMS signal sampled on the rising edge of the TCK signal.

As Figure 13-5 shows, holding TMS at logic 1 while clocking TCK through a sufficient number of rising edges also causes the state machine to enter the test-logic-reset state.



NOTE: The value shown adjacent to each state transition in this figure represents the value of TMS at the time of a rising edge of TCK.

Figure 13-5. IEEE 1149.1-2001 TAP Controller Finite State Machine

13.1.4.3.1 Enabling the TAP Controller

The JTAGC TAP controller is enabled by setting JCOMP to a logic 1 value.

13.1.4.3.2 Selecting an IEEE 1149.1-2001 Register

Access to the JTAGC data registers is done by loading the instruction register with any of the JTAGC instructions while the JTAGC is enabled. Instructions are shifted in via the select-IR-scan path and loaded in the update-IR state. At this point, all data register access is performed via the select-DR-scan path.

The select-DR-scan path is used to read or write the register data by shifting in the data (LSB first) during the shift-DR state. When reading a register, the register value is loaded into the IEEE 1149.1-2001 shifter during the capture-DR state. When writing a register, the value is loaded from the IEEE 1149.1-2001 shifter to the register during the update-DR state. When reading a register, there is no requirement to shift out the entire register contents. Shifting can be terminated after fetching the required number of bits.

13.1.4.4 JTAGC Instructions

This section gives an overview of each instruction, refer to the IEEE 1149.1-2001 standard for more details.

The JTAGC implements the IEEE 1149.1-2001 defined instructions listed in Table 13-3.

Table 13-3. JTAG Instructions

Instruction	Code[4:0]	Instruction Summary
IDCODE	00001	Selects device identification register for shift
SAMPLE/PRELOAD	00010	Selects boundary scan register for shifting, sampling, and preloading without disturbing functional operation
SAMPLE	00011	Selects boundary scan register for shifting and sampling without disturbing functional operation
EXTEST	00100	Selects boundary scan register while applying preloaded values to output pins and asserting functional reset
ENABLE_CENSOR_CTRL	00111	Selects CENSOR_CTRL register
HIGHZ	01001	Selects bypass register while three-stating all output pins and asserting functional reset
CLAMP	01100	Selects bypass register while applying preloaded values to output pins and asserting functional reset
ACCESS_AUX_TAP_NPC	10000	Enables access to the NPC TAP controller
ACCESS_AUX_TAP_OnCE	10001	Enables access to the primary e200 OnCE TAP controller (Primary CPU, core 0)
ACCESS_AUX_TAP_eTPU	10010	Enables access to the eTPU Nexus TAP controller (eTPU_A, eTPU_B, CDC_AB)
ACCESS_AUX_TAP_NXDM	10011	Enables access to the eDMA_A Nexus TAP controller (for Data Trace)
ACCESS_AUX_TAP_NXFR	10100	Enables access to the FlexRay Nexus TAP controller (for Data Trace)

Table 13-3. JTAG Instructions (continued)

Instruction	Code[4:0]	Instruction Summary
ACCESS_AUX_TAP_eTPU_SECONDARY	10110	Enables access to a secondary set of eTPU modules (eTPU_C, eTPU_D, CDC_CD)
ACCESS_AUX_TAP_NXDM_B	10111	Enables access to the eDMA_B Nexus TAP controller (for Data Trace)
ACCESS_AUX_TAP_OnCE_1	11001	Enables access to the secondary OnCE TAP controller (core 1)
ACCESS_AUX_TAP_MULTI	11100	Serialize the JTAG Instruction all internal cores
BYPASS	11111	Selects bypass register for data operations
Factory Debug Reserved[1]	00101 00110 01010	Intended for factory debug only
Reserved[2]	All Other Codes	Decoded to select bypass register

[1] Intended for factory debug, and not customer use

[2] Freescale reserves the right to change the decoding of reserved instruction codes

13.1.4.4.1 BYPASS Instruction

BYPASS selects the bypass register, creating a single-bit shift register path between TDI and TDO. BYPASS enhances test efficiency by reducing the overall shift path when no test operation of the MCU is required. This allows more rapid movement of test data to and from other components on a board that are required to perform test functions. While the BYPASS instruction is active the system logic operates normally.

13.1.4.4.2 ACCESS_AUX_TAP_*x* Instructions

The ACCESS_AUX_TAP_*x* instructions allow the Nexus modules on the MCU to take control of the TAP. When this instruction is loaded, control of the TAP pins is transferred to the selected auxiliary TAP controller. Any data input via TDI and TMS is passed to the selected TAP controller, and any TDO output from the selected TAP controller is sent back to the JTAGC to be output on the pins. The JTAGC regains control of the JTAG port during the UPDATE-DR state if the PAUSE-DR state was entered. Auxiliary TAP controllers are held in RUN-TEST/IDLE while they are inactive.

13.1.4.4.3 CLAMP Instruction

CLAMP allows the state of signals driven from MCU pins to be determined from the boundary scan register while the bypass register is selected as the serial path between TDI and TDO. CLAMP enhances test efficiency by reducing the overall shift path to a single bit (the bypass register) while conducting an EXTEST type of instruction through the boundary scan register. CLAMP also asserts the internal system reset for the MCU to force a predictable internal state.

13.1.4.4.4 EXTEST — External Test Instruction

EXTEST selects the boundary scan register as the shift path between TDI and TDO. It allows testing of off-chip circuitry and board-level interconnections by driving preloaded data contained in the boundary scan register onto the system output pins. Typically, the preloaded data is loaded into the boundary scan register using the SAMPLE/PRELOAD instruction before the selection of EXTEST. EXTEST asserts the internal system reset for the MCU to force a predictable internal state while performing external boundary scan operations.

13.1.4.4.5 ENABLE_CENSOR_CTRL Instruction

The ENABLE_CENSOR_CTRL instruction selects the CENSOR_CTRL register for connection as the shift path between TDI and TDO.

13.1.4.4.6 HIGHZ Instruction

HIGHZ selects the bypass register as the shift path between TDI and TDO. While HIGHZ is active, all output drivers are placed in an inactive drive state (for example, high impedance). HIGHZ also asserts the internal system reset for the MCU to force a predictable internal state.

13.1.4.4.7 IDCODE Instruction

IDCODE selects the 32-bit device identification register as the shift path between TDI and TDO. This instruction allows interrogation of the MCU to determine its version number and other part identification data. IDCODE is the instruction placed into the instruction register when the JTAGC is reset.

13.1.4.4.8 SAMPLE Instruction

The SAMPLE instruction obtains a sample of the system data and control signals present at the MCU input pins and just before the boundary scan register cells at the output pins. This sampling occurs on the rising edge of TCK in the capture-DR state when the SAMPLE instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output during the Shift-DR state. There is no defined action in the update-DR state. Both the data capture and the shift operation are transparent to system operation.

13.1.4.4.9 SAMPLE/PRELOAD Instruction

The SAMPLE/PRELOAD instruction has two functions:

- The SAMPLE part of the instruction samples the system data and control signals on the MCU input pins and just before the boundary scan register cells at the output pins. This sampling occurs on the rising-edge of TCK in the capture-DR state when the SAMPLE/PRELOAD instruction is active. The sampled data is viewed by shifting it through the boundary scan register to the TDO output during the shift-DR state. Both the data capture and the shift operation are transparent to system operation.
- The PRELOAD part of the instruction initializes the boundary scan register cells before selecting the EXTEST or CLAMP instructions to perform boundary scan tests. This is achieved by shifting in initialization data to the boundary scan register during the shift-DR state. The initialization data is transferred to the parallel outputs of the boundary scan register cells on the falling edge of TCK

in the update-DR state. The data is applied to the external output pins by the EXTEST or CLAMP instruction. System operation is not affected.

13.1.4.5 Boundary Scan

The boundary scan technique allows signals at component boundaries to be controlled and observed through the shift-register stage associated with each pad. Each stage is part of a larger boundary scan register cell, and cells for each pad are interconnected serially to form a shift-register chain around the border of the design. The boundary scan register consists of this shift-register chain, and is connected between TDI and TDO when the EXTEST, SAMPLE, or SAMPLE/PRELOAD instructions are loaded. The shift-register chain contains a serial input and serial output, as well as clock and control signals.

13.1.5 Initialization/Application Information

The test logic is a static logic design, and TCK can be stopped in either a high or low state without loss of data. However, the system clock is not synchronized to TCK internally. Any mixed operation using both the test logic and the system functional logic requires external synchronization.

To initialize the JTAGC module and enable access to registers, the following sequence is required:

1. Set the JCOMP signal to logic 1, thereby enabling the JTAGC TAP controller.
2. Load the appropriate instruction for the test or action to be performed.

13.2 Nexus Development Interface (NDI)

13.2.1 Introduction

The device microcontroller contains multiple Nexus clients that communicate over a single IEEE®-ISTO 5001™-2003 Nexus class 3 combined JTAG IEEE 1149.1/auxiliary out interface. Combined, all of the Nexus clients are referred to as the Nexus development interface (NDI). Class 3 Nexus allows for program, data, and ownership trace of the microcontroller execution without access to the external data and address buses.

This chapter is organized into sections that provide a high level view of the Nexus development interface: Section 13.2.1, “Introduction” through Section 13.2.8, “NPC Initialization and Application Information.”

The chapter contains sections that discuss the modules of the Nexus development interface:

- Nexus eTPU development interfaces (NDEDI and NSEDI). The device has three eTPU engines. See Section 13.2.9, “Nexus TPU Development Interfaces (NDEDI and NSEDI)” and the *eTPU Reference Manual* for more information.
- Nexus e200z7 core interface (NZ7C3). In this chapter, the NZ7C3 interface is discussed in Section 13.2.10, “e200z7 Class 3 Nexus Module (NZ7C3) through Section 13.2.11, “NZ7C3 Memory Map and Register Definition.”
- Nexus crossbar eDMA interface (NXDM) and Nexus FlexRay interface (NXFR). Refer to Section 13.2.15, “Nexus eDMA Interface (NXDM) and Nexus FlexRay Interface (NXFR).”

Communication to the NDI is managed via the auxiliary port and the JTAG port.

- The auxiliary port is comprised of 17 or 21 output pins and 1 input pin. The output pins include one message clock out (MCKO) pin, 12 or 16 message data out (MDO) pins, two message start/end out ($\overline{\text{MSEO}}$) pins, one ready ($\overline{\text{RDY}}$) pin, and one event out ($\overline{\text{EVTO}}$) pin. Event in ($\overline{\text{EVTI}}$) is the only input pin for the auxiliary port.
- The JTAG port consists of four inputs and one output. These pins include JTAG compliance select (JCOMP), test data input (TDI), test data output (TDO), test mode select (TMS), and test clock input (TCK). TDI, TDO, TMS, and TCK are compliant with the IEEE 1149.1-2001 standard and are shared with the NDI through the test access port (TAP) interface. JCOMP along with power-on reset and the TAP state machine are used to control reset for the NDI module. Ownership of the TAP is achieved by loading the appropriate enable instruction for the desired Nexus client in the JTAG controller (JTAGC) when JCOMP is asserted. See Table 13-7 for the JTAGC opcodes to access the different Nexus clients.

13.2.1.1 Block Diagram

Figure 13-6 shows a general block diagram of the NDI components

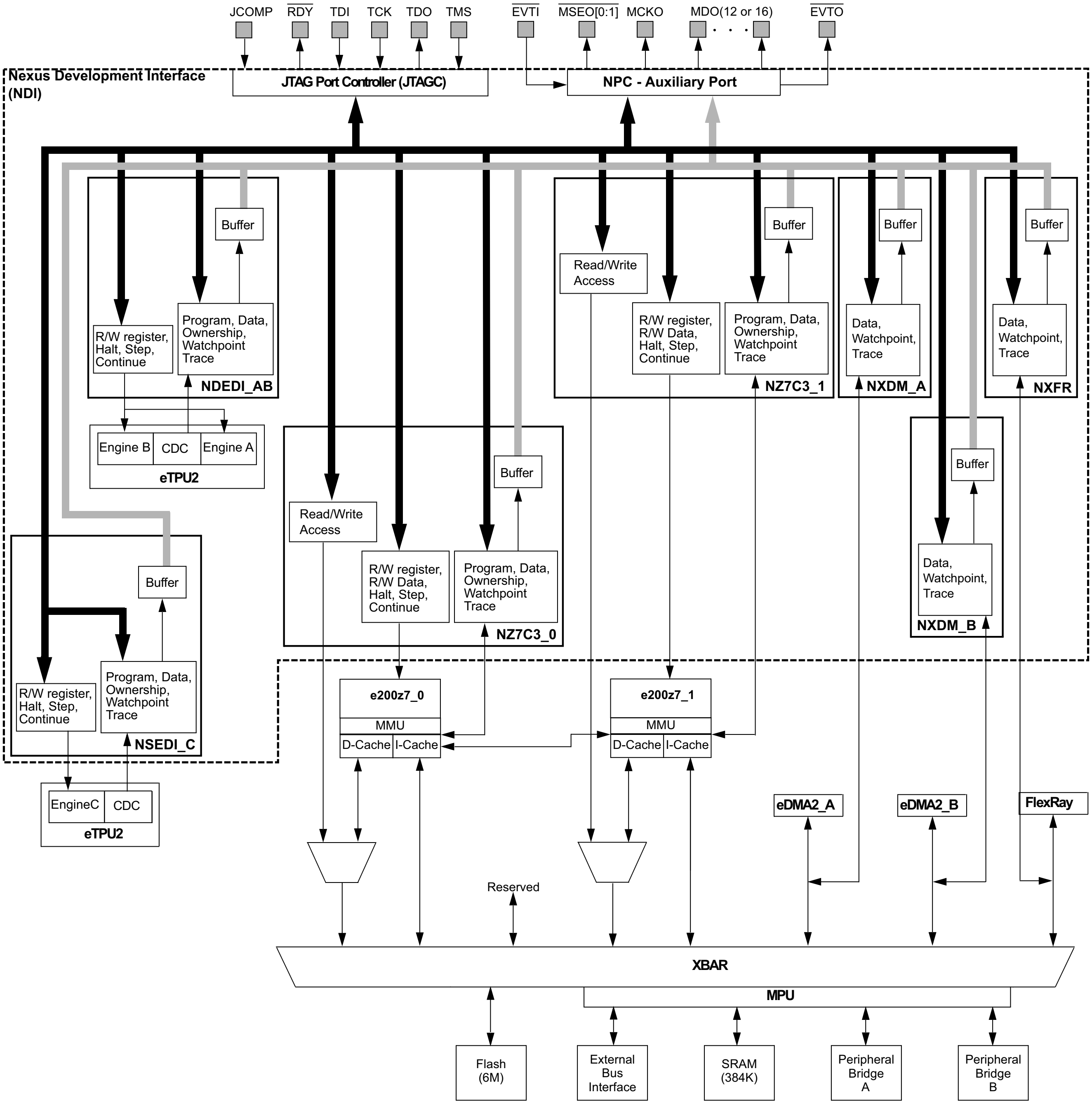


Figure 13-6. NDI General Block Diagram

13.2.1.2 Features

The NDI module is compliant with the IEEE-ISTO 5001-2003 standard. The following features are implemented:

- Full duplex pin interface for medium and high visibility throughput
  - One of two modes selected by register configuration: full port mode (FPM) and reduced port mode (RPM). FPM comprises 16 MDO pins, and RPM comprises 12 MDO pins.
  - Auxiliary output port
    - One MCKO (message clock out) pin
    - 16 MDO (message data out) pins
    - Two $\overline{\text{MSEO}}$ (message start/end out) pins
    - One $\overline{\text{RDY}}$ (ready) pin
    - One $\overline{\text{EVTO}}$ (event out) pin
  - Auxiliary input port uses one $\overline{\text{EVTI}}$ (event in) pin
  - Five-pin JTAG port (JCOMP, TDI, TDO, TMS, and TCK)
- Two host processor (e200z7) development support features (NZ7C3)
  - IEEE-ISTO 5001-2003 standard class 3 compliant.
  - Data trace via data write messaging (DWM) and data read messaging (DRM). This allows the development tool to trace reads and/or writes to selected internal memory resources.
  - Ownership trace via ownership trace messaging (OTM). OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated. An ownership trace message is transmitted when a new process/task is activated, allowing development tools to trace ownership flow.
  - Program trace via branch trace messaging (BTM). Branch trace messaging displays program flow discontinuities (direct branches, indirect branches, exceptions, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus, static code can be traced.
  - Watchpoint messaging (WPM) via the auxiliary port.
  - Watchpoint trigger enable of program and/or data trace messaging.
  - Data tracing of instruction fetches via private opcodes.
  - Subset of Power Architecture Book E software debug facilities with OnCE block (Nexus class 1 features).
- Two eDMA development support features (NXDM)
  - Data trace via data write messaging (DWM) and data read messaging (DRM). This allows the development tool to trace DMA generated reads and/or writes to selected address ranges in the device’s memory map.
  - Watchpoint messaging (WPM) via the auxiliary port.
  - Watchpoint trigger enable/disable of data trace messaging.
- FlexRay development support features (NXFR)

  - FlexRay Nexus trace via data write messaging (DWM) and data read messaging (DRM). This allows the development tool to trace FlexRay generated reads and/or writes to selected address ranges in the device's memory map.
  - Watchpoint messaging (WPM) via the auxiliary port.
  - Watchpoint trigger enable/disable of data trace messaging.
- eTPU development support features (NDEDI and NSEDI)
  - IEEE-ISTO 5001-2002 standard Class 3 compliant for the eTPU engines.
  - Data trace via data write messaging and data read messaging. This allows the development tool to trace reads and writes to selected shared parameter RAM (SPRAM) address ranges. Four data trace windows are shared by the two eTPU engines.
  - Ownership trace via ownership trace messaging (OTM). OTM facilitates ownership trace by providing visibility of which channel is being serviced. An ownership trace message is transmitted to indicate when a new channel service request is scheduled, allowing the development tools to trace task flow. A special OTM is sent when the engine enters in idle state, meaning that all requests were serviced and no new requests are yet scheduled.
  - Program trace via branch trace messaging. BTM displays program flow discontinuities (start, jumps, return, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus static code can be traced. The branch trace messaging method uses the branch/predicate method to reduce the number of generated messages.
  - Watchpoint messaging via the auxiliary port. WPM provides visibility of the occurrence of the eTPU's' watchpoints and breakpoints.
  - Nexus based breakpoint/watchpoint configuration and single step support.
- Run-time access to the on-chip memory map via the Nexus read/write access protocol. This feature supports accesses for run-time internal visibility, calibration variable acquisition, calibration constant tuning, and external rapid prototyping for powertrain automotive development systems.
- All features are independently configurable and controllable via the IEEE 1149.1 I/O port.
- The NDI block reset is controlled with JCOMP, power-on reset, and the TAP state machine. These sources are independent of system reset.
- NDI port ready status indication via MDO[0] following power-on reset.

13.2.1.3 Modes of Operation

The NDI block is in reset when the TAP controller state machine is in the TEST-LOGIC-RESET state. The TEST-LOGIC-RESET state is entered on the assertion of the power-on reset signal, negation of JCOMP, or through state machine transitions controlled by TMS. Assertion of JCOMP allows the NDI to move out of the reset state, and is a prerequisite to grant Nexus clients control of the TAP. Ownership of the TAP is achieved by loading the appropriate enable instruction for the desired Nexus client in the JTAGC controller (JTAGC) block when JCOMP is asserted.

Following negation of power-on reset, the NPC remains in reset until the system clock achieves lock. In PLL bypass mode, the NDI can transition out of the reset state immediately following negation of power-on reset. See Section 13.2.4.5, “Nexus Port Ready Status” for more details.

13.2.1.3.1 Nexus Reset Mode

In Nexus reset mode, the following actions occur:

- Register values default back to their reset values.
- The message queues are marked as empty.
- The auxiliary output port pins are negated if the NDI controls the pads.
- The TDO output buffer is disabled if the NDI has control of the TAP.
- The TDI, TMS, and TCK inputs are ignored.
- The NDI block indicates to the MCU that it is not using the auxiliary output port. This indication can be used to three-state the output pins or use them for another function.

13.2.1.3.2 Full-Port Mode

In full-port mode, all the available MDO pins are used to transmit messages. All trace features are enabled or can be enabled by writing the configuration registers via the JTAG port. The number of MDO pins available is 16.

13.2.1.3.3 Reduced-Port Mode

In reduced-port mode, a subset of the available MDO pins are used to transmit messages. All trace features are enabled or can be enabled by writing the configuration registers via the JTAG port. The number of MDO pins available is 12. Unused MDO pins can be used as GPIO. Details on GPIO functionality configuration can be found in Section 3.2.1.13, “Pad Configuration Registers (SIU_PCR).”

13.2.1.3.4 Disabled-Port Mode

In disabled-port mode, message transmission is disabled. Any debug feature that generates messages can not be used. The primary features available are class 1 features and read/write access.

13.2.1.3.5 Censored Mode

When the device is in censored mode, reading the contents of internal flash externally is not allowed. To prevent Nexus modules from violating censorship, the NPC is held in reset when in censored mode, asynchronously holding all other Nexus modules in reset as well. This prevents Nexus read/write to memory mapped resources and the transmission of Nexus trace messages. See Chapter 24, “Flash Memory Array and Control,” for information on Nexus port enabling and disabling regarding censorship.

13.2.2 External Signal Description

The auxiliary and JTAG pin interfaces provide for the transmission of messages from Nexus modules to the external development tools and for access to Nexus client registers. The auxiliary/JTAG pin definitions are outlined in Table 13-4.

Table 13-4. Signal Properties

Signal Name	Port	Function	Reset State
$\overline{\text{EVTO}}$	Auxiliary	Event out pin	Negated
$\overline{\text{EVTI}}$	Auxiliary	Event in pin	Pullup
MCKO	Auxiliary	Message clock out pin (from NPC)	Enabled
MDO[11:0] or MDO[15:0]	Auxiliary	Message data out pins	Driven Low[1]
$\overline{\text{MSEO}}$[1:0]	Auxiliary	Message start/end out pins	Negated
$\overline{\text{RDY}}$	Auxiliary	Ready out pin	Negated
JCOMP	JTAG	JTAG compliancy and TAP sharing control	Pulldown
TCK	JTAG	Test clock input	Pulldown
TDI	JTAG	Test data input	Pullup
TDO	JTAG	Test data output	High Z / Pullup
TMS	JTAG	Test mode select input	Pullup

[1] Following a power-on reset, MDO[0] remains asserted until power-on reset is exited and the system clock achieves lock.

13.2.2.1 Detailed Signal Descriptions

This section describes each of the signals listed in Table 13-4 in more detail.

13.2.2.1.1 Event Out ($\overline{\text{EVTO}}$)

$\overline{\text{EVTO}}$ is an output pin that is asserted upon breakpoint occurrence to provide breakpoint status indication or to signify that an event has occurred. The $\overline{\text{EVTO}}$ output of the NPC is generated based on the values of the individual $\overline{\text{EVTO}}$ signals from all Nexus modules that implement the signal.

13.2.2.1.2 Event In ($\overline{\text{EVTI}}$)

$\overline{\text{EVTI}}$ is used to initiate program and data trace synchronization messages or to generate a breakpoint. $\overline{\text{EVTI}}$ is edge-sensitive for synchronization and breakpoint generation.

13.2.2.1.3 Message Data Out (MDO[11:0] or [15:0])

Message data out (MDO) are output pins used for uploading OTM, BTM, DTM, and other messages to the development tool. The development tool must sample MDO on the rising edge of MCKO. The width of the MDO bus used is determined by the Nexus PCR[FPM] configuration.

Following a power-on reset, MDO[0] remains asserted until power-on reset is exited and the system clock achieves lock.

NOTE

MDO output pins should not be connected directly to ground or a power supply.

13.2.2.1.4 Message Start/End Out ($\overline{\text{MSEO}}$[1:0])

$\overline{\text{MSEO}}$[1:0] are output pins that indicates when a message on the MDO pins has started, when a variable length packet has ended, or when the message has ended. The development tool must sample the $\overline{\text{MSEO}}$ pins on the rising edge of MCKO.

NOTE

MSEO output pins should not be connected directly to ground or a power supply.

13.2.2.1.5 Ready ($\overline{\text{RDY}}$)

$\overline{\text{RDY}}$ is an output pin that indicates when a device is ready for the next access.

13.2.2.1.6 JTAG Compliancy (JCOMP)

The JCOMP signal enables or disables the TAP controller. The TAP controller is enabled when JCOMP asserts, otherwise the TAP controller remains in reset.

13.2.2.1.7 Test Data Output (TDO)

The TDO pin transmits serial output for instructions and data. TDO is tri-stateable and is actively driven in the SHIFT-IR and SHIFT-DR controller states. TDO is updated on the falling edge of TCK and sampled by the development tool on the rising edge of TCK.

13.2.2.1.8 Test Clock Input (TCK)

The TCK pin is used to synchronize the test logic and control register access through the JTAG port.

13.2.2.1.9 Test Data Input (TDI)

The TDI pin receives serial test instruction and data. TDI is sampled on the rising edge of TCK.

13.2.2.1.10 Test Mode Select (TMS)

The TMS pin is used to sequence the IEEE 1149.1-2001 TAP controller state machine. TMS is sampled on the rising edge of TCK.

13.2.3 Memory Map

The NDI block contains no memory mapped registers. Nexus registers are accessed by the development tool via the JTAG port using a register index and a client select value. The client select is controlled by loading the correct access instruction into the JTAG controller; see Table 13-7. OnCE registers are accessed by loading the appropriate value in the RS[0:6] field of the OnCE command register (OCMD) via the JTAG port.

Table 13-5 shows the NDI registers and their Index values.

Table 13-5. Nexus Development Interface (NDI) Registers

Index	Register
NPC Registers	
0	Device ID Register (DID)
127	Port Configuration Register (PCR)
e200z7 Control and Status Registers[1]	
2	e200z7 Development Control1 (NZ7C3_DC1)
3	e200z7 Development Control2 (NZ7C3_DC2)
4	e200z7 Development Control3 (NZ7C3_DC3)
5	e200z7 Development Control4 (NZ7C3_DC4)
7	Read/Write Access Control/Status (NZ7C3_RWCS)
9	Read/Write Access Address (NZ7C3_RWA)
10	Read/Write Access Data (NZ7C3_RWD)
11	e200z7 Watchpoint Trigger (NZ7C3_WT)
13	e200z7 Data Trace Control (NZ7C3_DTC)
14	e200z7 Data Trace Start Address 1 (NZ7C3_DTSA1)
15	e200z7 Data Trace Start Address 2 (NZ7C3_DTSA2)
16	e200z7 Data Trace Start Address 3 (NZ7C3_DTSA3)
17	e200z7 Data Trace Start Address 4 (NZ7C3_DTSA4)
18	e200z7 Data Trace End Address 1 (NZ7C3_DTEA1)
19	e200z7 Data Trace End Address 2 (NZ7C3_DTEA2)
20	e200z7 Data Trace End Address 3 (NZ7C3_DTEA3)
21	e200z7 Data Trace End Address 4 (NZ7C3_DTEA4)
48	Development Status (DS)
50	Overrun Control (OVCR)
51	Watchpoint Mask (WMSK)
53	Program Trace Start Trigger Control (PTSTC)

Table 13-5. Nexus Development Interface (NDI) Registers (continued)

Index	Register
54	Program Trace End Trigger Control (PTETC)
55	Data Trace Start Trigger Control (DTSTC)
56	Data Trace End Trigger Control (DTETC)
eDMA_A/B Control and Status Registers	
2	eDMA_*x* Development Control (NXDM_DC)
11	eDMA_*x* Watchpoint Trigger (NXDM_WT)
13	eDMA_*x* Data Trace Control (NXDM_DTC)
14	eDMA_*x* Data Trace Start Address 0 (NXDM_DTSA1)
15	eDMA_*x* Data Trace Start Address 1 (NXDM_DTSA2)
18	eDMA_*x* Data Trace End Address 0 (NXDM_DTEA1)
19	eDMA_*x* Data Trace End Address 1 (NXDM_DTEA2)
22	eDMA_*x* Breakpoint and Watchpoint Control 1 (NXDM_BWC1)
23	eDMA_*x* Breakpoint and Watchpoint Control 2 (NXDM_BWC2)
30	eDMA_*x* Breakpoint and Watchpoint Address 1 (NXDM_BWA1)
31	eDMA_*x* Breakpoint and Watchpoint Address 2 (NXDM_BWA2)
FlexRay Control and Status Registers	
2	FlexRay Development Control (NXFR_DC)
11	FlexRay Watchpoint Trigger (NXFR_WT)
13	FlexRay Data Trace Control (NXFR_DTC)
14	FlexRay Data Trace Start Address 0 (NXFR_DTSA1)
15	FlexRay Data Trace Start Address 1 (NXFR_DTSA2)
18	FlexRay Data Trace End Address 0 (NXFR_DTEA1)
19	FlexRay Data Trace End Address 1 (NXFR_DTEA2)
22	FlexRay Breakpoint and Watchpoint Control 1 (NXFR_BWC1)
23	FlexRay Breakpoint and Watchpoint Control 2 (NXFR_BWC2)
30	FlexRay Breakpoint and Watchpoint Address 1 (NXFR_BWA1)
31	FlexRay Breakpoint and Watchpoint Address 2 (NXFR_BWA2)
eTPU Engines Control/Status Registers	
0	Device ID (DID)
1	Client Select Control (CSC), eTPU Engine 1 / eTPU Engine 2
2	eTPU2 Development Control (NDEDI/NSEDI_eTPU2_DC)
4	eTPU2 Development Status (NDEDI/NSEDI_eTPU2_DS)
11	eTPU2 Watchpoint Trigger (NDEDI/NSEDI_eTPU2_WT)

Table 13-5. Nexus Development Interface (NDI) Registers (continued)

Index	Register
13	eTPU2 Data Trace Control (NDEDI/NSEDI_eTPU2_DTC)
22	eTPU2 Breakpoint/Watchpoint Control 1 (NDEDI/NSEDI_eTPU2_BWC1)
23	eTPU2 Breakpoint/Watchpoint Control 2 (NDEDI/NSEDI_eTPU2_BWC2)
24	eTPU2 Breakpoint/Watchpoint Control 3 (NDEDI/NSEDI_eTPU2_BWC3)
30	eTPU2 Breakpoint/Watchpoint Address 1 (NDEDI/NSEDI_eTPU2_BWA1)
31	eTPU2 Breakpoint/Watchpoint Address 2 (NDEDI/NSEDI_eTPU2_BWA2)
38	eTPU2 Breakpoint/Watchpoint Data 1 (NDEDI/NSEDI_eTPU2_BWD1)
39	eTPU2 Breakpoint/Watchpoint Data 1 (NDEDI/NSEDI_eTPU2_BWD2)
64	eTPU2 Program Trace Channel Enable (NDEDI/NSEDI_eTPU2_PTCE)
69	eTPU2 Microinstruction Debug Register (NDEDI/NSEDI_eTPU2_INST)
70	eTPU2 Microprogram Counter Debug Register (NDEDI/NSEDI_eTPU2_MPC)
71	eTPU2 Channel Flag Status Register (NDEDI/NSEDI_eTPU2_CFSR)

[1] These e200z7 registers are described in the *e200z7 Core Reference Manual.*

Table 13-6 shows the OnCE register addressing.

Table 13-6. e200z7 OnCE Register Addressing

OCMD, RS[0:6]	Register Selected
000 0000–000 0001	Invalid value
000 0010	JTAG DID (read-only)
000 0011–000 1111	Invalid value
001 0000	CPU Scan Register (CPUSCR)
001 0001	No Register Selected (Bypass)
001 0010	OnCE Control Register (OCR)
001 0011–001 1111	Invalid value
010 0000	Instruction Address Compare 1 (IAC1)
010 0001	Instruction Address Compare 2 (IAC2)
010 0010	Instruction Address Compare 3 (IAC3)
010 0011	Instruction Address Compare 4 (IAC4)
010 0100	Data Address Compare 1 (DAC1)
010 0101	Data Address Compare 2 (DAC2)
010 0110	Data Value Compare 1 (DVC1)
010 0111	Data Value Compare 2 (DVC2)
010 1000	Instruction Address Compare 5 (IAC5)

Table 13-6. e200z7 OnCE Register Addressing (continued)

OCMD, RS[0:6]	Register Selected
010 1001	Instruction Address Compare 6 (IAC6)
010 1010	Instruction Address Compare 7 (IAC7)
010 1011	Instruction Address Compare 8 (IAC8)
010 1100	Debug Counter Register (DBCNT)
010 1101	Debug PCFIFO (PCFIFO) (read-only)
010 1110	External Debug Control Register 0 (EDBCR0)
010 1111	External Debug Status Register 0 (EDBSR0)
011 0000	Debug Status Register (DBSR)
011 0001	Debug Control Register 0 (DBCR0)
011 0010	Debug control register 1 (DBCR1)
011 0011	Debug control register 2 (DBCR2)
011 0100	Debug control register 3 (DBCR3)
011 0101	Debug control register 4 (DBCR4)
011 0110	Debug control register 5 (DBCR5)
011 0111	Debug control register 6 (DBCR6)
011 1000–011 1011	Invalid value (do not access)
011 1100	External Debug Status Register MASK 0 (EDBSRMSK0)
011 1101	Debug Data Acquisition Message Register (DDAM)
011 1110	Debug Event Control (DEVENT)
011 1111	Debug External Resource Control (DBERC0)
100 0000–110 1111	Invalid value (do not access)
111 0000–111 1001	General purpose register selects [0:9]
111 1010	Cache Debug Access Control Register (CDACNTL)
111 1011	Cache Debug Access Data Register (CDADATA)
111 1100	Nexus3-access
111 1101	LSRL select
111 1110	Enable_OnCE (and bypass)
111 1111	Bypass

13.2.4 NDI Functional Description

13.2.4.1 Enabling Nexus Clients for TAP Access

After the NDI is out of the reset state, the loading of a specific instruction in the JTAG controller (JTAGC) block is required to grant the NDI ownership of the TAP. Each Nexus client has its own JTAGC instruction opcode for ownership of the TAP, granting that client the means to read/write its registers. The JTAGC instruction opcode for each Nexus client is shown in Table 13-7. After the JTAGC opcode for a client has been loaded, the client is enabled by loading its NEXUS-ENABLE instruction. The NEXUS-ENABLE instruction opcode for each Nexus client is listed in Table 13-8. Opcodes for all other instructions supported by Nexus clients can be found in the relevant sections of this chapter.

Table 13-7. JTAG Client Select Instructions

JTAGC Instruction	Opcode	Description
ACCESS_AUX_TAP_NPC	10000	Enables access to the NPC TAP controller
ACCESS_AUX_TAP_OnCE	10001	Enables access to the primary e200z7 OnCE TAP controller (Primary CPU, core 0)
ACCESS_AUX_TAP_eTPU	10010	Enables access to the primary eTPU Nexus TAP controller (eTPU_A, eTPU_B, CDC_AB)
ACCESS_AUX_TAP_NXDM	10011	Enables access to the eDMA_A Nexus TAP controller
ACCESS_AUX_TAP_NXFR	10100	Enables access to the FlexRay Nexus TAP controller
ACCESS_AUX_TAP_eTPU_SECONDARY	10110	Enables access to the secondary eTPU Nexus TAP controller (eTPU_C)
ACCESS_AUX_TAP_NXDM_B	10111	Enables access to the eDMA_B Nexus TAP controller
ACCESS_AUX_TAP_OnCE_1	11001	Enables access to the secondary e200z7 OnCE TAP controller (Secondary CPU, core 1)
ACCESS_AUX_TAP_MULTI	11100	Serialize the JTAG instructions to all internal cores
BYPASS	11111	Bypass the TAP controller

Table 13-8. Nexus Client JTAG Instructions

Instruction	Description	Opcode
NPC JTAG Instruction Opcodes		
NEXUS_ENABLE	Opcode for NPC Nexus Enable instruction (4-bits)	0x0
BYPASS	Opcode for the NPC BYPASS instruction (4-bits)	0xF
e200z7 OnCE JTAG Instruction Opcodes[1]		
NEXUS3_ACCESS	Opcode for e200z7 OnCE Nexus Enable instruction (10-bits)	0x7C
BYPASS	Opcode for the e200z7 OnCE BYPASS instruction (10-bits)	0x7F
eDMA Nexus JTAG Instruction Opcodes		
NEXUS_ACCESS	Opcode for eDMA Nexus Enable instruction (4-bits)	0x0
BYPASS	Opcode for the eDMA Nexus BYPASS instruction (4-bits)	0xF

Table 13-8. Nexus Client JTAG Instructions

Instruction	Description	Opcode
FlexRay Nexus JTAG Instruction Opcodes		
NEXUS_ACCESS	Opcode for FlexRay Nexus Enable instruction (4-bits)	0x0
BYPASS	Opcode for the FlexRay Nexus BYPASS instruction (4-bits)	0xF

[1] See the e200z7 Reference Manual for a complete list of available OnCE instructions.

13.2.4.2 Configuring the NDI for Nexus Messaging

The NDI is placed in disabled mode upon exit of power-on reset. If message transmission via the auxiliary port is desired, a write to the port configuration register (PCR) located in the NPC is then required to enable the NDI and select the mode of operation. Asserting MCKO_EN in the PCR places the NDI in enabled mode and enables MCKO. The frequency of MCKO is selected by writing the MCKO_DIV field. Asserting or negating the FPM bit selects full-port or reduced-port mode, respectively. When writing to the PCR, the PCR lsb (least significant bit) must be written to a logic 0. Setting the lsb of the PCR enables factory debug mode and prevents the transmission of Nexus messages.

Table 13-9 describes the NDI configuration options.

Table 13-9. NDI Configuration Options

JCOMP Asserted	MCKO_EN bit of the Port Configuration Register	FPM bit of the Port Configuration Register	Configuration
No	X	X	Reset
Yes	0	X	Disabled
Yes	1	1	Full-port mode
Yes	1	0	Reduced-port mode

13.2.4.3 Programmable MCKO Frequency

MCKO is an output clock to the development tools used for the timing of $\overline{\text{MSEO}}$ and MDO pin functions. MCKO is derived from the system clock, and its frequency is determined by the value of the MCKO_DIV field in the port configuration register (PCR) located in the NPC. Possible operating frequencies include one-half, one-quarter, and one-eighth system clock speed.

Table 13-10 shows the MCKO_DIV encodings. In this table, SYS_CLK represents the core clock frequency. The default value selected if a reserved encoding is programmed is SYS_CLK divided by two

NOTE:

The SYS_CLK setting for MCKO should only be used if this setting does not violate the maximum operating frequency of the auxiliary port pins.

Table 13-10. MCKO_DIV Values

MCKO_DIV[2:0]	MCKO Frequency[1]
0b000	SYS_CLK
0b001	SYS_CLK ÷ 2
0b010	Invalid value
0b011	SYS_CLK ÷ 4
0b100	Invalid value
0b101	Invalid value
0b110	Invalid value
0b111	SYS_CLK ÷ 8

[1] The SYS_CLK setting for MCKO should only be used if this setting does not violate the maximum operating frequency of the auxiliary port pins (specified in the *MPC5676R Data Sheet*).

13.2.4.4 Nexus Messaging

Most of the messages transmitted by the NDI include a SRC field. This field is used to identify which source generated the message. Table 13-11 shows the values used for the SRC field by the different clients on the device. These 4-bit values are specific to the device. The same values are used for the client select values written to the client select control register.

Table 13-11. SRC Packet Encodings

SRC[3:0]	Client
0b0000	e200z7 (core 0)
0b0001	eDMA_A
0b0010	eTPU_A
0b0011	eTPU_B
0b0100	eTPU CDC[1]
0b0101	Reserved
0b0110	FlexRay
0b0111	eDMA_B
0b1000	e200z7 (core 1)
0b1001	Reserved
0b1010	eTPU_C
0b1011	Reserved
0b1100	eTPU CDC2

Table 13-11. SRC Packet Encodings (continued)

SRC[3:0]	Client
0b1101	Reserved
0b1110	Reserved
0b1111	Reserved

[1] CDC is the eTPU Coherent Dual-Parameter Controller. See the *eTPU Reference Manual* for more information.

13.2.4.5 Nexus Port Ready Status

Following a power-on reset, the lsb of the auxiliary output port pins (MDO[0]) can be monitored to provide the ready status of the Nexus port. MDO[0] is driven to a logic one until the system is ready to communicate after exiting power-on reset. Once ready, MDO[0] is negated and tools can begin Nexus configuration.

13.2.4.6 Breakpoint Interconnections

Table 13-12 lists the breakpoint and watchpoint interconnections within this device. Each processor core's breakpoint and watchpoint events are routed to the other core and to the Nexus Port Controller (NPC) for distribution to other modules. The eTPU NDEDI (eTPUA and eTPUB) and NSEDI (eTPUC) peripheral breakpoint events are capable of halting processor core 0 and/or core 1 as well as any other eTPU module. Breakpoint destination modules can be programmed to halt on the event input signals shown below.

Table 13-12. Breakpoint Routing Table

Breakpoint Source		Breakpoint Destination	
Module	**Signal**	**Module**	**Signal**
core 0	nex_evto_b	NPC	nex_evto_b
core 0	nex_wevto[0]	core 1	p_devt2
core 0	nex_wevto[1]	NPC	nex_bkpt_req
core 1	nex_evto_b	NPC	nex_evto_b
core 1	nex_wevto[0]	core 0	p_devt2
core 1	nex_wevto[1]	NPC	nek_bkpt_req
NDEDI	EVTO	NPC	nex_evto_b
NDEDI	etpu_n3_bkpt_req	NPC	nex_bkpt_req
NSEDI	EVTO	NPC	nex_evto_b
NSEDI	etpu_n3_bkpt_req	NPC	nex_bkpt_req
NPC	nex_de_rq OR ipg_debug	core 0	p_devt1
NPC	nex_de_rq OR ipg_debug	core 1	p_devt1

Table 13-12. Breakpoint Routing Table (continued)

Breakpoint Source		Breakpoint Destination	
NPC	nex_de_rq OR ipg_debug‘	NDEDI	ipg_debug
NPC	nex_de_rq OR ipg_debug	NSEDI	ipg_debug
DMA_A	EVTO	NPC	nex_evto_b
DMA_B	EVTO	NPC	nex_evto_b
Flexray	EVTO	NPC	nex_evto_b
DTS	EVTO	NPC	nex_evto_b

Debug mode entry requests from the eTPUs (etpu_n3_bkpt_req) and core watchpoint events (nex_wevto[1]) are sent to the NPC input ports nex_bkpt_req[], ORed together, and sent back out on nex_de_req, which is returned to all cores and eTPUs. Core0 and core1 entry to debug mode is indicated on their respective jd_debug_b signals, which are ORed and sent to the NPC on nex_debug_b. The NPC block inverts this signal and returns it to ipg_debug as a device wide signal to other modules. ipg_debug is also ORed with nex_de_req to provide a means for any core to halt any eTPU block.

13.2.5 Nexus Port Controller (NPC)

The Nexus port controller (NPC) is that part of the NDI that controls access and arbitration of the device’s internal Nexus modules. The NPC contains the port configuration register (PCR) and the device identification register (DID). The contents of the NPC DID are the same as the JTAGC device identification register.

13.2.5.1 Overview

The device incorporates multiple modules that require development support. Each of these modules implements a development interface based on the IEEE-ISTO 5001-2001 standard and must share the input and output ports that interface with the development tool. The NPC controls the usage of these ports in a manner that allows the individual modules to share the ports, while appearing to the development tool as a single module.

13.2.5.2 Features

The NPC performs the following functions:

- Controls arbitration for ownership of the Nexus auxiliary output port
- Nexus device identification register and messaging
- Generates MCKO enable and frequency division control signals
- Controls sharing of $\overline{\text{EVTO}}$
- Control of the device-wide debug mode

- Generates asynchronous reset signal for Nexus modules based on JCOMP input, censorship status, and power-on reset status
- System clock locked status indication via MDO[0] during Nexus reset
- Provides Nexus support for censorship mode

13.2.6 NPC Memory Map and Register Definition

This section provides a detailed description of the NPC registers accessible to the end user. Individual bit-level descriptions and reset states of the registers are included.

13.2.6.1 Memory Map

Table 13-13 shows the NPC registers by index values. The registers are not memory-mapped and can only be accessed via the TAP. The NPC does not implement the client select control register because the value does not matter when accessing the registers. The bypass register (see Section 13.2.6.2.1, “Bypass Register”) and instruction register (see Section 13.2.6.2.2, “Instruction Register”) have no index values. These registers are not accessed in the same manner as Nexus client registers.

Table 13-13. NPC Memory Map

Index	Register Name	Register Description	Bits
0	DID	Device ID register	32
127	PCR	Port configuration register	32

13.2.6.2 Register Descriptions

This section consists of NPC register descriptions. Additional information regarding references to the TAP controller state can be found in Section 13.1.4.3, “TAP Controller State Machine.”

13.2.6.2.1 Bypass Register

The bypass register is a single-bit shift register path selected for serial data transfer between TDI and TDO when the BYPASS instruction or any unimplemented instructions are active. After entry into the Capture-DR state, the single-bit shift register is set to a logic 0. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0.

13.2.6.2.2 Instruction Register

The NPC uses a 4-bit instruction register as shown in Figure 13-7. The instruction register is accessed via the SELECT_IR_SCAN path of the tap controller state machine, and allows instructions to be loaded into the module to enable the NPC for register access (NEXUS_ENABLE) or select the bypass register as the shift path from TDI to TDO (BYPASS or unimplemented instructions).

Instructions are shifted in through TDI while the TAP controller is in the Shift-IR state, and latched on the falling edge of TCK in the Update-IR state. The latched instruction value can only be changed in the Update-IR and test-logic-reset TAP controller states. Synchronous entry into the test-logic-reset state results in synchronous loading of the BYPASS instruction. Asynchronous entry into the test-logic-reset state results in asynchronous loading of the BYPASS instruction. During the Capture-IR TAP controller state, the instruction register is loaded with the value of the previously executed instruction, making this value the register's read value when the TAP controller is sequenced into the Shift-IR state.

	3	2	1	0
R	Previous Instruction Opcode			
W	Instruction Opcode			
Reset:	BYPASS Instruction Opcode (0xF)			

Figure 13-7. 4-Bit Instruction Register

13.2.6.2.3 Nexus Device ID Register (DID)

The NPC device identification register, shown in Figure 13-8, allows the part revision number, design center, part identification number, and manufacturer identity code of the part to be determined through the auxiliary output port.

Reg Index: 0

Access: User R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	Part Revision Number				Design Center						Part Identification Number					
W																
Reset	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	Part Identification Number (*continued*)				Manufacturer Identity Code											1
W																
Reset	0	1	1	0	0	0	0	0	0	0	0	1	1	1	0	1

Figure 13-8. Nexus Device ID Register (DID)

Table 13-14. DID Register Field Descriptions

Field	Description
31–28 PRN	Part revision number. Contains the revision number of the part. This field changes with each revision of the device or module.
27–22 DC	Design center. Indicates the Freescale design center. This value is 0x20.
21–12 PIN	Part identification number. Contains the part number of the device. The PIN for the MPC5676R is 0x276.
11–1 MIC	Manufacturer identity code. Contains the reduced Joint Electron Device Engineering Council (JEDEC) ID for Freescale, 0x000E.
0	Fixed per JTAG 1149.1 Always set to 1.

13.2.6.2.4 Port Configuration Register (PCR)

The PCR, shown in Figure 13-9, is used to select the NPC mode of operation, enable MCKO and select the MCKO frequency, and enable or disable MCKO gating. This register must be configured as soon as the NPC is enabled.

NOTE

> The mode (MCKO_GT) or clock division (MCKO_DIV) bits must not be modified after MCKO has been enabled. Changing the mode or clock division while MCKO is enabled can produce unpredictable results.

Reg Index: 127 Access: R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FPM	MCKO_GT	MCKO_EN	MCKO_DIV			0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R																PSTAT_EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-9. Port Configuration Register (PCR)

Table 13-15. PCR Field Descriptions

Field	Description
31 FPM	Full port mode. Determines if the auxiliary output port uses the full MDO port or a reduced MDO port to transmit messages. 0 The subset of MDO[11:0] pins are used to transmit messages. 1 All MDO[15:0] pins are used to transmit messages. Section 3.2.1.13, “Pad Configuration Registers (SIU_PCR)” shows how GPIO is enabled or disabled by the FPM setting.
30 MCKO_GT	MCKO clock gating control. Enables or disables MCKO clock gating. If clock gating is enabled, the MCKO clock is gated when the NPC is in enabled mode but not actively transmitting messages on the auxiliary output port. When clock gating is disabled, MCKO is allowed to run even if no auxiliary output port messages are being transmitted. 0 MCKO gating is disabled. 1 MCKO gating is enabled.
29 MCKO_EN	MCKO enable. Enables the MCKO clock. When enabled, the frequency of MCKO is determined by the MCKO_DIV field. 0 MCKO clock is driven to zero. 1 MCKO clock is enabled.

Table 13-15. PCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>28–26
MCKO_DIV
[2:0]</td><td>MCKO division factor. Determines the frequency of MCKO relative to the system clock frequency when MCKO_EN is asserted. The table below shows the meaning of MCKO_DIV values. In this table, SYS_CLK represents the system clock frequency.
<table>
<tr><th>MCKO_DIV[2:0]</th><th>MCKO Frequency</th></tr>
<tr><td>0</td><td>SYS_CLK[1]</td></tr>
<tr><td>1</td><td>SYS_CLK ÷ 2</td></tr>
<tr><td>2</td><td>Invalid value</td></tr>
<tr><td>3</td><td>SYS_CLK ÷ 4</td></tr>
<tr><td>4</td><td>Invalid value</td></tr>
<tr><td>5</td><td>Invalid value</td></tr>
<tr><td>6</td><td>Invalid value</td></tr>
<tr><td>7</td><td>SYS_CLK ÷ 8</td></tr>
</table>
[1] The SYS_CLK setting for MCKO should only be used if this setting does not violate the maximum operating frequency of the auxiliary port pins.</td></tr>
<tr><td>25–1</td><td>Reserved</td></tr>
<tr><td>0
PSTAT_EN</td><td>Processor status mode enable. Enables processor status (PSTAT) mode. In PSTAT mode, all auxiliary output port MDO pins are used to transmit processor status information, and Nexus messaging is unavailable.
0 PSTAT mode disabled
1 PSTAT mode enabled
Note: PSTAT mode is intended for factory processor debug only. The PSTAT_EN bit must be written to disable PSTAT mode by the customer. No Nexus messages are transmitted under any circumstances when PSTAT mode is enabled</td></tr>
</table>

13.2.7 NPC Functional Description

13.2.7.1 NPC Reset Configuration

The NPC is placed in disabled mode upon exit of reset. If message transmission via the auxiliary port is desired, a write to the PCR is then required to enable the NPC and select the mode of operation. Asserting MCKO_EN places the NPC in enabled mode and enables MCKO. The frequency of MCKO is selected by writing the MCKO_DIV field. Asserting or negating the FPM bit selects full-port or reduced-port mode, respectively.

Table 13-9 describes the NPC reset configuration options.

13.2.7.2 Auxiliary Output Port

The auxiliary output port is shared by each of the Nexus modules on the device. The NPC communicates with each of the individual modules and arbitrates for access to the port. Additional information about the auxiliary port is found in Section 13.2.2, “External Signal Description.”

13.2.7.2.1 Output Message Protocol

The protocol for transmitting messages via the auxiliary port is accomplished with the $\overline{\text{MSEO}}$ functions. The $\overline{\text{MSEO}}$ pins are used to signal the end of variable-length packets and the end of messages. They are not required to indicate the end of fixed-length packets. MDO and $\overline{\text{MSEO}}$ are sampled on the rising edge of MCKO.

Figure 13-10 illustrates the state diagram for $\overline{\text{MSEO}}$ transfers. All transitions not included in the figure are reserved, and must not be used.

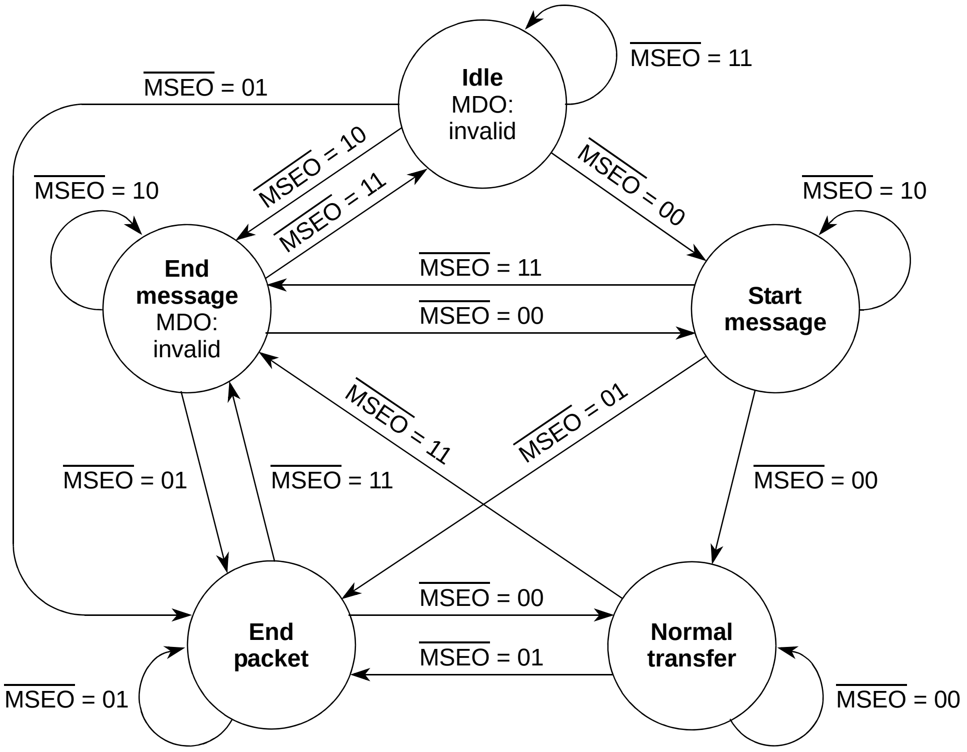


Figure 13-10. $\overline{\text{MSEO}}$ Transfers

13.2.7.2.2 Output Messages

In addition to sending out messages generated in other Nexus modules, the NPC can also output the device ID message contained in the device ID register on the MDO pins. The device ID message can also be sent out serially through TDO.

Table 13-16 describes the device ID message that the NPC can transmit on the auxiliary port. The TCODE is the first packet transmitted.

Table 13-16. NPC Output Messages

Message Name	Min. Packet Bits	Max Packet Bits	Packet Type	Packet Name	Packet Description
Device ID Message	6	6	Fixed	TCODE	Value = 1
	32	32	Fixed	ID	DID register contents

Figure 13-11 shows the various message formats that the pin interface formatter has to encounter.

Message	TCODE	Field #1	Field #2	Field #3	Field #4	Field #5	Min. Size[1] Bits	Max. Size[2] Bits
Device ID Message	1	Fixed = 32	—	—	—	—	38	38

[1] Minimum information size. The actual number of bits transmitted depends on the number of MDO pins

[2] Maximum information size. The actual number of bits transmitted depends on the number of MDO pins

Figure 13-11. Message Field Sizes

The double edges in Figure 13-11 indicate the starts and ends of messages. Fields without shaded areas between them are grouped into super-fields and can be transmitted together without end-of-packet indications between them.

Rules of Messages

The rules of messages include the following:

- A variable-sized field within a message must end on a port boundary. (Port boundaries depend on the number of MDO pins active with the current reset configuration.)
- A variable-sized field can start within a port boundary only when following a fixed-length field.
- Super-fields must end on a port boundary.
- When a variable-length field is sized such that it does not end on a port boundary, it is necessary to extend and zero fill the remaining bits after the highest order bit so that it can end on a port boundary.
- Multiple fixed-length packets can start and/or end on a single clock.
- When any packet follows a variable-length packet, it must start on a port boundary.
- The field containing the TCODE number is always transferred out first, followed by subsequent fields of information.
- Within a field, the lowest significant bits are shifted out first. Figure 13-12 shows the transmission sequence of a message that is made up of a TCODE followed by three fields.

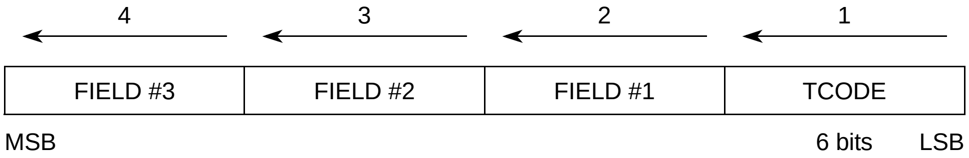


Figure 13-12. Transmission Sequence of Messages

13.2.7.2.3 IEEE 1149.1-2001 (JTAG) TAP

The NPC uses the IEEE 1149.1-2001 TAP for accessing registers. Each of the individual Nexus modules on the device implements a TAP controller for accessing its registers as well. TAP signals include TCK, TDI, TMS, and TDO. Detailed information about the TAP controller state machine can be found in Section 13.1.4.3, “TAP Controller State Machine.”

The IEEE 1149.1-2001 specification can be ordered for further detail on electrical and pin protocol compliance requirements.

The NPC implements a Nexus controller state machine that transitions based on the state of the IEEE 1149.1-2001 state machine shown in Figure 13-8. The Nexus controller state machine is defined by the IEEE-ISTO 5001-2003 standard. It is shown in Figure 13-15.

The instructions implemented by the NPC TAP controller are listed in Table 13-17. The value of the NEXUS-ENABLE instruction is 0b0000. Each unimplemented instruction acts like the BYPASS instruction. The size of the NPC instruction register is 4-bits.

Table 13-17. Implemented Instructions

Instruction Name	Private/Public	Opcode	Description
NEXUS-ENABLE	Public	0x0	Activate Nexus controller state machine to read and write NPC registers.
BYPASS	Private	0xF	NPC BYPASS instruction. Also the value loaded into the NPC IR upon exit of reset.

Data is shifted between TDI and TDO starting with the least significant bit as illustrated in Figure 13-13. This applies for the instruction register and all Nexus tool-mapped registers.

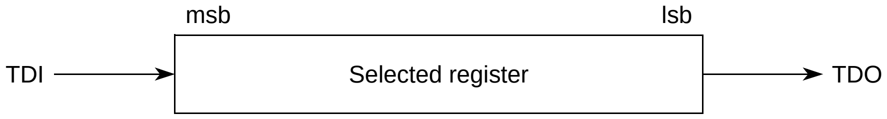
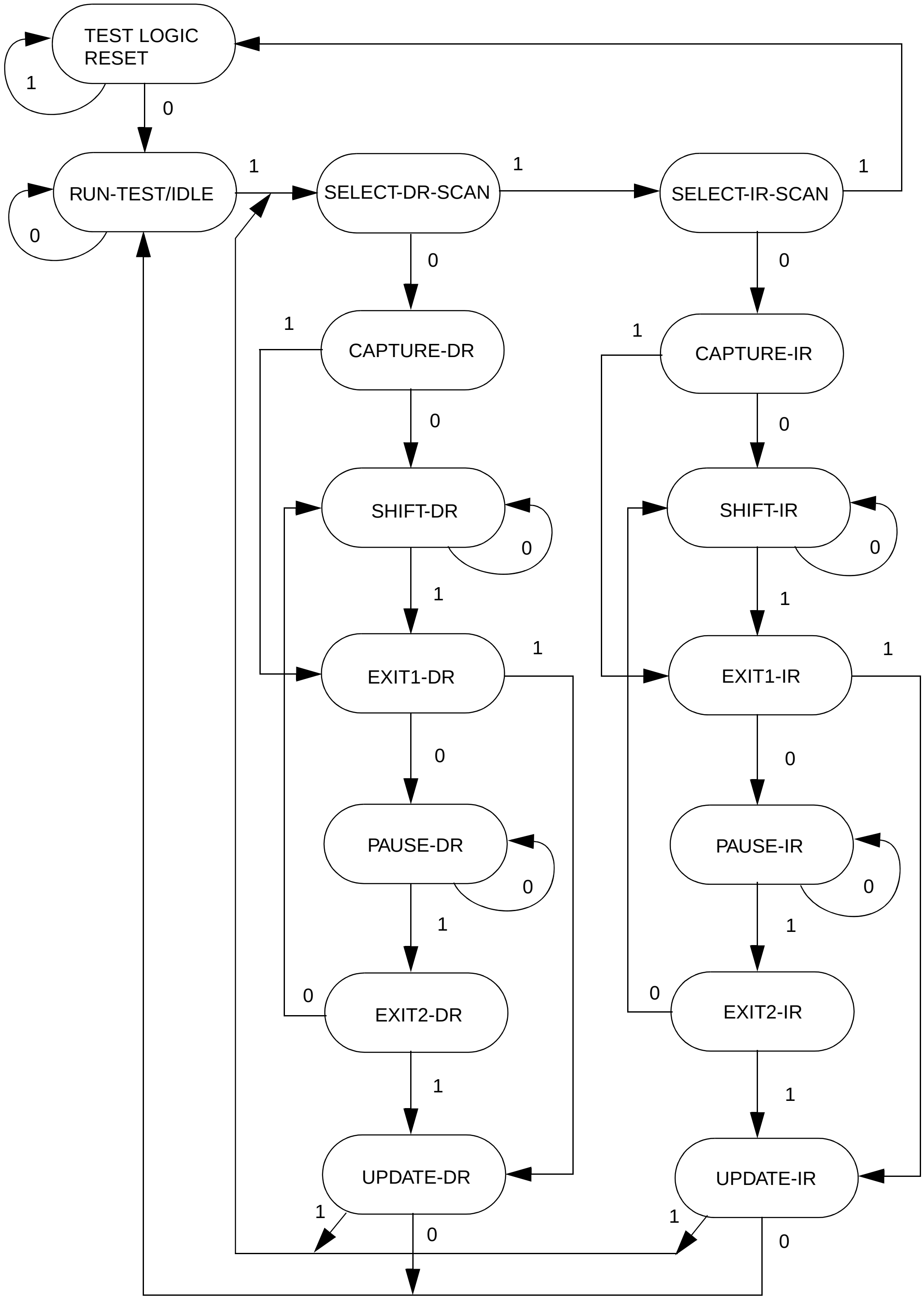


Figure 13-13. Shifting Data Into a Register

Enabling the NPC TAP Controller

Assertion of the power-on reset signal or negating JCOMP resets the NPC TAP controller. When not in power-on reset, the NPC TAP controller is enabled by asserting JCOMP and loading the ACCESS_AUX_TAP_NPC instruction in the JTAGC. Loading the NEXUS-ENABLE instruction then grants access to NPC registers.



NOTE: The value shown adjacent to each state transition in this figure represents the value of TMS at the time of a rising edge of TCK.

Figure 13-14. IEEE 1149.1-2001 TAP Controller State Machine

Retrieving Device IDCODE

The Nexus TAP controller does not implement the IDCODE instruction. However, the device identification message can be output by the NPC through the auxiliary output port or shifted out serially by accessing the NPC device ID register through the TAP. If the NPC is enabled, transmission of the device identification message on the auxiliary output port MDO pins occurs immediately after a write to the PCR. Transmission of the device identification message serially through TDO is achieved by performing a read of the register contents as described in Section , “Selecting a Nexus Client Register.”

Loading NEXUS-ENABLE Instruction

Access to the NPC registers is enabled by loading the NPC NEXUS-ENABLE instruction when NPC has ownership of the TAP. This instruction is shifted in via the SELECT-IR-SCAN path and loaded in the UPDATE-IR state. At this point, the Nexus controller state machine, shown in Figure 13-15, transitions to the REG_SELECT state. The Nexus controller has three states: idle, register select, and data access. Table 13-18 illustrates the IEEE 1149.1 sequence to load the NEXUS-ENABLE instruction.

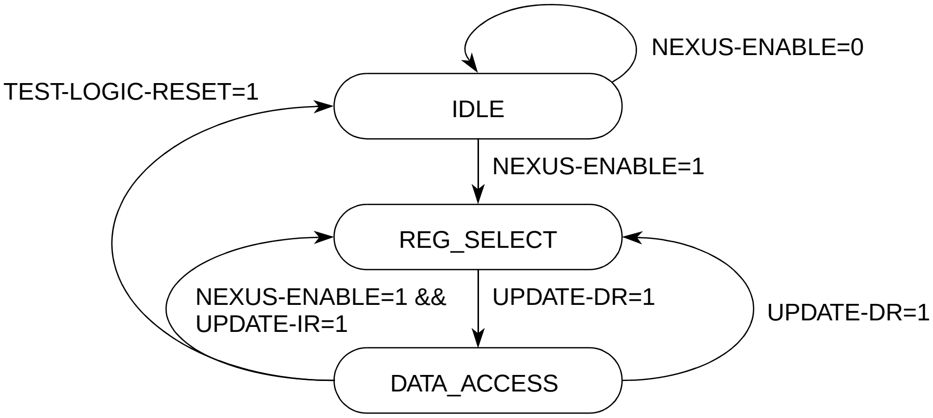


Figure 13-15. NEXUS Controller State Machine

Table 13-18. Loading NEXUS-ENABLE Instruction

Clock	TDI	TMS	IEEE 1149.1 State	Nexus State	Description
0	—	0	RUN-TEST/IDLE	IDLE	IEEE 1149.1-2001 TAP controller in idle state
1	—	1	SELECT-DR-SCAN	IDLE	Transitional state
2	—	1	SELECT-IR-SCAN	IDLE	Transitional state
3	—	0	CAPTURE-IR	IDLE	Internal shifter loaded with current instruction
4	—	0	SHIFT-IR	IDLE	TDO becomes active, and the IEEE 1149.1-2001 shifter is ready. Shift in all but the last bit of the NEXUS_ENABLE instruction.
5–7	0	0	3 TCKS in SHIFT-IR	IDLE	
8	0	1	EXIT1-IR	IDLE	Last bit of instruction shifted in
9	—	1	UPDATE-IR	IDLE	NEXUS-ENABLE loaded into instruction register
10	—	0	RUN-TEST/IDLE	REG_SELECT	Ready to be read/write Nexus registers

Selecting a Nexus Client Register

When the NEXUS-ENABLE instruction is decoded by the TAP controller, the input port allows development tool access to all Nexus registers. Each register has a 7-bit address index.

All register access is performed via the SELECT-DR-SCAN path of the IEEE 1149.1–2001 TAP controller state machine. The Nexus controller defaults to the REG_SELECT state when enabled. Accessing a register requires two passes through the SELECT-DR-SCAN path: one pass to select the register and the second pass to read/write the register.

The first pass through the SELECT-DR-SCAN path is used to enter an 8-bit Nexus command consisting of a read/write control bit in the lsb followed by a 7-bit register address index, as illustrated in Figure 13-16. The read/write control bit is set to 1 for writes and 0 for reads.

MSB	LSB
7-bit register index	R/W

Figure 13-16. IEEE 1149.1 Controller Command Input

The second pass through the SELECT-DR-SCAN path is used to read or write the register data by shifting in the data (lsb first) during the SHIFT-DR state. When reading a register, the register value is loaded into the IEEE 1149.1-2001 shifter during the CAPTURE-DR state. When writing a register, the value is loaded from the IEEE 1149.1-2001 shifter to the register during the UPDATE-DR state. When reading a register, there is no requirement to shift out the entire register contents. Shifting can be terminated after the required number of bits have been acquired.

Table 13-19 illustrates a sequence that writes a 32-bit value to a register.

Table 13-19. Write to a 32-Bit Nexus Client Register

Clock	TMS	IEEE 1149.1 State	Nexus State	Description
0	0	RUN-TEST/IDLE	REG_SELECT	IEEE 1149.1-2001 TAP controller in idle state
1	1	SELECT-DR-SCAN	REG_SELECT	First pass through SELECT-DR-SCAN path
2	0	CAPTURE-DR	REG_SELECT	Internal shifter loaded with current value of controller command input.
3	0	SHIFT-DR	REG_SELECT	TDO becomes active, and write bit and 6 bits of register index shifted in.
7 TCKs				
11	1	EXIT1-DR	REG_SELECT	Last bit of register index shifted into TDI
12	1	UPDATE-DR	REG_SELECT	Controller decodes and selects register
13	1	SELECT-DR-SCAN	DATA_ACCESS	Second pass through SELECT-DR-SCAN path
14	0	CAPTURE-DR	DATA_ACCESS	Internal shifter loaded with current value of register
15	0	SHIFT-DR	DATA_ACCESS	TDO becomes active, and outputs current value of register while new value is shifted in through TDI
31 TCKs				
47	1	EXIT1-DR	DATA_ACCESS	Last bit of current value shifted out TDO. Last bit of new value shifted in TDI.

Table 13-19. Write to a 32-Bit Nexus Client Register (continued)

Clock	TMS	IEEE 1149.1 State	Nexus State	Description
48	1	UPDATE-DR	DATA_ACCESS	Value written to register
49	0	RUN-TEST/IDLE	REG_SELECT	Controller returned to idle state. It could also return to SELECT-DR-SCAN to write another register.

13.2.7.2.4 Nexus Auxiliary Port Sharing

Each of the Nexus modules on the MCU implements a request/grant scheme to arbitrate for control of the Nexus auxiliary port when Nexus data is ready to be transmitted.

All modules arbitrating for the port are given fixed priority levels relative to each other. If multiple modules have the same request level, this priority level is used as a tie-breaker. To avoid monopolization of the port, the module given the highest priority level alternates following each grant. Immediately out of reset the order of priority, from highest to lowest, is: NPC, NZ7C3_0, NZ7C3_1, NDEDI, NXDM, NXFR, NSEDI, and NXDM_B. This arbitration mechanism is controlled internally and is not programmable by tools or the user.

13.2.7.2.5 Nexus JTAG Port Sharing

Each of the individual Nexus modules on the device implements a TAP controller for accessing its registers. When JCOMP is asserted, only the module whose ACCESS_AUX_TAP instruction is loaded has control of the TAP (see Section 13.1.4.4, “JTAGC Instructions”). This allows the interface to all of these individual TAP controllers to appear to be a single port from outside the device. After a Nexus module has ownership of the TAP, that module acts like a single-bit shift register, or bypass register, if no register is selected as the shift path.

13.2.7.2.6 MCKO

MCKO is an output clock to the development tools used for the timing of $\overline{\text{MSEO}}$ and MDO pin functions. MCKO is derived from the system clock and its frequency is determined by the value of the MCKO_DIV[2:0] field in the PCR. Possible operating frequencies include one-half, one-quarter, and one-eighth system clock speed. MCKO is enabled by setting the MCKO_EN bit in the PCR.

The NPC also controls dynamic MCKO clock gating when in full- or reduced-port modes. The setting of the MCKO_GT bit inside the PCR determines whether or not MCKO gating control is enabled. The MCKO_GT bit resets to a logic 0. In this state gating of MCKO is disabled. To enable gating of MCKO, the MCKO_GT bit in the PCR is written to a logic 1. When MCKO gating is enabled, MCKO is driven to a logic 0 if the auxiliary port is enabled but not transmitting messages and there are no pending messages from Nexus clients.

13.2.7.2.7 $\overline{\text{EVTO}}$ Sharing

The NPC controls sharing of the $\overline{\text{EVTO}}$ output between all Nexus clients that produce an $\overline{\text{EVTO}}$ signal. $\overline{\text{EVTO}}$ is driven for one MCKO period whenever any module drives its $\overline{\text{EVTO}}$. When there is no active MCKO, such as in disabled mode, the NPC assumes an MCKO frequency of one-half system clock speed when driving $\overline{\text{EVTO}}$. $\overline{\text{EVTO}}$ sharing is active as long as the NPC is not in reset.

13.2.7.2.8 Nexus Reset Control

The JCOMP input that is used as the primary reset signal for the NPC is also used by the NPC to generate a single-bit reset signal for other Nexus modules. If JCOMP is negated, an internal reset signal is asserted, indicating that all Nexus modules must be held in reset. This internal reset signal is also asserted during a power-on reset. This single bit reset signal functions much like the IEEE 1149.1-2001 defined $\overline{\text{TRST}}$ signal and allows JCOMP reset information to be provided to the Nexus modules without each module having to sense the JCOMP signal directly.

13.2.8 NPC Initialization and Application Information

13.2.8.1 Accessing NPC Tool-Mapped Registers

To initialize the TAP for NPC register accesses, the following sequence is required:

1. Enable the NPC TAP controller. This is achieved by asserting JCOMP and loading the ACCESS_AUX_TAP_NPC instruction in the JTAGC.
2. Load the TAP controller with the NEXUS-ENABLE instruction.

To write control data to NPC tool-mapped registers, the following sequence is required:

1. Write the 7-bit register index and set the write bit to select the register with a pass through the SELECT-DR-SCAN path in the TAP controller state machine.
2. Write the register value with a second pass through the SELECT-DR-SCAN path. The prior value of this register is shifted out during the write.

To read status and control data from NPC tool-mapped registers, the following sequence is required:

1. Write the 7-bit register index and clear the write bit to select register with a pass through SELECT-DR-SCAN path in the TAP controller state machine.
2. Read the register value with a second pass through the SELECT-DR-SCAN path. Data shifted in is ignored.

See the IEEE-ISTO 5001-2010 standard for more detail.

13.2.9 Nexus TPU Development Interfaces (NDEDI and NSEDI)

The enhanced timing processor units (eTPU) have their own Nexus class 3 interfaces, the Nexus dual eTPU development interface (NDEDI) covering eTPU_A and eTPU_B and the Nexus Single eTPU development interface covering eTPU_C. Each interface also includes a coherent dual-parameter controller (CDC) for coherent access to eTPU parameters. In each interface, the eTPU modules and CDC unit appear as separate Nexus clients. See the *Enhanced Time Processor Unit Reference Manual* for more information about the NDEDI and NSEDI modules.

Reg Index: 0 Access: R/O

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PRN				DC						PIN					
W																
Reset (NDEDI)	0	0	0	0	1	0	0	0	0	0	1	0	0	0	1	0
Reset(NSEDI)	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PIN				MIC											1
W																
Reset(NDEDI)	0	1	0	0	0	0	0	0	0	0	0	1	1	1	0	1
Reset(NSEDI)	0	1	0	0	0	0	0	0	0	0	0	1	1	1	0	1

Figure 13-17. NDEDI Device ID Register (DID)

Table 13-20. NDEDI DID Register Field Descriptions

Field	Description
31–28 PRN	Part revision number. Contains the revision number of the part. This field changes with each revision of the device or module.
27–22 DC	Design center. Indicates the Freescale design center. This value is 0x20.
21–12 PIN	Part identification number. Contains the part number of the device.
11–1 MIC	Manufacturer identity code. Contains the reduced Joint Electron Device Engineering Council (JEDEC) ID for Freescale, 0xE.
0	Fixed per JTAG 1149.1 1 Always set

13.2.10 e200z7 Class 3 Nexus Module (NZ7C3)

The NZ7C3 module provides real-time development capabilities for the device core in compliance with the IEEE-ISTO Nexus 5001-2003 standard. This module provides development support capabilities without requiring the use of address and data pins for internal visibility.

13.2.10.1 Introduction

This section defines the auxiliary pin functions, transfer protocols and standard development features of the NZ7C3 module. The development features supported are Program trace, data trace, watchpoint messaging, ownership trace, and read/write access via the JTAG interface.

NOTE

Throughout this section references are made to the auxiliary port and its specific signals, such as MCKO, $\overline{\text{MSEO}}$[1:0], MDO[15:0] and others. The device NPC module arbitrates the access of the single auxiliary port. To simplify the description of the function of the NZ7C3 module, the interaction of the NPC is omitted and the configuration in this chapter describes an NPC with a dedicated auxiliary port. The auxiliary port is fully described in Section 13.2.2, “External Signal Description.”

13.2.10.2 Block Diagram

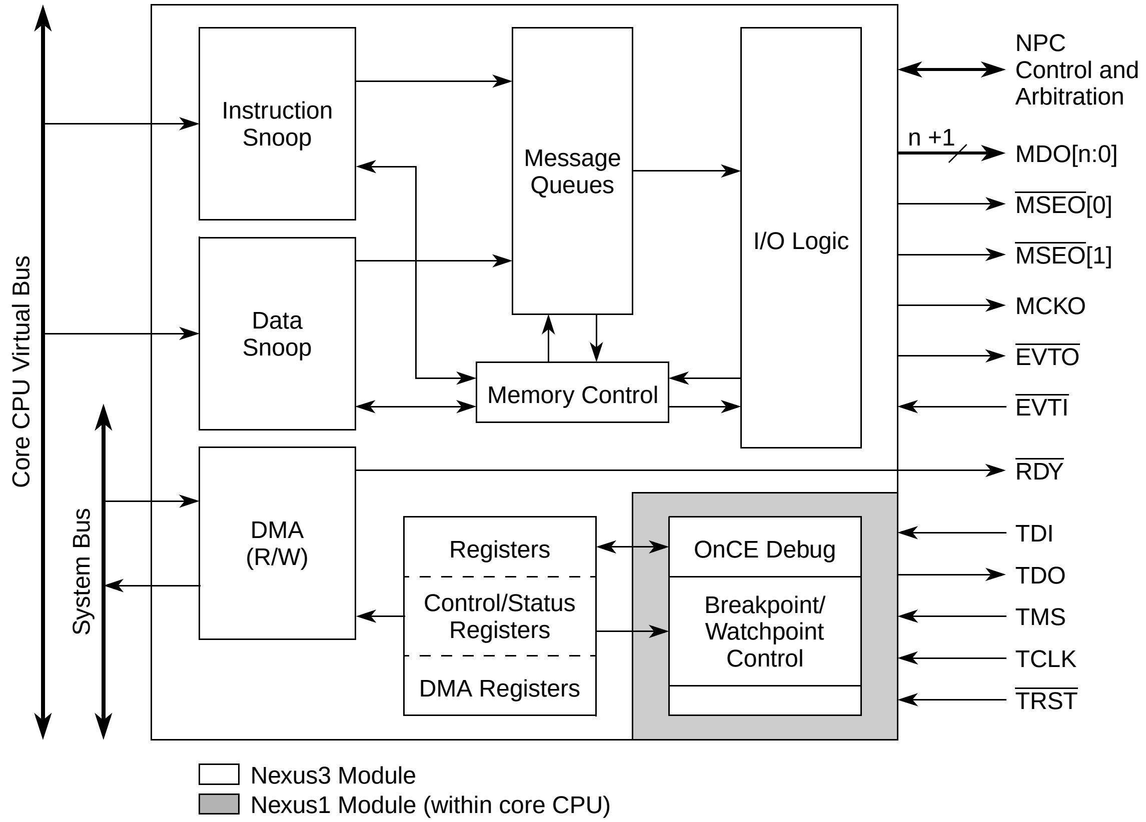


Figure 13-18. e200z7 Nexus3 Functional Block Diagram

13.2.10.3 Overview

Table 13-21 contains a set of terms and definitions associated with the NZ7C3 module.

Table 13-21. Terms and Definitions

Term	Description
IEEE-ISTO 5001	Consortium and standard for real-time embedded system design. World wide Web documentation at http://www.ieee-isto.org/Nexus5001
Auxiliary Port	Refers to Nexus auxiliary port. Used as auxiliary port to the IEEE 1149.1 JTAG interface.
Branch Trace Messaging (BTM)	Visibility of addresses for taken branches and exceptions, and the number of sequential instructions executed between each taken branch.
Client	A functional block on an embedded processor which requires development visibility and controllability. Examples are a central processing unit (CPU) or an intelligent peripheral.
Data Read Message (DRM)	External visibility of data reads to memory-mapped resources.
Data Write Message (DWM)	External visibility of data writes to memory-mapped resources.

Table 13-21. Terms and Definitions (continued)

Term	Description
Data Trace Messaging (DTM)	External visibility of how data flows through the embedded system. This can include DRM and/or DWM.
JTAG Compliant	Device complying to IEEE 1149.1 JTAG standard
JTAG IR & DR Sequence	JTAG instruction register (IR) scan to load an opcode value for selecting a development register. The JTAG IR corresponds to the OnCE command register (OCMD). The selected development register is then accessed via a JTAG data register (DR) scan.
Nexus1	The e200z7 (OnCE) debug module. This module integrated with each e200z7 processor provides all static (core halted) debug functionality. This module is compliant with Class1 of the IEEE-ISTO 5001 standard.
Ownership Trace Message (OTM)	Visibility of process/function that is currently executing.
Public Messages	Messages on the auxiliary pins for accomplishing common visibility and controllability requirements
Standard	The phrase ‘according to the standard’ is used to indicate according to the IEEE-ISTO 5001 standard.
Transfer Code (TCODE)	Message header that identifies the number and/or size of packets to be transferred, and how to interpret each of the packets.
Watchpoint	A data or instruction breakpoint which does not cause the processor to halt. Instead, a pin is used to signal that the condition occurred. A watchpoint message is also generated.

13.2.10.4 Features

The NZ7C3 module is compliant with Class 3 of the IEEE-ISTO 5001-2003 standard. The following features are implemented:

- Program trace via branch trace messaging (BTM). Branch trace messaging displays program flow discontinuities (direct and indirect branches, exceptions, etc.), allowing the development tool to interpolate what transpires between the discontinuities. Thus static code can be traced.
- Data trace via data write messaging (DWM) and data read messaging (DRM). This provides the capability for the development tool to trace reads and/or writes to selected internal memory resources.
- Ownership trace via ownership trace messaging (OTM). OTM facilitates ownership trace by providing visibility of which process ID or operating system task is activated. An ownership trace message is transmitted when a new process/task is activated, allowing the development tool to trace ownership flow.
- Run-time access to embedded processor registers and memory map via the JTAG port. This allows for enhanced download/upload capabilities.
- Watchpoint messaging via the auxiliary pins.
- Watchpoint trigger enable of program and/or data trace messaging.
- High-speed data input/output via the auxiliary port.
- Auxiliary interface for higher data input/output

  - Configurable (minimum and maximum) message data out pins
  - One read/write ready pin
  - One watchpoint-event pin
  - One event-in pin
  - One MCKO (message clock out) pin
- Registers for program trace, data trace, ownership trace and watchpoint trigger.
- All features controllable and configurable via the JTAG port.

13.2.10.5 Enabling Nexus3 Operation

The Nexus module is enabled by loading a single instruction (ACCESS_AUX_TAP_ONCE, as shown in Table 13-7) into the JTAGC instruction register (IR), and then loading the corresponding OnCE OCMD register with the NEXUS3_ACCESS instruction (see Table 13-8). For the e200z7 Class 3 Nexus module, the OCMD value is 0b00_0111_1100. After it is enabled, the module is ready to accept control input via the JTAG pins. See Section 13.2.7, “NPC Functional Description” for more information.

The Nexus module is disabled when the JTAG state machine reaches the test-logic-reset state. This state can be reached by asserting the JCOMP pin or cycling through the state machine using the TMS pin. The Nexus module is also disabled if a power-on-reset (POR) event occurs. If the Nexus3 module is disabled, no trace output is provided, and the module disables (drive inactive) auxiliary port output pins MDO[n:0], $\overline{\text{MSEO}}$[1:0], MCKO. Nexus registers are not available for reads or writes.

13.2.10.6 TCODEs Supported by NZ7C3

The Nexus3 pins allow for flexible transfer operations via public messages. A TCODE defines the transfer format, the number and/or size of the packets to be transferred, and the purpose of each packet. The IEEE-ISTO 5001-2003 standard defines a set of public messages. The NZ7C3 module supports the public TCODEs seen in Table 13-22. Each message contains multiple packets transmitted in the order shown in the table.

Table 13-22. Public TCODEs Supported by NZ7C3

Message Name	Packet Size (bits)		Packet Name	Packet Type	Packet Description
	Min	Max			
Debug Status	6	6	TCODE	Fixed	TCODE number = 0 (0x00)
	4	4	SRC	Fixed	Source processor identifier
	8	8	STATUS	Fixed	Debug status register (DS[31:24])
Ownership Trace Message	6	6	TCODE	Fixed	TCODE number = 2 (0x02)
	4	4	SRC	Fixed	Source processor identifier
	32	32	PROCESS	Fixed	Task/Process ID tag

Table 13-22. Public TCODEs Supported by NZ7C3 (continued)

Message Name	Packet Size (bits)		Packet Name	Packet Type	Packet Description
	Min	Max			
Program Trace - Direct Branch Message[1]	6	6	TCODE	Fixed	TCODE number = 3 (0x03)
	4	4	SRC	Fixed	Source processor identifier
	1	8	I-CNT	Variable	Number of sequential instructions executed since last taken branch
Program Trace - Indirect Branch Message[1]	6	6	TCODE	Fixed	TCODE number = 4 (0x04)
	4	4	SRC	Fixed	Source processor identifier
	1	8	I-CNT	Variable	Number of sequential instructions executed since last taken branch
	1	32	U-ADDR	Variable	Unique part of target address for taken branches/exceptions
Data Trace - Data Write Message	6	6	TCODE	Fixed	TCODE number = 5 (0x05)
	4	4	SRC	Fixed	Source processor identifier
	3	3	DSIZ	Fixed	Data size (see Table 13-26)
	1	32	U-ADDR	Variable	Unique portion of the data write address
	1	64	DATA	Variable	Data write values (see Section 13.2.14.6, “Data Trace,” for details)
Data Trace - Data Read Message	6	6	TCODE	Fixed	TCODE number = 6 (0x06)
	4	4	SRC	Fixed	Source processor identifier
	3	3	DSIZ	Fixed	Data size (see Table 13-26)
	1	32	U-ADDR	Variable	Unique portion of the data read address
	1	64	DATA	Variable	Data read values (see Section 13.2.14.6, “Data Trace,” for details)
Error Message	6	6	TCODE	Fixed	TCODE number = 8 (0x08)
	4	4	SRC	Fixed	Source processor identifier
	5	5	ECODE	Fixed	Error code
Program Trace - Direct Branch Message w/ Sync[1]	6	6	TCODE	Fixed	TCODE number = 11 (0x0B)
	4	4	SRC	Fixed	Source processor identifier
	1	8	I-CNT	Variable	Number of sequential instructions executed since last taken branch
	1	32	F-ADDR	Variable	Full target address (leading zeros truncated)

Table 13-22. Public TCODEs Supported by NZ7C3 (continued)

Message Name	Packet Size (bits)		Packet Name	Packet Type	Packet Description
	Min	Max			
Program Trace - Indirect Branch Message w/ Sync[1]	6	6	TCODE	Fixed	TCODE number = 12 (0x0C)
	4	4	SRC	Fixed	Source processor identifier
	1	8	I-CNT	Variable	Number of sequential instructions executed since last taken branch
	1	32	F-ADDR	Variable	Full target address (leading zeros truncated)
Data Trace - Data Write Message w/ Sync	6	6	TCODE	Fixed	TCODE number = 13 (0x0D)
	4	4	SRC	Fixed	Source processor identifier
	3	3	DSZ	Fixed	Data size (see Table 13-26)
	1	32	F-ADDR	Variable	Full access address (leading zeros truncated)
	1	64	DATA	Variable	Data write values (see Section 13.2.14.6, "Data Trace," for details)
Data Trace - Data Read Message w/ Sync	6	6	TCODE	Fixed	TCODE number = 14 (0x0E)
	4	4	SRC	Fixed	Source processor identifier
	3	3	DSZ	Fixed	Data size (see Table 13-26)
	1	32	F-ADDR	Variable	Full access address (leading zeros truncated)
	1	64	DATA	Variable	Data read values (see Section 13.2.14.6, "Data Trace," for details)
Watchpoint Message	6	6	TCODE	Fixed	TCODE number = 15 (0x0F)
	4	4	SRC	Fixed	Source processor identifier
	4	4	WPHIT	Fixed	Number indicating watchpoint sources
Resource Full Message	6	6	TCODE	Fixed	TCODE number = 27 (0x1B)
	4	4	SRC	Fixed	Source processor identifier
	4	4	RCODE	Fixed	Resource code indicates which resource is the cause of this message (See RCODE values in Table 13-25)
	1	32	RDATA	Variable	Branch / predicate instruction history (see Section 13.2.13.1, "Branch Trace Messaging (BTM)")

Table 13-22. Public TCODEs Supported by NZ7C3 (continued)

Message Name	Packet Size (bits)		Packet Name	Packet Type	Packet Description
	Min	Max			
Program Trace - Indirect Branch History Message	6	6	TCODE	Fixed	TCODE number = 28 (0x1C) (see footnote 1 below)
	4	4	SRC	Fixed	Source processor identifier
	1	8	I-CNT	Variable	Number of sequential instructions executed since last taken branch
	1	32	U-ADDR	Variable	Unique part of target address for taken branches/exceptions
	1	32	HIST	Variable	Branch / predicate instruction history (see Section 13.2.13.1, “Branch Trace Messaging (BTM)”)
Program Trace - Indirect Branch History Message w/ Sync	6	6	TCODE	Fixed	TCODE number = 29 (0x1D) (see footnote 1 below)
	4	4	SRC	Fixed	Source processor identifier
	1	8	I-CNT	Variable	Number of sequential instructions executed since last taken branch
	1	32	F-ADDR	Variable	Full target address (leading zero (0) truncated)
	1	32	HIST	Variable	Branch / predicate instruction history (see Section 13.2.13.1, “Branch Trace Messaging (BTM)”)
Program Trace - Program Correlation Message	6	6	TCODE	Fixed	TCODE number = 33 (0x21)
	4	4	SRC	Fixed	Source processor identifier
	4	4	EVCODE	Fixed	Event correlated w/ program flow (see Table 13-25)
	1	8	I-CNT	Variable	Number of sequential instructions executed since last taken branch
	1	32	HIST	Variable	Branch / predicate instruction history (see Section 13.2.13.1, “Branch Trace Messaging (BTM)”)

[1] You can select between the two types of program trace. The advantages for each are discussed in Section 13.2.13.1, “Branch Trace Messaging (BTM). If the branch history method is selected, the shaded TCODES above are not messaged out.

Table 13-23 shows the error code encodings used when reporting an error via the Nexus3 Error Message.

Table 13-23. Error Code Encoding (TCODE = 8)

Error Code (ECODE)	Description
00000	Ownership trace overrun
00001	Program trace overrun
00010	Data trace overrun
00011	Read/write access error

Table 13-23. Error Code Encoding (TCODE = 8) (continued)

Error Code (ECODE)	Description
00101	Invalid access opcode (Nexus register unimplemented)
00110	Watchpoint overrun
00111	(Program trace or data trace) and ownership trace overrun
01000	(Program trace or data trace or ownership trace) and watchpoint overrun
01001–10111	Invalid value
11000	BTM lost due to collision w/ higher priority message
11001–11111	Invalid value

Table 13-24 shows the encodings used for resource codes for certain messages.

Table 13-24. RCODE values (TCODE = 27)

Resource Code (RCODE)	Description	Resource Data (RDATA)
0000	Program Trace Instruction Counter overflow (reached 255 and was reset)	0xFF
0001	Program Trace, Branch and Predicate Instruction History. This type of packet is terminated by a stop bit set to 1 after the last history bit.	Branch History. This type of packet is terminated by a stop bit set to a 1 after the last history bit.

Table 13-25 shows the event code encodings used for certain messages.

Table 13-25. Event Code Encoding (TCODE = 33)

Event Code	Description
0000	Entry into Debug Mode
0001	Entry into Low Power Mode (CPU only)[1]
0010–0011	Invalid value. Reserved for future functionality
0100	Disabling Program Trace

[1] The device enters Low Power Mode when the Nexus stall mode is enabled (NZ7C3_DC1[OVC]=0b011) and a trace message is in danger of over-flowing the Nexus queue.

Table 13-26 shows the data trace size encodings used for certain messages.

Table 13-26. Data Trace Size Encodings (TCODE = 5, 6, 13, 14)

DTM Size Encoding	Transfer Size
000	Byte
001	Halfword (two bytes)
010	Word (four bytes)

Table 13-26. Data Trace Size Encodings (TCODE = 5, 6, 13, 14) (continued)

DTM Size Encoding	Transfer Size
011	Doubleword (eight bytes)
100	String (three bytes)
101–111	Invalid value

NOTE

Program trace can be implemented using either branch history/predicate instruction messages, or traditional direct and indirect branch messages. The user can select between the two types of Program Trace. The advantages for each are discussed in Section 13.2.13.1, “Branch Trace Messaging (BTM).” If the Branch History method is selected, the shaded TCODES are not messaged out.

13.2.11 NZ7C3 Memory Map and Register Definition

Refer to the Nexus 3 Programmer’s Model section in the *e200z7 PowerPC*$^{\text{TM}}$ *Core Reference Manual* for the description of the NZ7C3 programmer’s model. NZ7C3 registers are accessed using the JTAG/OnCE port in compliance with IEEE 1149.1.

NOTE

Nexus 3 registers and output signals are numbered using bit 0 as the least significant bit. This bit ordering is consistent with the ordering defined by the IEEE-ISTO 5001 standard.

Table 13-27. NZ7C3 Memory Map

Nexus Register	Nexus Access Opcode	Read/Write	Read Address	Write Address
Client Select Control (CSC)[1]	0x1	R	0x02	—
Port Configuration Register (PCR)[1]	PCR_INDEX2	R/W	—	—
Development Control 1 (DC1)	0x2	R/W	0x04	0x05
Development Control 2 (DC2)	0x3	R/W	0x06	0x07
Development Control 3 (DC3)	0x4	R/W	0x08	0x09
Development Control 4 (DC4)	0x5	R/W	0x0A	0x0B
Read/Write Access Control/Status (RWCS)	0x7	R/W	0x0E	0x0F
Read/Write Access Address (RWA)	0x9	R/W	0x12	0x13
Read/Write Access Data (RWD)	0xA	R/W	0x14	0x15
Watchpoint Trigger (WT)	0xB	R/W	0x16	0x17
Reserved	0xC	R/W	0x18	0x19

Table 13-27. NZ7C3 Memory Map (continued)

Nexus Register	Nexus Access Opcode	Read/Write	Read Address	Write Address
Data Trace Control (DTC)	0xD	R/W	0x1A	0x1B
Data Trace Start Address 1 (DTSA1)	0xE	R/W	0x1C	0x1D
Data Trace Start Address 2 (DTSA2)	0xF	R/W	0x1E	0x1F
Data Trace Start Address 3 (DTSA3)	0x10	R/W	0x20	0x21
Data Trace Start Address 4 (DTSA4)	0x11	R/W	0x22	0x23
Data Trace End Address 1 (DTEA1)	0x12	R/W	0x24	0x25
Data Trace End Address 2 (DTEA2)	0x13	R/W	0x26	0x27
Data Trace End Address 3 (DTEA3)	0x14	R/W	0x28	0x29
Data Trace End Address 4 (DTEA4)	0x15	R/W	0x2A	0x2B
Reserved	0x16 – 0x2F	—	0x28 – 0x5E	0x29 – 0x5F
Development Status (DS)	0x30	R	0x60	—
Reserved	0x31	R/W	0x62	0x63
Overrun Control (OVCR)	0x32	R/W	0x64	0x65
Watchpoint Mask (WMSK)	0x33	R/W	0x66	0x67
Reserved	0x34	—	0x68	0x69
Program Trace Start Trigger Control (PTSTC)	0x35	R/W	0x6A	0x6B
Program Trace End Trigger Control (PTETC)	0x36	R/W	0x6C	0x6D
Data Trace Start Trigger Control (DTSTC)	0x37	R/W	0x6E	0x6F
Data Trace End Trigger Control (DTETC)	0x38	R/W	0x70	0x71
Reserved	0x39 – 0x3F	—	0x72 – 0x7E	0x73 – 0x7F

[1] The CSC and PCR registers are shown in this table as part of the Nexus programmer's model. They are only present at the top level Nexus controller in a multi-Nexus implementation, not in the Nexus 3 module. The CSC Register is readable through Nexus, but the PCR is shown for reference only here.

13.2.11.1 NZ7C3 Register Access via JTAG / OnCE

Access to Nexus3 register resources is enabled by loading a single instruction (ACCESS_AUX_TAP_ONCE) into the JTAGC instruction register (IR), and then loading the corresponding OnCE OCMD register with the NEXUS3_ACCESS instruction (see Table 13-8). For the NZ7C3 module, the OCMD value is 0b00_0111_1100.

After the ACCESS_AUX_TAP_ONCE instruction has been loaded, the JTAG/OnCE port allows tool/target communications with all Nexus3 registers according to the register map in Table 13-27.

Reading/writing of a NZ7C3 register then requires two (2) passes through the data-scan (DR) path of the JTAG state machine (see 13.2.14.10).

1. The first pass through the DR selects the NZ7C3 register to be accessed by providing an index (see Table 13-27), and the direction (read/write). This is achieved by loading an 8-bit value into the JTAG data register (DR). This register has the following format:

(7-bits)	(1-bit)
Nexus register index	R/W

RESET Value: 0x00

Nexus Register Index:	Selected from values in Table 13-27
Read/Write (R/W):	0 Read 1 Write

2. The second pass through the DR then shifts the data in or out of the JTAG port, lsb first.
   a) During a read access, data is latched from the selected Nexus register when the JTAG state machine passes through the capture-DR state.
   b) During a write access, data is latched into the selected Nexus register when the JTAG state machine passes through the update-DR state.

13.2.12 Ownership Trace

This section details the ownership trace features of the NZ7C3 module.

Ownership trace provides a macroscopic view, such as task flow reconstruction, when debugging software written in a high level (or object-oriented) language. It offers the highest level of abstraction for tracking operating system software execution. This is especially useful when the developer is not interested in debugging at lower levels.

13.2.12.1 Ownership Trace Messaging (OTM)

Ownership trace information is messaged via the auxiliary port using an ownership trace message (OTM). The e200z7 processor contains a Power Architecture Book E defined process ID register within the CPU.

The process ID register is updated by the operating system software to provide task/process ID information. The contents of this register are replicated on the pins of the processor and connected to Nexus. The process ID register value can be accessed using the **mfspr**/**mtspr** instructions. Please see the *e200z7 PowerPC*^{TM} *Core Reference Manual* for more details on the process ID register.

The only condition that causes an ownership trace message occurs when the OTR register is updated or process ID register by the e200z7 processor, the data is latched within Nexus, and is messaged out via the auxiliary port, allowing development tools to trace ownership flow.

Ownership trace information is messaged out in the following format:

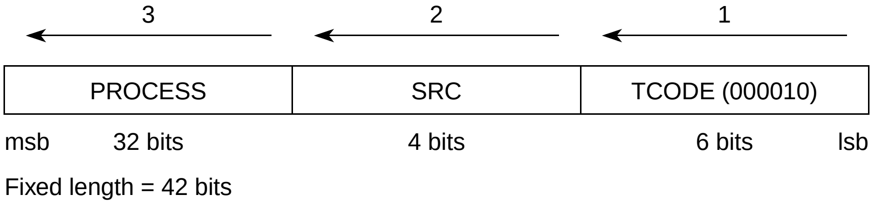


Figure 13-19. Ownership Trace Message Format

13.2.12.2 OTM Error Messages

An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards incoming messages until the queue the queue is completely empty. After it empties, an error message is queued. The error encoding indicate the message types denied service to the queue while the FIFO was emptying.

If an OTM message only attempts to enter the queue while the queue is emptying, the error message incorporates the OTM error encoding (00000) only. If OTM and either BTM or DTM messages attempt to enter the queue, the error message incorporates the OTM and (program or data) trace error encoding (00111). If a watchpoint also attempts to enter the queue while the FIFO is emptying, then the error message incorporates error encoding (01000).

NOTE

The OVC bits within the DC1 register can be set to delay the CPU to alleviate (but not eliminate) potential overrun situations.

Error information is messaged out in the following format (see Table 13-23):

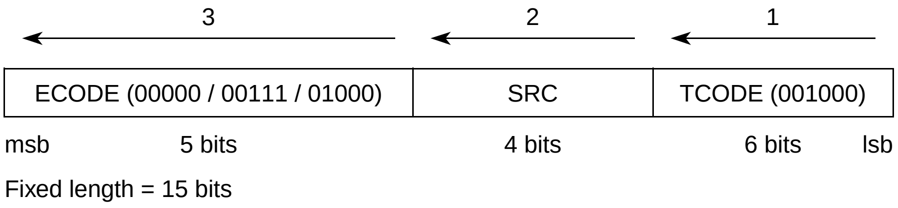


Figure 13-20. Error Message Format

13.2.12.3 OTM Flow

Ownership trace messages are generated when the operating system writes to the e200z7 process ID register or the memory mapped ownership trace register.

The following flow describes the OTM process:

1. The process ID register is a system control register. It is internal to the e200z7 processor and can be accessed by using PPC instructions **mtspr** and **mfspr**. The contents of this register are replicated on the pins of the processor and connected to Nexus.
2. OTR/process ID register reads do not cause ownership trace messages to be transmitted by the NZ7C3 module.
3. If the periodic OTM message counter expires (after 255 queued messages without an OTM), an OTM is sent using the latched data from the previous OTM or process ID register write.

13.2.13 Program Trace

This section details the program trace mechanism supported by NZ7C3 for the MPC5676R processor. Program trace is implemented via branch trace messaging (BTM) as per the Class 3 IEEE-ISTO 5001-2003 standard definition. Branch trace messaging for e200z7 processors is accomplished by snooping the e200z7 virtual address bus (between the CPU and MMU), attribute signals, and CPU status.

13.2.13.1 Branch Trace Messaging (BTM)

Traditional branch trace messaging facilitates program trace by providing the following types of information:

- Messages generated for direct branches that were taken indicate the number of sequential instructions executed since the last branch or exception. Branches not taken (direct or indirect) are not counted as sequential instructions.
- Messages generated for indirect branches and exceptions that were taken indicate:
  - Number of sequential instructions executed since the last branch that was taken
  - Exception with the unique portion of the branch target address or exception vector address
- History field in the branch and predicate instructions that can generate the following messages for program trace:
  - Number of sequential instructions executed since the last indirect branch was taken, as well as the unique portion of the indirect branch address

  - Number of sequential instructions executed since the last exception was processed, as well as the unique portion of the exception vector address
  - Number of sequential instructions executed since the last predicate instruction was taken
  - History field in the branch and predicate instruction unique to the branch target address or exception vector address. Each bit in the history field represents a direct branch or predicated instruction where a value of one (1) indicates taken, and a value of zero (0) indicates not taken. Certain instructions (**evsel**) generate a pair of predicate bits which are both reported as consecutive bits in the history field.

13.2.13.1.1 e200z7 Indirect Branch Message Instructions (Power Architecture Book E)

Table 13-28 shows the types of instructions and events which cause indirect branch messages or branch history messages to be encoded.

Table 13-28. Indirect Branch Message Sources

Source of Indirect Branch Message	Instructions
Taken branch relative to a register value	bcctr, bcctrl, bclr, bclrl
System call / trap exceptions taken	sc, tw, twi
Return from interrupts / exceptions	rfi, rfci, rfdi

13.2.13.1.2 e200z7 Direct Branch Message Instructions (Power Architecture Book E)

Table 13-29 shows the instruction types that cause direct branch messages, or toggle a bit in the instruction history buffer in a resource full message or branch history message before it is sent out.

Table 13-29. Direct Branch Message Sources

Source of Direct Branch Message	Instructions
Taken direct branch instructions	b, ba, bl, bla, bc, bca, bcl, bcla
Instruction synchronize	isync

13.2.13.1.3 BTM Using Branch History Messages

Traditional BTM messaging can accurately track the number of sequential instructions between branches, but cannot accurately indicate which instructions were conditionally executed, and which were not.

Branch history messaging solves this problem by providing a predicated instruction history field in each indirect branch message. Each bit in the history represents a predicated instruction or direct branch. A value of one (1) indicates the conditional instruction was executed or the direct branch was taken. A value of zero (0) indicates the conditional instruction was not executed or the direct branch was not taken. Certain instructions (**evsel**) generate a pair of predicate bits which are both reported as consecutive bits in the history field.

Branch history messages solve predicated instruction tracking and save bandwidth since only indirect branches cause messages to be queued.

13.2.13.1.4 BTM Using Traditional Program Trace Messages

Based on the PTM bit in the DC register (DC[PTM]), program tracing can use:

- Branch history messages (DC[PTM] = 1); or
- Traditional direct and indirect branch messages (DC[PTM] = 0)

Branch history saves bandwidth and keeps consistency between methods of program trace, yet can lose temporal order between BTM messages and other types of messages. Since direct branches are not messaged, but are instead included in the history field of the indirect branch history message, other types of messages can enter the FIFO between branch history messages. The development tool cannot determine the order of “events” that occurred for direct branches by the order in which messages are sent out.

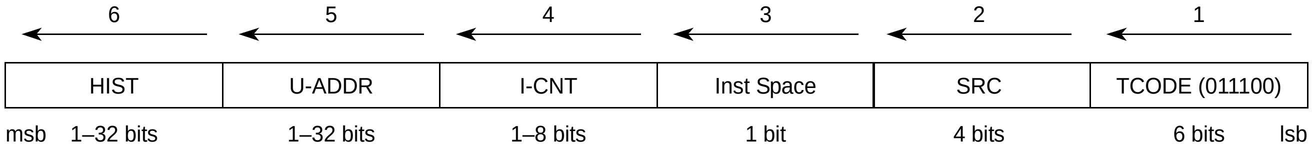
Traditional BTM messages maintain their temporal ordering because each event that queues a message to the FIFO is processed and sent in the order it was generated, and the message order is maintained when it is transmitted.

13.2.13.2 BTM Message Formats

The e200z7 Nexus3 module supports three types of traditional BTM messages—direct, indirect, and synchronization messages. It supports two types of branch history BTM messages—indirect branch history, and indirect branch history with synchronization messages. Debug status messages and error messages are also supported.

13.2.13.2.1 Indirect Branch Messages (History)

Indirect branches include all taken branches whose destination is determined at run time, interrupts and exceptions. If DC[PTM] is set, indirect branch information is messaged out in the following format:



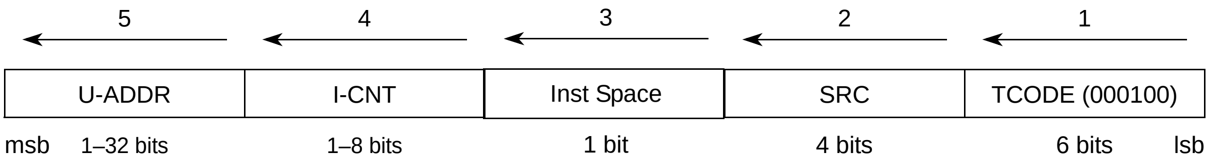
6	5	4	3	2	1
HIST	U-ADDR	I-CNT	Inst Space	SRC	TCODE (011100)
msb 1–32 bits	1–32 bits	1–8 bits	1 bit	4 bits	6 bits lsb

Max length = 83 bits; Min length = 14 bits

Figure 13-21. Indirect Branch Message (History) Format

13.2.13.2.2 Indirect Branch Messages (Traditional)

If DC[PTM] is cleared, indirect branch information is messaged out in the following format:



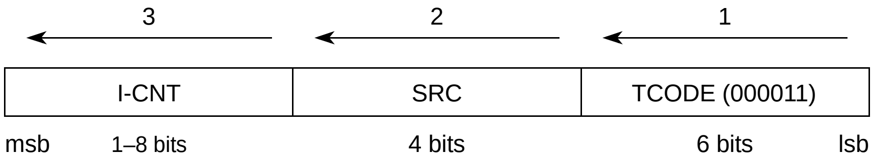
5	4	3	2	1
U-ADDR	I-CNT	Inst Space	SRC	TCODE (000100)
msb 1–32 bits	1–8 bits	1 bit	4 bits	6 bits lsb

Max length = 51 bits; Min length = 13 bits

Figure 13-22. Indirect Branch Message Format

13.2.13.2.3 Direct Branch Messages (Traditional)

Direct branches (conditional or unconditional) are all taken branches whose destination is fixed in the instruction opcode. Direct branch information is messaged out in the following format:



Max length = 18 bits; Min length = 11 bits

Figure 13-23. Direct Branch Message Format

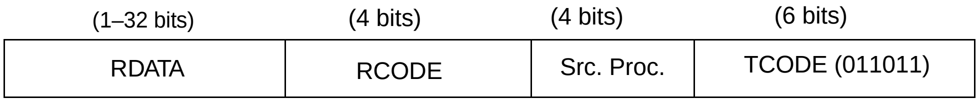
NOTE

> When DC[PTM] is set, direct branch messages are not transmitted. Instead, each direct branch or predicated instruction toggles a bit in the history buffer.

13.2.13.2.4 Resource Full Messages

The resource full message is used in conjunction with the branch history messages. The resource full message is generated when the internal branch/predicate history buffer is full, or if the BTM Instruction sequence counter (I-CNT) overflows. If synchronization is needed at the time this message is generated, the synchronization is delayed until the next branch trace message that is not a resource full message.

The current value of the history buffer is transmitted as part of the resource full message. This information can be concatenated by the tool with the branch/predicate history information from subsequent messages to obtain the complete branch history for a message. The internal history value is reset by this message, and the I-CNT value is reset as a result of a bit being added to the history buffer.



Max length = 46 bits; Min length = 15 bits

Figure 13-24. Resource Full Message Format

13.2.13.2.5 Debug Status Messages

NOTE

Debug Status Messages (DSMs) are enabled if the Nexus module is enabled.

Debug status messages report low power mode and debug status. Entering/exiting debug mode as well as entering a low power mode triggers a debug status message.

Debug status information is sent out in the following format:

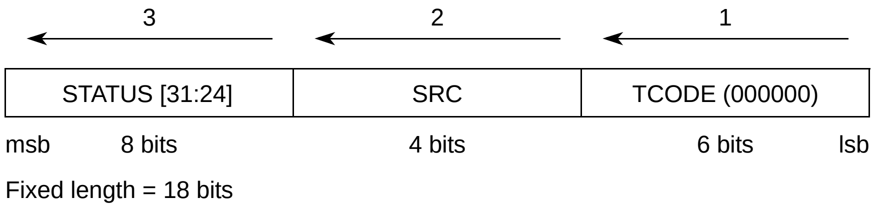


Figure 13-25. Debug Status Message Format

13.2.13.2.6 Program Correlation Messages

Program correlation messages are used to correlate events to the program flow that are not necessarily associated with the instruction stream. To maintain accurate instruction tracing information when entering debug mode or a CPU low power mode (where tracing can be disabled), this message is sent upon entry into one of these two modes and includes the instruction count and branch history. Program correlation is messaged out in the following format:

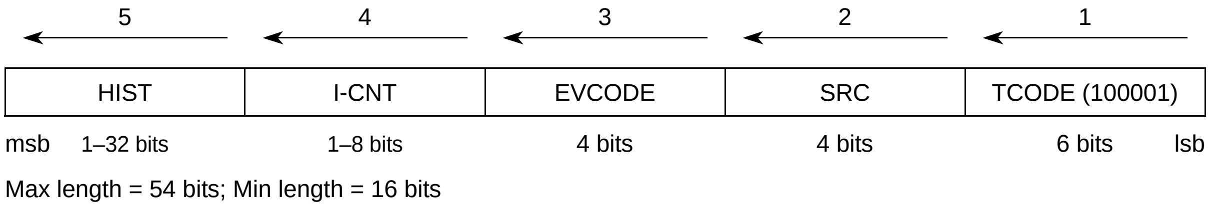


Figure 13-26. Program Correlation Message Format

13.2.13.2.7 BTM Overflow Error Messages

An error message occurs when a new message cannot be queued because the message queue is full. The FIFO discards incoming messages until the queue is completely empty. After it is empty, an error message is queued. The error encoding indicates which message types were denied queueing while the FIFO was emptying.

If only a program trace message attempts to enter the queue while it is being emptied, the error message incorporates the program trace only error encoding (00001). If both OTM and program trace messages attempt to enter the queue, the error message incorporates the OTM and program trace error encoding (00111). If a watchpoint also attempts to be queued while the FIFO is being emptied, then the error message incorporates error encoding (01000).

NOTE

The OVC bits within the DC1 register can be set to delay the CPU to alleviate (but not eliminate) potential overrun situations.

Error information is messaged out in the following format:

	3	2	1	
	ECODE (00001 / 00111 / 01000)	SRC	TCODE (001000)	
msb	5 bits	4 bits	6 bits	lsb

Fixed length = 15 bits

Figure 13-27. Error Message Format

13.2.13.3 Program Trace Synchronization Messages

A program trace direct/indirect branch with sync message is messaged via the auxiliary port (provided program trace is enabled) for the following conditions (see Table 13-30):

- Initial program trace message upon the first direct/indirect branch after exit from system reset or whenever program trace is enabled
- Upon direct/indirect branch after returning from a CPU low power state
- Upon direct/indirect branch after returning from debug mode
- Upon direct/indirect branch after occurrence of queue overrun (can be caused by any trace message), provided program trace is enabled
- Upon direct/indirect branch after the periodic program trace counter has expired indicating 255 *without-sync* program trace messages have occurred since the last *with-sync* message occurred
- Upon direct/indirect branch after assertion of the event in (EVTI) pin if the EIC bits within the DC1 register have enabled this feature
- Upon direct/indirect branch after the sequential instruction counter has expired indicating 255 instructions have occurred between branches
- Upon direct/indirect branch after a BTM message was lost due to an attempted access to a secure memory location.
- Upon direct/indirect branch after a BTM message was lost due to a collision entering the FIFO between the BTM message and either a watchpoint message or an ownership trace message

If the NZ7C3 module is enabled at reset, a EVTI assertion initiates a program trace direct/indirect branch with sync message (if program trace is enabled) upon the first direct/indirect branch. The format for program trace direct/indirect branch with sync messages is as follows:

	5	4	3	2	1	
	F-ADDR	I-CNT	Inst Space	SRC	TCODE (001011 or 001100)	
msb	1–32 bits	1–8 bits	1 bit	4 bits	6 bits	lsb

Max length = 51 bits; Min length = 13 bits

Figure 13-28. Direct/Indirect Branch with Sync Message Format

The formats for program trace direct/indirect branch with sync. messages and indirect branch history with sync. messages are as follows:

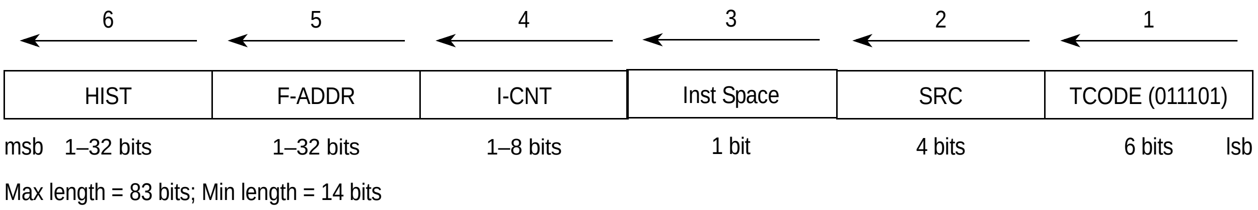


Figure 13-29. Indirect Branch History with Sync. Message Format

Exception conditions that result in program trace synchronization are summarized in Table 13-30.

Table 13-30. Program Trace Exception Summary

Exception Condition	Exception Handling
System Reset Negation	At the negation of JTAG reset (JCOMP), queue pointers, counters, state machines, and registers within the NZ7C3 module are reset. Upon the first branch out of system reset (if program trace is enabled), the first program trace message is a direct/indirect branch with sync. message.
Program Trace Enabled	The first program trace message (after program trace has been enabled) is a synchronization message.
Exit from Low Power/Debug	Upon exiting from the low power or debug modes, the next direct/indirect branch is converted to a direct/indirect branch with sync. message.
Queue Overrun	An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards messages until the queue is completely empty. After it is empty, an error message is queued. The error encoding indicates the message types denied queueing while the FIFO was emptying. The next BTM message in the queue is a direct/indirect branch with sync. message.
Periodic Program Trace Sync.	A forced synchronization occurs periodically after 255 program trace messages have been queued. A direct/indirect branch with sync. message is queued. The periodic program trace message counter then resets.
Event In	If the Nexus module is enabled, an EVTI assertion initiates a direct/indirect branch with sync. message upon the next direct/indirect branch (if program trace is enabled and the EIC bits of the DC1 register have enabled this feature).
Sequential Instruction Count Overflow	When the sequential instruction counter reaches its maximum count (up to 255 sequential instructions can be executed), a forced synchronization occurs. The sequential counter then resets. A program trace direct/indirect branch with sync.message is queued upon execution of the next branch.
Attempted Access to Secure Memory	For devices which implement security, any attempt to branch to secure memory locations temporarily disables program trace and cause the corresponding BTM to be lost. The following direct/indirect branch queues a direct/indirect branch with sync. message. The count value within this message can be inaccurate since re-enabling program trace does not guarantee alignment on an instruction boundary.
Collision Priority	All messages have the following priority: WPM → OTM → BTM → DTM. A BTM message which attempts to enter the queue at the same time as a watchpoint message or ownership trace message is lost. An error message is sent indicating the BTM was lost. The following direct/indirect branch queues a direct/indirect branch with sync. message. The count value within this message reflects the number of sequential instructions executed after the last successful BTM message was generated. This count includes the branch which did not generate a message due to the collision.

13.2.14 BTM Operation

13.2.14.1 Enabling Program Trace

Both types of branch trace messaging can be enabled in one of two ways:

- Setting the TM field of the DC1 register to enable program trace (DC1[TM])
- Using the PTS field of the WT register to enable program trace on watchpoint hits (e200z7 watchpoints are configured within the CPU)

13.2.14.2 Relative Addressing

The relative address feature is compliant with the IEEE-ISTO 5001-2003 standard recommendations, and is designed to reduce the number of bits transmitted for addresses of indirect branch messages.

The address transmitted is relative to the target address of the instruction which triggered the previous indirect branch (or sync) message. It is generated by XOR'ing the new address with the previous address, and then using only the results up to the most significant 1 in the result. To recreate this address, an XOR of the (most-significant 0-padded) message address with the previously decoded address gives the current address.

```
Previous address (A1) =0x0003FC01, New address (A2) = 0x0003F365
```

Message Generation:

A1 = 0000 0000 0000 0011 1111 1100 0000 0001
A2 = 0000 0000 0000 0011 1111 0011 0110 0101

A1 ⊕ A2 = 0000 0000 0000 0000 0000 1111 0110 0100

Address Message (M1) = 1111 0110 0100

Address Re-creation:

A1 ⊕ M1 = A2
A1 = 0000 0000 0000 0011 1111 1100 0000 0001
M1 = 0000 0000 0000 0000 0000 1111 0110 0100

A2 = 0000 0000 0000 0011 1111 0011 0110 0101

Figure 13-30. Relative Address Generation and Re-creation

13.2.14.3 Branch and Predicate Instruction History (HIST)

If DC[PTM] is set, BTM messaging uses the branch history format. The branch history (HIST) packet in these messages provides a history of direct branch execution used for reconstructing the program flow. This packet is implemented as a left-shifting shift register. The register is always pre-loaded with a value of one (1). This bit acts as a stop bit so that the development tools can determine which bit is the end of the history information. The pre-loaded bit itself is not part of the history, but is transmitted with the packet.

A value of one (1) is shifted into the history buffer on a taken branch (condition or unconditional) and on any instruction whose predicate condition executed as true. A value of zero (0) is shifted into the history buffer on any instruction whose predicate condition executed as false as well as on branches not taken. This includes indirect as well as direct branches were not taken. For the **evsel** instruction, two bits are shifted in, corresponding to the low element (shifted in first) and the high element (shifted in second) conditions.

13.2.14.4 Sequential Instruction Count (I-CNT)

The I-CNT packet, is present in all BTM messages. For traditional branch messages, I-CNT represents the number of sequential instructions, or non-taken branches in between direct/indirect branch messages.

For branch history messages, I-CNT represents the number of instructions executed since the last taken/non-taken direct branch, last taken indirect branch or exception. Not taken indirect branches are considered sequential instructions and cause the instruction count to increment. I-CNT also represents the number of instructions executed since the last predicate instruction.

The sequential instruction counter overflows when its value reaches 255. The next BTM message is converted to a synchronization type message.

13.2.14.5 Program Trace Queueing

NZ7C3 implements a message queue. Messages that enter the queue are transmitted via the auxiliary pins in the order in which they are queued.

NOTE

> If multiple trace messages must be queued at the same time, Watchpoint Messages have the highest priority (WPM → OTM → BTM → DTM).

13.2.14.5.1 Program Trace Timing Diagrams

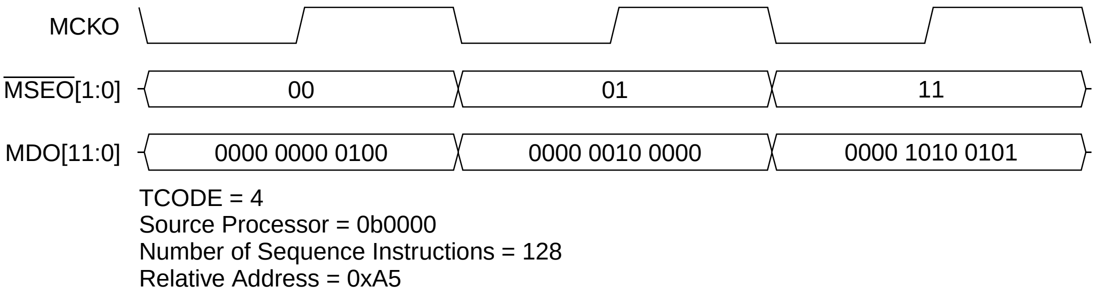
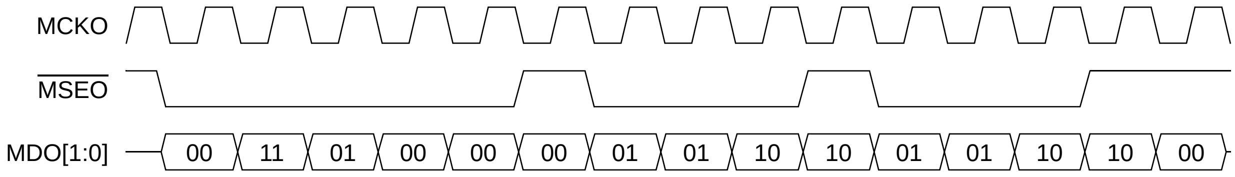
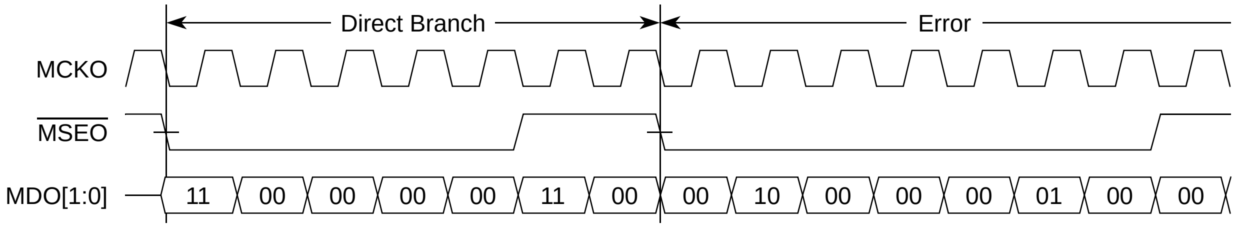


Figure 13-31. Program Trace (MDO = 12)—Indirect Branch Message (Traditional)



TCODE = 28
Source Processor = 0b0000
Number of Sequential Instructions = 0
Relative Address = 0xA5
Branch History = 0b1010_0101 (with Stop)

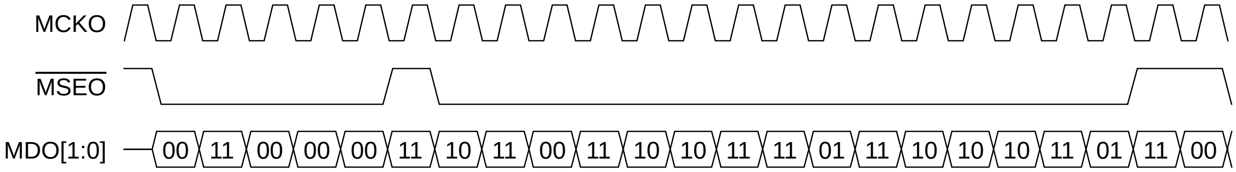
Figure 13-32. Program Trace (MDO = 2)—Indirect Branch Message (History)



DBM:
TCODE = 3
Source Processor = 0b0000
Number of Sequential Instructions = 3

Error:
TCODE = 8
Source Processor = 0b0000
Error Code = 1 (Queue Overrun ,ÄöÐÑÐ¹ÐêÐë

Figure 13-33. Program Trace—Direct Branch (Traditional) and Error Messages



TCODE = 12
Source Processor = 0b0000
Number of Sequential Instructions = 3
Full Target Address = 0xDEAD_FACE

Figure 13-34. Program Trace—Indirect Branch with Sync. Message

13.2.14.6 Data Trace

This section deals with the data trace mechanism supported by the NZ7C3 module. Data trace is implemented via data write messaging (DWM) and data read messaging (DRM), as per the IEEE-ISTO 5001-2003 standard.

13.2.14.6.1 Data Trace Messaging (DTM)

Data trace messaging for e200z7 is accomplished by snooping the e200z7 virtual data bus (between the CPU and MMU), and storing the information for qualifying accesses (based on enabled features and matching target addresses). The NZ7C3 module traces all data access that meet the selected range and attributes.

NOTE

> Data trace is only performed on the e200z7 virtual data bus. This allows for data visibility for the incorporated data cache. Only e200z7 CPU initiated accesses are traced. No DMA accesses to the NXDM system bus are traced.

Data trace messaging can be enabled in one of two ways:

- Setting the TM field of the DC1 register to enable data trace (DC1[TM]).
- Using WT[DTS] to enable data trace on watchpoint hits (e200z7 watchpoints are configured within the Nexus1 module)

13.2.14.6.2 DTM Message Formats

The Nexus3 module supports five types of DTM messages: data write, data read, data write synchronization, data read synchronization and error messages.

Data Write Messages

The data write message contains the data write value and the address of the write access, relative to the previous data trace message. Data write message information is messaged out in the following format:

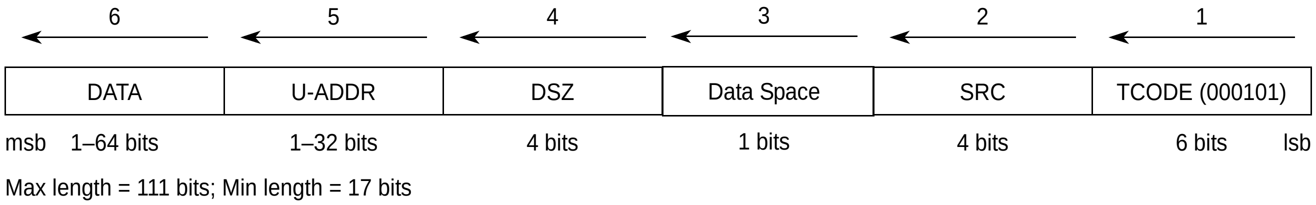


Figure 13-35. Data Write Message Format

Data Read Messages

The data read message contains the data read value and the address of the read access, relative to the previous data trace message. Data read message information is messaged out in the following format:

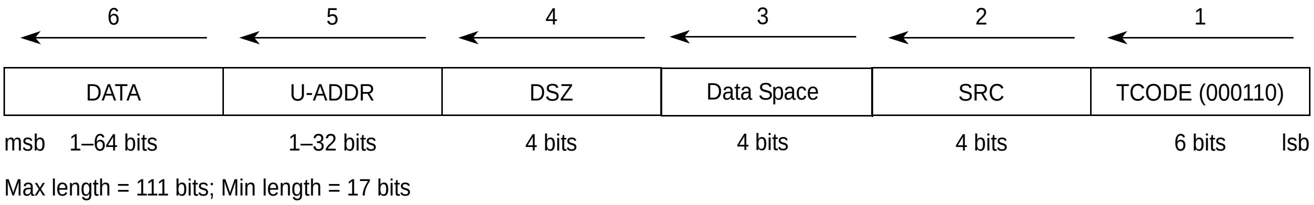


Figure 13-36. Data Read Message Format

NOTE

> For the e200z7 based CPU, the doubleword encoding (data size = 0b000) indicates a doubleword access and sends out as a single data trace message with a single 64-bit data value.

DTM Overflow Error Messages

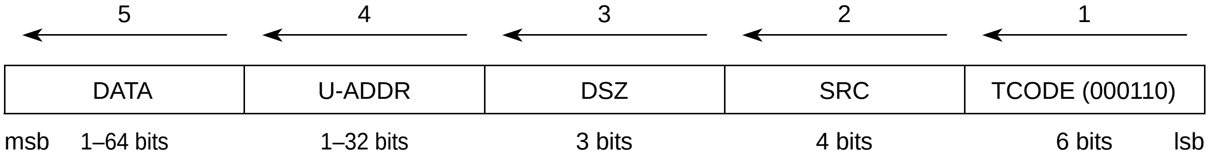
An error message occurs when the next message is denied service because the message queue is full. The FIFO discards all incoming messages until the queue is completely empty. After it is empty, an error message is queued that indicates the message types denied into the queue while the FIFO is emptying.

If a data trace message only attempts to enter the queue while it is emptying, the error message incorporates the data trace only error encoding (00010). If both OTM and data trace messages attempt to enter the queue, the error message incorporates the OTM and data trace error encoding (00111). If a watchpoint also attempts to be queued while the FIFO is being emptied, then the error message incorporates error encoding (01000).

NOTE

> The OVC bits within the DC1 register can be set to delay the CPU to alleviate (but not eliminate) potential overrun situations.

Error information is messaged out in the following format:



Max length = 109 bits; Min length = 15 bits

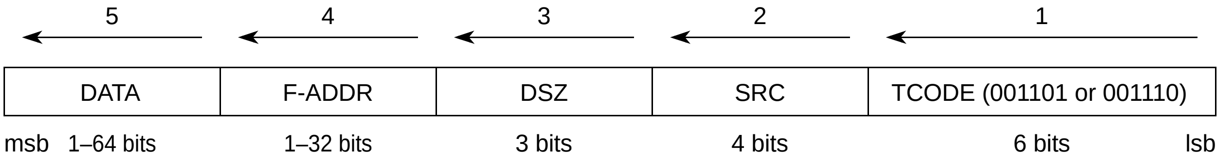
Figure 13-37. Error Message Format

Data Trace Synchronization Messages

A data trace write/read with sync. message is messaged via the auxiliary port (provided data trace is enabled) for the following conditions (see Table 13-31):

- Initial data trace message after exit from system reset or whenever data trace is enabled
- Upon exiting debug mode
- After occurrence of queue overrun (can be caused by any trace message), provided data trace is enabled
- After the periodic data trace counter has expired indicating 255 *without-sync* data trace messages have occurred since the last *with-sync* message occurred
- Upon assertion of the event in (EVTI) pin, the first data trace message is a synchronization message if the EIC bits of the DC1 register have enabled this feature
- Upon data trace write/read after the previous DTM message was lost due to an attempted access to a secure memory location
- Upon data trace write/read after the previous DTM message was lost due to a collision entering the FIFO between the DTM message and any of the following: watchpoint message, ownership trace message, or branch trace message

Data trace synchronization messages provide the full address (without leading zeros) and insure that development tools fully synchronize with data trace regularly. Synchronization messages provide a reference address for subsequent data messages, in which only the unique portion of the data trace address is transmitted. The format for data trace write/read with sync. messages is as follows:



Max length = 109 bits; Min length = 15 bits

Figure 13-38. Data Write/Read with Sync. Message Format

Exception conditions that result in data trace synchronization are summarized in Table 13-31.

Table 13-31. Data Trace Exception Summary

Exception Condition	Exception Handling
System Reset Negation	At the negation of JTAG reset (JCOMP), queue pointers, counters, state machines, and registers within the NZ7C3 module are reset. If data trace is enabled, the first data trace message is a data write/read with sync. message.
Data Trace Enabled	The first data trace message (after data trace has been enabled) is a synchronization message.
Exit from Low Power/Debug	Upon exiting from low power or debug modes, the next data trace message is converted to a data write/read with sync. message.
Queue Overrun	An error message occurs when a new message cannot be queued due to a full message queue. The FIFO discards messages until it has completely emptied the queue. After the queue is empty, an error message is queued that indicates the message types denied queuing while the FIFO was emptying. The next DTM message in the queue is a data write/read with sync. message.
Periodic Data Trace Sync.	A forced synchronization occurs periodically after 255 data trace messages have been queued. A data write/read with sync. message is queued. The periodic data trace message counter then resets.
Event In	If the Nexus module is enabled, a EVTI assertion initiates a data trace write/read with sync. message upon the next data write/read (if data trace is enabled and the EIC bits of the DC1 register have enabled this feature).
Attempted Access to Secure Memory	For devices which implement security, any attempted read or write to secure memory locations temporarily disables data trace and loses the DTM. A subsequent read/write queues a data trace read/write with sync. message.
Collision Priority	All messages have the following priority: WPM –> OTM –> BTM –> DTM. A DTM message which attempts to enter the queue at the same time as a watchpoint message or ownership trace message or branch trace message can be lost. A subsequent read/write queues a data trace read/write with sync. message.

13.2.14.6.3 DTM Operation

DTM Queueing

NZ7C3 implements a message queue for DTM messages. Messages that enter the queue are transmitted via the auxiliary pins in the order in which they are queued.

NOTE

If multiple trace messages must be queued at the same time, watchpoint messages have the highest priority (WPM → OTM → BTM –> DTM).

Relative Addressing

The relative address feature is compliant with the IEEE-ISTO 5001-2003 standard recommendations, and is designed to reduce the number of bits transmitted for addresses of data trace messages. See Section 13.2.14.2, “ Relative Addressing for details.

Data Trace Windowing

Data write/read messages are enabled via the RWT1(2) field in the data trace control register (DTC) for each DTM channel. Data trace windowing is achieved via the address range defined by the DTEA and DTSA registers and by the RC1(2) field in the DTC. All e200z7 initiated read/write accesses which fall inside or outside these address ranges, as programmed, are candidates to be traced.

Data Access/Instruction Access Data Tracing

The Nexus3 module is capable of tracing both instruction access data or data access data. Each trace window can be configured for either type of data trace by setting the DI1(2) field within the data trace control register for each DTM channel.

e200z7 Bus Cycle Special Cases

Table 13-32. e200z7 Bus Cycle Cases

Special Case	Action
e200z7 bus cycle aborted	Cycle ignored
e200z7 bus cycle with data error ($\overline{\text{TEA}}$)	Data Trace Message discarded
e200z7 bus cycle completed without error	Cycle captured & transmitted
e200z7 bus cycle initiated by NZ7C3	Cycle ignored
e200z7 bus cycle is an instruction fetch	Cycle ignored
e200z7 bus cycle accesses misaligned data (across 64-bit boundary)—both 1st and 2nd transactions within data trace range	1st and 2nd cycle captured, and 2 DTM’s transmitted (see Note)

Table 13-32. e200z7 Bus Cycle Cases (continued)

Special Case	Action
e200z7 bus cycle accesses misaligned data (across 64-bit boundary)—1st transaction within data trace range; 2nd transaction out of data trace range	1st cycle captured and transmitted; 2nd cycle ignored
e200z7 bus cycle accesses misaligned data (across 64-bit boundary)—1st transaction out of data trace range; 2nd transaction within data trace range	1st cycle ignored; 2nd cycle capture and transmitted

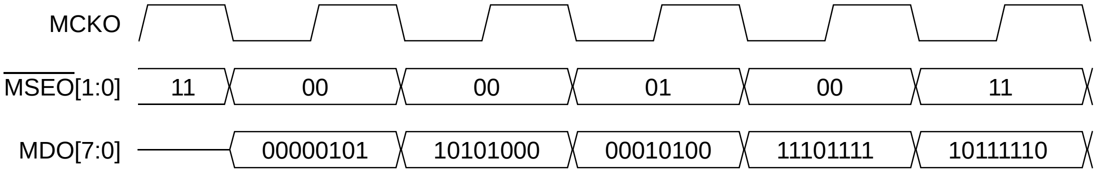
NOTE

For misaligned accesses (crossing 64-bit boundary), the access is broken into two accesses. If both accesses are within the data trace range, two DTMs are sent: one with a size encoding indicating the size of the original access (a word), and one with a size encoding for the portion which crossed the boundary (3 bytes).

NOTE

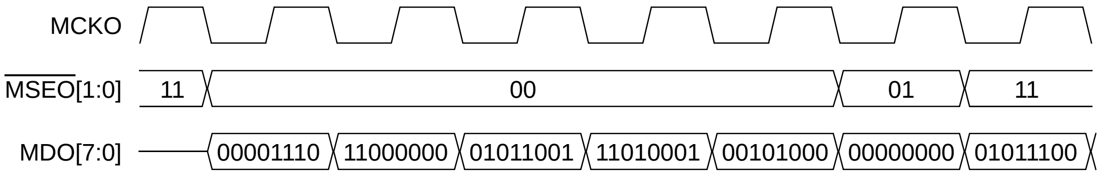
An STM to the cache's store buffer within the data trace range initiates a DTM message. If the corresponding memory access causes an error, a checkstop condition occurs. The debug/development tool must use this indication to invalidate the previous DTM.

13.2.14.6.4 Data Trace Timing Diagrams (Eight MDO Configuration)



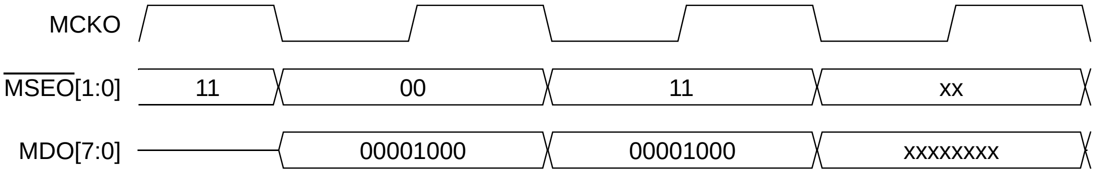
TCODE = 5
Source Processor = 0b0000
Data Size = 010 (halfword)
Relative Address = 0xA5
Write Data = 0xBEEF

Figure 13-39. Data Trace—Data Write Message



TCODE = 14
Source Processor = 0b0000
Data Size = 000 (Byte)
Full Access Address = 0x0146_8ACE
Write Data = 0x5C

Figure 13-40. Data Trace—Data Read with Sync Message



TCODE = 8
Source Processor = 0b0000
Error Code = 2 (Queue Overrun ,ÄöÐÑÐ¹ÐêÐë

Figure 13-41. Error Message (Data Trace only encoded)

13.2.14.7 Watchpoint Support

This section details the watchpoint features of the NZ7C3 module.

13.2.14.7.1 Overview

The NZ7C3 module provides watchpoint messaging via the auxiliary pins, as defined by the IEEE-ISTO 5001-2003 standard.

NZ7C3 is not compliant with Class4 breakpoint/watchpoint requirements defined in the standard. The breakpoint/watchpoint control register is not implemented.

13.2.14.7.2 Watchpoint Messaging

Enabling watchpoint messaging is done by setting the watchpoint enable bit in the DC1 register. Setting the individual watchpoint sources is supported through the e200z7 Nexus1 module. The e200z7 Nexus1 module is capable of setting multiple address and/or data watchpoints. Please see the e200z7 Core Reference Manual for more information on watchpoint initialization.

When these watchpoints occur, a watchpoint event signal from the Nexus1 module causes a message to be sent to the queue to be messaged out. This message includes the watchpoint number indicating which watchpoint caused the message.

The occurrence of any of the e200z7 defined watchpoints can be programmed to assert the event out $\overline{\text{EVTO}}$ pin for one period of the output clock (MCKO).

Watchpoint information is messaged out in the following format:

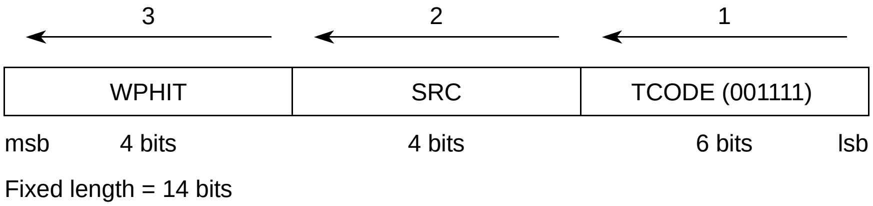


Figure 13-42. Watchpoint Message Format.

Table 13-33. Watchpoint Source Encoding

Watchpoint Source (8 bits)	Watchpoint Description
00000001	e200z7 Watchpoint #0 (IAC1 from Nexus1)
00000010	e200z7 Watchpoint #1 (IAC2 from Nexus1)
00000100	e200z7 Watchpoint #2 (IAC3 from Nexus1)
00001000	e200z7 Watchpoint #3 (IAC4 from Nexus1)
00010000	e200z7 Watchpoint #4 (DAC1 from Nexus1)
00100000	e200z7 Watchpoint #5 (DAC2 from Nexus1)
01000000	e200z7 Watchpoint #6 (DCNT1 from Nexus1)
10000000	e200z7 Watchpoint #7 (DCNT2 from Nexus1)

13.2.14.7.3 Watchpoint Error Message

An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards messages until it has completely emptied the queue. After it is emptied, an error message is queued. The error encoding indicates the types of messages that attempted to be queued while the FIFO was being emptied.

If only a watchpoint message attempts to enter the queue while it is being emptied, the error message incorporates the watchpoint only error encoding (00110). If an OTM and/or program trace and/or data trace message also attempts to enter the queue while it is being emptied, the error message incorporates error encoding (01000).

NOTE

> The OVC bits within the DC1 register can be set to delay the CPU to alleviate (but not eliminate) potential overrun situations.

Error information is messaged out in the following format (see Table 13-23):

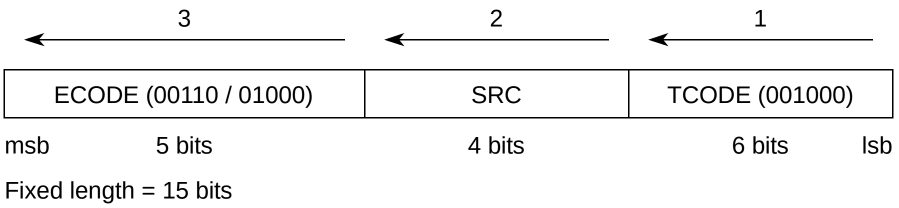


Figure 13-43. Error Message Format

13.2.14.7.4 Watchpoint Timing Diagram (Two MDO and One MSEO Configuration)

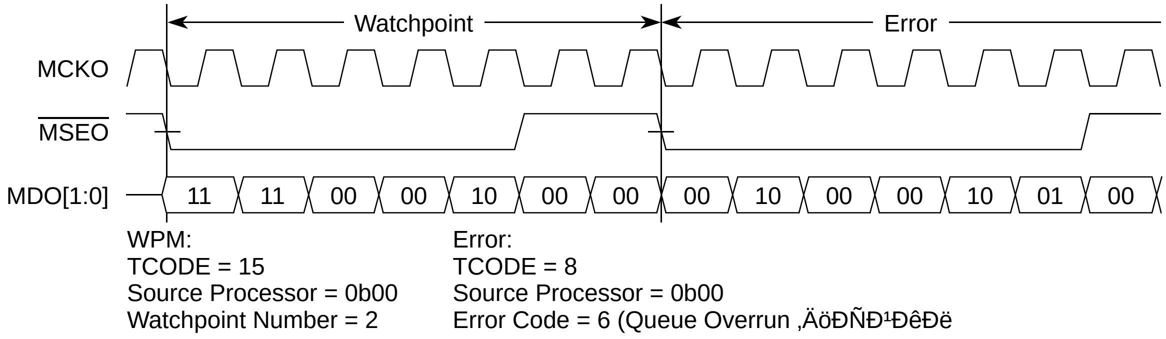


Figure 13-44. Watchpoint Message and Watchpoint Error Message

13.2.14.8 NZ7C3 Read/Write Access to Memory-Mapped Resources

The read/write access feature allows access to memory-mapped resources via the JTAG/OnCE port. The read/write mechanism supports single as well as block reads and writes to e200z7 system bus resources.

The NZ7C3 module is capable of accessing resources on the e200z7 system bus, with multiple configurable priority levels. Memory-mapped registers and other non-cached memory can be accessed via the standard memory map settings.

All accesses are setup and initiated by the read/write access control/status register (RWCS), as well as the read/write access address (RWA) and read/write access data registers (RWD).

Using the read/write access registers (RWCS/RWA/RWD), memory-mapped e200z7 system bus resources can be accessed through NZ7C3. The following subsections describe the steps which are required to access memory-mapped resources.

NOTE

> Read/write access can only access memory mapped resources when system reset is de-asserted.
>
> Misaligned accesses are NOT supported in the e200z7 Nexus3 module.

13.2.14.8.1 Single Write Access

1. Initialize the read/write access address register (RWA) through the access method outlined in Section 13.2.11.1, “ NZ7C3 Register Access via JTAG / OnCE” using the Nexus register index of 0x9 (see Table 13-27). Configure the write address to 0xnnnnnnnn (write address).

2. Initialize the read/write access control/status register (RWCS) through the access method outlined in Section 13.2.11.1, " NZ7C3 Register Access via JTAG / OnCE," using the Nexus Register Index of 0x7 (see Table 13-27). Configure the bits as follows:
   - Access Control RWCS[AC] → 0b1 (to indicate start access)
   - Map Select RWCS[MAP] → 0b000 (primary memory map)
   - Access Priority RWCS[PR] → 0b00 (lowest priority)
   - Read/Write RWCS[RW] → 0b1 (write access)
   - Word Size RWCS[SZ] → 0b0xx (32-bit, 16-bit, 8-bit)
   - Access Count RWCS[CNT] → 0x0000 or 0x0001 (single access)

NOTE

Access count RWCS[CNT] of 0x0000 or 0x0001 performs a single access.

3. Initialize the read/write access data register (RWD) through the access method outlined in Section 13.2.11.1, " NZ7C3 Register Access via JTAG / OnCE," using the Nexus register index of 0xA (see Table 13-27). Configure the write data to 0xnnnnnnnn (write data).
4. The NZ7C3 module then arbitrates for the system bus and transfer the data value from the data buffer RWD register to the memory mapped address in the read/write access address register (RWA). When the access has completed without error (ERR=1'b0), NZ7C3 asserts the RDY pin and clears the DV bit in the RWCS register. This indicates that the device is ready for the next access.

NOTE

Only the RDY pin as well as the DV and ERR bits within the RWCS provide read/write access status to the external development tool.

13.2.14.8.2 Block Write Access (Non-Burst Mode)

1. For a non-burst block write access, follow Steps 1, 2, and 3 outlined in Section 13.2.14.8.1, "Single Write Access to initialize the registers," but using a value greater than one (0x1) for the RWCS[CNT] field.
2. The NZ7C3 module then arbitrates for the system bus and transfer the first data value from the RWD register to the memory mapped address in the read/write access address register (RWA). When the transfer has completed without error (ERR = 0), the address from the RWA register is incremented to the next word size (specified in the SZ field) and the number from the CNT field is decremented. Nexus then asserts the RDY pin. This indicates it is ready for the next access.
3. Repeat step 3 in Section 13.2.14.8.1, "Single Write Access" until the internal CNT value is zero (0). When this occurs, the DV bit within the RWCS is cleared to indicate the end of the block write access.

13.2.14.8.3 Block Write Access (Burst Mode)

1. For a burst block write access, follow Steps 1 and 2 outlined in Section 13.2.14.8.1, "Single Write Access" to initialize the registers, using a value of four (doublewords) for the CNT field and a RWCS[SZ] field indicating 64-bit access.

2. Initialize the burst data buffer (read/write access data register) through the access method outlined in Section 13.2.11.1, “ NZ7C3 Register Access via JTAG / OnCE,” using the Nexus register Index of 0xA (see Table 13-27).
3. Repeat step 2 until all doubleword values are written to the buffer.

NOTE

The data values must be shifted in 32-bits at a time lsb first (that is, doubleword write = two word writes to the RWD).

4. The Nexus module then arbitrates for the system bus and transfer the burst data values from the data buffer to the system bus beginning from the memory mapped address in the read/write access address register (RWA). For each access within the burst, the address from the RWA register is incremented to the next doubleword size (specified in the SZ field) modulo the length of the burst, and the number from the CNT field is decremented.
5. When the entire burst transfer has completed without error (ERR = 0), NZ7C3 then asserts the RDY pin, and the DV bit within the RWCS is cleared to indicate the end of the block write access.

NOTE

The actual RWA value as well as the CNT field within the RWCS are not changed when executing a block write access (burst or non-burst). The original values can be read by the external development tool at any time.

13.2.14.8.4 Single Read Access

1. Initialize the read/write access address register (RWA) through the access method outlined in Section 13.2.11.1, “ NZ7C3 Register Access via JTAG / OnCE,” using the Nexus register index of 0x9 (see Table 13-27). Configure as follows:
    - Read Address → 0xnnnnnnnn (read address)
2. Initialize the read/write access control/status register (RWCS) through the access method outlined in Section 13.2.11.1, “ NZ7C3 Register Access via JTAG / OnCE,” using the Nexus register index of 0x7 (see Table 13-27). Configure the bits as follows:
    - Access Control RWCS[AC]→ 0b1 (to indicate start access)
    - Map Select RWCS[MAP] → 0b000 (primary memory map)
    - Access Priority RWCS[PR] → 0b00 (lowest priority)
    - Read/Write RWCS[RW] → 0b0 (read access)
    - Word Size RWCS[SZ] → 0b0xx (32-bit, 16-bit, 8-bit)
    - Access Count RWCS[CNT]→ 0x0000 or 0x0001 (single access)

NOTE

Access Count (CNT) of 0x0000 or 0x0001 performs a single access.

3. The NZ7C3 module then arbitrates for the system bus and the read data is transferred from the system bus to the RWD register. When the transfer is completed without error (ERR = 0), Nexus asserts the RDY pin and sets the DV bit in the RWCS register. This indicates that the device is ready for the next access.

4. The data can then be read from the read/write access data register (RWD) through the access method outlined in Section 13.2.11.1, “ NZ7C3 Register Access via JTAG / OnCE,” using the Nexus register index of 0xA (see Table 13-27).

NOTE

Only the RDY pin as well as the DV and ERR bits within the RWCS provide Read/Write Access status to the external development tool.

13.2.14.8.5 Block Read Access (Non-Burst Mode)

1. For a non-burst block read access, follow Steps 1 and 2 outlined in Section 13.2.14.8.4, “Single Read Access” to initialize the registers, but using a value greater than one (0x1) for the CNT field in the RWCS register.
2. The NZ7C3 module then arbitrates for the system bus and the read data is transferred from the system bus to the RWD register. When the transfer has completed without error (ERR=0b0), the address from the RWA register is incremented to the next word size (specified in the SZ field) and the number from the CNT field is decremented. Nexus then asserts the RDY pin. This indicates that the device is ready for the next access.
3. The data can then be read from the read/write access data register (RWD) through the access method outlined in Section 13.2.11.1, “ NZ7C3 Register Access via JTAG / OnCE,” using the Nexus register index of 0xA (see Table 13-27).
4. Repeat steps 3 and 4 in Section 13.2.14.8.4, “Single Read Access” until the CNT value is zero (0). When this occurs, the DV bit within the RWCS is set to indicate the end of the block read access.

13.2.14.8.6 Block Read Access (Burst Mode)

1. For a burst block read access, follow Steps 1 and 2 outlined in Section 13.2.14.8.4, “Single Read Access” to initialize the registers, using a value of four (doublewords) for the CNT field and an RWCS[SZ] field indicating 64-bit access.
2. The NZ7C3 module then arbitrates for the system bus and the burst read data is transferred from the system bus to the data buffer (RWD register). For each access within the burst, the address from the RWA register is incremented to the next doubleword (specified in the SZ field) and the number from the CNT field is decremented.
3. When the entire burst transfer has completed without error (ERR = 0), Nexus then asserts the RDY pin and the DV bit within the RWCS is set to indicate the end of the block read access.
4. The data can then be read from the burst data buffer (read/write access data register) through the access method outlined in Section 13.2.11.1, “ NZ7C3 Register Access via JTAG / OnCE,” using the Nexus register index of 0xA (see Table 13-27).
5. Repeat step 3 until all doubleword values are read from the buffer.

NOTE

The data values must be shifted out 32-bits at a time lsb first (that is, doubleword read = two word reads from the RWD).

NOTE

The actual RWA value as well as the CNT field within the RWCS are not changed when executing a block read access (burst or non-burst). The original values can be read by the external development tool at any time.

13.2.14.8.7 Error Handling

The NZ7C3 module handles various error conditions as follows:

System Bus Read/Write Error

All address and data errors that occur on read/write accesses to the e200z7 system bus returns a transfer error. If this occurs:

1. The access is terminated without re-trying (AC bit is cleared).
2. The ERR bit in the RWCS register is set.
3. The error message is sent (TCODE = 8) indicating read/write error.

Access Termination

The following cases are defined for sequences of the read/write protocol that differ from those described in the above sections:

1. If the AC bit in the RWCS register is set to start read/write accesses and invalid values are loaded into the RWD and/or RWA, then a system bus access error can occur. This is handled as described above.
2. If a block access is in progress (all cycles not completed), and the RWCS register is written, then the original block access is terminated at the boundary of the nearest completed access.
   a) If the RWCS is written with the AC bit set, the next read/write access begins and the RWD can be written to/ read from.
   b) If the RWCS is written with the AC bit cleared, the read/write access is terminated at the nearest completed access. This method can be used to break (early terminate) block accesses.

13.2.14.8.8 Read/Write Access Error Message

The read/write access error message is sent out when an system bus access error (read or write) occurs.

Error information is messaged out in the following format:

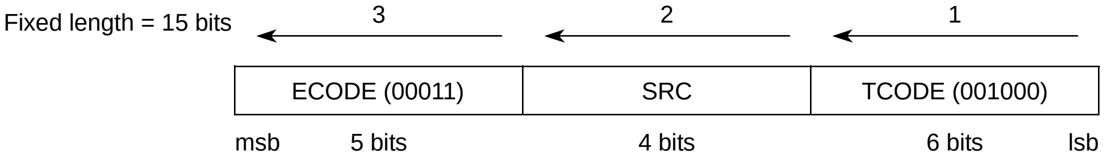


Figure 13-45. Error Message Format

13.2.14.9 Examples

The following are examples of program trace and data trace messages.

Table 13-34 illustrates an example indirect branch message with an eight MDO and two MSEO configuration.

T0 and S0 are the least significant bits where:

- Tx = TCODE number (fixed)
- Sx = Source processor (fixed)
- Ix = Number of instructions (variable)
- Ax = Unique portion of the address (variable)

Table 13-34. Indirect Branch Message Example (12 MDO and Two MSEO)

Clock	MDO[11:0]												$\overline{\text{MSEO}}$[1:0]		State
	11	10	9	8	7	6	5	4	3	2	1	0			
0	X	X	X	X	X	X	X	X	X	X	X	X	1	1	Idle (or end of last message)
1	I1	I0	S3	S2	S1	S0	T5	T4	T3	T2	T1	T0	0	0	Start Message
2	0	0	0	0	0	0	0	0	I5	I4	I3	I2	0	1	End Packet
3	0	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	1	1	End Packet/End Message
4	X	X	S3	S2	S1	S0	T5	T4	T3	T2	T1	T0	0	0	Start of Next Message

Table 13-35 illustrates an example of direct branch message with 12 MDO and two MSEO.

T0 and I0 are the least significant bits where:

- Tx = TCODE number (fixed)
- Sx = Source processor (fixed)
- Ix = Number of instructions (variable)

Table 13-35. Direct Branch Message Example (12 MDO and Two $\overline{\text{MSEO}}$)

Clock	MDO[11:0]												$\overline{\text{MSEO}}$[1:0]		State
	11	10	9	8	7	6	5	4	3	2	1	0			
0	X	X	X	X	X	X	X	X	X	X	X	X	1	1	Idle (or end of last message)
1	I1	I0	S3	S2	S1	S0	T5	T4	T3	T2	T1	T0	0	0	Start Message
2	0	0	0	0	0	0	0	0	0	0	I3	I2	1	1	End Packet and End Message
3	X	X	X	X	S1	S0	T5	T4	T3	T2	T1	T0	0	0	Start of Next Message

Table 13-36 an example data write message with 12 MDO and two MSEO configuration.

T0, A0, D0 are the least significant bits (LSB) where:

- Tx = TCODE number (fixed)
- Sx = Source processor (fixed)

- Zx = Data size (fixed)
- Ax = Unique portion of the address (variable)
- Dx = Write data (variable: 8-, 16- or 32-bit)

Table 13-36. Direct Write Message Example (12 MDO and Two $\overline{\text{MSEO}}$)

Clock	MDO[11:0]												MSEO[1:0]		State
	11	10	9	8	7	6	5	4	3	2	1	0			
0	X	X	X	X	X	X	X	X	X	X	X	X	1	1	Idle (or end of last message)
1	Z1	Z0	S3	S2	S1	S0	T5	T4	T3	T2	T1	T0	0	0	Start Message
2	0	0	0	0	0	0	0	A3	A2	A1	A0	Z2	0	1	End Packet
3	X	X	X	X	D7	D6	D5	D4	D3	D2	D1	D0	1	1	End Packet/End Message

13.2.14.10 IEEE 1149.1 (JTAG) RD/WR Sequences

This section contains example JTAG/OnCE sequences used to access resources.

13.2.14.10.1 JTAG Sequence for Accessing Internal Nexus Registers

Table 13-37. Accessing Internal Nexus3 Registers via JTAG/OnCE

Step #	TMS Pin	Description
1	1	IDLE –> SELECT-DR_SCAN
2	0	SELECT-DR_SCAN –> CAPTURE-DR (Nexus command register value loaded in shifter)
3	0	CAPTURE-DR –> SHIFT-DR
4	0	(7) TCK clocks issued to shift in direction (read/write) bit and first 6 bits of Nexus reg. addr.
5	1	SHIFT-DR –> EXIT1-DR (7th bit of Nexus reg. shifted in)
6	1	EXIT1-DR –> UPDATE-DR (Nexus shifter is transferred to Nexus command register)
7	1	UPDATE-DR –> SELECT-DR_SCAN
8	0	SELECT-DR_SCAN –> CAPTURE-DR (Register value is transferred to Nexus shifter)
9	0	CAPTURE-DR –> SHIFT-DR
10	0	(31) TCK clocks issued to transfer register value to TDO pin while shifting in TDI value
11	1	SHIFT-DR –> EXIT1-DR (msb of value is shifted in/out of shifter)
12	1	EXIT1-DR –> UPDATE -DR (if access is write, shifter is transferred to register)
13	0	UPDATE-DR–> RUN-TEST/IDLE (transfer complete - Nexus controller to reg. select state)

13.2.14.10.2 JTAG Sequence for Read Access of Memory-Mapped Resources

Table 13-38. Accessing Memory-Mapped Resources (Reads)

Step #	TCLK clocks	Description
1	13	Nexus Command = write to read/write access address register (RWA)
2	37	Write RWA (initialize starting read address—data input on TDI)
3	13	Nexus Command = write to read/write control/status register (RWCS)
4	37	Write RWCS (initialize read access mode and CNT value—data input on TDI)
5	—	Wait for falling edge of RDY pin
6	13	Nexus Command = read the read/write access data register (RWD)
7	37	Read RWD (data output on TDO)
8	—	If CNT > 0, go back to Step #5

13.2.14.10.3 JTAG Sequence for Write Access of Memory-Mapped Resources

Table 13-39. Accessing Memory-Mapped Resources (Writes)

Step #	TCLK clocks	Description
1	13	Nexus Command = write to read/write access control/status register (RWCS)
2	37	Write RWCS (initialize write access mode and CNT value—data input on TDI)
3	13	Nexus Command = write to read/write address register (RWA)
4	37	Write RWA (initialize starting write address—data input on TDI)
5	13	Nexus Command = read the read/write access data register (RWD)
6	37	Write RWD (data output on TDO)
7	—	Wait for falling edge of RDY pin
8	—	If CNT > 0, go back to Step #5

13.2.15 Nexus eDMA Interface (NXDM) and Nexus FlexRay Interface (NXFR)

The third module of the device NDI interface is the e200z7 eDMA Nexus module (NXDM) which is compliant with the Class 3 defined data trace feature of the IEEE-ISTO 5001-2003 standard.The fourth module of the device NDI interface is the FlexRay module (NXFR) which is compliant with the Class 3 defined data trace feature of the IEEE-ISTO 5001-2003 standard. The NXDM can be programmed to trace data accesses for the eDMA module on the system bus. The NXFR can be programmed to trace data accesses for the FlexRay module on the system bus. This eDMA module and FlexRay module as well as the Nexus module are components of the e200z7 platform. All output messages and register accesses are compliant with the protocol defined in the IEEE-ISTO 5001 standard.

NOTE

The auxiliary port and its signals, such as MCKO, MSEO[1:0], MDO[15:0] and others, are referenced. The device NPC module arbitrates the access of the single auxiliary port. The functions of the NXDM and FlexRay modules are described without the interaction of the NPC, as if Nexus has a dedicated auxiliary port, to simplify the description. The auxiliary port function is described in full in Section 13.2.2, “External Signal Description.”

13.2.15.1 Block Diagrams

Figure 13-46 shows a block diagram of the NXDM. The block diagram of the NXFR is the same.

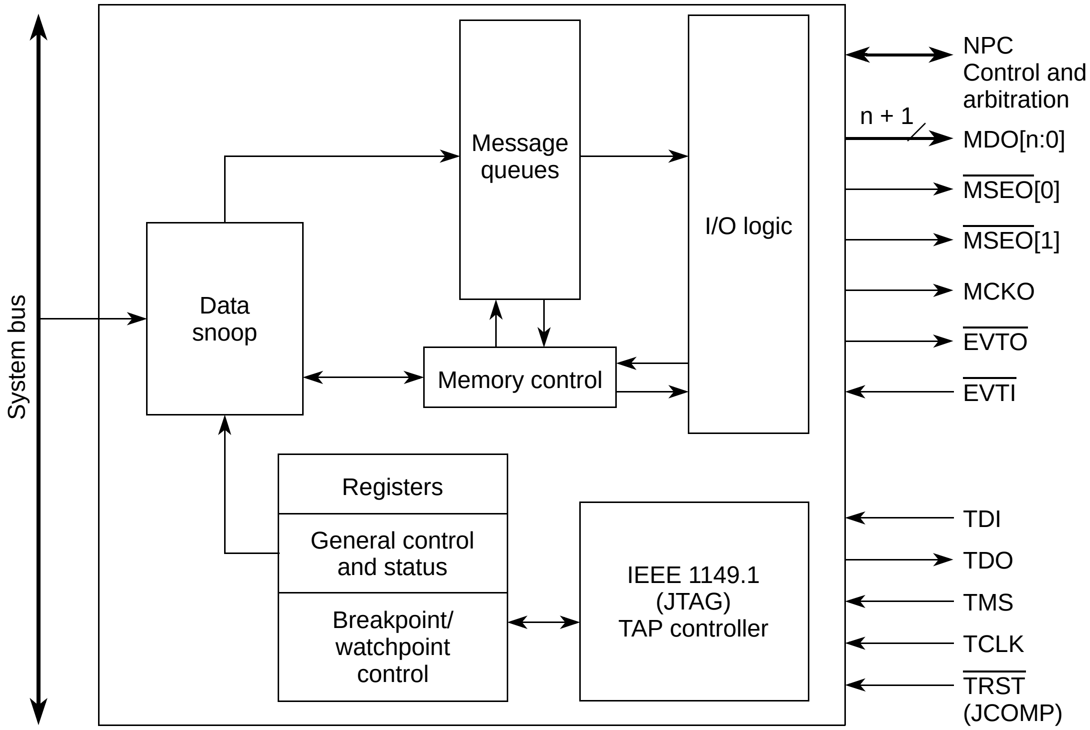


Figure 13-46. NXDM and NXFR Block Diagram

13.2.15.2 Features

Features include the following:

- Data trace via data write messaging (DWM) and data read messaging (DRM). This provides the capability for the development tool to trace reads and/or writes through the eDMA and FlexRay modules to (selected) internal memory resources.
- Watchpoint messaging via the auxiliary pins.
- Watchpoint trigger enable of data trace messaging (DTM).
- Registers for data trace, watchpoint generation, and watchpoint trigger.
- All features controllable and configurable via the JTAG port.
- Power management.

- — Low power design
- — Dynamic power management of FIFOs and control logic

13.2.16 External Signal Description

The NXDM (and NXFR) module uses the same pins and pin protocol as defined in Section 13.2.2, “External Signal Description.”

13.2.16.1 Rules for Output Messages

The NXDM (and NXFR) module observe the same rules for output messages as the NPC. Refer to Section , “Rules of Messages.”

13.2.16.2 Auxiliary Port Arbitration

The NXDM (and NXFR) module arbitrate for the shared Nexus port. This arbitration is handled by the NPC (Refer to Section 13.2.5, “Nexus Port Controller (NPC)”) based on prioritized requests from the NXDM, NXFR, and the other Nexus clients sharing the port.

13.2.17 NXDM and NXFR Programmers Model

This section describes the programmers model. Nexus registers are accessed using the JTAG port in compliance with IEEE 1149.1. Refer to Section 13.1, “IEEE 1149.1 Test Access Port Controller (JTAGC)” and Section 13.2.7.2.3, “IEEE 1149.1-2001 (JTAG) TAP” for details on Nexus register access.

13.2.17.1 NXDM and NXFR Nexus Register Map

Table 13-40. NXDM and NXFR Register Map

Nexus Register	Nexus Access Opcode	Read/Write	Read Address	Write Address
Client Select Control (CSC) [1]	0x1	R	0x02	–
Port Configuration Register (PCR) [1]	Refer to NPC	R/W	–	–
Development Control 1 (DC1_*n*)	0x2	R/W	0x04	0x05
Development Control 2 (DC2_*n*)	0x3	R/W	0x05	0x06
Watchpoint Trigger (WT_*n*)	0xB	R/W	0x16	0x17
Data Trace Control (DTC_*n*)	0xD	R/W	0x1A	0x1B
Data Trace Start Address 1 (DTSA1_*n*)	0xE	R/W	0x1C	0x1D
Data Trace Start Address 2 (DTSA2_*n*)	0xF	R/W	0x1E	0x1F
Data Trace End Address 1 (DTEA1_*n*)	0x12	R/W	0x24	0x25
Data Trace End Address 2 (DTEA2_*n*)	0x13	R/W	0x26	0x27
Breakpoint/Watchpoint Control Register 1 (BWC1_*n*)	0x16	R/W	0x2C	0x2D

Table 13-40. NXDM and NXFR Register Map (continued)

Nexus Register	Nexus Access Opcode	Read/Write	Read Address	Write Address
Breakpoint/Watchpoint Control Register 2 (BWC2_*n*)	0x17	R/W	0x2E	0x2F
Breakpoint/Watchpoint Address Register 1 (BWA1_*n*)	0x1E	R/W	0x3C	0x3D
Breakpoint/Watchpoint Address Register 2 (BWA2_*n*)	0x1F	R/W	0x3E	0x3F
Reserved	0x20–0x3F	–	0x40–0x7E	0x41–0x7F

[1] The CSC and PCR registers are shown in this table as part of the Nexus programmer's model. They are only present at the top level Nexus3 controller (NPC), not in the NXDM or NXFR module. The device's CSC register is readable through Nexus3; the PCR is shown for reference only.

13.2.17.2 NXDM and NXFR Registers

Detailed register definitions for the NXDM and NXFR implementation are as follows:

13.2.17.2.1 Development Control Registers (DC1 and DC2)

The development control registers control the basic development features of the NXDM and NXFR modules.

Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	OPC	MCK_DIV		EOC		0	0	WEN	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	C	0	0	0	0	0	0	0	0	0	EIC		TM		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-47. Development Control Register 1 (DC1)

Table 13-41. DC1 Field Description

Field	Description
31 OPC[1]	Output port mode control. 0 Reduced port mode configuration 1 Full port mode configuration
30–29 MCK_DIV[1]	Nexus message clock divide ratio. 00 MCKO is 1x system bus clock frequency. 01 MCKO is 1/2x system bus clock frequency. 10 MCKO is 1/4x system bus clock frequency. 11 MCKO is 1/8x system bus clock frequency.

Table 13-41. DC1 Field Description (continued)

Field	Description
28–27 EOC	$\overline{\text{EVTO}}$ control. 00 $\overline{\text{EVTO}}$ upon occurrence of watchpoint (internal or external) 01 $\overline{\text{EVTO}}$ upon entry into system-level debug mode (ipg_debug) 1X Invalid value
26–25	Reserved, read as 0.
24 WEN	Watchpoint trace enable. 0 Watchpoint messaging disabled 1 Watchpoint messaging enabled.
23–5	Reserved, read as 0.
4–3 EIC	$\overline{\text{EVTI}}$ control. 00 $\overline{\text{EVTI}}$ for synchronization (Data Trace) 01 Invalid value 10 $\overline{\text{EVTI}}$ disabled for this module 11 Invalid value
2–0 TM	Trace mode. 000 No Trace 1XX Program trace enabled (not supported)[2] X1X Data trace enabled XX1 Ownership Trace enable (not supported)[2]

[1] The output port mode control bit (OPC) and MCKO divide bits (MCK_DIV) are shown for clarity. These functions are controlled globally by the NPC port control register (PCR).

[2] The XBAR bus Nexus trace module only supports data trace and watchpoint trace, and does not support program or ownership trace.

Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	EWC								0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-48. Development Control Register 2 (DC2)

Table 13-42. DC2 Field Description

Field	Description
31–24 EWC[1]	$\overline{\text{EVTO}}$ Watchpoint Configuration 00000000 = No watchpoints trigger $\overline{\text{EVTO}}$ 1XXXXXXX = Invalid value X1XXXXXX = Invalid value XX1XXXXX = Invalid value XXX1XXXX = Invalid value XXXX1XXX = Internal watchpoint #1 triggers $\overline{\text{EVTO}}$ XXXXX1XX = Internal watchpoint #2 triggers $\overline{\text{EVTO}}$ XXXXXX1X = Invalid value XXXXXXX1 = Invalid value
23–0	Reserved, read as 0.

[1] The EOC bits in DC1 must be programmed to trigger $\overline{\text{EVTO}}$ on watchpoint occurrence for the EWC bits to have any effect.

13.2.17.2.2 Watchpoint Trigger Register (WT)

The watchpoint trigger register allows the watchpoints defined internally to the NXDM and NXFR modules to trigger actions. These watchpoints can control data trace enable and disable. The WT bits can be used to produce an address related window for triggering trace messages.

Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	DTS			DTE			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-49. Watchpoint Trigger Register (WT)

Table 13-43. WT Field Description

Field	Description
31–26	Reserved, read as 0.
25–23 DTS	DTS - Data trace start control 000 Trigger disabled 001–100 Invalid value 101 Use internal watchpoint #1 (BWA1 register) 110 Use internal watchpoint #2 (BWA2 register) 111 Invalid value
22–20 DTE	DTE - Data trace end control 000 Trigger disabled 001–100 Invalid value 101 Use internal watchpoint #1 (BWA1 register) 110 Use internal watchpoint #2 (BWA2 register) 111 Invalid value
19–0	Reserved, read as 0.

NOTE

The WT bits ONLY enable data trace if the tm bits within the development control register (DC) have not already been set to enable data trace.

13.2.17.2.3 Data Trace Control Register (DTC)

The data trace control register controls whether DTM Messages are restricted to reads, writes or both for a user programmable address range. There are two data trace channels controlled by the DTC for the NXDM and NXFR modules.

Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	RWT1		RWT2		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RC1	RC2	0	0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-50. Data Trace Control Register (DTC)

Table 13-44. DTC Field Description

Bit	Description
31–30 RWT1	Read/write trace 1 00 No trace messages generated X1 Enable data read trace 1X Enable data write trace
29–28 RWT2	Read/write trace 2 00 No trace messages generated X1 Enable data read trace 1X Enable data write trace
27–8	Reserved, read as 0.
7 RC1	Range control 1 0 Condition trace on address within range (endpoints inclusive) 1 Condition trace on address outside of range (endpoints exclusive)
6 RC2	Range control 2 0 Condition trace on address within range (endpoints inclusive) 1 Condition trace on address outside of range (endpoints exclusive)
5–0	Reserved, read as 0.

13.2.17.2.4 Data Trace Start Address Registers 1 and 2 (DTSA1 and DTSA2)

The data trace start address registers define the start addresses for each trace channel.

Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	DATA TRACE START ADDRESS																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-51. Data Trace Start Address Registers (DTSA1, DTSA2)

13.2.17.2.5 Data Trace End Address Registers 1 and 2 (DTEA1 and DTEA2)

The data trace end address registers define the end addresses for each trace channel.

Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DATA TRACE END ADDRESS																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-52. Data Trace Start Address Registers (DTEA1, DTEA2)

Table 13-45 illustrates the range that is selected for data trace for various cases of DTSA being less than, greater than, or equal to DTEA.

Table 13-45. Data Trace Address Range Options

Programmed Values	Range Control Bit Value	Range Selected
DTSA < or = DTEA	0	DTSA→ ←DTEA
DTSA < or = DTEA	1	← DTSA DTEA →
DTSA > DTEA	—	Invalid range, no trace

NOTE

DTSA must be less than (or equal to) DTEA to guarantee correct data write/read traces. When the range control bit is 0 (internal range), accesses to DTSA and DTEA addresses are traced. When the range control bit is 1 (external range), accesses to DTSA and DTEA are not traced.

13.2.17.2.6 Breakpoint / Watchpoint Control Register 1 (BWC1)

Breakpoint/watchpoint control register 1 controls attributes for generation of NXDM and NXFR watchpoint number 1.

Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BWE1		BRW1												BWR1		BWT1															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-53. Break / Watchpoint Control Register 1 (BWC1)

Table 13-46. BWC1 Field Description

Field	Description
31–30 BWE1	Breakpoint/watchpoint #1 enable 00 Internal Nexus watchpoint #1 disabled 01–10 Invalid value 11 Internal Nexus watchpoint #1 enabled
29–28 BRW1	Breakpoint/watchpoint #1 read/write select 00 Watchpoint #1 hit on read accesses 01 Watchpoint #1 hit on write accesses 10 Watchpoint #1 on read or write accesses 11 Invalid value
27–18	Reserved, read as 0.
17–16 BWR1	Breakpoint/watchpoint #1 register compare 00 No register compare (same as BWC1[31:30] = 2’b00) 01 Invalid value 10 Compare with BWA1 value 11 Invalid value
15 BWT1	Breakpoint/watchpoint #1 type 0 Invalid value 1 Watchpoint #1 on data accesses
14–0	Reserved, read as 0.

13.2.17.2.7 Breakpoint / Watchpoint Control Register 2 (BWC2)

Breakpoint/watchpoint control register2 controls attributes for generation of NXDM and NXFR watchpoint number 2.

Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BWE2		BRW2												BWR2		BWT2															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-54. Break / Watchpoint Control Register 2 (BWC2)

Table 13-47. BWC2 Field Description

Field	Description
31–30 BWE2	Breakpoint/watchpoint #2 enable 00 Internal Nexus watchpoint #2 disabled 01–10 Invalid value 11 Internal Nexus watchpoint #2 enabled
29–28 BRW2	Breakpoint/watchpoint #2 read/write select 00 Watchpoint #2 hit on read accesses 01 Watchpoint #2 hit on write accesses 10 Watchpoint #2 on read or write accesses 11 Invalid value
27–18	Reserved, read as 0.

Table 13-47. BWC2 Field Description (continued)

Field	Description
17–16 BWR2	Breakpoint/watchpoint #2 register compare 00 No register compare (same as BWC1[31:30] = 2’b00) 01 Invalid value 10 Compare with BWA2 value 11 Invalid value
15 BWT2	Breakpoint/watchpoint #2 Type 0 Invalid value 1 Watchpoint #2 on data accesses
14–0	Reserved, read as 0.

13.2.17.2.8 Breakpoint/Watchpoint Address Registers 1 and 2 (BWA1 and BWA2)

The breakpoint/watchpoint address registers are compared with bus addresses to generate internal watchpoints.

Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	BREAKPOINT / WATCHPOINT ADDRESS																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13-55. Breakpoint / Watchpoint Address Registers (BWA1, BWA2)

13.2.17.2.9 Unimplemented Registers

Unimplemented registers are those with client select and index value combinations other than those listed in Table 13-40. For unimplemented registers, the NXDM and NXFR modules drives TDO to zero during the “SHIFT-DR” state. It also transmits an error message with the invalid access opcode encoding.

13.2.17.2.10 Programming Considerations ($\overline{\text{RESET}}$)

If Nexus3 register configuration is to occur during system reset (as opposed to debug mode), all NXDM configuration should be completed between the negation of JCOMP and system reset de-assertion, after the JTAG DID register has been read by the tool.

13.2.17.2.11 IEEE 1149.1 (JTAG) Test Access Port

The NXDM and NXFR modules uses the IEEE 1149.1 TAP controller for accessing Nexus resources. The JTAG signals themselves are shared by all TAP controllers on the device. Refer to Section 13.1, “IEEE 1149.1 Test Access Port Controller (JTAGC),” for more information on the JTAG interface.

The NXDM and NXFR modules implements a 4-bit instruction register (IR). The valid instructions and method for register access are outlined in Section 13.2.7.2.3, “IEEE 1149.1-2001 (JTAG) TAP.”

NXDM and NXFR JTAG DID Register

This JTAG DID register that is included in the NXDM and NXFR modules provides key development attributes to the development tool concerning the NXDM and NXFR blocks. The register is accessed through the standard JTAG IR/DR paths. Refer to the PMC chapter.

Access: R/W

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PRN				DC						PIN										MIC											1
W																																
Reset	0	0	0	0	0	1	1	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1

Figure 13-56. NXDM and NXFR JTAG DID Register

Table 13-48. NXDM and NXFR JTAG DID Field Descriptions

Field	Description
31–28 PRN[1]	Embedded part revision number (0x0)
27–22 DC	Freescale design center ID number (0x1F)
21–12 PIN	NXDM and NXFR module part identification number, defines the features set. (0x60)
11–1 MIC	Manufacturer identity code 0x00E Freescale
0	Fixed per JTAG 1149.1 1 Always set

[1] The revision number is initially 0 and could change in the future.

Enabling the NXDM and NXFR TAP Controllers

Assertion of a power-on-reset signal or assertion of the JCOMP pin resets all TAP controllers on the device. Upon exit from the test-logic-reset state, the IR value is loaded with the JTAG DID. When the NXDM or NXFR TAP is accessed, this information helps the development tool obtain information about the Nexus module it is accessing, such as version, sequence, feature set, and so forth.

NXDM and NXFR Register Access via JTAG

Access to Nexus register resources is enabled by loading a single instruction (NEXUS_ACCESS) into the JTAG Instruction Register (IR). This IR is part of the IEEE 1149.1 TAP controller within the NXDM and NXFR modules. Refer to Section 13.1.4.4, "JTAGC Instructions."

After the JTAG NEXUS_ACCESS instruction has been loaded, the JTAG port allows tool/target communications with all Nexus registers according to the map in Table 13-40.

Reading/writing of a Nexus register then requires two (2) passes through the data-scan (DR) path of the JTAG state machine (refer to Section 13.1, "IEEE 1149.1 Test Access Port Controller (JTAGC)").

1. The first pass through the DR selects the Nexus register to be accessed by providing an index (refer to Table 13-40), and the direction (read/write). This is achieved by loading an 8-bit value into the JTAG data register (DR). This register has the following format:

Access: R/W

	7	6	5	4	3	2	1	0
R	Nexus Register Index							R/W
W								
Reset								

Figure 13-57. JTAG DR for NEXUS Register Access

Table 13-49. DR Read/Write Encoding

Nexus Register Index	Description
Read/Write (R/W)	0 Read 1 Write

2. The second pass through the DR then shifts the data in or out of the JTAG port, lsb first.
   a) During a read access, data is latched from the selected Nexus register when the JTAG state machine passes through the capture-DR state.
   b) During a write access, data is latched into the selected Nexus register when the JTAG state machine passes through the update-DR state.

13.2.17.3 Functional Description

13.2.17.4 Enabling NXDM and NXFR Operation

The NXDM (and NXFR) module is enabled by loading a single instruction (ACCESS_AUX_TAP_DMA or ACCESS_AUX_TAP_NXFR as shown in Table 13-7) into the JTAG instruction register (IR), and then loading the corresponding OnCE OCMD register with the NEXUS_ACCESS instruction (refer to Table 13-8). After it is enabled, the module is ready to accept control input via the JTAG pins.

The Nexus module is disabled when the JTAG state machine reaches the test-logic-reset state. This state can be reached by the assertion of the JCOMP pin or by cycling through the state machine using the TMS pin. The Nexus module is also disabled if a power-on reset (POR) event occurs.

If the NXDM (and NXFR) module is disabled, no trace output is provided, and the module disables (drive inactive) auxiliary port output pins (MDO[15:0], $\overline{\text{MSEO}}$[1:0], MCKO). Nexus registers are not be available for reads or writes.

13.2.17.5 TCODEs Supported by NXDM and NXFR

The NXDM and NXFR pins allow for flexible transfer operations via public messages. A TCODE defines the transfer format, the number and/or size of the packets to be transferred, and the purpose of each packet. The IEEE-ISTO 5001-2003 standard defines a set of public messages. The NXDM and NXFR blocks currently support the public TCODEs seen in Table 13-50.

Table 13-50. Public TCODEs Supported

Message Name	Packet Size Bits		Packet Name	Packet Type	Packet Description
	Min	Max			
Data Trace - Date Write Message	6	6	TCODE	Fixed	TCODE number = 5
	4	4	SRC	Fixed	Source processor identifier (multiple Nexus configuration)
	3	3	DSZ	Fixed	Data size (refer to Table 13-52)
	1	32	U-ADDR	Variable	Unique portion of the data write value
	1	64	DATA	Variable	Data write value
Data Trace - Data Read Message	6	6	TCODE	Fixed	TCODE number = 6
	4	4	SRC	Fixed	Source processor identifier (multiple Nexus configuration)
	3	3	DSZ	Fixed	Data size (refer to Table 13-52)
	1	32	U-ADDR	Variable	Unique portion of the data read value
	1	64	DATA	Variable	Data read value
Error Message	6	6	TCODE	Fixed	TCODE number = 8
	4	4	SRC	Fixed	Source processor identifier (multiple Nexus configuration)
	5	5	ECODE	Fixed	Error code (refer to Table 13-51)

Table 13-50. Public TCODEs Supported (continued)

Message Name	Packet Size Bits		Packet Name	Packet Type	Packet Description
	Min	Max			
Data Trace - Data Write Message w/ Sync	6	6	TCODE	Fixed	TCODE number = 13 (0xD)
	4	4	SRC	Fixed	Source processor identifier (multiple Nexus configuration)
	3	3	DSZ	Fixed	Data size (refer to Table 13-52)
	1	32	F-ADDR	Variable	Full access address (leading zero (0) truncated)
	1	64	DATA	Variable	Data write value
Data Trace - Data Read Message w/ Sync	6	6	TCODE	Fixed	TCODE number = 14 (0xE)
	4	4	SRC	Fixed	Source processor identifier (multiple Nexus configuration)
	3	3	DSZ	Fixed	Data size (refer to Table 13-52)
	1	32	F-ADDR	Variable	Full access address (leading zero (0) truncated)
	1	64	DATA	Variable	Data read valued
Watchpoint Message	6	6	TCODE	Fixed	TCODE number = 15 (0xF)
	4	4	SRC	Fixed	Source processor identifier (multiple Nexus configuration)
	4	4	WPHIT	Fixed	Number indicating watchpoint sources

Table 13-51. Error Code (ECODE) Encoding (TCODE = 8)

Error Code (ECODE)	Description
00000	Invalid value
00001	Invalid value
00010	Data Trace overrun
00011	Invalid value
00100	Invalid value
00101	Invalid access opcode (Nexus Register unimplemented)
00110	Watchpoint overrun
00111	Invalid value
01000	Data Trace and Watchpoint overrun
01001–11111	Invalid value

Table 13-52. Data Trace Size (DSZ) Encodings (TCODE = 5, 6, 13, 14)

DTM Size Encoding	Transfer Size
000	Byte
001	Halfword (two bytes)

Table 13-52. Data Trace Size (DSZ) Encodings (TCODE = 5, 6, 13, 14)

DTM Size Encoding	Transfer Size
010	Word (four bytes)
011	Doubleword (eight bytes)
100–111	Invalid value

13.2.17.5.1 Data Trace

This section deals with the data trace mechanism supported by the NXDM and NXFR modules. Data trace is implemented via data write messaging (DWM) and data read messaging (DRM).

13.2.17.5.2 Data Trace Messaging (DTM)

NXDM and NXFR data trace messaging is accomplished by snooping the NXDM and NXFR data bus, and storing the information for qualifying accesses (based on enabled features and matching target addresses). The NXDM (and NXFR) module traces all data access that meet the selected range and attributes.

NOTE

Data trace is ONLY performed on DMA or FlexRay accesses to the system bus.

13.2.17.5.3 DTM Message Formats

The NXDM (and NXFR) block supports five types of DTM Messages — data write, data read, data write synchronization, data read synchronization and error messages.

Data Write and Data Read Messages

The data write and data read messages contain the data write/read value and the address of the write/read access, relative to the previous data trace message. Data write message and data read message information is messaged out in the following format:

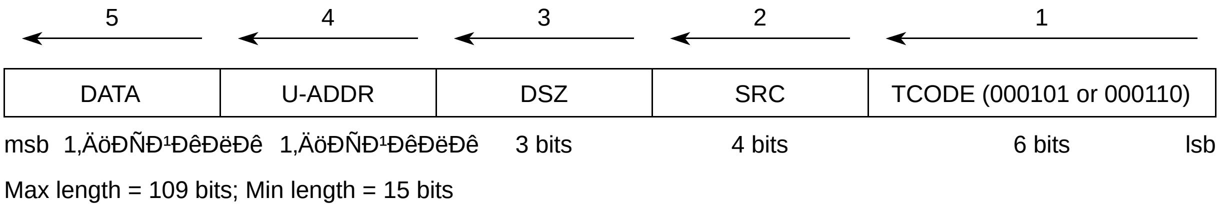


Figure 13-58. Data Write/Read Message Format

DTM Overflow Error Messages

An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards incoming messages until it has completely emptied the queue. After it is emptied, an error message is queued. The error encoding indicates which types of messages attempted to be queued while the FIFO was being emptied.

If only a data trace message attempts to enter the queue while it is being emptied, the error message incorporates the data trace only error encoding (00010). If a watchpoint also attempts to be queued while the FIFO is being emptied, then the error message incorporates error encoding (01000).

Error information is messaged out in the following format:

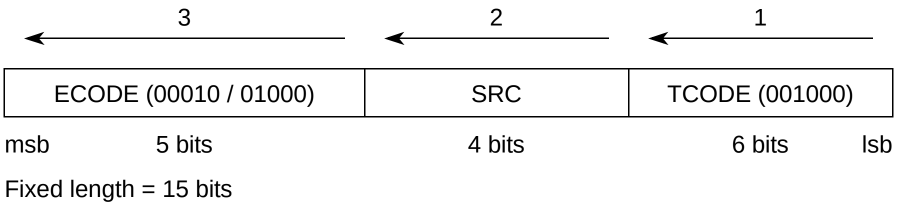


Figure 13-59. Error Message Format

Data Trace Synchronization Messages

A data trace write/read w/ sync. message is messaged via the auxiliary port (provided data trace is enabled) for the following conditions (refer to Table 13-53):

- Initial data trace message upon exit from system reset or whenever data trace is enabled is a synchronization message.
- Upon returning from a low power state, the first data trace message is a synchronization message.
- Upon returning from debug mode, the first data trace message is a synchronization message.
- After occurrence of queue overrun (can be caused by any trace message), the first data trace message is a synchronization message.
- After the periodic data trace counter has expired indicating 255 *without-sync* data trace messages have occurred since the last *with-sync* message occurred.
- Upon assertion of the Event In ($\overline{\text{EVTI}}$) pin, the first data trace message is a synchronization message if the eic bits of the dc register have enabled this feature.
- Upon data trace write/read after the previous dtm message was lost due to an attempted access to a secure memory location.
- Upon data trace write/read after the previous dtm message was lost due to a collision entering the fifo between the dtm message and any of the following: error message, or watchpoint message.

Data trace synchronization messages provide the full address (without leading zeros) and insure that development tools fully synchronize with data trace regularly. Synchronization messages provide a reference address for subsequent DTMs, in which only the unique portion of the data trace address is transmitted. The format for data trace write/read w/ sync. messages is as follows:

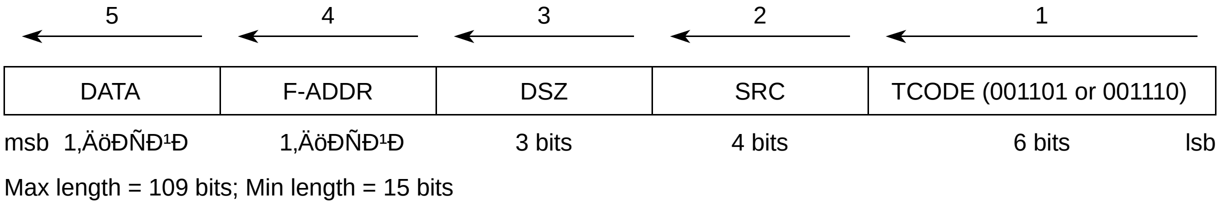


Figure 13-60. Data Write/Read w/ Sync Message Format

Exception conditions that result in data trace synchronization are summarized in Table 13-53., “Data Trace Exception Summary.”

Table 13-53. Data Trace Exception Summary

Exception Condition	Exception Handling
System Reset Negation	At the negation of JTAG reset (JCOMP), queue pointers, counters, state machines, and registers within the NXDM and NXFR module are reset. If data trace is enabled, the first data trace message is a data write/read w/ sync. message.
Data Trace Enabled	The first data trace message (after data trace has been enabled) is a synchronization message.
Exit from Low Power/Debug	Upon exit from a low power mode or debug mode the next data trace message is converted to a data write/read w/ sync. message.
Queue Overrun	An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards messages until it has completely emptied the queue. After it is emptied, an error message is queued. The error encoding indicates the types of messages that attempted to be queued while the FIFO was being emptied. The next DTM message in the queue is a data write/read w/ sync. message.
Periodic Data Trace Synchronization	A forced synchronization occurs periodically after 255 data trace messages have been queued. A data write/read w/ sync. message is queued. The periodic data trace message counter then resets.
Event In	If the nexus module is enabled, an $\overline{\text{EVTI}}$ assertion initiates a data trace write/read w/ sync. message upon the next data write/read (if data trace is enabled and the eic bits of the dc register have enabled this feature).
Attempted Access to Secure Memory	Any attempted read or write to secure memory locations temporarily disable data trace & cause the corresponding DTM to be lost. A subsequent read/write queues a data trace read/write with sync. message.
Collision Priority	All messages have the following priority: Error → WPM → DTM. A DTM message which attempts to enter the queue at the same time as an error message, or watchpoint message is lost. A subsequent read/write queues a data trace read/write with sync. message.

13.2.17.5.4 DTM Operation

Enabling Data Trace Messaging

Data trace messaging can be enabled in one of two ways.

- Setting the DC1[TM] field to enable data trace
- Using the WT[DTS] field to enable data trace on watchpoint hits

DTM Queueing

NXDM and NXFR implements a programmable depth queue for queuing all messages. Messages that enter the queue are transmitted via the auxiliary pins in the order in which they are queued.

NOTE

If multiple trace messages must be queued at the same time, watchpoint messages have the highest priority (WPM → DTM).

Relative Addressing

The relative address feature is compliant with IEEE-ISTO Nexus 5001-2003 and is designed to reduce the number of bits transmitted for addresses of data trace messages. Relative addressing is the same as described for the NZ7C3 in Section 13.2.14.2, " Relative Addressing."

Data Trace Windowing

Data write/read messages are enabled via the RWT1(2) field in the data trace control register (DTC) for each DTM channel. Data trace windowing is achieved via the address range defined by the DTEA and DTSA registers and by the RC1(2) field in the DTC. All eDMA or FlexRay initiated read/write accesses that fall inside or outside these address ranges, as programmed, are candidates to be traced.

System Bus Cycle Special Cases

Table 13-54. System Bus Cycle Special Cases

Special Case	Action
System bus cycle aborted (DABORT asserted)	Cycle ignored
System bus cycle with data error	Data Trace Message discarded
System bus cycle completed without error	Cycle captured and transmitted
System bus cycle is an instruction fetch	Cycle ignored

13.2.17.5.5 Data Trace Timing Diagrams (Eight MDO configuration)

Data trace timing for the NXDM and NXFR is the same as for the NZ7C3. Refer to Section 13.2.14.6.4, " Data Trace Timing Diagrams (Eight MDO Configuration)."

13.2.17.6 Watchpoint Support

The NXDM and NXFR module provides watchpoint messaging via the auxiliary pins, as defined by IEEE-ISTO 5001-2003.

Watchpoint messages can be generated using the NXDM and NXFR defined internal watchpoints.

13.2.17.6.1 Watchpoint Messaging

Enabling watchpoint messaging is accomplished by setting the watchpoint messaging enable bit, DC1[WEN]. Using the BWC1 and BWC2 registers, two independently controlled internal watchpoints can be initialized. When a DMA or FlexRay access address matches on BWA1 or BWA2, a watchpoint message is transmitted.

The Nexus module provides watchpoint messaging using the TCODE. When either of the two possible watchpoint sources asserts, a message is sent to the queue to be messaged out. This message indicates the watchpoint number.

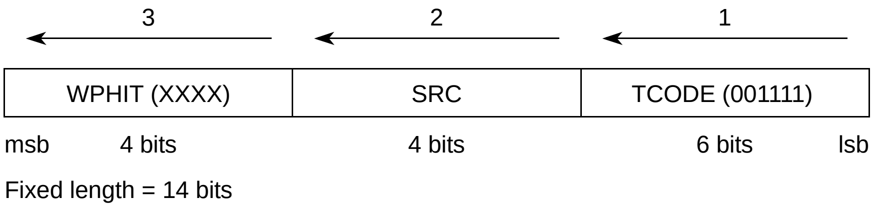


Figure 13-61. Watchpoint Message Format

Table 13-55. Watchpoint Source Description

Watchpoint Source (4 bits)	Watchpoint Description
XXX1	Invalid value
XX1X	Invalid value
X1XX	Internal Watchpoint #1 (BWA1 match)
1XXX	Internal Watchpoint #2 (BWA2 match)

13.2.17.6.2 Watchpoint Error Message

An error message occurs when a new message cannot be queued due to the message queue being full. The FIFO discards messages until it has completely emptied the queue. After it is emptied, an error message is queued. The error encoding indicates which types of messages attempted to be queued while the FIFO was being emptied.

If only a watchpoint message attempts to enter the queue while it is being emptied, the error message incorporates the watchpoint only error encoding (00110). If a data trace message also attempts to enter the queue while it is being emptied, the error message incorporates error encoding (01000).

Error information is messaged out in the following format (refer to Figure 13-62).

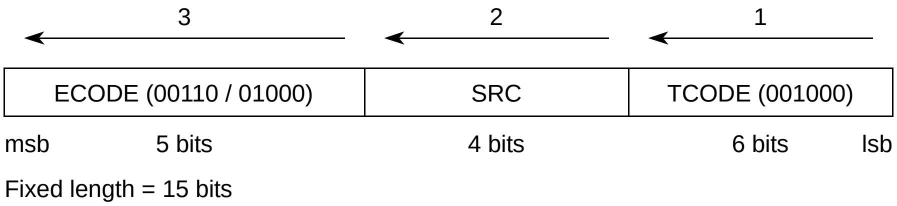


Figure 13-62. Error Message Format

Chapter 14
Decimation Filter

14.1 Overview

This document describes the Decimation Filter block functionality for the MPC5676R.

There are 12 independent Decimation Filter blocks (A through L) on the MPC5676R device. Each Decimation Filter block contains a multiply-accumulate (MAC) unit capable of implementing a 16-bit, 4th order IIR or 8th order FIR filter. The Decimation Filter blocks can be cascaded to create larger filters. Filter output data can be decimated at a programmable rate in the range 1 to 16.

Each Decimation Filter has a 32-bit integrator unit, which allows the block to sum a series of signed or absolute value filter outputs. The integrator input can be selected from before or after the decimator. The integrator can be enabled and disabled, cleared, and read by software or by hardware triggers from certain channels of the eTPU modules on the device. The integrator control triggers are selected in the SIU_DECFIL1 and SIU_DECFIL2 and SIU_DECFIL3 registers.

The data and control registers for each Decimation Filter block are independently accessed by the CPU. All Decimation Filters support DMA writes to the input data register, and DMA reads of the output data register. The eQADC_A and eQADC_B blocks have an internal hardware link to Decimation Filters A through L. This allows CPU independent transfer of eQADC_A and eQADC_B analog to digital conversion result data to the Decimation Filter input data registers, and filter output data back to the eQADC_A and eQADC_B conversion result FIFOs.

Each Decimation Filter block supports 4 combinations of input data source and output result destination:

1. **Input from eQADC/Output to eQADC**: In this mode, the Decimation Filter receives analog to digital conversion data samples from the eQADC block. The output result from the Decimation Filter is automatically returned to the RFIFO in eQADC that was specified in the conversion command for the ADC input.
2. **Input from system RAM/Output to system RAM**: In this mode, the input data to the Decimation filter is supplied from system RAM by the CPU or DMA. The filter data and associated commands are written to a memory mapped input register, and the output results are written to a memory mapped register, where they can be read by either CPU or DMA.
3. **Input from eQADC/Output to system RAM**: In this mode, the Decimation Filters can be configured for input data from eQADC and output result to the system RAM (CPU or DMA).
4. **Input from system RAM/Output to eQADC**: In this mode, the Decimation Filter input data from system RAM (CPU or DMA), and the output result is sent to an eQADC result FIFO.

Each Decimation Filter block can be programmed to generate interrupt requests when input data is received from the eQADC, when the filter result is written to the output buffer, if a filter overflow occurs, and if either input or output buffer overruns occur. An interrupt can also be generated when an integrator result is ready to be read.

Figure 14-1 shows a system-level block diagram of a single Decimation Filter. All Decimation Filters share the same configuration, with some differences in the hardware trigger sources for the integrator controls.

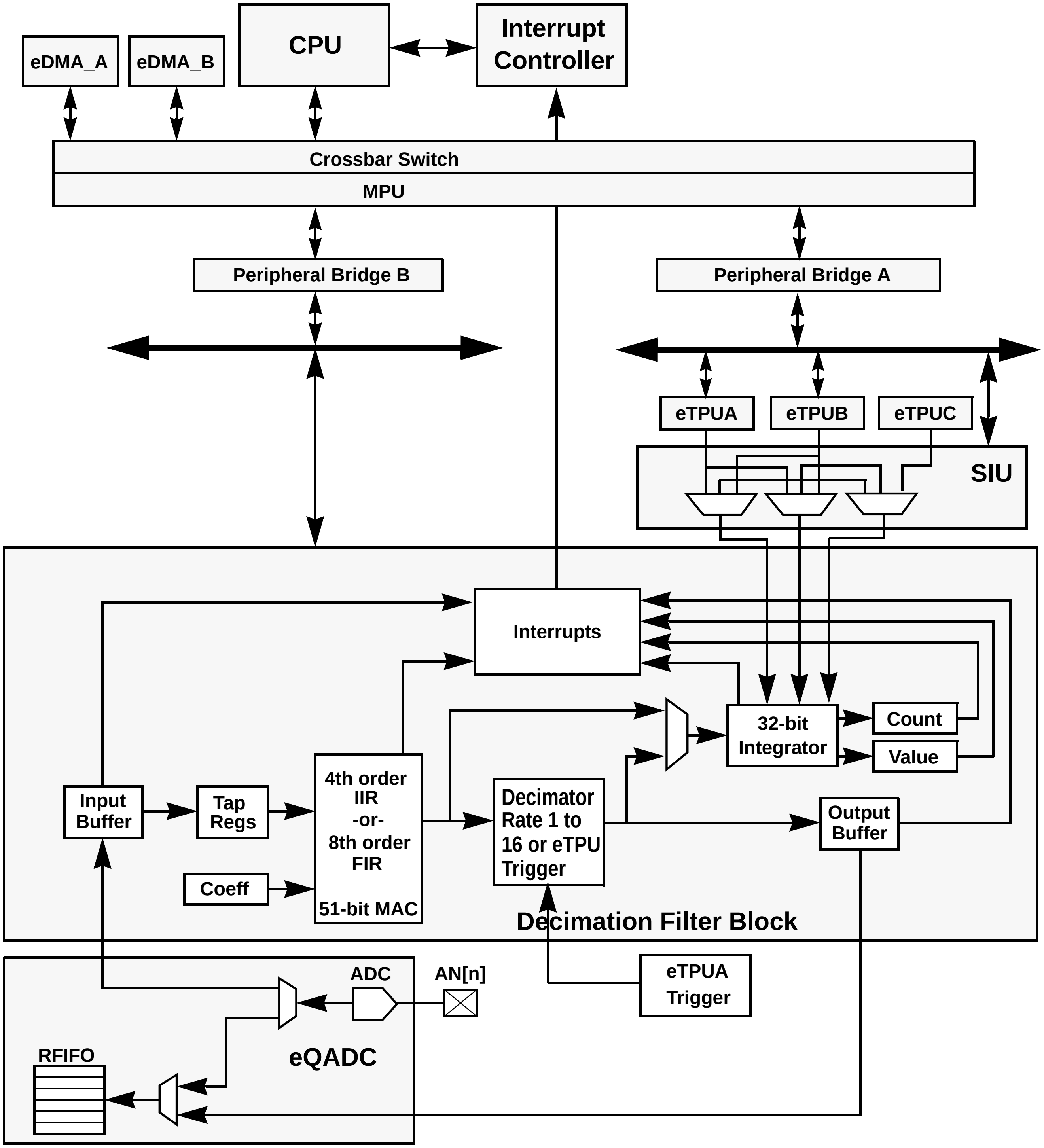


Figure 14-1. Decimation Filter Block Diagram

14.1.1 Features

The decimation filter block includes these features:

- Selectable 4th order IIR filter, or 8th order FIR filter
  - 16-bit, two’s complement signed input/output data
  - Internal taps with 16-bit (feed-forward taps) and 24-bit (feedback taps) resolutions (fixed point) for two’s complement signed value
  - 24-bit programmable filter coefficients (fixed point) for two’s complement signed value
  - MAC unit with 51-bit fixed point accumulator
  - Convergent rounding methodology
  - Two’s complement overflow or saturation selection
  - 58 system (core frequency divided by two) clock cycles to process the input
- Direct data input from the on-chip eQADC_A and eQADC_B block
- DMA access to input and output data buffers
- CPU accessible status/configuration registers and data input/output
- Filter initialization (flush) and stabilization (prefill) commands
- Timestamp support
- Programmable integer decimation rates of 1 to 16
- 32-bit, fixed point, signed or unsigned integrator unit with hardware triggered windowing
- Cascading of two or more blocks to create more complex filters

14.1.2 Modes of Operation Overview

This section provides an overview of the operational modes of the Decimation Filter. The modes are selected using the Module Configuration Register fields MDIS, FREN, and FRZ (see Section 14.2.2.1, “Decimation Filter Module Configuration Register (DECFILT_x_MCR)”. The mode selection is summarized in Table 14-1.

Table 14-1. Operation Mode Selection

Mode	MDIS	FREN, FRZ
Normal	0	(0,0), (0,1) or (1,0)
Freeze	0	1, 1
Disabled	1	X

14.1.2.1 Normal Mode

This is the default operational mode of the Decimation Filter. In this mode, the Decimation Filter processes each new input in the input buffer according the filter, decimator, and integrator configuration.

14.1.2.2 Freeze Mode

This mode is also known as debug mode. All filter action is frozen. If the filter is processing an input, it enters freeze mode only after the current processing finishes. More details of freeze mode may be found in Section 14.3.8, “Freeze Mode”.

14.1.2.3 Disabled Mode

Each Decimation Filter block may be disabled by setting the DECFILT_x_MCR[MDIS] bit in the respective block instance.

14.2 Memory Map and Register Definition

This section provides the memory maps and detailed descriptions of all registers. There are 12 Decimation Filter blocks on the MPC5676R device. Each Decimation Filter block has its own independent set of registers as defined in Section 14.2.2, "Decimation Filter Register Descriptions". The base address of each Decimation Filter block is given in Table 14-2.

Table 14-2. Decimation Filter Modules Base Address

Module	Address
DECFILT_A_ BASE	0xFFF8_8000
DECFILT_B_ BASE	0xFFF8_8800
DECFILT_C_ BASE	0xFFF8_9000
DECFILT_D_ BASE	0xFFF8_9800
DECFILT_E_ BASE	0xFFF8_A000
DECFILT_F_ BASE	0xFFF8_A800
DECFILT_G_ BASE	0xFFF8_B000
DECFILT_H_ BASE	0xFFF8_B800
DECFILT_I_ BASE	0xFFF8C000
DECFILT_J_ BASE	0xFFF8C800
DECFILT_K_ BASE	0xFFF8D000
DECFILT_L_ BASE	0xFFF8D800

14.2.1 Decimation Filter Memory Map

The addresses of the Decimation Filter registers are specified as offsets from the module's base address as described in Table 14-3.

Table 14-3. Block Memory Map

Address	Register	Bits	Access	Reset Value	Section/Page
DECFILT_x_ BASE + 0x000	DECFILTER_MCR — Module Configuration Register	32	R/W	0x0000_0000	14.2.2.1/14-7
DECFILT_x_ BASE + 0x004	DECFILTER_MSR — Module Status Register	32	R/W	0x0000_0000	14.2.2.2/14-11
DECFILT_x_ BASE + 0x008	DECFILTER_MXCR — Module Extended Configuration Register	32	R/W	0x0000_0000	14.2.2.3/14-13
DECFILT_x_ BASE + 0x00C	DECFILTER_MXSR — Module Extended Status Register	32	R/W	0x0000_0000	14.2.2.4/14-16

Table 14-3. Block Memory Map (continued)

Address	Register	Bits	Access	Reset Value	Section/Page
DECFILT_x_ BASE + 0x010	DECFILTER_IB — Interface Input Buffer	32	R/W	0x0000_0000	14.2.2.5/14-19
DECFILT_x_ BASE + 0x014	DECFILTER_OB — Interface Output Buffer	32	R	0x0000_0000	14.2.2.6/14-20
DECFILT_x_ BASE + 0x018–0x01F	Reserved				
DECFILT_x_ BASE + 0x020	DECFILTER_COEF0 — Filter Coefficient 0	32	R/W	0x0000_0000	14.2.2.7/14-20
DECFILT_x_ BASE + 0x024	DECFILTER_COEF1 — Filter Coefficient 1	32	R/W	0x0000_0000	14.2.2.7/14-20
DECFILT_x_ BASE + 0x028	DECFILTER_COEF2 — Filter Coefficient 2	32	R/W	0x0000_0000	14.2.2.7/14-20
DECFILT_x_ BASE + 0x02C	DECFILTER_COEF3 — Filter Coefficient 3	32	R/W	0x0000_0000	14.2.2.7/14-20
DECFILT_x_ BASE + 0x030	DECFILTER_COEF4 — Filter Coefficient 4	32	R/W	0x0000_0000	14.2.2.7/14-20
DECFILT_x_ BASE + 0x034	DECFILTER_COEF5 — Filter Coefficient 5	32	R/W	0x0000_0000	14.2.2.7/14-20
DECFILT_x_ BASE + 0x038	DECFILTER_COEF6 — Filter Coefficient 6	32	R/W	0x0000_0000	14.2.2.7/14-20
DECFILT_x_ BASE + 0x03C	DECFILTER_COEF7 — Filter Coefficient 7	32	R/W	0x0000_0000	14.2.2.7/14-20
DECFILT_x_ BASE + 0x040	DECFILTER_COEF8 — Filter Coefficient 8	32	R/W	0x0000_0000	14.2.2.7/14-20
DECFILT_x_ BASE + 0x044–0x077	Reserved				
DECFILT_x_ BASE + 0x078	DECFILTER_TAP0 — Filter TAP[1] 0 Register	32	R	0x0000_0000	14.2.2.8/14-21
DECFILT_x_ BASE + 0x07C	DECFILTER_TAP1 — Filter TAP 1 Register	32	R	0x0000_0000	14.2.2.8/14-21
DECFILT_x_ BASE + 0x080	DECFILTER_TAP2 — Filter TAP 2 Register	32	R	0x0000_0000	14.2.2.8/14-21
DECFILT_x_ BASE + 0x084	DECFILTER_TAP3 — Filter TAP 3 Register	32	R	0x0000_0000	14.2.2.8/14-21
DECFILT_x_ BASE + 0x088	DECFILTER_TAP4 — Filter TAP 4 Register	32	R	0x0000_0000	14.2.2.8/14-21
DECFILT_x_ BASE + 0x08C	DECFILTER_TAP5 — Filter TAP 5 Register	32	R	0x0000_0000	14.2.2.8/14-21
DECFILT_x_ BASE + 0x090	DECFILTER_TAP6 — Filter TAP 6 Register	32	R	0x0000_0000	14.2.2.8/14-21

Table 14-3. Block Memory Map (continued)

Address	Register	Bits	Access	Reset Value	Section/Page
DECFILT_x_ BASE + 0x094	DECFILTER_TAP7 — Filter TAP 7 Register	32	R	0x0000_0000	14.2.2.8/14-21
DECFILT_x_ BASE + 0x098–0x0CF	Reserved				
DECFILT_x_ BASE + 0x0D0	DECFILTER_EDID — Enhanced Debug Input Data	32	R	0x0000_0000	14.2.2.9/14-22
DECFILT_x_ BASE + 0x0D4–0x0DF	Reserved				
DECFILT_x_ BASE + 0x0E0	DECFILTER_FINTVAL — Final Integr. Value Register	32	R	0x0000_0000	14.2.2.10/14-22
DECFILT_x_ BASE + 0x0E4	DECFILTER_FINTCNT — Final Integr. Count Register	32	R	0x0000_0000	14.2.2.11/14-23
DECFILT_x_ BASE + 0x0E8	DECFILTER_CINTVAL — Current Integr. Value Register	32	R	0x0000_0000	14.2.2.12/14-24
DECFILT_x_ BASE + 0x0EC	DECFILTER_CINTCNT — Current Integr. Count Register	32	R	0x0000_0000	14.2.2.13/14-24
DECFILT_x_ BASE + 0x0F0–0x1FF	Reserved				

---

[1] The TAP register stores, on each filter node, the input sample data and, for the IIR type, the filter intermediary results.

14.2.2 Decimation Filter Register Descriptions

All registers are 32-bit wide, and all Decimation Filters on the device have the same register interface.

14.2.2.1 Decimation Filter Module Configuration Register (DECFILT_x_MCR)

The Decimation Filter module configuration register provides configuration control bits for the Decimation Filter internal logic.

NOTE

Do not modify this register’s contents when the status bit BSY is set, except for fields FREN, FRZ and IDIS. To guarantee that BSY does not set during the read-modify-write operation, it is advisable to set IDIS=1 and wait for BSY=0 beforehand.

Address: DECFILT_x_BASE + 0x000

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MDIS	FREN	0	FRZ	0	CASCD[1:0]		IDEN	ODEN	ERREN	0	FTYPE[1:0]		0	SCAL[1:0]	
W					SRES											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IDIS	SAT	IO_SEL[1:0]		DEC_RATE[3:0]				SDIE	DSEL	IBIE	OBIE	EDME	TORE	TMODE	
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-2. Decimation Filter Module Configuration Register (DECFILT_x_MCR)

Table 14-4. DECFILT_x_MCR Field Descriptions

Field	Description
0 MDIS	Module Disable—Puts the Decimation Filter in low power mode. Communication through the eQADC Interface is ignored in this mode. Writes to the configuration register are allowed with the exception of writes to the FREN and SRES bits, which are ignored. Writes to the Coefficient registers are allowed. The Decimation Filter cannot enter Freeze mode once in disable mode. 0 Normal Mode 1 Low Power Mode
1 FREN	Freeze Enable—Enables the Decimation Filter to enter freeze mode See Section 14.3.8, “Freeze Mode”, for more details. 0 Freeze mode disabled 1 Freeze mode enabled
2	Reserved
3 FRZ	Freeze Mode—Controls the freeze mode of the Decimation Filter. For this bit to take effect the FREN freeze enable bit also needs to be asserted. While in freeze mode the MAC operations are halted. See Section 14.3.8, “Freeze Mode”, for more details. 0 Normal Mode 1 Freeze Mode
4 SRES	Software-reset bit—A self-negated bit which provides the capability to initialize the Decimation Filter interface. This bit always reads as zero. See Section 14.3.7, “Soft Reset Command”, for more details. 0 No action 1 Software-Reset
5–6 CASCD	Cascade Mode Configuration—Configures the block to work in cascade mode of operation. For more details about the cascade mode, see Section 14.3.14, “Cascade Mode”. 00 No cascade mode (single block) 01 Cascade Mode, Head block config 10 Cascade Mode, Tail block config 11 Cascade Mode, Middle block config **Note:** Any change to this field must follow the procedure described in the Section 14.3.14.2, “Cascade Freeze, Stop, and Configuration Change Procedures”.
7 IDEN	Input Data Interrupt Enable—Enables the Decimation Filter to generate interrupt requests on all new input data written to the Interface Input Buffer register. 0 Input Data Interrupt Disabled 1 Input Data Interrupt Enabled

Table 14-4. DECFILT_x_MCR Field Descriptions (continued)

Field	Description
8 ODEN	Output Data Interrupt Enable—Enables the Decimation Filter to generate interrupt requests on all new data written to the filter Output buffer. This is independent of the IO_SEL field setting. 0 Output Data Interrupt Disabled 1 Output Data Interrupt Enabled
9 ERREN	Error Interrupt Enable—Enables the Decimation Filter to generate interrupt requests based on the assertion of the DECFILTER_MSR register error flags OVF, DIVR, SVR, OVR or IVR. 0 Error Interrupts Disabled 1 Error Interrupts Enabled
10	Reserved
11–12 FTYPE	Filter Type Selection bits—Selects the filter type. 00 Filter Bypass 01 IIR Filter - 1 x 4th order 10 FIR Filter - 1 x 8th order 11 Reserved
13	Reserved
14–15 SCAL	Filter Scaling Factor—Selects the scaling factor used by the filter algorithm. 00 Scaling Factor = 1 01 Scaling Factor = 4 10 Scaling Factor = 8 11 Scaling Factor = 16
16 IDIS	Input Disable—Disables the block input, so that writes to the input buffer have no effect and input DMA or interrupt requests are not issued. Input disabling is needed to change the block configuration to or from cascade mode. 0 Input enabled 1 Input disabled
17 SAT	Saturation Enable—Enables the saturation of the filter output. See Section 14.3.11, "Saturation", for more details. 0 Disables Saturation 1 Enable Saturation
18–19 IO_SEL	Input Data Source and Output Result Destination Selection—Selects the source of the input data to the Decimation Filter, and the destination for the filter output result. The IO_SEL[1:0] encoding and associated source and destination definitions is given below. Note that when Decimation Filters are cascaded to form larger filters, the IO_SEL[1:0] field only is applicable to the input data source for the head filter in the cascade, and to the output result destination for the tail filter in the cascade. Filters in the middle of the cascade receive input and send output to their adjacent filters in the cascade. Regardless of the IO_SEL setting, the Decimation Filter input and output buffer registers can be read by the CPU/DMA at any time, and the output buffer register is updated in the case of the eQADC result destination. Note that the eQADC module has to be configured to send conversion results to a decimation filter in addition to setting the IO_SEL field.

IO_SEL[1]	IO_SEL[0]	Input Data Source	Output Result Destination
0	0	eQADC A/D conversion result	eQADC RFIFO
0	1	eQADC A/D conversion result	Output Buffer Register (CPU/DMA)
1	0	Input Buffer Register (CPU/DMA)	Output Buffer Register (CPU/DMA)
1	1	Input Buffer Register (CPU/DMA)	eQADC RFIFO

Table 14-4. DECFILT_x_MCR Field Descriptions (continued)

Field	Description
20–23 DEC_RATE	Decimation Rate Selection—Selects the decimation rate used by the Decimation Filter. The decimation rate defines the number of data samples from the master block that is required to generate one decimated result in the Decimation Filter output. 0000 No Decimation: one filter output for each sample input 0001–1111 One filter output for each (DEC_RATE+1) sample inputs
24 SDIE	Integrator Data Interrupt Enable—Enables output buffer interrupts due to integrator data result being ready. 0 Integration ready does not cause an output interrupt. 1 Integration ready causes an output interrupt
25 DSEL	DMA Selection—Determines whether the data transfers — to the input buffer (write to) and from the output buffer (read from) — are performed by DMA requests or by interrupt requests. 0 Interrupt requests are generated 1 DMA requests are generated
26 IBIE	Input Buffer Interrupt Request Enable—Enables the Decimation Filter to generate interrupt requests when: • CPU/DMA is selected (IO_SEL[1]=1), DSEL=0, and the input buffer is available to receive new data; • eQADC input is selected with Enhanced debug (IO_SEL[1]=0, EDME=1), DSEL=0, and the input buffer has data to be read by the CPU. 0 Input Buffer Interrupt Request Disabled 1 Input Buffer Interrupt Request Enabled
27 OBIE	Output Buffer Interrupt Request Enable—Enables the Decimation Filter interrupt requests when outputs are directed to the CPU/DMA and DMA requests is not selected (DSEL=0). 0 Output Buffer Interrupt Request Disabled 1 Output Buffer Interrupt Request Enabled
29 EDME	Enhanced Debug Monitor Enable—Defines the enhanced debug monitor when input selection is from eQADC (IO_SEL[1]=0). 0 Enhanced debug monitor disabled 1 Enhanced debug monitor enabled
29 TORE	Triggered Output Result Enable—Enables an eTPU signal to transfer the filter result to its destination, using the mode specified by TMODE[1:0]. 0 Output buffer transfer by eTPU signal disabled 1 Output buffer transfer by eTPU signal enabled[1]
30–31 TMODE	Trigger Mode—Selects the way the eTPU signal controls the transfer of a new output result 00 result is transferred at the rising edge of the eTPU signal 01 result is transferred while the eTPU signal is 0 10 result is transferred at the falling edge of the eTPU signal 11 result is transferred while the eTPU signal is 1

[1] Refer to Section 3.2.1.24, “Decimation Filter Register 4, 5 (SIU_DECFIL4, SIU_DECFIL5) for details of available signals.

14.2.2.2 Decimation Filter Module Status Register (DECFILT_x_MSR)

Address: DECFILT_x_BASE + 0x004

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BSY	0	DEC_COUNTER[3:0]				0	0	0	0	0	0	0	0	0	0
W							IDFC	ODFC		IBIC	OBIC		DIVRC	OVFC	OVRC	IVRC
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	IDF	ODF	0	IBIF	OBIF	0	DIVR	OVF	OVR	IVR
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-3. Decimation Filter Status Register (DECFILT_x_MSR)

Table 14-5. DECFILT_x_MSR Field Descriptions

Field	Description
0 BSY	Decimation Filter Busy indication—The BSY bit indicates that the Decimation Filter is actively processing a new input data sample. BSY is not asserted when the filter is disabled (FTYPE = 00). The BSY bit is asserted when the soft reset is executed. 0 Decimation Filter Idle 1 Decimation Filter Busy
1	Reserved
2–5 DEC_ COUNTER	Decimation Counter—The DEC_COUNTER[3:0] field indicates the current value of the DEC_COUNTER Decimation Counter, which counts the number of input data samples received by the Decimation Filter. When the value of this counter matches the DEC_RATE[3:0] Configuration Register field, one result is available and the DEC_COUNTER counter is re-initialized at zero. This register is cleared by a soft reset or a flush command.
6 IDFC	Input Data Flag Clear bit—The IDFC bit clears the IDF Flag bit in the Status Register. This bit is self negated, therefore it is always read as zero. 0 No action 1 Clears IDF
7 ODFC	Output Data Flag Clear bit—The ODFC bit clears the ODF Flag bit in the Status Register. This bit is self negated, therefore it is always read as zero. 0 No action 1 Clears ODF
8	Reserved
9 IBIC	Input Buffer Interrupt Request Clear bit—The IBIC bit clears the IBIF Flag bit in the Status Register. This bit is self negated, therefore it is always read as zero. 0 No action 1 Clears IBIF
10 OBIC	Output Buffer Interrupt Request Clear bit—The OBIC bit clears the OBIF Flag bit in the Status Register. This bit is self negated, therefore it is always read as zero. 0 No action 1 Clears OBIF
11	Reserved

Table 14-5. DECFILT_x_MSR Field Descriptions (continued)

Field	Description
12 DIVRC	DIVR Clear bit—The DIVRC bit clears the DIVR Debug Filter Input Data Read Overrun indication bit in the Status Register. This bit is self negated, therefore it is always read as zero. 0 No action 1 Clears DIVR
13 OVFC	OVF Clear bit—The OVFC bit clears the OVF Output Overflow bit in the Status Register. This bit is self negated, therefore it is always read as zero. 0 No action 1 Clears OVF
14 OVRC	OVR Clear bit—The OVRC bit clears the OVR Output Overrun bit in the Status Register. This bit is self negated, therefore it is always read as zero. 0 No action 1 Clears OVR
15 IVRC	IVR Clear bit—The IVRC bit clears the IVR Filter Input Overrun indication bit in the Status Register. This bit is self negated, therefore it is always read as zero. 0 No action 1 Clears IVR
16–21	Reserved
22 IDF	Input Data Flag—The IDF bit flag indicates when new data is available at the DECFILT_x_IB register or at the DECFILT_x_IOB register. This flag generates an Interrupt Request if enabled by the IDEN bit in the Configuration Register. This Flag is cleared by setting the IDFC Status bit or by a soft reset of the decimation filter. 0 Sample not received 1 New Sample received **Note:** This flag is not used for read / write requests. It is used only to announce the input data event. For read / write request flag, refer to IBIF.
23 ODF	Output Data Flag—The ODF bit flag indicates when a new decimated sample is available at the DECFILT_x_OB register or at the DECFILT_x_IOB register. This flag generates an Interrupt Request if enabled by the ODEN bit in the Configuration Register. This Flag is cleared by setting the ODFC Status bit or by a soft reset of the decimation filter. 0 No new Decimated Output Sample available 1 New Decimated Output Sample available **Note:** This flag is not used for read requests. It is used only to announce the output data event. For read request flag, refer to OBIF.
24	Reserved
25 IBIF	Input Buffer Interrupt Request Flag—The IBIF bit flag indicates that the input buffer DECFILT_x_IB is available to be filled with new data, when Enhanced Debug Monitor is off. In Enhanced Debug Monitor, it indicates the input buffer DECFILT_x_IB was filled with a new sample and is ready to be read. This flag is cleared by setting the IBIC bit or by a soft reset of the decimation filter. 0 No action 1 New Sample is requested (IO_SEL[1] = 1, EDME=0) or new sample is available in Enhanced Debug Monitor (IO_SEL[1]=0, EDME=1).
26 OBIF	Output Buffer Interrupt Request Flag—The OBIF bit flag indicates that either a new decimated sample is available at the DECFILT_x_OB register. This flag is cleared by setting the OBIC bit or by a soft reset of the decimation filter. 0 No new Decimated Output available 1 New Decimated Output available
27	Reserved

Table 14-5. DECFILT_x_MSR Field Descriptions (continued)

Field	Description
28 DIVR	Enhanced Debug Monitor Input Data Read Overrun—The DIVR bit indicates that a received sample in the Filter Interface Input Register was overwritten by a new sample and was not read by the CPU/DMA. This flag generates an Interrupt Request if enabled by the ERREN bit in the Configuration Register. This Flag is cleared by the DIVRC Status bit or by a soft reset of the decimation filter. 0 Input Data Read Overrun did not occur in Enhanced Debug monitor 1 Enhanced Debug Monitor Input Data Read Overrun occurred
29 OVF	Filter Overflow Flag—The OVF bit indicates that an overflow occurred in the filtered sample result. This flag generates an Interrupt Request if enabled by the ERREN bit in the Configuration Register. This Flag is cleared by the OVFC Status bit or by a soft reset of the decimation filter. 0 No overflow 1 Overflow occurred
30 OVR	Output Interface Buffer Overrun—The OVR bit indicates that a decimated sample was overwritten by a new sample in the Interface Output Buffer Register. This flag generates an Interrupt Request if enabled by the ERREN bit in the Configuration Register. This Flag is cleared by the OVRC Status bit or by a soft reset of the decimation filter. 0 No Output Overrun 1 Filter Output Overrun occurred
31 IVR	Input Interface Buffer Overrun—The IVR bit indicates that a received sample in the Filter Interface Input Register was overwritten by a new sample. This flag generates an Interrupt Request if enabled by the ERREN bit in the Configuration Register. This Flag is cleared by the IVRC Status bit or by a soft reset of the decimation filter. 0 Input Buffer Overrun did not occur 1 Input Buffer Overrun occurred **Note:** This bit does not set on an input register write when the input is disabled (DECFILTER_x_MCR[IDIS]=1).

14.2.2.3 Decimation Filter Module Extended Configuration Register (DECFILT_x_MXCR)

Address: DECFILT_x_BASE + 0x008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SDMAE	SSIG	SSAT	SCSAT	0	0	0	0	0	0	0	0	0	0	0	0
W															SRQ	SZRO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SISEL	0	SZROSEL		0	0	SHLTSEL		0	SRQSEL			0	0	SENSEL	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-4. Decimation Filter Extended Configuration Register (DECFILT_x_MXCR)

Table 14-6. DECFILT_x_MXCR Field Descriptions

Field	Description
0 SDMAE	Integrator DMA Enable—The SDMAE bit enables a DMA request when an integrator output is available (see Section 14.3.13.2, “Integrator Output”). 0 Integrator DMA request disabled 1 Integrator DMA request enabled **Note:** The DMA channel used is the same one used for filter outputs, and any configuration that generates DMA requests from both of those sources is not allowed.
1 SSIG	Integrator Signal operation selection—The SSIG bit defines how the filtered data signal is treated for integration: 0 Integrator input takes the absolute value of filter output 1 Integrator input takes the signed filter output
2 SSAT	Integrator Saturated operation selection—The SSAT bit defines how the integrator accumulator behaves in case of an overflow. 0 Integrator accumulator holds a modulo 2^{17} value (considering the 15-bit fractional part) on an overflow. 1 Integrator accumulator saturates on an overflow **Note:** In saturated operation the overflown integration sum holds the value 0xFFFFFFFF for absolute integration (SSIG=0), or values 0x7FFFFFFF (positive saturation) and 0x80000000 (negative saturation) for signed integration (SSIG=1). **Note:** Non-saturated mode is not supported with signed integration, therefore one must not configure SSIG=1 and SSAT=0.
3 SCSAT	Integrator Counter Saturated operation selection—The SCSAT bit defines how the integrator sample counter behaves in case of an overflow. 0 Integrator sample counter holds a modulo 2^{32} value on an overflow. 1 Integrator sample counter saturates on an overflow, holding a value of 0xFFFFFFFF.
4–13	Reserved
14 SRQ	Integrator Output Request—The SRQ bit is used to command the update of the integrator output, reflected in the registers DECFILT_x_FINTVAL and DECFILT_x_FINTCNT. It may also cause a DMA or interrupt request, depending on the DECFILT_x_MCR bit SDIE and DECFILT_x_MXCR bit SDMAE. This is a write-only bit, so reads always return 0. For more details see Section 14.3.13.2, “Integrator Output”. 0 No integrator output update request 1 Requests integrator output update
15 SZRO	Integrator Zero—The SZRO bit is used to zero the integrator sum. This is a write-only bit, reads always return 0. For more details see Section 14.3.13.3, “Integrator Reset”. 0 Does not zero integrator sum 1 Zeroes integrator sum **Note:** If bits SRQ and SZRO are both written 1 at the same time, the integrator is reset only after the registers DECFILT_x_FINTVAL and DECFILT_x_FINTCNT are updated.
16 SISEL	Integrator Input Selection—The SISEL bit selects the input of the integrator. For more details see Section 14.3.13.1, “Integrator Inputs”. 0 Decimated filter outputs feed the integrator 1 Filter outputs before the decimation feed the integrator
17	Reserved

Table 14-6. DECFILT_x_MXCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>18–19
SZROSEL</td><td>Integrator Zero Control Mode Selection—The SZROSEL field defines the use of the integrator zero hardware input signal. For more details see Section 14.3.13.3, “Integrator Reset”.
<table>
<tr><th>SZROSEL[1:0]</th><th>Description</th></tr>
<tr><td>0 0</td><td>Hardware integrator zero request disabled</td></tr>
<tr><td>0 1</td><td>Integrator zero on toggle of hardware signal</td></tr>
<tr><td>1 0</td><td>Integrator zero on rising edge of hardware signal</td></tr>
<tr><td>1 1</td><td>Integrator zero on falling edge of hardware signal</td></tr>
</table></td></tr>
<tr><td>20–21</td><td>Reserved</td></tr>
<tr><td>22–23
SHLTSEL</td><td>Integrator Halt Control Selection—The SHLTSEL field defines the integrator halting mechanism. When the integrator is halted, the integration accumulator remains unaltered on filter outputs independently of the enabling selected by SENSEL. For more details see Section 14.3.13.4, “Integrator Enabling and Halting”.
<table>
<tr><th>SHLTSEL[1:0]</th><th>Description</th></tr>
<tr><td>0 0</td><td>Hardware halt control signal disabled</td></tr>
<tr><td>0 1</td><td>Integrator halted, independently of the hardware signal</td></tr>
<tr><td>1 0</td><td>Integrator halted when signal is at logical 0</td></tr>
<tr><td>1 1</td><td>Integrator halted when signal is at logical 1</td></tr>
</table></td></tr>
<tr><td>24</td><td>Reserved</td></tr>
<tr><td>25–27
SRQSEL</td><td>Integrator Output Read Request Mode Selection—The SRQSEL field defines the use of the integrator output read request hardware input signal. An integrator output read request updates the registers DECFILT_x_FINTVAL and DECFILT_x_FINTCNT, also causing a DMA or interrupt request. Note that DMA or interrupt requests due to integrator output updates depend on the DECFILT_x_MXCR bit SDMAE and DECFILT_x_MCR bit SDIE.
When continuous output is on, an integrator output read request is automatically issued whenever a new filter output is accumulated. For more details see Section 14.3.13.2, “Integrator Output”.
<table>
<tr><th>SRQSEL[2:0]</th><th>Description</th></tr>
<tr><td>0 0 0</td><td>Hardware output request disabled</td></tr>
<tr><td>0 0 1</td><td>Integrator output request on toggle of hardware signal</td></tr>
<tr><td>0 1 0</td><td>Integrator output request on rising edge of hardware signal</td></tr>
<tr><td>0 1 1</td><td>Integrator output request on falling edge of hardware signal</td></tr>
<tr><td>1 0 0</td><td>Reserved</td></tr>
<tr><td>1 0 1</td><td>Continuous output request on, independently of hardware signal</td></tr>
<tr><td>1 1 0</td><td>Continuous output request on when signal is at logical 0</td></tr>
<tr><td>1 1 1</td><td>Continuous output request on when signal is at logical 1</td></tr>
</table></td></tr>
</table>

Table 14-6. DECFILT_x_MXCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>28–29</td><td>Reserved</td></tr>
<tr><td>30–31
SENSEL</td><td>Integrator Enable Control Selection—The SENSEL field defines the integrator enabling mechanism. When the integrator is enabled, filter outputs selected by the SISEL bit are added to the integration accumulator. When the integrator is disabled, the integration accumulator remains unaltered on filter outputs. For more details see Section 14.3.13.4, “Integrator Enabling and Halting”.
<table>
<tr><th>SENSEL[1:0]</th><th>Description</th></tr>
<tr><td>0 0</td><td>Integrator disabled, independently of the hardware enable control signal</td></tr>
<tr><td>0 1</td><td>Integrator enabled, independently of the hardware signal</td></tr>
<tr><td>1 0</td><td>Integrator enabled when signal is at logical 0</td></tr>
<tr><td>1 1</td><td>Integrator enabled when signal is at logical 1</td></tr>
</table>
</td></tr>
</table>

14.2.2.4 Decimation Filter Module Extended Status Register (DECFILT_x_MXSR)

Address: DECFILT_x_BASE + 0x00C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W								SDFC			SSEC	SCEC		SSOVFC	SCOVFC	SVRC
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	SDF	0	0	SSE	SCE	0	SSOVF	SCOVF	SVR
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-5. Decimation Filter Extended Status Register (DECFILT_x_MXSR)

Table 14-7. DECFILT_x_MXSR Field Descriptions

Field	Description
0–6	Reserved
7 SDFC	Integrator Output Data Flag Clear bit—The SDFC bit clears the SDF Flag bit in the Status Register. This bit is self negated, therefore it is always read as zero. 0 No action 1 Clears SDF
8–9	Reserved

Table 14-7. DECFILT_x_MXSR Field Descriptions (continued)

Field	Description
10 SSEC	SSEC — Integrator Sum Exception Clear bit—The SSEC bit clears the SSE flag bit in the Status Register. This bit is self negated, therefore it is always read as zero. 0 No action 1 Clears SSE
11 SCEC	Integrator Count Exception Clear bit—The SCEC bit clears the SCE flag bit in the Status Register. This bit is self negated, therefore it is always read as zero. 0 No action 1 Clears SCE
12	Reserved
13 SSOVFC	Integrator Sum Overflow Clear bit—The SSOVFC bit clears the SSOVF Flag bit in the Status Register. This bit is self negated, therefore it is always read as zero. 0 No action 1 Clears SSOVF
14 SCOVFC	Integrator Count Overflow Clear bit—The SCOVFC bit clears the SCOVF Flag bit in the Status Register. This bit is self negated, therefore it is always read as zero. 0 No action 1 Clears SCOVF
15 SVRC	SVR Clear bit—The SVRC bit clears the SVR Integrator Data Overrun indication bit in the Status Register. This bit is self negated, therefore it is always read as zero. 0 No action 1 Clears SVR
16–22	Reserved
23 SDF	Integrator Data Flag—The SDF bit flag indicates when a new integrator result is available at the DECFILT_x_FINTVAL register. This flag generates an Interrupt Request if enabled by the SDIE bit in the Configuration Register. This Flag is cleared by the SDFC Status bit or by a soft reset of the decimation filter. 0 No new integrator result available 1 New integrator result available
24–25	Reserved
26 SSE	Integrator Sum Exception flag—The SSE bit indicates an exceptional condition of the integrator accumulator. This flag generates an Interrupt Request if enabled by the DECFILT_x_MCR bit ERREN, and it is cleared by the SSEC bit or by a soft reset. Integrator exceptions are defined in Section 14.3.13.5, “Integrator Exceptions”. 0 No exception in the integrator accumulator. 1 Integrator accumulator exception.
27 SCE	Integrator Count Exception flag—The SCE bit indicates an exceptional condition of the integrator counter. This flag generates an Interrupt Request if enabled by the DECFILT_x_MCR bit ERREN, and it is cleared by the SCEC bit or by a soft reset. Integrator exceptions are defined in Section 14.3.13.5, “Integrator Exceptions”. 0 No exception in the integrator counter. 1 Integrator counter exception.
28	Reserved

Table 14-7. DECFILT_x_MXSR Field Descriptions (continued)

Field	Description
29 SSOVF	Integrator Sum Overflow Flag—The SSOVF bit indicates an overflow of the integrator accumulator. This Flag is cleared by the SSOVFC bit or by a soft reset. 0 No overflow in the integrator accumulator. 1 Integrator accumulator overflown. **Note:** The SSOVF bit samples the integrator accumulator overflow condition when and only when either registers DECFILT__FINTVAL or DECFILT__CINTCNT are updated. Therefore, only one of the register pairs (DECFILT_x_FINTVAL/DECFILT_x_FINTCNT and DECFILT_x_CINTVAL/DECFILT_x_CINTCNT) must be used by the application, in order to avoid races.
30 SCOVF	Integrator Count Overflow Flag—The SCOVF bit flag indicates an overflow of the internal integrated sample counter. This Flag is cleared by the SCOVFC bit or by a soft reset. 0 No overflow in the integrator sample counter. 1 Integrator sample counter overflown. **Note:** The SCOVF bit samples the integrator accumulator overflow condition when and only when either registers DECFILT_x_FINTVAL or DECFILT_x_CINTCNT are updated. Therefore, only one of the register pairs (DECFILT_x_FINTVAL/DECFILT_x_FINTCNT and DECFILT_x_CINTVAL/DECFILT_x_CINTCNT) must be used by the application, in order to avoid races.
31 SVR	Integrator Data Overrun—The SVR bit indicates that an integration value and count in the registers DECFILT_x_FINTVAL and DECFILT_x_FINTCNT was overwritten by a new integrator output request and was not read by the CPU or DMA. This flag generates an Interrupt Request if enabled by the ERREN bit in the Configuration Register. This Flag is cleared by the SVRC bit or by a soft reset. 0 Integrator Data Overrun did not occur. 1 Integrator Data Overrun occurred.

14.2.2.5 Decimation Filter Interface Input Buffer Register (DECFILT_x_IB)

The Input Buffer Register provides access to the Input buffer of the decimation filter when the filter is receiving input data from the CPU/DMA (DECFILT_x_MCR[IO_SEL[1]] = 1). Writes to this register are interpreted as requests to the Decimation Filter to process new sample data. Writes to this register when DECFILT_x_MCR[IO_SEL[1]] = 0 have no effect.

Address: DECFILT_x_BASE + 0x010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	OSEL	INTAG[3:0]				0	0	0	0	0	0	PRE FILL	FLUSH
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	INPBUF[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-6. Decimation Filter Interface Input Buffer Register (DECFILT_x_IB)

Table 14-8. DECFILT_x_IB Field Descriptions

Field	Description
0–2	Reserved
3 OSEL	The OSEL bit indicates whether the filter output returns to eQADC_A or eQADC_B. This field has no meaning when both the data source and destination are not eQADC modules. 0 Result data is returned to eQADC_A 1 Result data is returned to eQADC_B
4–7 INTAG	Decimation filter input tag bits —The INTAG[3:0] bit field is defined as a selector signal and it is used to identify different destinations for the INBUF[15:0] data. When the input data source is the CPU/DMA and the output destination is an eQADC, INTAG is used to address the appropriate RFIFO in the eQADC block.
8–13	Reserved
14 PREFILL	Decimation Filter Prefill/Filter control bit—The PREFILL bit selects the Decimation Filter operation mode. For more details, see Section 14.3.4, “Filter Prefill Control”. 0 Decimation Filter normal sample 1 Decimation Filter prefill sample
15 FLUSH	Decimation Filter Flush control bit—Assertion of the FLUSH bit initializes the Decimation Filter to a initial state, as defined in Section 14.3.6, “Flush Command”. This bit is self negated and it is cleared only when the data is read and the flush is executed. 0 No flush request 1 Flush request
16–31 INPBUF	Input Buffer Data—The INPBUF[15:0] bit field carries the sample data to be filtered. This data buffer can be written from the eQADC or by the CPU/DMA. See Section 14.3.1, “Decimation Filter Input”, for more details.

14.2.2.6 Decimation Filter Interface Output Buffer Register (DECFILT_x_OB)

Address: DECFILT_x_BASE + 0x014 Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	OSEL	OUTTAG[3:0]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OUTBUF[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-7. Decimation Filter Interface Output Buffer Register (DECFILT_x_OB)

Table 14-9. DECFILT_x_OB Field Descriptions

Field	Description
0–10	Reserved
11 OSEL	The OSEL bit indicates whether the filter output returns to eQADC_A or eQADC_B. This field has no meaning when the destination is not an eQADC module. 0 Result data is returned to eQADC_A 1 Result data is returned to eQADC_B
12–15 OUTTAG	Decimation filter output tag bits—The OUTTAG[3:0] bit field is defined as a selector signal and it is used to identify different destinations for the OUTBUF[15:0] data. When the output result destination is an eQADC, OUTTAG holds the same value as the DECFILT_x_IB[INTAG], which is used to address the destination RFIFO.
16–31 OUTBUF	Output Buffer Data—The OUTPBUF[15:0] bit field is the result data in the decimation filter Output Buffer. It represents a fixed point signed number in two's complement format and is updated only when a decimated result is ready to be transmitted, meaning it contains the last decimated result from the filter.

14.2.2.7 Decimation Filter Coefficient n Register (DECFILT_x_COEFn)

Address: DECFILT_x_BASE + 0x020–0x040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	8{COEFn[23]}								COEFn[23:16]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	COEFn[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-8. Decimation Filter Coefficient n Register (DECFILT_x_COEFn)

Table 14-10. DECFILT_x_COEFn Field Descriptions

Field	Description
0–31 COEFn	Coefficient n field—The COEFn[23:0] bit fields are the digital filter coefficients registers. The coefficients are fractional signed values in two's complement format, in the range (-1 ≤ coef < 1). **Note:** Reads to this register are sign-extended, meaning the coefficient's sign bit is copied to all 8 most significant register bits. **Note:** Writing to these fields when BSY=1 is not allowed.

14.2.2.8 Decimation Filter TAPn Register (DECFILT_x_TAPn)

Address: DECFILT_x_BASE + 0x078–0x094

Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	8{TAPn[23]}								TAPn[23:16]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TAPn[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-9. Decimation Filter TAPn Register (DECFILT_x_TAPn)

Table 14-11. DECFILT_x_TAPn Field Descriptions

Field	Description
0–31 TAPn	TAPn Register—The read-only TAPn[23:0] bit fields shows the contents of the digital filter tap registers, as fractional signed values in two's complement format, in the range (-1 ≤ coef < 1). The tap registers hold the input data delay line (Xn, Xn-1,......,Xn-7 for 8th order FIR). **Note:** Reads to this register are sign-extended, meaning the coefficient's sign bit is copied to all 8 most significant register bits. **Note:** The content of these registers is meaningless when BSY=1.

14.2.2.9 Decimation Filter Interface Enhanced Debug Input Data Register (DECFILT_x_EDID)

The Enhanced Debug Input Data Register provides read-only access to the sample data received by the Decimation Filter when the input is selected from an eQADC module (DECFILT_x_MCR[IO_SEL[1]] =0), allowing the monitoring of input data from an eQADC. See Section 14.3.15, “Enhanced Debug Monitor Description”, for more details, and see Section 14.3.12.1.2, “Input Buffer Enhanced Debug Monitor Interrupt”, and Section 14.3.12.1.4, “Input Buffer Enhanced Debug Monitor DMA Request”, for more information on interrupt and DMA requests associated with the Enhanced Debug Monitor. Writes to this register have no effect.

Address: DECFILT_x_BASE + 0x0D0 Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SAMP_DATA[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-10. Decimation Filter Interface Input Buffer Register (DECFILT_x_EDID)

Table 14-12. DECFILT_x_EDID Field Descriptions

Field	Description
0–15	Reserved
16–31 SAMP_DATA	Conversion Sample Data—The SAMP_DATA[15:0] bit field carries the data that was loaded into the Decimation Filter input buffer to be processed by the FIR/IIR sub-block. This value is only updated by input data received from an eQADC (DECFILT_x_MCR[IO_SEL[1]] = 0), and the Enhanced Debug Monitor is enabled (DECFILT_x_MCR[EDME] = 1).

14.2.2.10 Decimation Filter Final Integration Value Register (DECFILT_x_FINTVAL)

Address: DECFILT_x_BASE + 0x0E0 Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SUM_VALUE[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SUM_VALUE[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-11. Decimation Filter Final Integration Value Register (DECFILT_x_FINTVAL)

Table 14-13. DECFILT_x_FINTVAL Field Descriptions

Field	Description
0–31 SUM_VALUE	Integration Sum Value—The SUM_VALUE[31:0] field holds the sum of filtered output values. The 17 most significant bits hold the integer part, and the 15 least significant ones the fractional part of the integration value. The control of the integration sum and update of this register is determined by the register DECFILT_x_MXCR (see Section 14.2.2.3, “Decimation Filter Module Extended Configuration Register (DECFILT_x_MXCR)”). The register is updated only upon an integration output request. SUM_VALUE should be taken as an unsigned number when the integrator is configured for absolute operation (DECFILTER_MXCR bit SSIG=0), and a two’s complement signed number otherwise. **Note:** If DEFILT_x_MXCR[SSAT]=0, DECFILT_x_FINTVAL holds the integration sum modulo 2^{17} (considering the 15-bit fractional part). **Note:** If DEFILT_x_MXCR[SSAT]=1, the integration sum is saturated, so that if the accumulation overflows DECFILT_x_FINTVAL holds the value 0xFFFFFFFF for absolute integration (SSIG=0), or values 0x7FFFFFFF (positive saturation) and 0x80000000 (negative saturation) for signed integration (SSIG=1).

14.2.2.11 Decimation Filter Final Integration Count Value Register (DECFILT_x_FINTCNT)

Address: DECFILT_x_BASE + 0x0E4 Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	COUNT[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	COUNT[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-12. Decimation Filter Final Integration Count Value Register (DECFILT_x_FINTCNT)

Table 14-14. DECFILT_x_FINTCNT Field Descriptions

Field	Description
0–31 COUNT	Integration Count Value—The COUNT field holds the count of filtered outputs integrated. The control of the integration sum and update of this register is determined by the register DECFILT_x_MXCR. The register is updated together with DECFILT_x_FINTVAL, only upon an integration output request.

14.2.2.12 Decimation Filter Current Integration Value Register (DECFILT_x_CINTVAL)

Address: DECFILT_x_BASE + 0x0E8 Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SUM_VALUE[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SUM_VALUE[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-13. Decimation Filter Current Integration Value Register (DECFILT_x_CINTVAL)

Table 14-15. DECFILT_x_CINTVAL Field Descriptions

Field	Description
0–31 SUM_VALUE	Integration Sum Value—The SUM_VALUE[31:0] field holds an unsigned number representing the sum of filtered output values, continuously updated as the integration proceeds. The control of the integration sum is determined by the register DECFILT_x_MXCR (see Section 14.2.2.3, “Decimation Filter Module Extended Configuration Register (DECFILT_x_MXCR)”).

14.2.2.13 Decimation Filter Current Integration Count Value Register (DECFILT_x_CINTCNT)

Address: DECFILT_x_BASE + 0x0EC Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	COUNT[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	COUNT[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-14. Decimation Filter Current Integration Count Value Register (DECFILT_x_CINTCNT)

Table 14-16. DECFILT_x_CINTCNT Field Descriptions

Field	Description
0–31 COUNT	Integration Count Value—The COUNT field holds the count of filtered outputs integrated. The value is updated only when register DECFILT_x_CINTVAL is read, to keep the coherency between the integration and count values.

14.3 Functional Description

The following subsections describe the functional operation of a single block and apply to all Decimation Filter blocks on the MPC5676R device.

14.3.1 Decimation Filter Input

The Decimation Filter receives input data from either the an eQADC block, or from the CPU using the memory mapped DECFILT_x_IB register. The data source is selected by the DECFILT_x_MCR[IO_SEL[1]] bit. Note that when the Decimation Filters are cascaded, filters other than the head filter receive input from the adjacent filter. See Section 14.3.14, “Cascade Mode”, for more information on cascade operation.

An interrupt or DMA request can be generated when the input buffer is empty, and the input data source is the CPU or DMA. A DMA request is generated when the DECFILT_x_MCR[DSEL] = 1, and the input buffer becomes empty. If DMA is not enabled, an interrupt is generated by setting the DECFILT_x_MCR[IBIE] = 1. If both an interrupt and DMA request are enabled, the DMA takes precedence.

An interrupt can also be generated when the input buffer is written with a new value. This interrupt is enabled by the DECFILT_x_MCR[IDEN] bit. When enabled and data is written to the filter input buffer, the DECFILT_x_MSR[IDF] bit is set. The IDF flag remains set, even after the input data has been consumed by the filter and the buffer is free, until IDF is cleared by software write to the IDFC bit.

See Section 14.3.12, “Interrupts and DMA Overview”, for more information on interrupt and DMA requests.

14.3.1.1 Input Buffer Overrun

An input overrun occurs when the input buffer is holding input data and new data is received by the filter. See Section 14.3.1, “Decimation Filter Input”, for details of the input buffer. When the decimation filter is idle (DECFILT_x_MSR[BSY] = 0) the filter can receive two consecutive input buffer writes without input overrun. Input buffer overrun is detected and flagged by the DECFILT_x_MSR[IVR] bit. The overrun interrupt is enabled by the DECFILT_x_MCR[ERREN] bit.

The input buffer overrun can occur only in the following cases:

- When the input buffer has sample data to be processed but the filter is busy and another input (data or timestamp) is received.
- When the input buffer has a timestamp, the internal timestamp register is loaded and the next input data is received.

As an example of the input data sequence, assume that the filter is enabled and not busy, and all registers are empty. Then a word of sample data is received followed by a timestamp and another word of sample data. No input overrun occurs in this case, because the first sample is immediately transferred to the tap input register, the timestamp is immediately transferred to the internal timestamp storage register, and the second sample can be held in the input buffer until the end of the processing of the first sample data by the filter. The input overrun may occur if more input is received before the end of the processing, or if the filter is busy at the beginning of the received sequence.

When the filter is bypassed (DECFILT_x_MCR[FTYPE] = 0b00) or in disable mode, the data from the input buffer is transferred to the output buffer, if it is not already full. If the output buffer is full, the input buffer is loaded, and another word of input data is sent, then an input overrun occurs.

14.3.2 Decimation Filter Output

NOTE

> Decimation Filter H is not writeable or readable from the eQADC, only by DMA or the CPU. All other decimation filters have no limitations.

The Decimation Filter can send filtered results directly to an eQADC, or to the CPU using the memory mapped DECFILT_x_OB output buffer register. The output destination for the filter result is determined by the DECFILT_x_MCR[IO_SEL] field. The filter result is written to the output buffer register when the decimation count is reached (either an eQADC or CPU destination). The DECFILT_x_MSR[ODF] flag bit is set when the output buffer is updated. When an eQADC is selected for the result destination, the result is automatically transferred to an eQADC RFIFO when the output buffer is updated.

The output buffer is not updated when the Decimation Filter is in prefill mode, so the DECFILT_x_MSR[ODF] flag is not set in that case.

An interrupt or DMA request can be generated when the output buffer is updated, and the output result destination is the CPU or DMA. A DMA request is generated when the DECFILT_x_MCR[DSEL] = 1, and the output buffer is updated. If DMA is not enabled, an interrupt is generated by setting the DECFILT_x_MCR[OBIE] = 1. If both an interrupt and DMA request are enabled, the DMA takes precedence.

When the filter is bypassed (DECFILT_x_MCR[FTYPE]=0b00), and the eQADC is selected as the output destination, the data written into the input buffer is delayed until the output buffer is empty and then written to the output buffer. When the filter is bypassed and the memory mapped register is selected as the output destination, the data written into the input buffer is immediately written into the output buffer, and the DECFILT_x_MSR[ODF] flag is set.

A Soft Reset (DECFILT_x_MSR[SRES]) clears the output buffer, and terminates output data transfer to the output buffer.

14.3.2.1 Output Buffer Overrun

An output overrun occurs when the output buffer (DECFILT_x_OB) is holding output data (sample or timestamp) that has not been read and it is overwritten with subsequent data (sample or timestamp). Output overruns are flagged by the DECFILT_x_MSR[OVR] bit. An output buffer overrun interrupt is enabled by the DECFILT_x_MCR[ERREN] bit. The output buffer empty condition depends on the mode and output selection as follows:

- When the output result destination is an eQADC, the output buffer is considered empty when the filter output transfer to an eQADC RFIFO is complete.
- When the output result destination is the CPU/DMA, the output buffer is considered empty after the buffer has been read and the DECFILT_x_MSR[ODF] flag is cleared.

Prefill inputs do not cause IIR or FIR output overrun for either output destination selection.

When bypass mode is selected, an output overrun cannot occur because the data written into the input buffer (DECFILT_x_IB) is written into the output buffer (DECFILT_x_OB) only when this buffer is empty, but an input overrun may still occur (see Section 14.3.1.1, “Input Buffer Overrun”).

14.3.3 Bypass Operation

Bypass operation is configured by setting the field FTYPE[1:0] of the Module Configuration Register DECFILT_x_MCR to 00. In this case, the input sample is sent to the output with no change. This behavior is independent of input data source or output result destination selections. The following applies to the bypass configuration:

- flush is ignored
- prefill is ignored
- decimation is ignored
- BSY bit is not set
- the input flag DECFILT_x_MSR[IDF] and output flag DECFILT_x_MSR[ODF] are set

14.3.4 Filter Prefill Control

A prefill indicates that the input data should be accepted by the Decimation Filter, but no decimated output should be generated while the control field indicates prefill. Therefore the prefill function is used in the beginning of the filter operation to initialize and stabilize the Decimation Filter without generating decimated samples. In addition, the prefill does not operate when the filter is in bypass (FTYPE=0b00).

When the input data source is an eQADC (DECFILT_x_MCR[IO_SEL[1]] = 0), prefill is enabled/disabled in the eQADC conversion command word (CCW). See Section 14.4.1, “eQADC Configuration for Decimation Filter Operation”, for more information on an eQADC configuration of decimation filter functionality. When the input data source is the CPU/DMA (DECFILT_x_MCR[IO_SEL[1] = 1]), prefill is controlled by the PREFILL field in the DECFILT_x_IB register.

When the prefill control is set, the decimation filter block operates as follows:

- Input data is processed normally by the digital filter and tap values are updated.
- The decimation counter is maintained in reset value.
- The output buffer is not updated and no output interrupt is generated.
- The accompanying timestamp for the identified prefill conversion data is not bypassed.
- The overflow detector/flag operates normally and the error interrupt request is set if enabled.

14.3.5 Timestamp Data Transmission

14.3.5.1 Timestamp Data

When the selected input source is the eQADC, the eQADC can be configured to send timestamp data after related sample data for filtering. The timestamp is sent back to the RFIFO following the respective filter output, using the same mechanism. Since the Decimation Filter takes several clock cycles to process a sample, the timestamp is copied into an internal timestamp register until the filtered output is sent out, therefore freeing the input buffer for more sample data.

Timestamp information is not supported when the input data source is the CPU or DMA.

14.3.5.2 Timestamp Management

The timestamp data input is automatically sent to the Decimation Filter by the eQADC when configured to do so in the eQADC conversion command word. However, some additional conditions are considered:

- The timestamp is additional information that accompanies a sample conversion data. The eQADC block sends the decimation filter the conversion data with control bits for either prefill or filter operation. This data may optionally be followed by the corresponding timestamp data. When the corresponding conversion data is marked for prefill, the timestamp data is not sent to the output buffer. This occurs because the filter result is not sent to the output buffer.
- Similarly, when the filter is decimating the results, the timestamp is only sent to the output buffer if the corresponding received conversion data has generated a filter output that is selected by the decimation counter to be sent to the output buffer. Other received timestamps that come with data not selected by the decimator are discarded.

14.3.6 Flush Command

The flush signal is used by the Decimation Filter to execute a partial reset of the filter. This is useful when the same filter is used on a new set of data samples after finishing the filtering of another set of data.

When the flush control is detected, all filter TAPs are cleared and the DEC_COUNTER[3:0] field in the status register DECFILT_x_MSR is reset.

The flush function does not clear the Coefficient registers (DECFILT_x_COEFn) in the Decimation Filter, thus it is not required to re-write these registers after a flush. The output buffer also keeps the last result and may be retrieved until the next output is posted.

The flush control precedes the input data to be filtered. Therefore, the corresponding sample data is processed by the block after the flush. When input is from an eQADC (DECFILT_x_MCR[IO_SEL[1] = 0]), the flush command is included in the eQADC conversion command word. See Section 14.4.1, “eQADC Configuration for Decimation Filter Operation”, for more information on an eQADC configuration of decimation filter functionality. When input is from the CPU/DMA (DECFILT_x_MCR[IO_SEL[1]] =1), the flush command comes from the FLUSH bit in the DECFILT_x_IB register. Note that a word of valid sample data can be available at the same time the flush signal is asserted. In this case the flush is executed and the sample is processed after the flush.

The flush command is ignored when the Decimation Filter is disabled (DECFILT_x_MCR[MDIS]=1).

14.3.7 Soft Reset Command

The Soft Reset command is requested through the self negated bit SRES of the DECFILT_x_MCR register and provides the CPU with the capability to initialize the Decimation Filter. After the software reset is issued, all internal Filter TAP registers, the decimation counter, and the state machine are put in to the reset state. The status register DECFILT_x_MSR is also cleared. The Coefficient registers are not affected by the Soft Reset. If the filter is currently processing data (the MAC is active and DECFILT_x_MSR[BSY]=1), the processing is aborted. In addition, any pending output data transfer to the eQADC RFIFO is terminated. The software reset command has precedence over all other register control bits except the module disable bit. The DECFILT_x_MSR[BSY] is set on the detection of the assertion of the DECFILT_x_MCR[SRES] bit, and remains set until the reset procedure is complete.

The configuration register DECFILT_x_MCR is also not affected by a soft reset, except for the self-negation of the SRES bit.

When in debug or freeze mode, the soft reset is executed but the filter remains in debug or freeze mode.

14.3.8 Freeze Mode

The freeze mode operation is entered using the FREN and FRZ bits in the DECFILT_x_MCR register, or when the entire SoC enters debug mode.

It is not possible to enter freeze mode when the Decimation Filter is disabled.

In case of a freeze mode request during the processing of an input sample, the current processing is finished and then the Decimation Filter block enters freeze mode.

Access to all memory mapped registers in the Decimation Filter remains active in freeze mode.

14.3.9 Filter Implementation

The filter hardware shown in Figure 14-15 contains eight taps which may be configured as an IIR or FIR filter. Multiplexer A controls the *bypass* filter path and multiplexer B controls/selects the filter mode of operation, to either IIR mode or FIR mode. The selection is controlled by the FTYPE[1:0] bits in the Filter Module Configuration register. The order of the filter can be controlled by setting the appropriate filter coefficients to zero. The IIR can be up to 4rd order and the FIR up to 8th order.

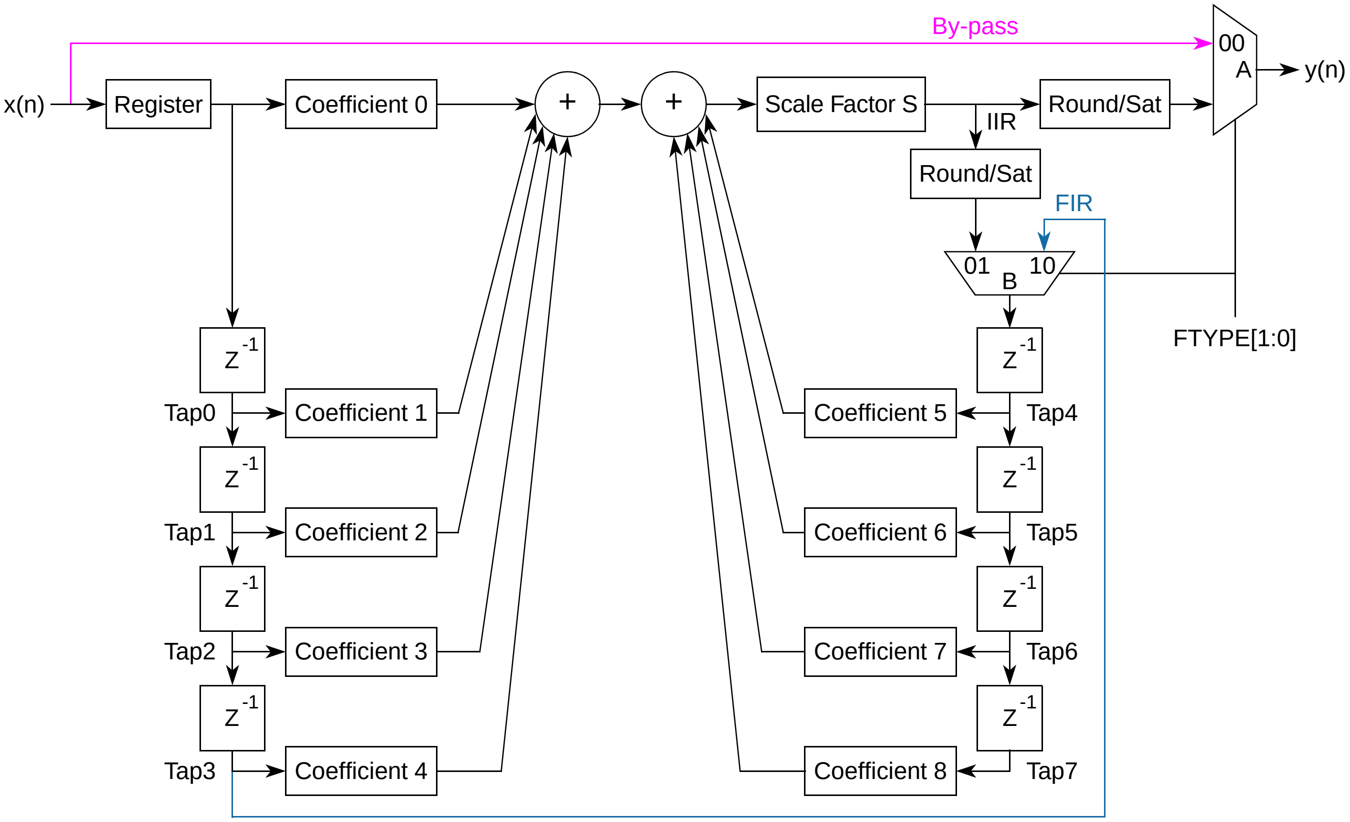


Figure 14-15. Filter Configuration Paths (FIR or 1x4Poles IIR)

14.3.10 Rounding

The Decimation Filter performs rounding operations in two different locations, as shown in Figure 14-15:

- to obtain the filter output result with 16 bits
- to obtain the IIR feedback result to be stored in tap4 registers with 24 bits

The rounding mechanism implements the Convergent Rounding methodology (also known as round-to-nearest even number), which makes the decision on rounding up or down based on the value of the lower portion of data to be rounded (LS_WORD). The rounding up/down condition is equal to the traditional rounding except when the LS_WORD has the format {1000...00}. In this particular case, the rounding procedure is like the example of Figure 14-16. If the MS_WORD is odd, the value is rounded up. Otherwise the value is rounded down.

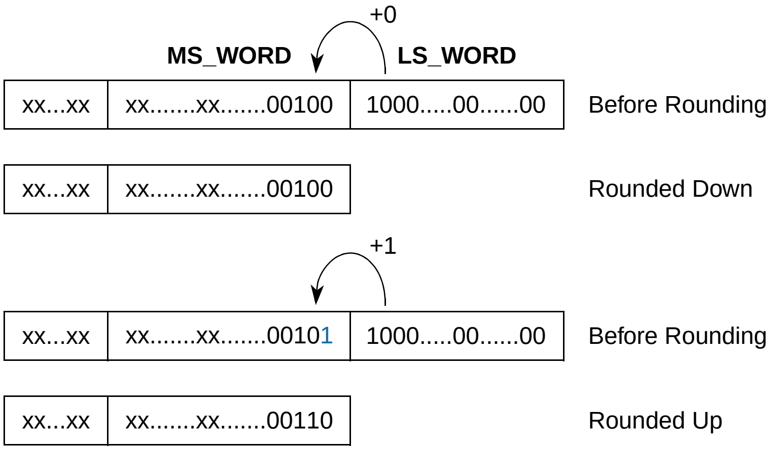


Figure 14-16. Convergent Rounding Methodology

14.3.11 Saturation

Filter output saturation occurs when an overflow or underflow condition of the filter is detected by dedicated logic, and if it is enabled by the SAT control bit of the configuration register DECFILT_x_MCR. In this condition, the filter output is set to a saturated value equal to the maximum or minimum value that can be represented by the 16-bit output port. Also, for the IIR filter an equivalent logic is used to assert the saturation for the 24-bit feedback result.

14.3.12 Interrupts and DMA Overview

There are several host request events that can be enabled using the module configuration register (MCR). An interrupt request can be issued under any of the following conditions:

- when a word of input data is received
- when the input buffer can receive data
- when a word of output data is available
- when an error has occurred.

The input data flag DECFILT_x_MSR[IDF] is set when a new input data is received from the CPU or an eQADC. Note that this flag is not used to generate read or write requests (as defined in Section 14.3.12.1, “Input Buffer Interrupt and DMA Requests”).

Output data is available and its flag (DECFILT_x_MSR[ODF]) is set when the input data sample is processed by the filter, the decimation counter matches the decimation rate value, and it is moved to the output buffer. It is not used to generate read requests (as defined in Section 14.3.12.2, “Output Buffer Interrupt and DMA Requests”).

An error event in the decimation filter block is defined as one of these events:

- Overflow in the filter, flagged by DECFILT_x_MSR[OVF]
- Overrun in the decimation filter input, flagged by DECFILT_x_MSR[IVR]
- Overrun in the decimation filter output, flagged by DECFILT_x_MSR[OVR]

- Overrun in enhanced debug monitor, flagged by DECFILT_x_MSR[DIVR]
- Integrator overrun, flagged by DECFILT_x_MSR[SVR]
- Integrator value exception, flagged by DECFILT_x_MSR[SSE]
- Integrator count exception, flagged by DECFILT_x_MSR[SCE]

A filter overflow occurs when the two's-complement result value from the MAC accumulator is out of the range of values that can be stored in tap register 4 (IIR) or in the output register.

An input overrun occurs when the input buffer is holding a word of input data and one more word of data is received by the filter. See Section 14.3.1.1, "Input Buffer Overrun", for more details.

An output overrun occurs when a new word of data is sent to the output buffer but the previous word of data has not been handled yet. See Section 14.3.2.1, "Output Buffer Overrun", for more details.

These flags can be set for input from or output to an eQADC, however they are only cleared by

- the CPU, or
- by the soft reset command (DECFILT_x_MCR[SRES]
- by the clear flag bits in the DECFILT_x_MSR register.

The DMA function for integrator result, input and output buffers is enabled setting the DECFILT_x_MCR[DSEL] = 1. The DMA request generally replaces the interrupt request that is normally generated by these registers/conditions and is discussed in more detail in Section 14.3.12.1.3, "Input Buffer DMA Request", Section 14.3.12.2.2, "Output Buffer DMA Request", and Section 14.3.12.3, "Integrator Interrupt and DMA Requests".

14.3.12.1 Input Buffer Interrupt and DMA Requests

14.3.12.1.1 Input Buffer Interrupt

This interrupt is enabled by the DECFILT_x_MCR[IBIE] bit and is asserted only when DECFILT_x_MCR[DSEL] = 0. If DECFILT_x_MCR[DSEL] = 1, a DMA request is generated instead (See Section 14.3.12.1.3, "Input Buffer DMA Request"). When this request is asserted, the DECFILT_x_MSR[IBIF] bit is set to indicate a pending interrupt.

When the input data source is the CPU, the input buffer interrupt request is asserted when the input buffer is empty, meaning the block is requesting data be written into the input buffer. The interrupt request is cleared when the CPU writes a one to the DECFILT_x_MSR[IBIC] bit, or by a soft reset command.

14.3.12.1.2 Input Buffer Enhanced Debug Monitor Interrupt

When the input data source is an eQADC, DMA is disabled, and enhanced debug is enabled, the input sample data can be read by the CPU when this interrupt request is asserted. The interrupt is asserted when a new word of sample data is supplied to the filter, and gives the application visibility into the input data being from an eQADC.

In enhanced debug mode, if the input buffer is overwritten by the next word of sample data, an input read overrun event can occur (the DECFILT_x_MSR[DIVR] bit is asserted) if the interrupt request is not cleared before, or at the same time as, the new sample arrives to set the interrupt. The

DECFILT_x_MSR[DIVR] bit is cleared by writing a one to the DECFILT_x_MSR[DIVRC] bit. Note however, in enhanced debug mode, the set condition has higher priority than the clear. This means that if the set condition occurs at the same time the CPU writes the clear bit (DECFILT_x_MSR[IBIC] to clear the interrupt, the interrupt remains asserted.

14.3.12.1.3 Input Buffer DMA Request

This DMA request is enabled by setting the IO_SEL field and the DSEL bit in the DECFILT_x_MCR register. When CPU/DMA is selected as the input data source, the input buffer DMA request is asserted when the input buffer is available to receive a conversion sample (it is not holding a word of data). This DMA request is cleared when an input data word is written to the input buffer. Therefore, the DMA request is always cleared before it is asserted again. This DMA request can also be cleared by a soft reset.

14.3.12.1.4 Input Buffer Enhanced Debug Monitor DMA Request

When an eQADC is the input data source, DMA is enabled, and enhanced debug is enabled, the input sample data can be read by DMA when this DMA request is asserted. The request is asserted when a new word of sample data is written into the input buffer to be processed. As this filter register is overwritten by the next word of sample data, a DMA read overrun event can occur (the DECFILT_x_MSR[DIVR] bit is asserted) if the DMA request is not cleared before, or at the same time as, a new sample arrives to set the DMA request. The DECFILT_x_MSR[DIVR] bit is cleared by writing one to the DECFILT_x_MSR[DIVRC] bit or by soft reset.

14.3.12.2 Output Buffer Interrupt and DMA Requests

14.3.12.2.1 Output Buffer Interrupt

This interrupt is enabled by setting the DECFILT_x_MCR[OBIE] bit, and is asserted when the output buffer is updated, CPU is selected for the output destination, and DMA is disabled. The DECFILT_x_MSR[OBIF] flag bit is set when the interrupt request is asserted.

The output buffer interrupt request can also be asserted when DECFILT_x_MCR[SDIE] = 1, and an integrator result is ready to be read. This condition is indicated when the DECFILT_x_MXSR[SDF] bit is set. Note that both the filter output and integrator output share the same interrupt source.

This interrupt request is cleared by writing a 1 to the bit DECFILT_x_MSR[OBIC] and/or the DECFILT_x_MXSR[SDFC] bits, or by a soft reset command.

14.3.12.2.2 Output Buffer DMA Request

When the CPU is selected as the output destination, and DMA is enabled, the output buffer can be read using DMA. The output buffer DMA request is asserted when the output buffer receives a new result from the filter. This DMA request is cleared when the output buffer is read by the processor, or a soft reset occurs.

14.3.12.3 Integrator Interrupt and DMA Requests

An interrupt or DMA request can be asserted when an integrator output is ready to be read or an overflow or error condition occurs in either the integrator sum or count values. Integrator interrupt requests are enabled by the DECFILT_x_MCR[SDIE] bit, and flagged in the DECFILT_x_MXSR[SDF] bit. Integrator DMA requests are enabled by the DECFILT_x_MXCR[SDMAE] bit.

14.3.13 Integrator

The decimation filter output result may be optionally routed to a dedicated hardware integrator. The integrator may be operated in a windowed mode, controlled by signals routed internally from eTPU2 channels, or operated in a continuous mode. Additionally, the integrator output may be configured to saturate at the maximum value of the supported range, or permitted to continue integration. Figure 14-17 shows the high level data flows and controls for the integrator. The RFIFO and Output Buffer data paths are shown in this figure to highlight that they are independent of the integrator. Note however that the Output Buffer shares a DMA and Interrupt request with the Integrator output value, FINTVAL (DECFILTER_FINTVAL).

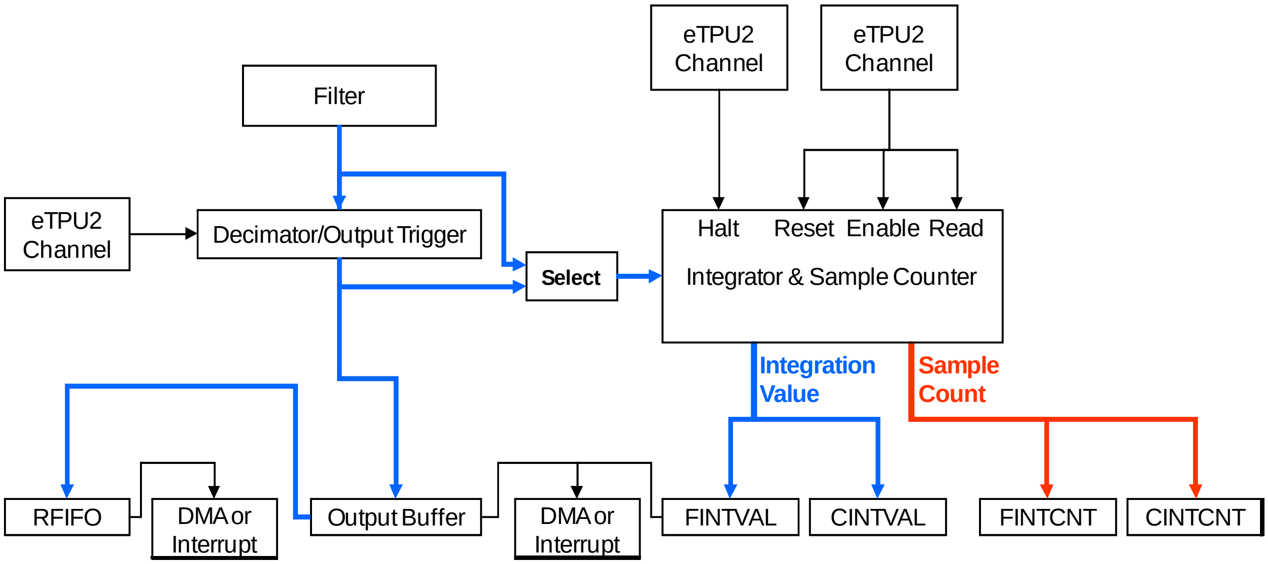


Figure 14-17. Integrator Data and Control

14.3.13.1 Integrator Inputs

The integrator input can come either directly from the filter output or from the decimator output, selected by the DECFILTER_MXCR[SISEL] (see Section 14.2.2.3, “Decimation Filter Module Extended Configuration Register (DECFILT_x_MXCR)”). Prior to integration a hardware option, controlled by DECFILTER_MXCR[SSIG], is provided to convert the input to its absolute value.

NOTE

The integrator accumulates input samples when bypass is selected in the filter.

14.3.13.2 Integrator Output

The integrator output is provided in two separate 32 bit registers: DECFILTER_FINTVAL and DECFILTER_CINTVAL. DECFILTER_FINTVAL is only updated when an output request is made by an eTPU2 channel or by a core or DMA access to a configuration register. DECFILTER_CINTVAL is updated whenever a new integration result is available or when a reset request is made by hardware or software. The output is in one of two forms:

- the 32-bit, fixed point unsigned accumulation of the absolute values from the filter output, when configured for absolute operation (DECFILTER_MXCR[SSIG] = 0). This allows a total of 131071 samples to be integrated before an overflow occurs.
- the 32-bit, fixed point signed two's complement accumulation of the signed values from the filter output, when configured for signed operation (DECFILTER_MXCR[SSIG] = 1). This allows a total of 65536 samples to be integrated before an overflow occurs

The fractional part of the accumulation is 15 bits wide in both forms.

An accumulation overflow is flagged in DECFILTER_MXSR[SSOVF] when an output request occurs. The accumulator can overflow in either of the ways described below, selected through the DECFILTER_MXCR[SSAT]:

- saturated accumulation (SSAT=1), so that an overflow results in the value of 0xFFFFFFFF for absolute value accumulation (SSIG=0), or 0x7FFFFFFF (positive) and 0x80000000 (negative) for signed accumulation (SSIG=1).
- non-saturated accumulation (SSAT=0), so that an overflow results in the modulo 2^{17} accumulation value. This operation is only allowed in absolute accumulation (SSIG=0).

NOTE

A non-saturated overflow that occurs before SSOVF is cleared is still flagged in the next output request.

The integrator output value becomes available in register DECFILTER_FINTVAL (see Section 14.2.2.10, "Decimation Filter Final Integration Value Register (DECFILT_x_FINTVAL)") when an integrator output request is issued. The integrator output request can be issued in the following ways:

- by an eTPU2 channel; the enabling and selection of the signal request modes is done through the DECFILTER_MXCR[SRQSEL] field (see Section 14.2.2.3, "Decimation Filter Module Extended Configuration Register (DECFILT_x_MXCR)"), and the channel selection is done through the ZSELn fields of SIU_DECFIL1, SIU_DECFIL2 and SIU_DECFIL3 registers in the SIU module.
- by software, writing 1 to the DECFILTER_MXCR[SRQ];

The integrator output request also updates the register DECFILTER_FINTCNT, which holds the number of samples accumulated into the register DECFILTER_FINTVAL. This accumulated sample counter can operate either in a saturated or "wrapped" count mode, as selected by DECFILTER_MXCR[SCSAT]. In both cases, a counter overflow is flagged by DECFILTER_MXSR[SCOVF].

NOTE

A non-saturated overflow that occurs before an SCOVF clear is still flagged in the next output request.

An integrator output update can also issue a DMA or interrupt request. The interrupt and DMA requests are the same ones used for the filter output buffer (see Section 14.3.12.2, “Output Buffer Interrupt and DMA Requests”, and Section 14.3.12.1.3, “Input Buffer DMA Request”). DECFILTER_MCR[SDIE] is used to enable integrator interrupts, and the DECFILTER_MXCR[SDMAE] enables the DMA integrator requests. The integrator DMA request uses the same signal as the filter output DMA request, so one must never use any configuration that allows both the integrator and filter output to make DMA requests.

Integrator output updates are flagged by DECFILTER_MXSR[SDF]. The integrator overrun is detected in the same way as a filter output buffer overrun, and is flagged by DECFILTER_MXSR[SVR]. An integrator overrun also generates an error interrupt if DECFILTER_MCR[ERREN] = 1 (see Section 14.3.2.1, “Output Buffer Overrun”).

Registers DECFILTER_CINTVAL and DECFILTER_CINTCNT provide a way to poll intermediate integration values and sample counts, respectively (see Section 14.2.2.12, “Decimation Filter Current Integration Value Register (DECFILT_x_CINTVAL)”, and Section 14.2.2.13, “Decimation Filter Current Integration Count Value Register (DECFILT_x_CINTCNT)”). DECFILTER_CINTVAL is updated whenever the integrator is reset or a new sample is accumulated. DECFILTER_CINTCNT is updated only when DECFILTER_CINTVAL is read, so that coherency between the value and count values is guaranteed. Therefore, the read access order of that pair of registers must be DECFILTER_CINTVAL first, followed by DECFILTER_CINTCNT.

NOTE

The flags SSOVF and SCOVF can also asserted when DECFILTER_CINTVAL is read. The SSOVF and SCOVF set and clearing rules apply for the DECFILTER_CINTVAL read the same way as for an integrator output request.

14.3.13.3 Integrator Reset

The integration value is reset to the value of zero, in the following ways:

- by hardware: on hardware reset, or controlled by an eTPU2 channel; the enabling and selection of the zero signal modes is done through DECFILTER_MXCR[SZROSEL] (see Section 14.2.2.3, “Decimation Filter Module Extended Configuration Register (DECFILT_x_MXCR)”), and eTPU2 channel selection is defined by the ZSELn fields of the SIU_DECFIL1, SIU_DECFIL2 and SIU_DECFIL3 registers in the SIU module.
- by software: on software reset, or writing 1 to the DECFILTER_MXCR bit SZRO;

The integrator reset also zeroes the internal counter of accumulated samples and the internal overflow state (but not SSOVF and SCOVF). Software and hardware reset resets all integrator registers immediately.

An Integrator zero command from an eTPU2 channel or by software (SZRO) affects the integrator registers and flags as follows:

- DECFILTER_CINTVAL resets immediately;
- DECFILTER_CINTCNT does not reset immediately; it is updated only upon a DECFILTER_CINTVAL read, loaded with the number of integrated samples occurred after the reset;

- DECFILTER_FINTVAL and DECFILTER_FINTCNT do not reset immediately; being updated only upon a new output request (see Section 14.3.13.2, “Integrator Output”); if a integrator software zero command (through SZRO bit) and an integrator output request (through SRQ bit) are made at the same time, the registers DECFILTER_FINTVAL and DECFILTER_FINTCNT are updated with the last internal values before reset; the same applies to simultaneous integrator zero command and output request by hardware signal;
- all internal overflow flags

NOTE

An integrator zero request does not negate the SSOVF and SCOVF flags

NOTE

The integrator reset does not depend on the integrator enabling (see Section 14.3.13.4, “Integrator Enabling and Halting”).

14.3.13.4 Integrator Enabling and Halting

Two mechanisms, enabling and halting, drive the integrator accumulation, allowing it to be controlled by a combination of two sources:

- both software
- both hardware (eTPU2 channels)
- one hardware (eTPU2 channel) and other software

Values are accumulated when the integrator is enabled and not halted. The integrator halt and enable states can be controlled in the following ways:

- by hardware, through eTPU2 channels; the enabling and the selection of the signal request modes is done through DECFILTER_MXCR[SENSEL] and DECFILTER_MXCR[SHLTSEL] fields, respectively (see Section 14.2.2.3, “Decimation Filter Module Extended Configuration Register (DECFILT_x_MXCR)”), and channel selection is done through the SIU_DECFIL1, SIU_DECFIL2 and SIU_DECFIL3 registers in the SIU module.
- by software, through the same DECFILTER_MXCR[SENSEL] and DECFILTER_MXCR[SHLTSEL] fields. Note that these fields are in different bytes, so that two distinct, concurrent software tasks can avoid coherency problems by changing the fields using byte read-modify-write accesses.

eTPU2 selection for the integrator enable state is defined by SIU_DECFILn[ZSELn] fields.

eTPU2 selection for the integrator halt state is defined by SIU_DECFILn[HSELn] fields.

NOTE

Enabling and halting does not affect output requests or integrator reset.

14.3.13.5 Integrator Exceptions

Integrator may run into exception states due to overflow, either of the accumulated value or the sample counter. Exceptions are flagged by the DECFILTER_MXSR bits SSE, for sum value exception, and SCE, for counter exception. These flags generate an error interrupt, if it is enabled (see Section 14.3.12, “Interrupts and DMA Overview”).

The accumulator exception condition depends on whether it operates in saturated mode or not, as follows:

- In Saturated operation (DECFILTER_MXCR[SSAT] = 1): a sum exception occurs (SSE=1) whenever an overflow is flagged; SSE asserts together with SSOVF.
- In Non-saturated operation (DECFILTER_MXCR[SSAT] = 0): a sum exception occurs (SSE=1) when an overflow is flagged and DECFILTER_MXSR[SSOVF] is already set to 1.
- In Non-saturated operation, an accumulator exception also occurs if the accumulator overflows twice without any update of the final integrator value DECFILTER_FINTVAL or the current integrator counter DECFILTER_CINTCNT (by a read to the DECFILTER_CINTVAL register), neither an integrator reset occurs. The SSOVF flag does not assert in this situation.

NOTE

The SSOVF flag can only be asserted upon a hardware request, a software request, or when DECFILTER_CINTVAL is read, based on the internal accumulator overflow state.

Similarly, the sample counter exception condition depends on whether it operates in saturated mode or not, as follows:

- In Saturated operation (DECFILTER_MXCR[SCSAT] = 1): a counter exception occurs (SCE=1) whenever an overflow is flagged; SCE asserts together with SCOVF.
- In Non-saturated operation (DECFILTER_MXCR[SCSAT] = 0): a counter exception occurs (SCE=1) when an overflow is flagged and the DECFILTER_MXSR bit SCOVF is already set to 1.
- In Non-saturated operation, a counter exception also occurs if the counter overflows twice without any update of the final count DECFILTER_FINTCNT or the current integrator counter DECFILTER_CINTCNT (by a read to the DECFILTER_CINTVAL register), neither an integration reset occurs. The SCOVF flag does not assert in this situation.

NOTE

The SCOVF flag can only be asserted upon a hardware request, a software request, or when DECFILTER_CINTVAL is read (also updating DECFILTER_CINTCNT), based on the internal counter overflow state.

14.3.14 Cascade Mode

Cascade mode is a configuration of the decimation filters where two or more filters are chained together serially to provide more complex filtering functions. All filters in the cascade arrangement are configured to operate in cascade mode using the CASCD[1:0] field in the DECFILT_x_MCR register. Figure 14-18 shows an example of a simple cascaded arrangement. This example shows the eQADC being used for both data input and output, but cascaded filters may also be configured to receive data from the CPU/DMA. The ‘head’ receives the raw data to be filtered from the eQADC. The bottom block, or ‘tail’, is the last filter block in the chain. It sends the output result to the selected data destination. The blocks in between, or ‘middle’ blocks, do not exchange data (receive/transmit) with the eQADC (or CPU/DMA), only with the preceding and following decimation filter blocks in the cascade. Middle blocks are optional. A minimum of two blocks, one head block feeding one tail block can be used in cascade.

NOTE

> The values passed between cascaded blocks can be monitored using Enhanced Debug Monitor (see Section 14.3.15, “Enhanced Debug Monitor Description”).

The following are general considerations for creating and using cascaded filters:

- The block configurations as head, tail or middle must respect their physical connections such that all the following apply:
  - a ‘head’ block must feed a ‘middle’ or a ‘tail’ block
  - a ‘middle’ must feed another ‘middle’ block or a ‘tail’ block
  - a ‘tail’ feeds no other block, and its output will be either the CPU/DMA interface or the eQADC FIFOs.
  - A ‘head’ is fed by no other block. Its input is either the CPU/DMA interface or the eQADC.
  - Cascaded filters must be sequential, that is; Filter A feeds Filter B which feeds Filter C etc.
- As a consequence of the conditions above, there must be one and only one ‘head’ block and one and only one ‘tail’ block in a cascade.
- More than one group of physically chained blocks can form a cascade. For example, Figure 14-19 shows two physical chains, with blocks A and B configured as head and tail, respectively, forming one cascaded filter block. Two of the remaining blocks form another cascaded filter block starting with block E (head), and ending with block F (tail).
- Blocks not used in a cascaded chain can be used normally by setting the DECFILT_x_MCR[CASCD] field to 0b00, as shown in Section 14.3.14.1.
- The optional connection show from block L to block A in Section 14.3.14.1 allows block L to be configured as head or middle, feeding block A configured as middle or tail, yielding more flexibility, as in the last example of Section 14.3.14.1.
- The input to a cascaded configuration is selected by the DECFILT_x_MCR[IO_SEL] bit-field of the head block. The output target of the cascaded blocks is selected by the DECFILT_x_MCR[IO_SEL] bit-field of the tail block.

14.3.14.1 Example Configurations

The following diagrams illustrate some of the possible options for using the decimation filters in a cascaded configuration. In all illustrated examples, the DECFILT_x_MCR[IO_SEL] bits are set to select the eQADC as the source for input conversion data, and the eQADC RFIFOs as destination for the outputs of the filter(s).

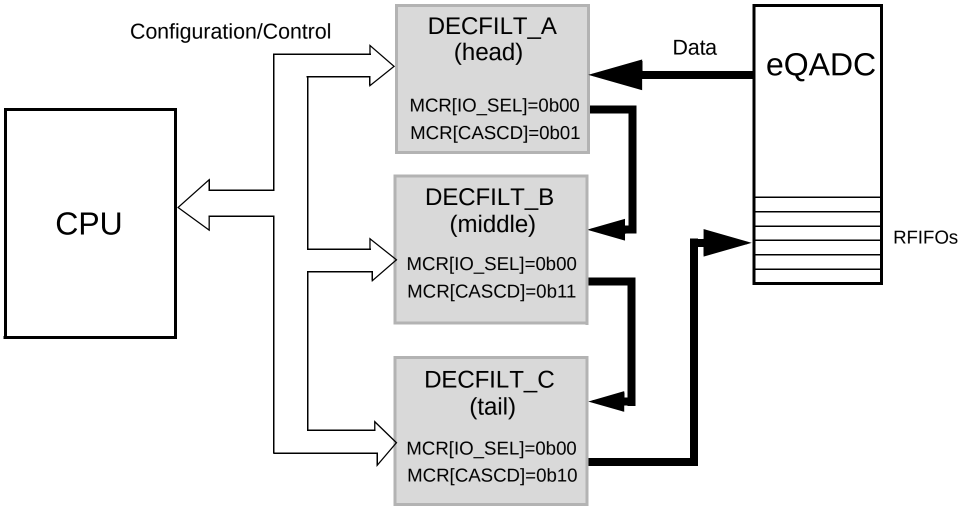


Figure 14-18. Cascaded Filters

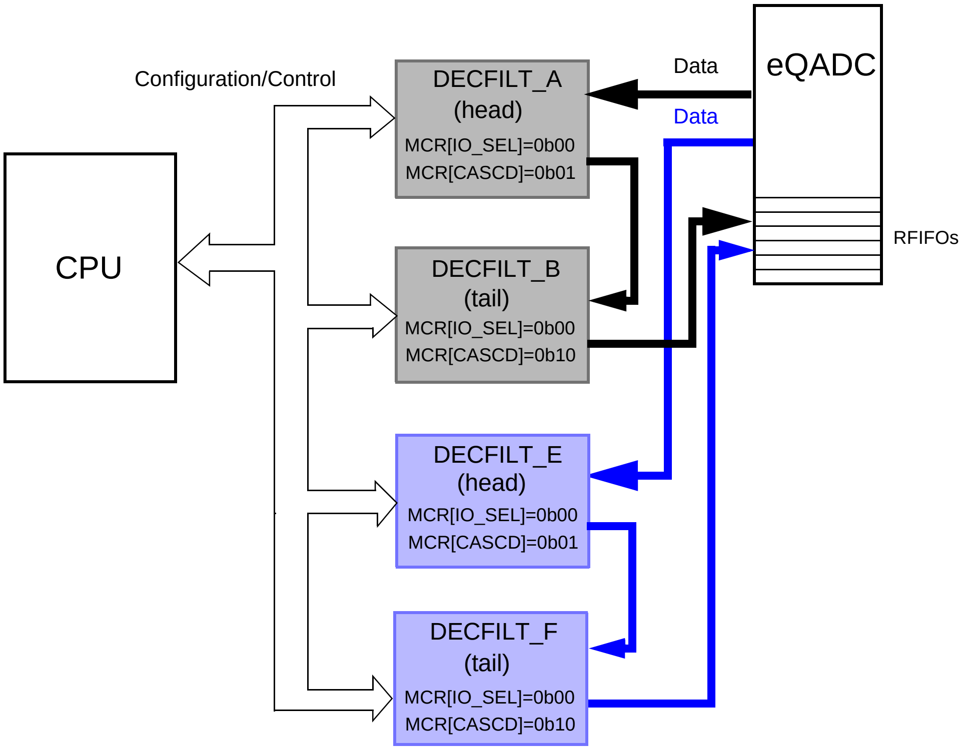


Figure 14-19. Multiple Cascaded Filters

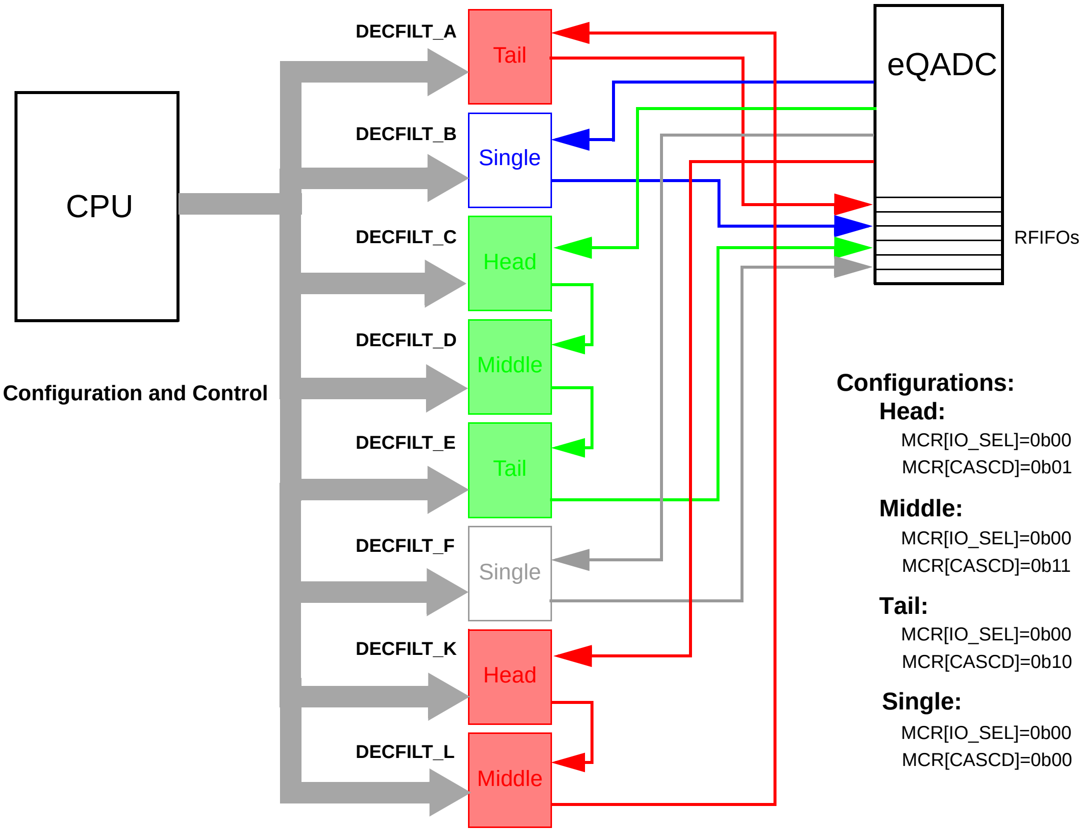


Figure 14-20. Mixed Cascaded and Single Blocks

14.3.14.2 Cascade Freeze, Stop, and Configuration Change Procedures

To change a block configuration mode to or from cascade mode, the following safe procedures must be observed:

- To modify a cascade combo, either to single or any other cascade combo combination, all the cascade combo blocks must have their inputs disabled (using DECFILT_x_MCR[IDIS]), in order, from the Head to the Tail block. After a block IDIS bit has been set to 1 (one), one must wait for its DECFILTER_x_MSR bit BSY to be 0 (zero) before disabling the input of the next block in the sequence.
- Each block in a new cascade combo must be configured with its input disabled. When the mode configuration is done, the combo blocks must have their inputs enabled in order, from the Tail towards the Head block.
- A single block must also be reconfigured the same way, to or from a cascade combo configuration: first disabling its input, and then waiting for a non-busy state before writing DECFILTER_x_MCR field CASCD.

To take cascade combo blocks to or from freeze or low power modes, a similar procedure must be used:

- Take the Head to freeze or low-power first, wait for DECFILTER_x_MSR bit BSY=0, and repeat the procedure for the other blocks in the chain in sequence, towards the Tail block.

Take the blocks out of freeze or low-power modes in the inverse sequence, from Tail to Head.

14.3.15 Enhanced Debug Monitor Description

This feature is enabled by the EDME bit in the configuration register DECFILTER_x_MCR. The monitoring operation is applicable only to an eQADC input data source, and applies when filters are cascaded. The Enhanced Debug Monitor feature makes the input sample data also available in the DECFILTER_x_EDID register. A DMA or interrupt request (selected by DECFILT_x_MCR[DSEL]) indicates a new input was fed and DECFILTER_x_EDID was updated. The input is processed normally by the filter.

An Enhanced Debug Input Data Register (DECFILTER_x_EDID) overrun can occur if a sample is not read by the CPU or DMA before overwritten by a new sample. The overrun is indicated in a separate flag DIVR in the status register DECFILTER_x_MSR. If the ERREN bit is set in the DECFILTER_x_MCR configuration register, this overrun asserts the an interrupt request.

14.4 Application Information

The following sections describe common use cases and configurations of the decimation filter block.

14.4.1 eQADC Configuration for Decimation Filter Operation

14.4.1.1 eQADC Configuration / Decimation Filter Input

In normal mode of operation of the Decimation Filter, filter data inputs are supplied by an eQADC block and the filter output is returned to the specified RFIFO in the eQADC. In standard eQADC operation, conversion results are routed directly from the converter to one of the local eQADC RFIFO buffers. However, by using a Conversion Command Format for Alternate word configurations in the eQADC, conversion results can be routed to a Decimation Filter block specified by an Alternate Configuration register. Additionally, the same eQADC conversion results can be routed to a second Decimation Filter block specified by an Extended Alternate Configuration register. The eQADC can send either conversion data or timestamp data. The conversion data is filtered by the decimation filter and the timestamp is bypassed and delayed until ready to be sent back to the eQADC when the relevant conversion data is filtered and available.

In the eQADC, the ALT_CONFIG_SEL field of the Conversion Command Word (CCW) specifies whether an alternate configuration is used for the ADC conversion (see Figure 19-47)(see “Figure 19-47”). The values for this field are given in Table 19-36. During an ADC conversion, any of the alternate configuration registers stored in the eQADC may be selected by the ALT_CONFIG_SEL field in the CCW. When the ALT_CONFIG_SEL field is set to 0x00, no alternate configuration is used, and the ADC conversion is processed according to the CCW.

Each eQADC alternate configuration uses the format specified in Figure 19-38, while the extended alternate configuration uses the format specified in Figure 19-41.

The Alternate and Extended Alternate Configurations are stored in the ADC by executing write commands to the On-Chip ADC Alternate and Extended Alternate Configuration Registers. The write command is specified in the Write Configuration Command Format for On-Chip ADC Operation section of the reference manual. The Alternate and Extended Alternate Configuration register addresses an eQADC are given in Table 19-20 and described in detail in the Section 19.6.3.6, "Alternate Configuration 1-14 Control Registers (ADC_ACR1-14)" and Section 19.6.3.7, "ADC0/1 Alternate Gain Registers (ADC0_AGR1-2 and ADC1_AGR1-2)".

14.5 Use Cases

The following use cases provide examples of how to configure the decimation filter for different modes. One use case describes the mode that filters the conversion results that directly transferred from the ADC. In this case, the appropriate software initialization of the ADC configuration register is described to enable the transfer of an ADC result to the Decimation Filter module without CPU intervention. The second use cases describes the stand-alone mode, where data previously stored in a memory table is transferred into the filter and subsequent filtered data extracted using CPU interaction. For all uses cases, the Decimation Filter is configured as an IIR filter, with example filter characteristics. Because all ADC results are routed to only one filter in this example, the Extended Alternate Configuration is unused. An explanation of the IIR configuration and coefficient selection is give in the next section.

14.5.1 IIR Filter Configuration

This section describes the topology of the IIR filter that is implemented in the Decimation Filter hardware and provides a definition of the coefficients for an elliptical low pass, 4th order IIR filter. Figure 14-21 shows the filter functional diagram, consisting of 4 feed-forward stages and 4 feedback stages, to provide a maximum of a 4th order IIR filter.

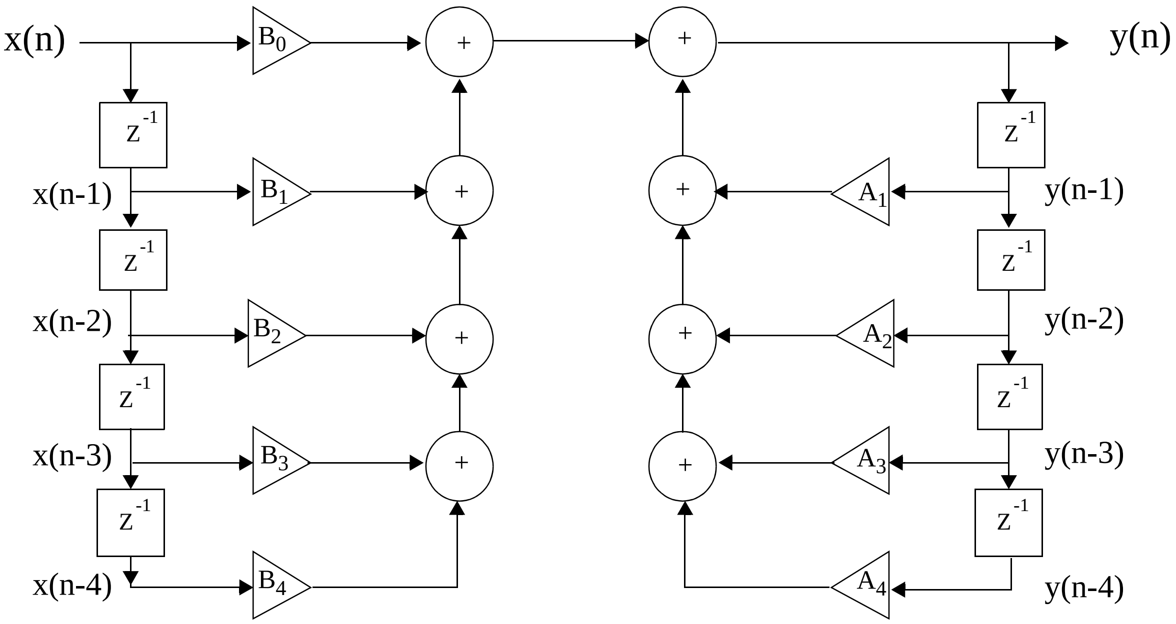


Figure 14-21. 1 x 4 Poles IIR Filter Functional Diagram

The generalized difference equation for the IIR filter of Figure 14-21 can be written as:

Eqn. 14-1

$$y(n) = \sum_{i=0}^{N} B_i x(n-i) + \sum_{j=1}^{M} A_j y(n-j)$$

where *x(n)* is the filter input at time *n*, *y(n)* is the filter output at time *n*, *N* is the number of feed-forward filter coefficients minus one, B_i are the feed-forward filter coefficients, *M* is the number of feed-back filter coefficients, and A_j are the feedback filter coefficients.

In order to optimize the hardware implementation, the coefficients must be scaled to a maximum range of +1 to -1. Taking scaling into account, Equation 14-1 can be expressed as:

Eqn. 14-2

$$y(n) = S\left\{\sum_{i=0}^{N} \frac{B_i}{S} x(n-i) + \sum_{j=1}^{M} \frac{A_j}{S} y(n-j)\right\}$$

All coefficients are scaled down by S, and the output of the accumulator is multiplied by S.

The block diagram shown in Figure 14-22 represents the hardware implementation of the Decimation Filter, which includes the scale factor hardware that compensates for the scaling of the coefficients. The scale factor, S, is set by the bit field DECFILT_x_MCR[SCAL].

Coefficients B_0 to B_4 correspond to DECFILT_x_COEF0 to DECFILT_x_COEF4

Coefficients A_1 to A_4 correspond to DECFILT_x_COEF5 to DECFILT_x_COEF8.

The delay taps labelled Tap0 to Tap7 correspond to the memory mapped registers DECFILT_x_TAP0 to DECFILT_x_TAP7.

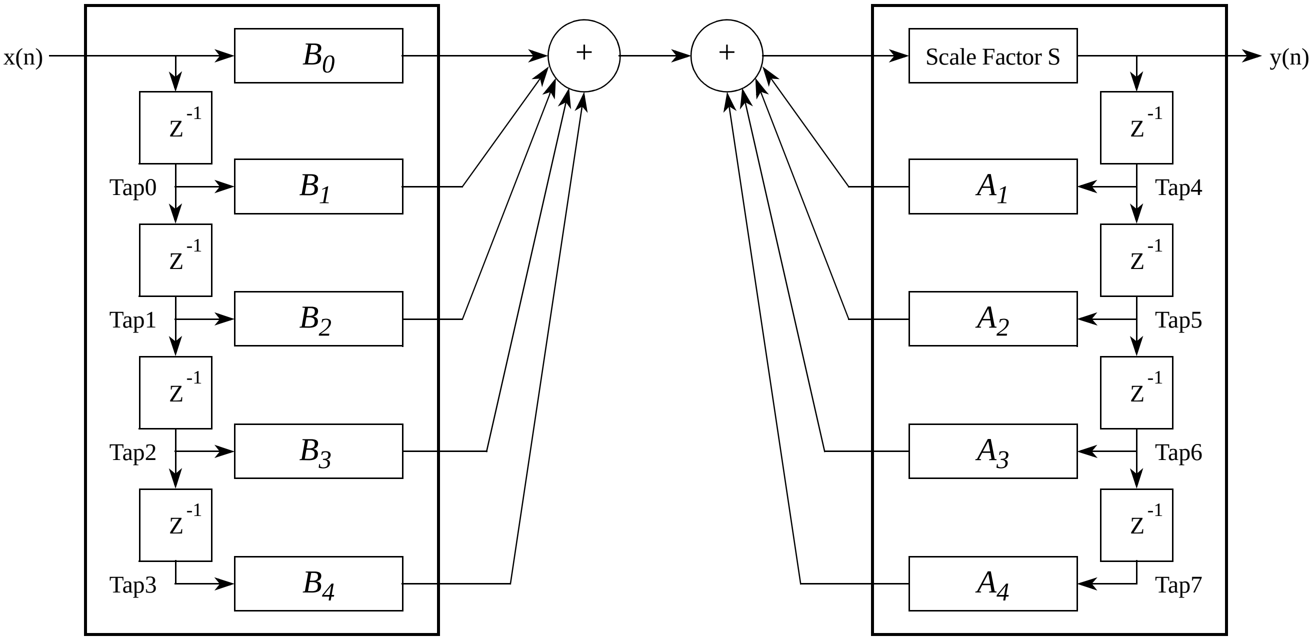


Figure 14-22. Fourth Order IIR Filter Implementation Block Diagram

The examples given here implement a filter with the following characteristics:

- Filter type: elliptic/low pass IIR
- Filter order: 4th order
- Input sample rate: 800k sample/s
- Passband edge: 100 kHz
- Stopband edge: 150 kHz
- Passband attenuation: ≤ 1 dB

Table 14-17 lists the computed coefficients, in real number notation, to achieve the above characteristics.

Table 14-17. Computed Coefficient Values

Coefficient	Decimal Value	Coefficient	Decimal Value
B0	0.0221455	A0	-1.0
B1	0.00445582948893748	A1	2.69772868375858

Table 14-17. Computed Coefficient Values (continued)

Coefficient	Decimal Value	Coefficient	Decimal Value
B2	0.0318517846509088	A2	-3.234056294853
B3	0.00445582948893748	A3	1.92028561712454
B4	0.0221455	A4	-0.47939080709495

Since the number range of the computed coefficients exceeds the range of the hardware, the coefficients must be scaled, by at least a factor of 4. In this example, a factor of 8 is chosen. Table 14-18 below lists the scaled values and their 24 bit signed fractional values, in hexadecimal notation.

Table 14-18. Coefficient Values for Decimation Filter

	Computed Value	Scaled Value (SCAL=8)	
Filter Coefficients	Decimal Value	Decimal Value	Hexadecimal Values (24 bits)
DECFILT_x_COEF0 = B0/SCAL	0.0221455	0.00276815891266	0x005AB5
DECFILT_x_COEF1 = B1/SCAL	0.00445582948893748	0.00055694580078	0x001240
DECFILT_x_COEF2 = B2/SCAL	0.0318517846509088	0.00398147106171	0x008277
DECFILT_x_COEF3 = B3/SCAL	0.00445582948893748	0.00055694580078	0x001240
DECFILT_x_COEF4 = B4/SCAL	0.0221455	0.00276815891266	0x005AB5
DECFILT_x_COEF5 = A1/SCAL	2.69772868375858	0.33721613883972	0x2B29E6
DECFILT_x_COEF6 = A2/SCAL	–3.234056294853	–0.40425717830658	0xCC414E
DECFILT_x_COEF7 = A3/SCAL	1.92028561712454	0.24003565311432	0x1EB97D
DECFILT_x_COEF8 = A4/SCAL	–0.47939080709495	–0.05992400646210	0xF8546A

14.5.2 Initialization Procedure

Specific initialization for two different operating modes is provided in the use case sections that follow this section. The general sequence of steps for initializing one Decimation Filter is:

1. Program the configuration registers DECFILT_x_MCR to configure the input data source and output result destination.
2. Write all filter coefficient registers DECFILT_x_COEFn with values that define the filter frequency response characteristics.
3. Execute a soft reset if the filter was previously configured, by writing DECFILT_x_MCR[SRES] = 1.
4. The module is ready to receive data and perform a filtering function.
5. If the input data source is an eQADC, these additional steps must be executed:
    a) Configure and enable the eQADC converter to transfer results to the filter.
    b) Configure and enable the eQADC commands to transfer filtered results to desired RFIFO.

   c) Configure and enable a DMA channel or CPU interrupt handler to transfer result from RFIFO to memory.

6. If the input data source is the CPU/DMA, configure a CPU interrupt handler or polling software to input unfiltered data and retrieve the filter data.

14.5.2.1 Use Case 1 - Normal mode, ADC conversion and filtering.

The input to Filter A is from an ADC conversion result.

Filter A is configured as a 4rd order low pass IIR.

The output from the filter is routed to RFIFO5 and transferred to memory by the DMA.

Decimation is not enabled.

Saturation is enabled.

14.5.2.1.1 DECFILT_A_MCR — Module Configuration Register settings

Enable IIR Filter type: FTYPE=1

Scale coefficients to a range of +1 to -1: SCAL=8

Enable saturation of filter output result: SAT=1

Select ‘normal” mode, to transfer data from ADC through filter, to RFIFO: IO_SEL[1]=0

Select no decimation: DEC_RATE=0

14.5.2.1.2 DECFILT_A_COEFn

Write the filter coefficients given in Table 14-18 to the following registers.

DECFILT_A_COEF0 = 0x005AB5

DECFILT_A_COEF1 = 0x001240

DECFILT_A_COEF2 = 0x008277

DECFILT_A_COEF3 = 0x001240

DECFILT_A_COEF4 = 0x005AB5

DECFILT_A_COEF5 = 0x2B29E6

DECFILT_A_COEF6 = 0xCC414E

DECFILT_A_COEF7 = 0x1EB97D

DECFILT_A_COEF8 = 0xF8546A

14.5.2.1.3 eQADC Configuration for Decimation Filter

Configure an Alternate Configuration Control Register (ADC_ACR1 to ADC_ACR8) to select Decimation Filter A as the destination for an ADC conversion, by setting ADC_ACRn[DEST] = 1. To select a signed format, ADC_ACRn[FMTA]=1.

The ADC_ACRn register format has the form:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RET_INH	0	DEST				FMTA	0	RESSEL		0	0	ATBSEL		PRE_GAIN	
W																

Note that the ADC_ADRn registers are internal to the ADC and can only be accessed indirectly by writing a register write command to the eQADC CFIFO. To do this the CPU should write a command to any eQADC CFIFO with the following format:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
EOQ	PAUSE	REP	RESERVED		EB (0b0)	BN	R/W (0b0)	ADC_REGISTER HIGH BYTE							
CFIFO Header						ADC Command									

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
ADC_REGISTER LOW BYTE								ADC_REG_ADDRESS							
ADC Command															

For example, to update ADC_ACR1[DEST] = 1, ADC_ACR1[FMTA]=1

- set the bit field ADC_REG_ADDRESS = 0x30
- set the bit field ADC_REGISTER LOW BYTE = 0x00
- set the bit field ADC_REGISTER HIGH BYTE = 0x06

14.5.2.1.4 eQADC Command

To cause the ADC conversion to be transferred to the Decimation filter selected by the method described in Section 14.5.2.1.3, a Conversion Command for Alternate Configurations must be applied to the eQADC CFIFO. The form of the command is as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
EOQ	PAUSE	REP	RESERVED		EB (0b0)	BN	CAL	MESSAGE_TAG				LST		TSR	FFMT
CFIFO Header						ADC Command									

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
CHANNEL_NUMBER								ALT_CONFIG_SEL							
ADC Command															

The ALT_CONFIG_SEL field should be set to select the ADC_ACRn register that was configured in Section 14.5.2.1.3. For example, if ADC_ACR1 were selected, then the ALT_CONFIG_SEL field should be 0x08.

14.5.2.2 Use Case 2 - Input/Output from/to the CPU/DMA, Stored data filtering.

The input to Filter C is through the memory mapped input register.

Filter C is configured as a 4th order low pass IIR.

The output from the filter is routed to a memory mapped output register and an interrupt is issued when new data is available.

In this example the ADC is not used at all, and no ADC configuration or commands are needed to support the Decimation Filter operation.

Decimation is not enabled.

Saturation is enabled.

14.5.2.2.1 DECFILT_C_MCR — Module Configuration Register settings

Enable interrupt when new filtered data is available: ODEN=1

Enable IIR Filter type: FTYPE=1

Scale coefficients to a range of +1 to -1: SCAL=8

Enable saturation of filter output result: SAT=1

Select the CPU/DMA as the output destination to transfer data to memory mapped output register: DECFILT_x_MCR[IO_SEL[0:1] = 01 or 10

Select no decimation: DEC_RATE=0

14.5.2.2.2 DECFILT_C_COEFn

Write the filter coefficients given in Table 14-18 to the following registers.

DECFILT_C_COEF0 = 0x005AB5

DECFILT_C_COEF1 = 0x001240

DECFILT_C_COEF2 = 0x008277

DECFILT_C_COEF3 = 0x001240

DECFILT_C_COEF4 = 0x005AB5

DECFILT_C_COEF5 = 0x2B29E6

DECFILT_C_COEF6 = 0xCC414E

DECFILT_C_COEF7 = 0x1EB97D

DECFILT_C_COEF8 = 0xF8546A

14.5.2.2.3 CPU Interrupt Handler

All sources of interrupts from the Decimation Filters A, B, C and D are routed to separate single vector numbers. Each of the remaining Decimation Filters has a single interrupt vector for all interrupt sources. Decimation Filter C uses vector numbers 467 to 469.

Set INTC vector number 468 to the address of Decimation Filter C output data interrupt handler.

Set the priority for the Decimation Filter and enable interrupts by lowering the processor priority.

When the interrupt occurs, if DECFILT_C_MSR[OBIF] = 1 then this indicates filter output data is available.

In this case, read the DECFILT_C_OB, then clear the output data interrupt by writing DECFILT_C_MSR[OBIC] = 1.

Note that DECFILT_C_MSR[IBIF] should also be set when the output data interrupt occurs, as a result of the previous write to the input data register, so it should be cleared also by writing DECFILT_C_MSR[IBIC] = 1. The CPU can now also write a new input value to the DECFILT_C_IB register.

The error flags OVF, OVR and IVR should also be checked at this point, and clear if set, and remedial action taken, if desired by the application.

Chapter 15
Deserial Serial Peripheral Interface (DSPI)

15.1 Introduction

Figure 15-1 is a block diagram of the Deserial Serial Peripheral Interface (DSPI) module.

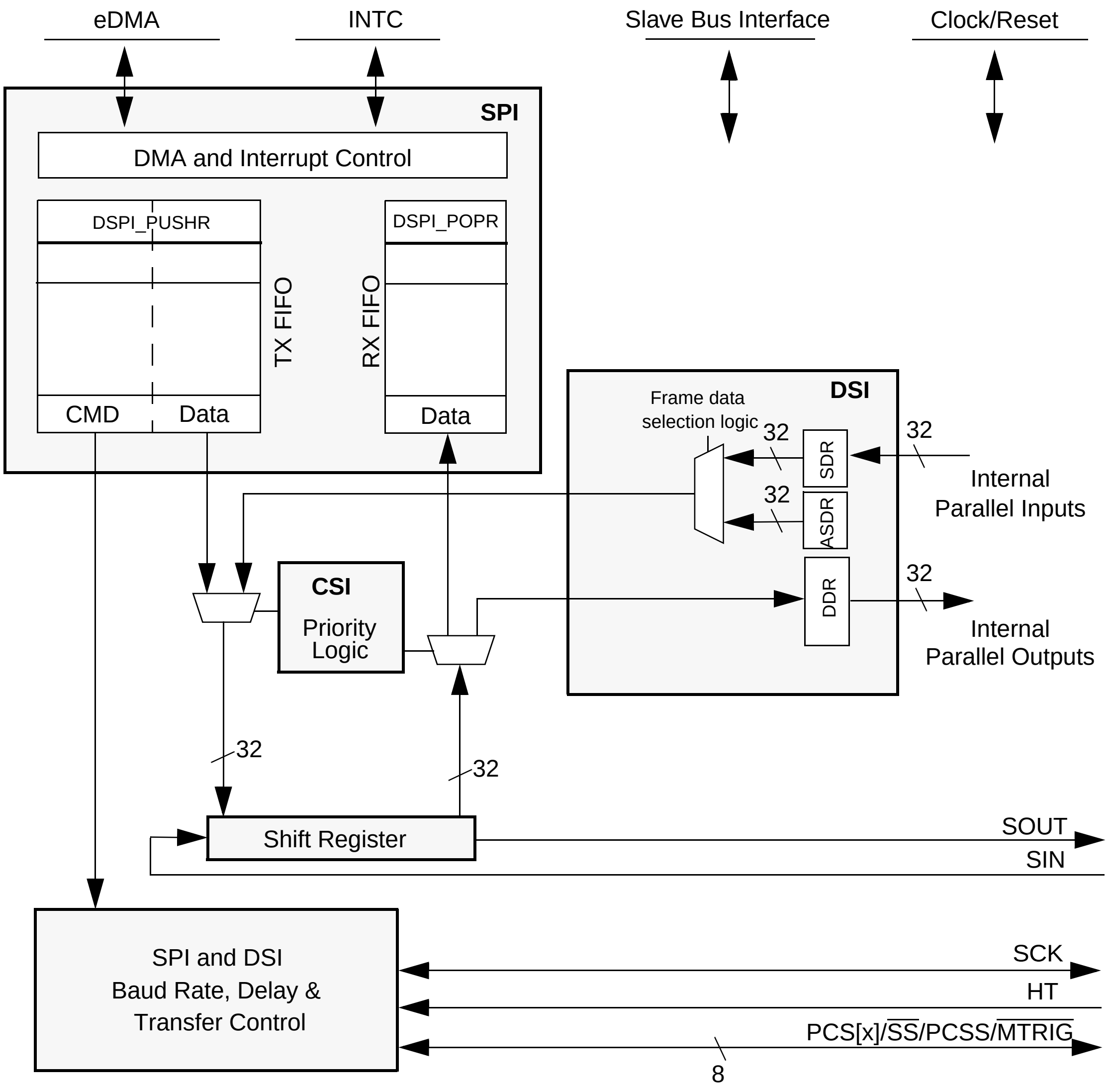


Figure 15-1. DSPI block diagram

15.2 Overview

The Deserial Serial Peripheral Interface (DSPI) module provides a synchronous serial interface for communication between the MPC5676R and external devices. The DSPI supports pin count reduction through serialization and deserialization of eTPU channels, eMIOS channels and memory-mapped registers. Incoming deserialized data can also be used to trigger external interrupt requests. The channels and register content are transmitted using a SPI-like protocol. There are five identical DSPI modules (DSPI_A, DSPI_B, DSPI_C, DSPI_D and DSPI_E) on the MPC5676R.

The DSPIs have three configurations:

- Serial Peripheral Interface (SPI)—DSPI operates as a SPI with support for queues
- Deserial Serial Interface (DSI)—DSPI serializes eTPU and eMIOS output channels and deserializes the received data by placing it on the eTPU and eMIOS input channels and as inputs to the External Interrupt Request sub-block of the SIU
- Combined Serial Interface (CSI)—DSPI operates in both SPI and DSI configurations interleaving DSI frames with SPI frames, giving priority to SPI frames

For queued operations, the SPI queues reside in system memory external to the DSPI. Data transfers between the memory and the DSPI FIFOs are accomplished through the use of the eDMA controller or through host software.

15.3 Features

The DSPI supports these SPI features:

- Full-duplex, synchronous transfers
- Selectable LVDS Pads working at 40 MHz for SOUT and SCK pins (only in DSPI_C)
- Master and Slave Mode
- Buffered transmit operation using the TX FIFO with depth of 4 entries
- Buffered receive operation using the RX FIFO with depth of 4 entries
- TX and RX FIFOs can be disabled individually for low-latency updates to SPI queues
- Visibility into the TX and RX FIFOs for ease of debugging
- FIFO Bypass Mode for low-latency updates to SPI queues
- Programmable transfer attributes on a per-frame basis:
  - Parameterized number of transfer attribute registers (from 2 to 8)
  - Serial clock with programmable polarity and phase
  - Various programmable delays:
    - PCS to SCK delay
    - SCK to PCS delay
    - Delay between frames
  - Programmable serial frame size of 4 to 32 bits, expandable with software control
  - Continuously held chip select capability
- Up to 6 Peripheral Chips Selects, expandable to 64 with external demultiplexer

- Deglitching support for up to 32 Peripheral Chip Selects with external demultiplexer
- DMA support for adding entries to TX FIFO and removing entries from RX FIFO:
  - TX FIFO is not full (TFFF)
  - RX FIFO is not empty (RFDF)
- 6 interrupt conditions:
  - End of queue reached (EOQF)
  - TX FIFO is not full (TFFF)
  - Transfer of current frame complete (TCF)
  - Attempt to transmit with an empty Transmit FIFO (TFUF) ‘OR’ Serial frame received while RX FIFO is full (RFOF). These two interrupts are ORed and given out as FIFO Overrun interrupt.
  - RX FIFO is not empty (RFDF)
  - FIFO Underrun (slave only and SPI mode, the slave is asked to transfer data when the Tx FIFO is empty)
- Modified transfer formats for communication with slower peripheral devices
- Continuous Serial Communications Clock (SCK)
- Enhanced DSI logic to implement a 32-bit Timed Serial Bus (TSB) configuration, supporting the Micro Second Channel (MSC) bus downstream frame format

The DSPIs also support these features unique to the DSI and CSI configurations:

- 2 sources of the serialized data:
  - eTPU_A and eMIOS output channels
  - Memory-mapped register in the DSPI
- Destinations for the deserialized data:
  - eTPU_A and eMIOS input channels
  - SIU External Interrupt Request inputs
  - Memory-mapped register in the DSPI
- Deserialized data is provided as Parallel Output signals and as bits in a memory-mapped register
- Transfer initiation conditions:
  - Continuous
  - Edge sensitive hardware triggered
  - Change in data
- Pin serialization/deserialization with interleaved SPI frames for control and diagnostics
- Continuous serial communications clock
- Support for parallel and serial chaining of up to 4 DSPI modules
- Parity generation and checking

15.4 DSPI configurations

The DSPI module can operate in three configurations: SPI, DSI and CSI.

15.4.1 SPI configuration

The SPI configuration allows the DSPI to send and receive serial data. This configuration allows the DSPI to operate as a basic SPI block with internal FIFOs supporting external queues operation. Transmit data and received data reside in separate FIFOs. The host CPU or a DMA controller read the received data from the receive FIFO and write transmit data to the transmit FIFO.

For queued operations the SPI queues can reside in system RAM, external to the DSPI. Data transfers between the queues and the DSPI FIFOs are accomplished by a DMA controller or host CPU. Figure 15-2 shows a system example with DMA, DSPI and external queues in system RAM.

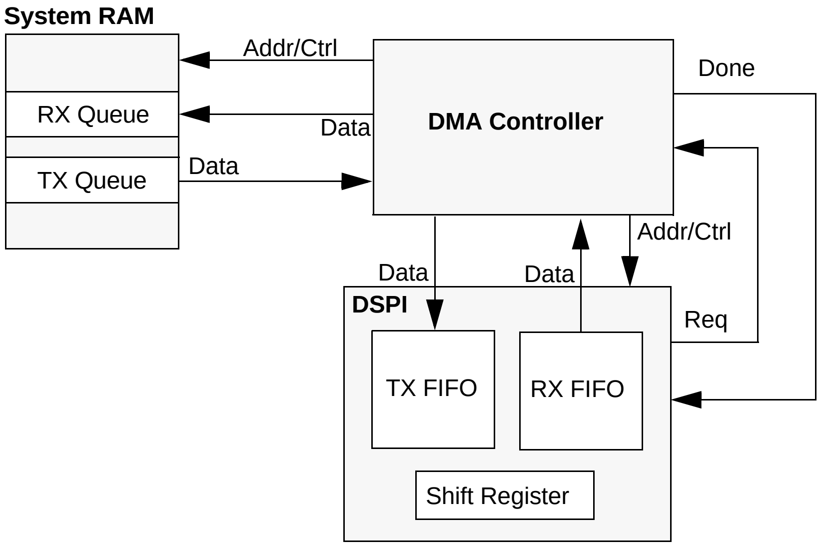


Figure 15-2. DSPI with queues and DMA

15.4.2 DSI configuration

The DSI configuration supports pin count reduction by serializing eTPU and eMIOS output channels or bits from a memory-mapped register and shifting them out with a SPI-like protocol. The DSPI deserializes the received data, and provides the received data to the eTPU's and eMIOS' input channels, the SIU IRQ inputs, or to a memory-mapped register in the DSPI. See Section 15.9.17, “DSPI connections to eTPU_A, eMIOS and SIU” for the source of the serialization data for each DSPI module.

Figure 15-3 shows an example of how a master DSPI block connects to a DSI slave in DSI configuration.

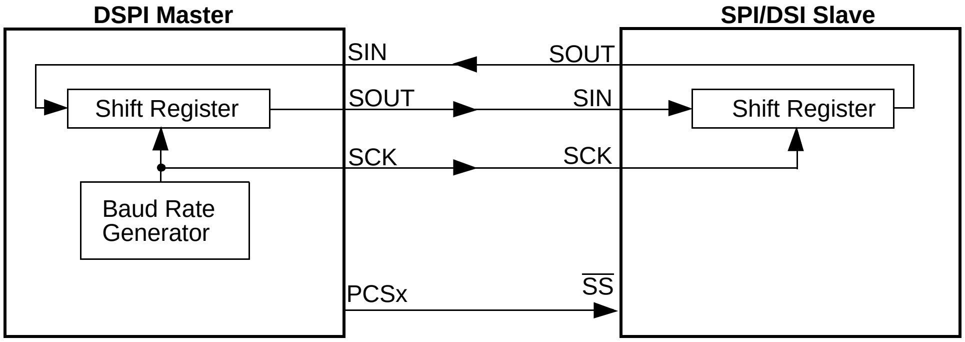


Figure 15-3. DSPI connections for SPI and DSI transfers

Specifically in the TSB configuration, detailed in Section 15.9.8, “Timed serial bus (TSB)”, the DSPI serializes from 4 to 32 Parallel Input signals or register bits. The TSB downstream frame used to communicate with a single slave is shown in Figure 15-37.

15.4.3 CSI configuration

The CSI configuration allows serialized data to be interleaved with configuration or diagnostic data and be transferred to a slave device using only one serial link. The CSI configuration supports SPI and DSI functionality on a frame by frame basis. CSI configuration allows interleaving of DSI data frames from the eTPU’s and eMIOS’ output channels with SPI commands and data from the TX FIFO. In the CSI configuration, transmission of SPI data has higher priority than DSI data. The data returned from the bus slave is either used to drive the eTPUs or eMIOS input channels, or the data is stored in the RX FIFO. The DSPI only supports CSI configuration in Master Mode. Figure 15-4 shows an example of how a DSPI can be used with a deserializing peripheral that supports SPI control for control and diagnostic frames.

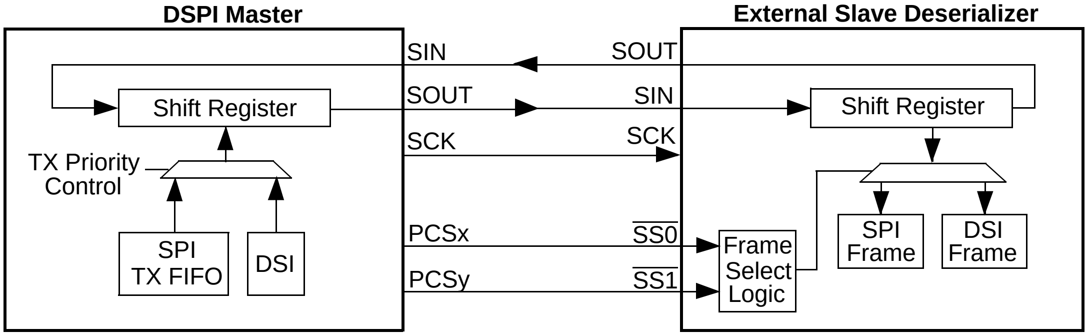


Figure 15-4. DSPI Connections for CSI Transfer

15.5 DSPI frequency support

The DSPI supports frequencies up to 40 MHz when used with LVDS output (DSPI_C only), and frequencies up to 20 MHz in non-LVDS mode. Table 15-1 shows possible divider settings to achieve maximum frequency for different system clock frequencies.

Table 15-1. DSPI channel frequency support

System clock (MHz)	DSPI use mode	Max. usable frequency (MHz)	Notes
150	LVDS	37.5	Use sys clock/4 divide ratio
	Non-LVDS	18.75	Use sys clock /8 divide ratio
120	LVDS	40	Use sys clock /3 divide ratio. Gives 33/66 duty cycle. Use DSPI configuration DBR = 0b1 (double baud rate), BR = 0b0000 (scaler value 2) and PBR = 0b01 (prescaler value 3).
	Non-LVDS	20	Use sys clock /6 divide ratio
80	LVDS	40	Use sys clock /2 divide ratio
	Non-LVDS	20	Use sys clock /4 divide ratio

15.6 Modes of operation

The DSPI has four modes of operation that can be divided into two categories: module-specific modes and an MCU-specific mode. Master Mode, Slave Mode and Module Disable Mode are the module-specific modes, and Debug Mode is the MCU-specific mode.

The module-specific modes are entered by host software writing to a register bit. The MCU-specific mode is selected by a signal external to the DSPI. The MCU-specific mode is a mode that MPC5676R may enter in parallel to the DSPI being in one of its module-specific modes.

15.6.1 Master mode

Master Mode allows the DSPI to initiate and control serial communication. In this mode the SCK, DSPI_x_PCS and SOUT signals are controlled by the DSPI and configured as outputs.

15.6.2 Slave mode

Slave Mode allows the DSPI to communicate with SPI/DSI bus masters. In this mode the DSPI responds to externally controlled serial transfers. The DSPI cannot initiate serial transfers in Slave Mode.

15.6.3 Module Disable mode

The Module Disable mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI is stopped while in Module Disable Mode. The DSPI enters the Module Disable Mode when bit DSPI_MCR[MDIS] is set.

15.6.4 Debug mode

Debug Mode is used for system development and debugging. If MPC5676R MCU enters Debug Mode while bit DSPI_MCR[FRZ] is set, the DSPI halts operation on the next frame boundary. If the MPC5676R enters Debug Mode while the FRZ bit is negated, the DSPI behavior is unaffected and remains dictated by the module-specific mode and configuration of the DSPI.

15.7 External signal description

15.7.1 Overview

Table 15-2 lists the signals that may connect off-chip depending on the device implementation.

Table 15-2. Signal properties

Name	I/O type	Function	
		Master mode	Slave mode
DSPI_x_PCS[0]/$\overline{SS}$	Output / Input	Peripheral Chip Select 0	Slave Select
DSPI_x_PCS[1] – PCS[3]	Output	Peripheral Chip Select 1 – 3	Unused
DSPI_x_PCS[4]/$\overline{MTRIG}$	Output	Peripheral Chip Select 4	Master Trigger
DSPI_x_PCS[5]/$\overline{PCSS}$	Output	Peripheral Chip Select 5 / Peripheral Chip Select Strobe	Unused
DSPI_x_SIN	Input	Serial Data In	Serial Data In
DSPI_x_SOUT	Output	Serial Data Out	Serial Data Out
DSPI_x_SCK	Output / Input	Serial Clock (output)	Serial Clock (input)
HT	Input	Hardware Trigger	Hardware Trigger

15.7.2 Detailed signal description

15.7.2.1 DSPI_x_PCS[0]/$\overline{SS}$ — Peripheral Chip Select/Slave Select

In master mode, the DSPI_x_PCS[0] signal is a Peripheral Chip Select output that selects which slave device the current transmission is intended for.

In slave mode, the active low $\overline{SS}$ signal is a Slave Select input signal that allows a SPI master to select the DSPI as the target for transmission.

15.7.2.2 DSPI_x_PCS[1] – PCS[3] — Peripheral Chip Selects 1 – 3

DSPI_x_PCS[1] – PCS[3] are Peripheral Chip Select output signals in master mode. In slave mode these signals are unused.

15.7.2.3 DSPI_x_PCS[4]/$\overline{\text{MTRIG}}$ — Peripheral Chip Select 4/Master Trigger

In master mode, DSPI_x_PCS[4] is a Peripheral Chip Select output signal.

In slave mode, the active low $\overline{\text{MTRIG}}$ is an output trigger signal that indicates that a change in data to be serialized has occurred. The $\overline{\text{MTRIG}}$ provides a pulse in DSI configuration when a change in data to be serialized occurs. The $\overline{\text{MTRIG}}$ pulse is four system clock cycles in duration. If the DSPI is in slave mode and the MTO is disabled, the DSPI_x_PCS[4]/$\overline{\text{MTRIG}}$ signal is unused.

15.7.2.4 DSPI_x_PCS[5]/$\overline{\text{PCSS}}$ — Peripheral Chip Select 5/Peripheral Chip Select Strobe

DSPI_x_PCS[5] is a Peripheral Chip Select output signal. When the DSPI is in master mode and the DSPI_MCR[PCSSE] bit is cleared, this signal selects which slave device the current transfer is intended for.

When the DSPI is in master mode and the DSPI_MCR[PCSSE] bit is set, the $\overline{\text{PCSS}}$ signal acts as a strobe to external peripheral chip select demultiplexer, which decodes the DSPI_x_PCS[0] – PCS[4] and DSPI_x_PCS[6] – PCS[7] signals, preventing glitches on the demultiplexer outputs.

This signal is not used in slave mode.

15.7.2.5 DSPI_x_SIN — Serial input

DSPI_x_SIN is a serial data input signal.

15.7.2.6 DSPI_x_SOUT — Serial output

DSPI_x_SOUT is a serial data output signal.

15.7.2.7 DSPI_x_SCK — Serial clock

DSPI_x_SCK is a serial communication clock signal. In master mode, the DSPI generates the SCK. In slave mode, SCK is an input from an external bus master.

15.7.2.8 HT — Hardware trigger

HT is a trigger input signal that is used with Multiple Transfer Operations in DSI configuration.

In master mode while in DSI or CSI configurations, the HT signal initiates a data transfer when the TRRE bit in the DSPI_DSICR is set and a rising or falling edge is detected on HT. Which edge to trigger on is determined by the TPOL bit in the DSPI_DSICR.

In slave mode, the DSPI generates a trigger pulse on the $\overline{\text{MTRIG}}$ pin, when a rising or falling edge is detected on HT. Which edge that generates an output pulse is selected by the TPOL bit in the DSPI_DSICR.

15.8 Memory map and register definition

15.8.1 Memory map

Register accesses to memory addresses that are reserved or undefined result in a transfer error. Write access to the DSPI_POPR also result in a transfer error.

Table 15-3. DSPI Base Addresses

DSPI Base	Addresses
DSPI_A	0xFFF9_0000
DSPI_B	0xFFF9_4000
DSPI_C	0xFFF9_8000
DSPI_D	0xFFF9_C000
DSPI_E	0xFFFA_0000

Table 15-4 shows the DSPI memory map.

Table 15-4. Memory map

Address	Register name	Location
DSPI_BASE	DSPI Module Configuration Register (DSPI_MCR)	on page 15-11
DSPI_BASE+0x4	Reserved	
DSPI_BASE+0x8	DSPI Transfer Count Register (DSPI_TCR)	on page 15-14
DSPI_BASE+0xC – DSPI_BASE+0x28	DSPI Clock and Transfer Attributes Register 0 (DSPI_CTAR0) – DSPI Clock and Transfer Attributes Register 7 (DSPI_CTAR7)	on page 15-14
DSPI_BASE+0x2C	DSPI Status Register (DSPI_SR)	on page 15-20
DSPI_BASE+0x30	DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)	on page 15-23
FIFO Registers		
DSPI_BASE+0x34	DSPI Push TX FIFO Register (DSPI_PUSHR)	on page 15-25
DSPI_BASE+0x38	DSPI Pop RX FIFO Register (DSPI_POPR)	on page 15-27
DSPI_BASE+0x3C – DSPI_BASE+0x48	DSPI Transmit FIFO Register 0 (DSPI_TXFR0) – DSPI Transmit FIFO Register 3 (DSPI_TXFR3)	on page 15-28
DSPI_BASE+0x4C – DSPI_BASE+0x78	Reserved	
DSPI_BASE+0x7C – DSPI_BASE+0x88	DSPI Receive FIFO Register 0 (DSPI_RXFR0) – DSPI Receive FIFO Register 3 (DSPI_RXFR3)	on page 15-29

Table 15-4. Memory map (continued)

Address	Register name	Location
DSPI_BASE+0x8C – DSPI_BASE+0xB8	Reserved	
DSI Registers		
DSPI_BASE+0xBC	DSPI DSI Configuration Register (DSPI_DSICR)	on page 15-29
DSPI_BASE+0xC0	DSPI DSI Serialization Data Register (DSPI_SDR)	on page 15-31
DSPI_BASE+0xC4	DSPI DSI Alternate Serialization Data Register (DSPI_ASDR)	on page 15-32
DSPI_BASE+0xC8	DSPI DSI Transmit Comparison Register (DSPI_COMPR)	on page 15-32
DSPI_BASE+0xCC	DSPI DSI Deserialization Data Register (DSPI_DDR)	on page 15-33
DSPI_BASE+0xD0	DSPI DSI Configuration Register 1 (DSPI_DSICR1)	on page 15-34
DSPI_BASE+0xD4	DSPI DSI Serialization Source Select Register (DSPI_SSR)	on page 15-35
DSPI_BASE+0xD8	DSPI DSI Parallel Input Select Register 0 (DPSI_PISR0)	on page 15-36
DSPI_BASE+0xDC	DSPI DSI Parallel Input Select Register 1 (DPSI_PISR1)	on page 15-36
DSPI_BASE+0xE0	DSPI DSI Parallel Input Select Register 2 (DPSI_PISR2)	on page 15-36
DSPI_BASE+0xE4	DSPI DSI Parallel Input Select Register 3 (DPSI_PISR3)	on page 15-36
DSPI_BASE+0xE8	DSPI DSI Deserialized Data Interrupt Mask Register (DSPI_DIMR)	on page 15-40
DSPI_BASE+0xEC	DSPI DSI Deserialized Data Polarity Interrupt Register (DSPI_DPIR)	on page 15-40

15.8.2 Register descriptions

15.8.2.1 DSPI Module Configuration Register (DSPI_MCR)

The DSPI_MCR contains bits which configure various attributes associated with DSPI operation. The HALT and MDIS bits can be changed at any time, but only take effect on the next frame boundary. Only the HALT and MDIS bits in the DSPI_MCR are allowed to be changed, while the DSPI is in the Running state.

Address: DSPI_BASE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MSTR	CONT_SCKE	DCONF		FRZ	MTFE	PCSSE	ROOE	0	0	PCSIS5	PCSIS4	PCSIS3	PCSIS2	PCSIS1	PCSIS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DOZE	MDIS	DIS_TXF	DIS_RXF	0	0	SMPL_PT		0	0	0	0	0	0	PES	HALT
W					CLR_TXF	CLR_RXF										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 15-5. DSPI Module Configuration Register (DSPI_MCR)

Table 15-5. DSPI_MCR field description

Field	Description
0 MSTR	Master/Slave Mode Select The MSTR bit configures the DSPI for either master mode or slave mode. 0 DSPI is in slave mode 1 DSPI is in master mode
1 CONT_SCKE	Continuous SCK Enable The CONT_SCKE bit enables the Serial Communication Clock (SCK) to run continuously. See Section 15.9.7, “Continuous serial communications clock,” for details. 0 Continuous SCK disabled 1 Continuous SCK enabled
2–3 DCONF[0:1]	DSPI Configuration The DCONF field selects between the three different configurations of the DSPI: 00 SPI 01 DSI 10 CSI 11 Reserved

Table 15-5. DSPI_MCR field description (continued)

Field	Description
4 FRZ	Freeze The FRZ bit enables the DSPI transfers to be stopped on the next frame boundary when the device enters Debug mode. 0 Do not stop serial transfers 1 Stop serial transfers
5 MTFE	Modified Timing Format Enable The MTFE bit enables a modified transfer format to be used. See Section 15.9.6.4, “Modified SPI/DSI transfer format (MTFE = 1, CPHA = 1),” for more information. 0 Modified SPI transfer format disabled 1 Modified SPI transfer format enabled
6 PCSSE	Peripheral Chip Select Strobe Enable The PCSSE bit enables the DSPI_x_PCS[5]/$\overline{\text{PCSS}}$ to operate as a PCS Strobe output signal. See Section 15.9.5.5, “Peripheral chip select strobe enable (PCSS),” for more information. 0 DSPI_x_PCS[5]/$\overline{\text{PCSS}}$ is used as the Peripheral Chip Select[5] signal 1 DSPI_x_PCS[5]/$\overline{\text{PCSS}}$ is used as an active-low PCS Strobe signal
7 ROOE	Receive FIFO Overflow Overwrite Enable The ROOE bit enables in RX FIFO overflow condition to ignore the incoming serial data or to overwrite existing data. If the RX FIFO is full and new data is received, the data from the transfer, generated the overflow, is ignored or shifted in to the shift register. See Section 15.9.10.6, “Receive FIFO overflow interrupt request,” for more information. 0 Incoming data is ignored 1 Incoming data is shifted in to the shift register
8–9	Reserved
10–15 PCSIS*x*	Peripheral Chip Select Inactive State The PCSIS bit determines the inactive state of the PCS*x* signal. 0 The inactive state of PCS*x* is low 1 The inactive state of PCS*x* is high
16 DOZE	Doze Enable The DOZE bit provides support for externally controlled Doze mode power-saving mechanism. See Section 15.9.18, “Power saving features,” for details. 0 Device Doze mode has no effect on DSPI. 1 Device Doze mode disables DSPI.
17 MDIS	Module Disable The MDIS bit allows the clock to be stopped to the non-memory mapped logic in the DSPI effectively putting the DSPI in a software controlled power-saving state. See Section 15.9.18, “Power saving features,” for more information. The reset value of the MDIS bit is parameterized, with a default reset value of ‘0’. 0 Enable DSPI clocks. 1 Allow external logic to disable DSPI clocks.
18 DIS_TXF	Disable Transmit FIFO When the TX FIFO is disabled, the transmit part of the DSPI operates as a simplified double-buffered SPI. See Section 15.9.2.3, “FIFO disable operation,” for details. 0 TX FIFO is enabled 1 TX FIFO is disabled

Table 15-5. DSPI_MCR field description (continued)

Field	Description
19 DIS_RXF	Disable Receive FIFO When the RX FIFO is disabled, the receive part of the DSPI operates as a simplified double-buffered SPI. See Section 15.9.2.3, “FIFO disable operation,” for details. 0 RX FIFO is enabled 1 RX FIFO is disabled
20 CLR_TXF	Clear TX FIFO CLR_TXF is used to flush the TX FIFO. Writing a ‘1’ to CLR_TXF clears the TX FIFO Counter. The CLR_TXF bit is always read as zero. 0 Do not clear the TX FIFO Counter 1 Clear the TX FIFO Counter
21 CLR_RXF	Clear RX FIFO CLR_RXF is used to flush the RX FIFO. Writing a ‘1’ to CLR_RXF clears the RX Counter. The CLR_RXF bit is always read as zero. 0 Do not clear the RX FIFO Counter 1 Clear the RX FIFO Counter
22–23 SMPL_PT	Sample Point SMPL_PT field controls when the DSPI master samples SIN in Modified Transfer Format. Figure 15-39 shows where the master can sample the SIN pin. 00 DSPI samples SIN at driving SCK edge. 01 DSPI samples SIN one system clock after driving SCK edge 10 DSPI samples SIN two system clocks after driving SCK edge 11 Reserved
24–29	Reserved, should be cleared.
30 PES	Parity Error Stop PES bit controls SPI operation when a parity error detected in received SPI frame. 0 SPI frames transmission continue. 1 SPI frames transmission stop.
31 HALT	Halt The HALT bit starts and stops DSPI transfers. See Section 15.9.1, “Start and stop of DSPI transfers,” for details on the operation of this bit. 0 Start transfers 1 Stop transfers

15.8.2.2 DSPI Transfer Count Register (DSPI_TCR)

The DSPI_TCR contains a counter, that indicates the number of SPI transfers made. The transfer counter is intended to assist in queue management. Do not write the DSPI_TCR, when the DSPI is in the Running state.

Address: DSPI_BASE + 0x8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-6. DSPI Transfer Count Register (DSPI_TCR)

Table 15-6. DSPI_TCR field description

Field	Description
0–15 TCNT[0:15]	SPI Transfer Counter The TCNT field counts the number of SPI transfers the DSPI makes. The TCNT field increments every time the last bit of a SPI frame is transmitted. A value written to TCNT presets the counter to that value. TCNT is reset to zero at the beginning of the frame when the CTCNT field is set in the executing SPI command. The Transfer Counter ‘wraps around’ i.e. incrementing the counter past 65535 resets the counter to zero.
16–31	Reserved, should be cleared.

15.8.2.3 DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTAR0–DSPI_CTAR7)

The DSPI_CTAR registers are used to define different transfer attributes. Each DSPI module on MPC5676R contains eight Clock and Transfer Attribute Registers (CTAR) to support compatibility with the QSPI block in the MPC5xx family of MCUs. Each DSPI_CTAR controls:

- Frame size
- Baud rate and transfer delay values
- Clock phase
- Clock polarity
- MSB/LSB first

In Master SPI configuration, the SPI command field selects which DSPI_CTAR to use on a per-frame basis. In Master DSI configuration, field DSPI_DSICR[DSICTAS] selects which DSPI_CTAR to use. In Slave SPI and Slave DSI configuration, DSPI_CTAR0 and DSPI_CTAR1 are used.

In master mode, the DSPI_CTAR0 – DSPI_CTAR7 registers define combinations of transfer attributes such as frame size, clock phase and polarity, data bit ordering, baud rate, and various delays. In slave mode, a subset of the bit-fields in the DSPI_CTAR0 and DSPI_CTAR1 registers are used to set the slave transfer attributes.

When the DSPI is configured as a SPI master, the CTAS field in the command portion of the TX FIFO entry selects which of the DSPI_CTAR registers is used. When the DSPI is configured as a SPI bus slave, the DSPI_CTAR0 register is used.

When the DSPI is configured as a DSI master, field DSPI_DSICR[DSICTAS] selects which of the DSPI_CTAR registers is used. When the DSPI is configured as a DSI bus slave, the DSPI_CTAR1 register is used.

In CSI configuration, the transfer attributes are selected based on whether the current frame is SPI data or DSI data. SPI transfers in CSI configuration follow the protocol described for SPI configuration, and DSI transfers in CSI configuration follow the protocol described for DSI configuration. CSI configuration is only valid in conjunction with master mode. See Section 15.9.4, “Combined serial interface (CSI) configuration,” for more details.

TSB mode sets some limitations on transfer attributes:

- Clock phase is forced to be CPHA = 1 and the CPHA bit setting has no effect.
- PCS lines are driven at the driving edge of the SCK clock together with SOUT, so PCS assertion and negation delays control is unavailable and PCSSCK, PASC, CSSCK and ASC fields have no effect.
- Delay after transfer can be set from 1 to 64 serial clocks with help of PDT and DT fields.

Address: DSPI_BASE + 0xC–DSPI_BASE + 0x28

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DBR	FMSZ				CPOL	CPHA	LSBFE	PCSSCK		PASC		PDT		PBR	
Reset	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CSSCK				ASC				DT				BR			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-7. DSPI Clock and Transfer Attributes Register 0–7 (DSPI_CTAR0–DSPI_CTAR7) in the master mode

Address: DSPI_BASE + 0xC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	FMSZ					CPOL	CPHA	PE	PP	Not used						
Reset	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	Not used															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-8. DSPI Clock and Transfer Attributes Register 0 (DSPI_CTAR0) in the slave mode

Table 15-7. DSPI_CTAR*n* field description in master mode

Field	Descriptions
0 DBR	Double Baud Rate The DBR bit doubles the effective baud rate of the Serial Communications Clock (SCK). This field is only used in master mode. It effectively halves the Baud Rate division ratio supporting faster frequencies and odd division ratios for the Serial Communications Clock (SCK). When the DBR bit is set, the duty cycle of the Serial Communications Clock (SCK) depends on the value in the Baud Rate Prescaler and the Clock Phase bit as listed in Table 15-8. See the BR field description for details on how to compute the baud rate. 0 The baud rate is computed normally with a 50/50 duty cycle 1 The baud rate is doubled with the duty cycle depending on the Baud Rate Prescaler
1–4 FMSZ[0:3]	Frame Size The number of bits transferred per frame is equal to FMSZ field value plus 1. Minimum valid FMSZ field value is 3. When operating in TSB mode, detailed in Section 15.9.8, “Timed serial bus (TSB),” the FMSZ field value plus 1 is equal the data frame bit number, where control of the PCS assertion switches from the DSPI_DSICR to the DSPI_DSICR1 register.
5 CPOL	Clock Polarity The CPOL bit selects the inactive state of the Serial Communications Clock (SCK). This bit is used in both master and slave mode. For successful communication between serial devices, the devices must have identical clock polarities. When the Continuous selection format is selected, switching between clock polarities without stopping the DSPI can cause errors in the transfer due to the peripheral device interpreting the switch of clock polarity as a valid clock edge. 0 The inactive state value of SCK is low 1 The inactive state value of SCK is high
6 CPHA	Clock Phase The CPHA bit selects which edge of SCK causes data to change and which edge causes data to be captured. This bit is used in both master and slave mode. For successful communication between serial devices, the devices must have identical clock phase settings. In Continuous SCK mode or TSB mode the bit value is ignored and the transfers are done as CPHA bit is set to 1. 0 Data is captured on the leading edge of SCK and changed on the following edge 1 Data is changed on the leading edge of SCK and captured on the following edge

Table 15-7. DSPI_CTAR*n* field description in master mode (continued)

Field	Descriptions
7 LSBFE	LSB First The LSBFE bit selects if the LSB or MSB of the frame is transferred first. When operating in TSB configuration, this bit should be set to be compliant to MSC specification. 0 Data is transferred MSB first 1 Data is transferred LSB first
8–9 PCSSCK[0:1]	PCS to SCK Delay Prescaler The PCSSCK field selects the prescaler value for the delay between assertion of PCS and the first edge of the SCK. See the CSSCK field description how to compute the PCS to SCK delay. In the TSB mode the PCSSCK field has no effect. 00 PCS to SCK prescaler value is 1 01 PCS to SCK prescaler value is 3 10 PCS to SCK prescaler value is 5 11 PCS to SCK prescaler value is 7
10–11 PASC[0:1]	After SCK Delay Prescaler The PASC field selects the prescaler value for the delay between the last edge of SCK and the negation of PCS. See the ASC field description how to compute the After SCK delay. In the TSB mode the PASC field has no effect. 00 After SCK delay prescaler value is 1 01 After SCK delay prescaler value is 3 10 After SCK delay prescaler value is 5 11 After SCK delay prescaler value is 7
12–13 PDT[0:1]	Delay after Transfer Prescaler The PDT field selects the prescaler value for the delay between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. The PDT field is only used in master mode. In the TSB mode the PDT field defines two MSB bits of the Delay after Transfer. See the DT field description for details on how to compute the Delay after Transfer. 00 Delay after Transfer prescaler value is 1 01 Delay after Transfer prescaler value is 3 10 Delay after Transfer prescaler value is 5 11 Delay after Transfer prescaler value is 7
14–15 PBR[0:1]	Baud Rate Prescaler The PBR field selects the prescaler value for the baud rate. This field is only used in master mode. The Baud Rate is the frequency of the Serial Communications Clock (SCK). The system clock is divided by the prescaler value before the baud rate selection takes place. See the BR field description for details on how to compute the baud rate. 00 Baud Rate prescaler value is 2 01 Baud Rate prescaler value is 3 10 Baud Rate prescaler value is 5 11 Baud Rate prescaler value is 7
16–19 CSSCK[0:3]	PCS to SCK Delay Scaler The CSSCK field selects the scaler value for the PCS to SCK delay. This field is only used in master mode. The PCS to SCK delay is the delay between the assertion of PCS and the first edge of the SCK. Table 15-9 list the scaler values.The PCS to SCK delay is a multiple of the system clock period and it is computed according to the following equation: $t_{CSC} = \frac{1}{f_{SYS}} \times PCSSCK \times CSSCK$ ***Eqn. 15-1*** See Section 15.9.5.2, "PCS to SCK delay (tCSC)," for more details.In the TSB mode the field has no effect.

Table 15-7. DSPI_CTAR*n* field description in master mode (continued)

Field	Descriptions
20–23 ASC[0:3]	After SCK Delay Scaler The ASC field selects the scaler value for the After SCK Delay. This field is only used in master mode. The After SCK Delay is the delay between the last edge of SCK and the negation of PCS. Table 15-9 list the scaler values.The After SCK Delay is a multiple of the system clock period, and it is computed according to the following equation: $t_{ASC} = \frac{1}{f_{SYS}} \times PASC \times ASC$ **Eqn. 15-2** See Section 15.9.5.3, "After SCK delay (tASC)," for more details. In the TSB mode the field has no effect.
24–27 DT[0:3]	Delay after Transfer Scaler The DT field selects the Delay after Transfer Scaler. This field is only used in master mode. The Delay after Transfer is the time between the negation of the PCS signal at the end of a frame and the assertion of PCS at the beginning of the next frame. Table 15-9 lists the scaler values. In the Continuous Serial Communications Clock operation the DT value is fixed to one SCK clock period, The Delay after Transfer is a multiple of the system clock period and it is computed according to the following equation: $t_{DT} = \frac{1}{f_{SYS}} \times PDT \times DT$ **Eqn. 15-3** In the TSB mode the Delay after Transfer is equal to a number formed by concatenation of PDT and DT fields plus 1 of the SCK clock periods. See Section 15.9.5.4, "Delay after transfer (tDT)," for more details.
28–31 BR[0:3]	Baud Rate Scaler The BR field selects the scaler value for the baud rate. This field is only used in master mode. The prescaled system clock is divided by the Baud Rate Scaler to generate the frequency of the SCK. Table 15-10 lists the Baud Rate Scaler values.The baud rate is computed according to the following equation: $\text{SCK baud rate} = \frac{f_{SYS}}{PBR} \times \frac{1 + DBR}{BR}$ **Eqn. 15-4** See Section 15.9.5.1, "Baud rate generator," for more details.

Table 15-8. DSPI SCK duty cycle

DBR	CPHA	PBR	SCK duty cycle
0	any	any	50/50
1	0	00	50/50
1	0	01	33/66
1	0	10	40/60
1	0	11	43/57
1	1	00	50/50
1	1	01	66/33
1	1	10	60/40
1	1	11	57/43

Table 15-9. Delay scaler encoding

Field value	Scaler value	Field value	Scaler value
0000	2	1000	512
0001	4	1001	1024
0010	8	1010	2048
0011	16	1011	4096
0100	32	1100	8192
0101	64	1101	16384
0110	128	1110	32768
0111	256	1111	65536

Table 15-10. DSPI baud rate scaler

BR	Baud rate scaler value	BR	Baud rate scaler value
0000	2	1000	256
0001	4	1001	512
0010	6	1010	1024
0011	8	1011	2048
0100	16	1100	4096
0101	32	1101	8192
0110	64	1110	16384
0111	128	1111	32768

Table 15-11. DSPI_CTAR0 field description in slave mode

Field	Descriptions
0–4 FMSZ[0:4]	Frame Size The number of bits transferred per frame is equal FMSZ field value plus 1. Minimum valid FMSZ field value is 3.
5 CPOL	Clock Polarity The CPOL bit selects the inactive state of the Serial Communications Clock (SCK). 0 The inactive state value of SCK is low 1 The inactive state value of SCK is high
6 CPHA	Clock Phase The CPHA bit selects which edge of SCK causes data to change and which edge causes data to be captured. 0 Data is captured on the leading edge of SCK and changed on the following edge 1 Data is changed on the leading edge of SCK and captured on the following edge

Table 15-11. DSPI_CTAR0 field description in slave mode (continued)

Field	Descriptions
7 PE	Parity Enable PE bit enables parity bit transmission and reception for the frame 0 No parity bit included/checked. 1 Parity bit is transmitted instead of last data bit in frame, parity checked for received frame.
8 PP	Parity Polarity PP bit controls polarity of the parity bit transmitted and checked 0 Even Parity: number of “1” bits in the transmitted frame is even. The DSPI_SR[SPEF] bit is set if in the received frame number of “1” bits is odd. 1 Odd Parity: number of “1” bits in the transmitted frame is odd. The DSPI_SR[SPEF] bit is set if in the received frame number of “1” bits is even.
29–31 —	Not used, write always zero to keep software compatible with future updates.

15.8.2.4 DSPI Status Register (DSPI_SR)

The DSPI_SR contains status and flag bits. The bits reflect the status of the DSPI and indicate the occurrence of events that can generate interrupt or DMA requests. Software can clear flag bits in the DSPI_SR by writing a ‘1’ to it. Writing a ‘0’ to a flag bit has no effect. This register may not be writable in module disable mode due to the use of power saving mechanisms.

Address: DSPI_BASE + 0x2C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF	TXRXS	0	EOQF	TFUF	0	TFFF	0	0	DPEF	SPEF	DDIF	RFOF	0	RFDF	0
W	w1c	w1c		w1c	w1c		w1c			w1c	w1c	w1c	w1c		w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TXCTR				TXNXTPTR				RXCTR				POPNXTPTR			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-9. DSPI Status Register (DSPI_SR)

Table 15-12. DSPI_SR field description

Field	Description
0 TCF	Transfer Complete Flag The TCF bit indicates that all bits in a frame have been shifted out. The TCF bit remains set until cleared by writing 1 to it. 0 Transfer not complete 1 Transfer complete

Table 15-12. DSPI_SR field description (continued)

Field	Description
1 TXRXS	TX & RX Status The TXRXS bit reflects the run status of the DSPI. Section 15.9.1, “Start and stop of DSPI transfers,” explains what causes this bit to be set or cleared. 0 TX and RX operations are disabled (DSPI is in STOPPED state) 1 TX and RX operations are enabled (DSPI is in RUNNING state)
2	Reserved, should be cleared.
3 EOQF	End of Queue Flag The EOQF bit indicates that the last entry in a queue has been transmitted when the DSPI in the master mode. The EOQF bit is set when TX FIFO entry has the EOQ bit set in the command halfword and the end of the transfer is reached. The EOQF bit remains set until cleared by writing 1 to it. When the EOQF bit is set, the TXRXS bit is automatically cleared. 0 EOQ is not set in the executed command 1 EOQ bit is set in the executed SPI command
4 TFUF	Transmit FIFO Underflow Flag The TFUF bit indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI operating in SPI slave mode is empty, and a transfer is initiated by an external SPI master. The TFUF bit remains set until cleared by writing 1 to it. 0 TX FIFO underflow has not occurred 1 TX FIFO underflow has occurred
5	Reserved, should be cleared.
6 TFFF	Transmit FIFO Fill Flag The TFFF bit provides a method for the DSPI to request more entries to be added to the TX FIFO. The TFFF bit is set while the TX FIFO is not full. The TFFF bit can be cleared by writing 1 to it or by acknowledgement from the DMA controller to the TX FIFO full request. 0 TX FIFO is full 1 TX FIFO is not full
7–8	Reserved, should be cleared.
9 DPEF	DSI Parity Error Flag The DPEF flag indicates that a DSI frame with parity error had been received. The bit remains set until cleared by writing 1 to it. 0 Parity Error has not occurred 1 Parity Error has occurred
10 SPEF	SPI Parity Error Flag The SPEF flag indicates that a SPI frame with parity error had been received. The bit remains set until cleared by writing 1 to it. 0 Parity Error has not occurred 1 Parity Error has occurred
11 DDIF	DSI data received with active bits The DDIF flag indicates that DSI frame had been received with bits, selected by DSPI_DIMR with active polarity, defined by DSPI_DPIR. The bit remains set until cleared by writing 1 to it. 0 No DSI data with active bits was received 1 DSI data with active bits was received

Table 15-12. DSPI_SR field description (continued)

Field	Description
12 RFOF	Receive FIFO Overflow Flag The RFOF bit indicates that an overflow condition in the RX FIFO has occurred. The bit is set when the RX FIFO and shift register are full and a transfer is initiated. The bit remains set until cleared by writing 1 to it. 0 RX FIFO overflow has not occurred 1 RX FIFO overflow has occurred
13	Reserved, should be cleared.
14 RFDF	Receive FIFO Drain Flag The RFDF bit provides a method for the DSPI to request that entries be removed from the RX FIFO. The bit is set while the RX FIFO is not empty. The RFDF bit can be cleared by writing 1 to it or by acknowledgement from the DMA controller when the RX FIFO is empty. 0 RX FIFO is empty 1 RX FIFO is not empty
15	Reserved.
16–20 TXCTR	TX FIFO Counter The TXCTR field indicates the number of valid entries in the TX FIFO. The TXCTR is incremented every time the DSPI _PUSHR is written. The TXCTR is decremented every time a SPI command is executed and the SPI data is transferred to the shift register.
20–23 TXNXTPTR	Transmit Next Pointer The TXNXTPTR field indicates which TX FIFO Entry is transmitted during the next transfer. The TXNXTPTR field is updated every time SPI data is transferred from the TX FIFO to the shift register. See Section 15.9.10.4, “Transmit FIFO underflow interrupt request,” for more details.
24–27 RXCTR	RX FIFO Counter The RXCTR field indicates the number of entries in the RX FIFO. The RXCTR is decremented every time the DSPI _POPR is read. The RXCTR is incremented every time data is transferred from the shift register to the RX FIFO.
28–31 POPNXTPTR	Pop Next Pointer The POPNXTPTR field contains a pointer to the RX FIFO entry that will be returned when the DSPI_POPR is read. The POPNXTPTR is updated when the DSPI_POPR is read. See Section 15.9.2.5, “Receive first-in first-out (RX FIFO) buffering mechanism,” for more details.

15.8.2.5 DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)

The DSPI_RSER controls DMA and interrupt requests. Do not write to the DSPI_RSER while the DSPI is in the Running state.

Address: DSPI_BASE + 0x30

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCF_RE	0	0	EOQFRE	TFUFRE	0	TFFFRE	TFFFDIRS	0	DPEFRE	SPEFRE	DDIFRE	RFOFRE	0	RFDFRE	RFDFDIRS
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-10. DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER)

Table 15-13. DSPI_RSER field description

Field	Description
0 TCF_RE	Transmission Complete Request Enable The TCF_RE bit enables TCF flag in the DSPI_SR to generate an interrupt request. 0 TCF interrupt requests are disabled 1 TCF interrupt requests are enabled
1–2	Reserved, should be cleared.
3 EOQFRE	DSPI Finished Request Enable The EOQFRE bit enables the EOQF flag in the DSPI_SR to generate an interrupt request. 0 EOQF interrupt requests are disabled 1 EOQF interrupt requests are enabled
4 TFUFRE	Transmit FIFO Underflow Request Enable The TFUFRE bit enables the TFUF flag in the DSPI_SR to generate an interrupt request. 0 TFUF interrupt requests are disabled 1 TFUF interrupt requests are enabled
5	Reserved, should be cleared.
6 TFFFRE	Transmit FIFO Fill Request Enable The TFFFRE bit enables the TFFF flag in the DSPI_SR to generate a request. The TFFFDIRS bit selects between generating an interrupt request or a DMA requests. 0 TFFF interrupt requests or DMA requests are disabled 1 TFFF interrupt requests or DMA requests are enabled
7 TFFFDIRS	Transmit FIFO Fill DMA or Interrupt Request Select The TFFFDIRS bit selects between generating a DMA request or an interrupt request. When the TFFF flag bit in the DSPI_SR is set, and the TFFFRE bit in the DSPI_RSER is set, this bit selects between generating an interrupt request or a DMA request. 0 Interrupt request will be generated 1 DMA request will be generated

Table 15-13. DSPI_RSER field description (continued)

Field	Description
8	Reserved, should be cleared.
9 DPEFRE	DSI Parity Error Request Enable The DPEFRE bits enables DPEF flag in the DSPI_SR to generate an interrupt requests. 0 PEF interrupt requests are disabled 1 PEF interrupt requests are enabled
10 SPEFRE	SPI Parity Error Request Enable The SPEFRE bits enables SPEF flag in the DSPI_SR to generate an interrupt requests. 0 PEF interrupt requests are disabled 1 PEF interrupt requests are enabled
11 DDIFRE	DSI data received with active bits Request Enable The DDIFRE bit enables the DDIF flag in the DSPI_SR to generate an interrupt requests. 0 DDIF interrupt requests are disabled 1 DDIF interrupt requests are enabled
12 RFOFRE	Receive FIFO Overflow Request Enable The RFOFRE bit enables the RFOF flag in the DSPI_SR to generate an interrupt requests. 0 RFOF interrupt requests are disabled 1 RFOF interrupt requests are enabled
13	Reserved, should be cleared.
14 RFDFRE	Receive FIFO Drain Request Enable The RFDFRE bit enables the RFDF flag in the DSPI_SR to generate a request. The RFDFDIRS bit selects between generating an interrupt request or a DMA request. 0 RFDF interrupt requests or DMA requests are disabled 1 RFDF interrupt requests or DMA requests are enabled
15 RFDFDIRS	Receive FIFO Drain DMA or Interrupt Request Select The RFDFDIRS bit selects between generating a DMA request or an interrupt request. When the RFDF flag bit in the DSPI_SR is set, and the RFDFRE bit in the DSPI_RSER is set, the RFDFDIRS bit selects between generating an interrupt request or a DMA request. 0 Interrupt request will be generated 1 DMA request will be generated

15.8.2.6 DSPI PUSH TX FIFO Register (DSPI_PUSHR)

The DSPI_PUSHR provides means to write to the TX FIFO. Data written to this register is transferred to the TX FIFO. See Section 15.9.2.4, “Transmit first-in first-out (TX FIFO) buffering mechanism” for more information. Eight or 16-bit write accesses to the DSPI_PUSHR transfers all 32 register bits to the TX FIFO.

The register structure is different in master and slave modes. In master mode the register provides 16-bit commands and 16-bit data to the TX FIFO. In slave mode all 32 register bits can be used as data, supporting up to 32-bit SPI frame operation.

Address: DSPI_BASE + 0x34

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CONT	CTAS			EOQ	CTCNT	PE	PP	0	0	PCS5	PCS4	PCS3	PCS2	PCS1	PCS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-11. DSPI PUSH TX FIFO Register (DSPI_PUSHR) in master mode

Table 15-14. DSPI_PUSHR field description in master mode

Field	Descriptions
0 CONT	Continuous Peripheral Chip Select Enable The CONT bit selects a Continuous Selection Format. The bit is used in SPI master mode. The bit enables the selected PCS signals to remain asserted between transfers. See Section 15.9.6.5, “Continuous selection format,” for more information. 0 Return Peripheral Chip Select signals to their inactive state between transfers 1 Keep Peripheral Chip Select signals asserted between transfers
1–3 CTAS[0:2]	Clock and Transfer Attributes Select The CTAS field selects the number of the DSPI_CTAR to be used to set the transfer attributes for the associated SPI frame. The field is only used in SPI master mode. In SPI slave mode DSPI_CTAR0 is used. The number of DSPI_CTAR registers is implementation specific and the CTAS should be set to select only implemented one.
4 EOQ	End Of Queue The EOQ bit provides a means for host software to signal to the DSPI that the current SPI transfer is the last in a queue. At the end of the transfer the EOQF bit in the DSPI_SR is set. 0 The SPI data is not the last data to transfer 1 The SPI data is the last data to transfer
5 CTCNT	Clear Transfer Counter The CTCNT bit clears field DSPI_TCR[TCNT]. The TCNT field is cleared before transmission of the current SPI frame begins. 0 Do not clear field DSPI_TCR[TCNT] 1 Clear field DSPI_TCR[TCNT]

Table 15-14. DSPI_PUSHR field description in master mode (continued)

Field	Descriptions
6 PE	Parity Enable PE bit enables parity bit transmission and parity reception check for the SPI frame 0 No parity bit included/checked. 1 Parity bit is transmitted instead of last data bit in frame, parity checked for received frame.
7 PP	Parity Polarity PP bit controls polarity of the parity bit transmitted and checked 0 Even Parity: number of “1” bits in the transmitted frame is even. The DSPI_SR[SPEF] bit is set if in the received frame number of “1” bits is odd. 1 Odd Parity: number of “1” bits in the transmitted frame is odd. The DSPI_SR[SPEF] bit is set if in the received frame number of “1” bits is even.
8–9	Reserved
10–15 PCS*x*	Peripheral Chip Select 0–5 The PCS bits select which PCS signals will be asserted for the transfer. 0 Negate the PCS[x] signal 1 Assert the PCS[x] signal
16–31 TXDATA[0:15]	Transmit Data The TXDATA field holds SPI data to be transferred according to the associated SPI command.

Address: DSPI_BASE + 0x34

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	TXDATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TXDATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-12. DSPI PUSH TX FIFO Register (DSPI_PUSHR) in slave mode

Table 15-15. DSPI_PUSHR field description in slave mode

Field	Descriptions
0–31 TXDATA[0:31]	Transmit Data The TXDATA field holds SPI data to be transferred.

15.8.2.7 DSPI POP RX FIFO Register (DSPI_POPR)

The DSPI_POPR provides the means to read the RX FIFO. See Section 15.9.2.5, “Receive first-in first-out (RX FIFO) buffering mechanism” for a description of the RX FIFO operations. Eight or 16-bit read accesses to the DSPI_POPR have the same effect on the RX FIFO as 32-bit read access.

Address: DSPI_BASE + 0x38

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-13. DSPI POP RX FIFO Register (DSPI_POPR)

Table 15-16. DSPI_POPR field description

Field	Description
0–31 RXDATA[0:31]	Received Data The RXDATA field contains the SPI data from the RX FIFO entry pointed to by the Pop Next Data Pointer.

15.8.2.8 DSPI Transmit FIFO Registers 0–15 (DSPI_TXFR0–DSPI_TXFR15)

The DSPI_TXFR0 – DSPI_TXFR15 registers provide visibility into the TX FIFO for debugging purposes. Each register is an entry in the TX FIFO. The registers are read-only and cannot be modified. Reading the DSPI_TXFRx registers does not alter the state of the TX FIFO. In MPC5676R, four-entry TX FIFO is implemented thus DSPI_TXFR0 – DSPI_TXFR3 are accessible.

Address: DSPI_BASE+0x3C–DSPI_BASE+0x78

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TXCMD/TXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-14. DSPI Transmit FIFO Register 0–15 (DSPI_TXFR0–DSPI_TXFR15)

Table 15-17. DSPI_TXFR*n* field description

Field	Description
0–15 TXCMD[0:15]/ TXDATA[0:15]	Transmit Command or Transmit Data In master mode the TXCMD field contains the command that sets the transfer attributes for the SPI data. See Section 15.8.2.6, “DSPI PUSH TX FIFO Register (DSPI_PUSHR),” for details on the command field. In slave mode the TXDATA contains 16 MSB bits of the SPI data to be shifted out
16–31 TXDATA[16:31]	Transmit Data The TXDATA field contains the SPI data to be shifted out.

15.8.2.9 DSPI Receive FIFO Registers 0–15 (DSPI_RXFR0–DSPI_RXFR15)

The DSPI_RXFR0 – DSPI_RXFR15 registers provide visibility into the RX FIFO for debugging purposes. Each register is an entry in the RX FIFO. The DSPI_RXFR registers are read-only. Reading the DSPI_RXFRx registers does not alter the state of the RX FIFO. In MPC5676R, four-entry RX FIFO is implemented thus DSPI_RXFR0 – DSPI_RXFR3 are accessible.

Address: DSPI_BASE + 0x7C–DSPI_BASE + 0xB8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	RXDATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-15. DSPI Receive FIFO Registers 0–15 (DSPI_RXFR0–DSPI_RXFR15)

Table 15-18. DSPI_RXFR*n* field description

Field	Description
0–31 RXDATA[0:31]	Receive Data The RXDATA field contains the received SPI data.

15.8.2.10 DSPI DSI Configuration Register (DSPI_DSICR)

The DSI Configuration Register selects various attributes associated with DSI and CSI Configurations. Do not write to the DSPI_DSICR, while the DSPI is in the Running state.

Address: DSPI_BASE + 0xBC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MTOE	FMSZ	MTOCNT						0	0	0	TSBC	TXSS	TPOL	TRRE	CID
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DCONT	DSICTAS			0	0	0	0	0	0	DPCS5	DPCS4	DPCS3	DPCS2	DPCS1	DPCS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-16. DSPI DSI Configuration Register (DSPI_DSICR)

Table 15-19. DSPI_DSICR field description

Field	Description
0 MTOE	Multiple Transfer Operation Enable The MTOE bit enables multiple DSPIs to be connected in a parallel or serial configuration. See Section 15.9.3.6, “Multiple transfer operation (MTO),” for more information. 0 Multiple Transfer Operation disabled 1 Multiple Transfer Operation enabled The MTOE and TSB bits should not be set simultaneously.
1 FMSZ	MSB of the Frame Size If the bit is set, 16 is added to the frame size, defined by field DSPI_CTARn[FMSZ]. DSPI_CTARn register is selected by field DSPI_DSICR[DSICTAS].
2–7 MTOCNT[0:5]	Multiple Transfer Operation Count The MTOCNT field selects number of bits to be shifted out during a transfer in Multiple Transfer Operation. The field sets the number of SCK cycles that the bus master will generate to complete the transfer. The number of SCK cycles used will be one more than the value in the MTOCNT field. The number of SCK cycles defined by MTOCNT must be equal to or greater than the frame size. When TSBC is set, MTOCNT field has no effect.
8–10	Reserved, should be cleared.
11 TSBC	Timed Serial Bus Configuration The TSBC bit enables the Timed Serial Bus Configuration. This configuration allows 32-bit data to be used. It also allows t_{DT} to be programmable. See Section 15.9.8, “Timed serial bus (TSB)” for detailed information. 0 Timed Serial Bus Configuration disabled 1 Timed Serial Bus Configuration enabled If this bit is clear the DSPI_DSICR1 register value has no effect.
12 TXSS	Transmit Data Source Select The TXSS bit selects the source of data to be serialized. The source can be either data from host Software written to the DSPI DSI Alternate Serialization Data Register (DSPI_ASDR), or Parallel Input pin states latched into the DSPI DSI Serialization Data Register (DSPI_SDR). 0 Source of serialized data is the DSPI_SDR 1 Source of serialized data is the DSPI_ASDR
13 TPOL	Trigger Polarity The TPOL bit selects the active edge of the hardware trigger input signal (HT). initiating DSI frames transfer. See Section 15.9.3.5, “DSI transfer initiation control,” for more information. 0 Falling edge will initiate a transfer 1 Rising edge will initiate a transfer
14 TRRE	Trigger Reception Enable The TRRE bit enables the DSPI to initiate DSI frames transfer with external trigger signal. See Section 15.9.3.5, “DSI transfer initiation control,” for more information. 0 Trigger signal reception disabled 1 Trigger signal reception enabled
15 CID	Change In Data Transfer Enable The CID bit enables a change in serialization data to initiate DSI frames transfer. in DSI and CSI configurations. When the CID bit is set, DSI frames are initiated when the current DSI data differs from the previous DSI data shifted out. Refer to Section 15.9.3.5, “DSI transfer initiation control,” for more information.

Table 15-19. DSPI_DSICR field description (continued)

Field	Description
16 DCONT	DSI Continuous Peripheral Chip Select Enable The DCONT bit enables the PCS signals to remain asserted between transfers. The DCONT bit only affects the PCS signals in DSI master mode. See Section 15.9.6.5, “Continuous selection format,” for details. When TSBC bit is set, DCONT bit has no effect. 0 Return Peripheral Chip Select signals to their inactive state after transfer is complete 1 Keep Peripheral Chip Select signals asserted after transfer is complete
17–19 DSICTAS[0:2]	DSI Clock and Transfer Attributes Select The DSICTAS field selects which of the DSPI_CTAR registers is used to provide transfer attributes for DSI frames. The DSICTAS field is used in DSI master mode. In DSI slave mode, the DSPI_CTAR1 is always selected.
20–25	Reserved, should be cleared.
26–31 DPCS*x*	DSI Peripheral Chip Select 0–5 The DPCS bits select which of the PCS signals to assert during a DSI master mode transfer. 0 Negate PCS[x] 1 Assert PCS[x]

15.8.2.11 DSPI DSI Serialization Data Register (DSPI_SDR)

The DSPI_SDR contains the states of the Parallel Input signals. The states of the Parallel Input signals are latched into the DSPI_SDR on the rising edge of every system clock. The DSPI_SDR is read-only. When the TXSS bit in the DSPI_DSICR is cleared, the data in the DSPI_SDR is used as the source of the DSI frames.

Address: DSPI_BASE + 0xC0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SER_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SER_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-17. DSPI DSI Serialization Data Register (DSPI_SDR)

Table 15-20. DSPI_SDR field description

Field	Description
0–31 SER_DATA [30:31]	Serialized Data The SER_DATA field contains the signal states of the Parallel Input signals.

15.8.2.12 DSPI DSI Alternate Serialization Data Register (DSPI_ASDR)

The DSPI_ASDR provides means for host software to write the data to be serialized. When the TXSS bit in the DSPI_DSICR is set, the data in the DSPI_ASDR is the source of the DSI frames. Writes to the DSPI_ASDR take effect on the next frame boundary.

Address: DSPI_BASE + 0xC4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ASER_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ASER_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-18. DSPI DSI Alternate Serialization Data Register (DSPI_ASDR)

Table 15-21. DSPI_ASDR field description

Field	Descriptions
0–31 ASER_DATA [0:31]	Alternate Serialized Data The ASER_DATA field holds the alternate data to be serialized.

15.8.2.13 DSPI DSI Transmit Comparison Register (DSPI_COMPR)

The DSPI_COMPR holds a copy of the last transmitted DSI data. The DSPI_COMPR is read-only. DSI data is transferred to this register as it is loaded into the TX Shift Register.

Address: DSPI_BASE + 0xC8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	COMP_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	COMP_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-19. DSPI DSI Transmit Comparison Register (DSPI_COMPR)

Table 15-22. DSPI_COMPR field description

Field	Description
0–31 COMP_DATA[0:31]	Compare Data The COMP_DATA field holds the last serialized DSI data.

15.8.2.14 DSPI DSI Deserialization Data Register (DSPI_DDR)

The DSPI_DDR holds the signal states for the Parallel Output signals. The DSPI_DDR is read-only and host software can read data from incoming DSI frames.

Address: DSPI_BASE + 0xCC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DESER_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DESER_DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-20. DSPI Deserialization Data Register (DSPI_DDR)

Table 15-23. DSPI_DDR field description

Field	Descriptions
0–31 DESER_DATA [0:31]	Deserialized Data The DESER_DATA field holds deserialized data which is presented as signal states to the Parallel Output signals.

15.8.2.15 DSPI DSI Configuration Register 1 (DSPI_DSICR1)

The DSI Configuration Register 1 selects various attributes associated with TSB Configuration. The user must not write to the DSPI_DSICR1 while the DSPI is in the Running state. If TSBC bit is cleared the register value is ignored.

Address: DSPI_BASE + 0xD0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	TSBCNT					0	0	0	0	0	0	DSE 1	DSE 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	DPCS1_5	DPCS1_4	DPCS1_3	DPCS1_2	DPCS1_1	DPCS1_0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-21. DSPI DSI Configuration Register 1 (DSPI_DSICR1)

Table 15-24. DSPI_DSICR1 field description

Field	Description
0–2	Reserved, should be cleared.
3–7 TSBCNT[0:4]	Timed Serial Bus Operation Count When TSBC is set, TSBCNT defines the length of the data frame. TSBCNT field valid value is from 3 to 31. The TSBCNT field selects number of data bits to be shifted out during a transfer in TSB mode. The number of data bits in the data frame is one more than the value in the TSBCNT field.
8–13	Reserved, should be cleared.
14 DSE1	Data Select Enable1 When TBSC bit is set, the DSE1 bit controls insertion of the zero bit (Data Select) in the middle of the data frame. The insertion bit position is defined by FMSZ field of DSPI_CTARn register, selected by DSICTAS field of the DSPI_DSICR register. 0 No Zero bit inserted in the middle of the data frame. 1 Zero bit is inserted at the middle of the data frame. Total number of bits in the data frame is increased by 1.
15 DSE0	Data Select Enable0. When TBSC bit is set, the DSE0 bit controls insertion of the zero bit (Data Select) in the beginning of the data frame. 0 No Zero bit inserted in the beginning of the frame 1 Zero bit is inserted at the beginning of the data frame. Total number of bits in the data frame is increased by 1.

Table 15-24. DSPI_DSICR1 field description (continued)

Field	Description
16-25	Reserved, should be cleared.
26–31 DPCS1_*x*	DSI Peripheral Chip Select 0–5 These bits define the PCSs to assert for the second part of the DSI frame when operating in TSB configuration with dual receiver. The DPCS1 bits select which of the PCS signals to assert during the second part of the DSI frame. The DPCS1 bits only control the assertions of the PCS signals in TSB mode. 0 Negate PCS[x] 1 Assert PCS[x]

15.8.2.16 DSPI DSI Serialization Source Select Register (DSPI_SSR)

DSPI DSI Serialization Source Select Register provides means to create combined frame for transmission, containing bits from DSPI_ASDR register and from DSPI_SDR register. Each bit in the DSPI_SSR register selects corresponding bit to be serialized. When DSPI_DSICR[TXSS] is set, the DSPI_SSR register value has no effect.

Address: DSPI_BASE + 0xD4 Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	SS															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	SS															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-22. DSPI DSI Serialization Source Select Register (DSPI_SSR)

Table 15-25. DSPI_SSR Field Descriptions

Field	Description
0–31 SS[0:31]	Source Select. The SS bits select serialization source for DSI frame. Each SS bit selects data for corresponded bit in the transmitted frame. 0 the bit in transmitted frame is taken from Parallel Input pin; 1 the bit in transmitted frame is taken from DSPI_ASDR register

15.8.2.17 DSPI DSI Parallel Input Select Registers 0 - 3 (DPSI_PISR0 - DPSI_PISR3)

DSPI DSI Parallel Input Select Registers 0 - 3 provide means to select each data bit for transmitted frame from 16 Parallel Input pins. Each Input Pin Select (IPS) field controls one bit in the transmitted frame. Each register contains control fields for 8 bits in the frame. The select field value is defined as 4 bits signed integer number. Selected Parallel Input pin number is defined as a difference between the field number and field value.

For example, if IPS16 is equal binary number 1111 (minus 1 decimal) bit 16 in the frame will be taken from Parallel Input pin number 17. When the IPS0 is equal -1, the bit 0 in the frame is taken from Parallel Input 1. When the IPS0 is equal +1, the bit 0 in the frame is taken from Parallel Input 31 and etc.

Please, note that the DSPI_PISR0-3 only preselect Parallel Input pins, final selection to the transmitted frame is done by DSPI_SSR register bits or DSPI_DSICR[TXSS] bit.

Address: DSPI_BASE + 0xD8 Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS24				IPS25				IPS26				IPS27			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IPS28				IPS29				IPS30				IPS31			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-23. DSPI DSI Parallel Input Select Register 0 (DSPI_PISR0)

Table 15-26. DSPI_PISR0 Field Descriptions

Field	Description
0–3 IPS24	Input Pin Select 24. The IPS24 field selects Parallel Input pin for transmitted frame bit 24.
4–7 IPS25	Input Pin Select 25. The IPS25 field selects Parallel Input pin for transmitted frame bit 25.
8–11 IPS26	Input Pin Select 26. The IPS26 field selects Parallel Input pin for transmitted frame bit 26.
12–15 IPS27	Input Pin Select 27. The IPS27 field selects Parallel Input pin for transmitted frame bit 27.
16–19 IPS28	Input Pin Select 28. The IPS28 field selects Parallel Input pin for transmitted frame bit 28.
20–23 IPS29	Input Pin Select 29. The IPS29 field selects Parallel Input pin for transmitted frame bit 29.

Table 15-26. DSPI_PISR0 Field Descriptions (continued)

Field	Description
24–27 IPS30	Input Pin Select 30. The IPS30 field selects Parallel Input pin for transmitted frame bit 30.
28–31 IPS31	Input Pin Select 31. The IPS31 field selects Parallel Input pin for transmitted frame bit 31.

Address: DSPI_BASE + 0xDC

Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS16				IPS17				IPS18				IPS19			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IPS20				IPS21				IPS22				IPS23			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-24. DSPI DSI Parallel Input Select Register 1 (DSPI_PISR1)

Table 15-27. DSPI_PISR1 Field Descriptions

Field	Description
0–3 IPS16	Input Pin Select 16. The IPS16 field selects Parallel Input pin for transmitted frame bit 16.
4–7 IPS17	Input Pin Select 17. The IPS17 field selects Parallel Input pin for transmitted frame bit 17.
8–11 IPS18	Input Pin Select 18. The IPS18 field selects Parallel Input pin for transmitted frame bit 18.
12–15 IPS19	Input Pin Select 19. The IPS19 field selects Parallel Input pin for transmitted frame bit 19.
16–19 IPS20	Input Pin Select 20. The IPS20 field selects Parallel Input pin for transmitted frame bit 20.
20–23 IPS21	Input Pin Select 21. The IPS21 field selects Parallel Input pin for transmitted frame bit 21.
24–27 IPS22	Input Pin Select 22. The IPS22 field selects Parallel Input pin for transmitted frame bit 22.
28–31 IPS23	Input Pin Select 23. The IPS23 field selects Parallel Input pin for transmitted frame bit 23.

Address: DSPI_BASE + 0xE0 Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS8				IPS9				IPS10				IPS11			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IPS12				IPS13				IPS14				IPS15			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-25. DSPI DSI Parallel Input Select Register 2 (DSPI_PISR2)

Table 15-28. DSPI_PISR2 Field Descriptions

Field	Description
0–3 IPS8	Input Pin Select 8. The IPS8 field selects Parallel Input pin for transmitted frame bit 8.
4–7 IPS9	Input Pin Select 9. The IPS9 field selects Parallel Input pin for transmitted frame bit 9.
8–11 IPS10	Input Pin Select 10. The IPS10 field selects Parallel Input pin for transmitted frame bit 10.
12–15 IPS11	Input Pin Select 11. The IPS11 field selects Parallel Input pin for transmitted frame bit 11.
16–19 IPS12	Input Pin Select 12. The IPS12 field selects Parallel Input pin for transmitted frame bit 12.
20–23 IPS13	Input Pin Select 13. The IPS13 field selects Parallel Input pin for transmitted frame bit 13.
24–27 IPS14	Input Pin Select 14. The IPS14 field selects Parallel Input pin for transmitted frame bit 14.
28–31 IPS15	Input Pin Select 15. The IPS15 field selects Parallel Input pin for transmitted frame bit 15.

Address: DSPI_BASE + 0xE4 Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	IPS0				IPS1				IPS2				IPS3			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	IPS4				IPS5				IPS6				IPS7			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-26. DSPI DSI Parallel Input Select Register 3 (DSPI_PISR3)

Table 15-29. DSPI_PISR3 Field Descriptions

Field	Description
0–3 IPS0	Input Pin Select 0. The IPS0 field selects Parallel Input pin for transmitted frame bit 0.
4–7 IPS1	Input Pin Select 1. The IPS1 field selects Parallel Input pin for transmitted frame bit 1.
8–11 IPS2	Input Pin Select 2. The IPS2 field selects Parallel Input pin for transmitted frame bit 2.
12–15 IPS3	Input Pin Select 3. The IPS3 field selects Parallel Input pin for transmitted frame bit 3.
16–19 IPS4	Input Pin Select 4. The IPS4 field selects Parallel Input pin for transmitted frame bit 4.
20–23 IPS5	Input Pin Select 5. The IPS5 field selects Parallel Input pin for transmitted frame bit 5.
24–27 IPS6	Input Pin Select 6. The IPS6 field selects Parallel Input pin for transmitted frame bit 6.
28–31 IPS7	Input Pin Select 7. The IPS7 field selects Parallel Input pin for transmitted frame bit 7.

15.8.2.18 DSPI DSI Deserialized Data Interrupt Mask Register (DSPI_DIMR)

The DSPI DSI Deserialized Data Interrupt Mask Register selects bits in the received DSI frame to be checked to generate the DDI interrupt.

Address: DSPI_BASE + 0xE8 Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MASK															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MASK															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-27. DSPI DSI Deserialized Data Interrupt Mask Register (DSPI_DIMR)

Table 15-30. DSPI_DIMR Field Descriptions

Field	Description
0–31 MASK[0:31]	MASK. The MASK bits define which bits in received deserialization data should be checked to produce the Deserialized Data Interrupt (DDI). 0 the bit in received DSI frame does not produce DDI interrupt. 1 the bit in received DSI frame can produce DDI interrupt if the data bit matches to configured polarity.

15.8.2.19 DSPI DSI Deserialized Data Polarity Interrupt Register (DSPI_DPIR)

The DSPI DSI Deserialized Data Polarity Interrupt Register defines what data bits value in the received DSI frame generates the DDI interrupt.

Address: DSPI_BASE + 0xEC Access:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DP															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DP															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-28. DSPI DSI Deserialized Data Polarity Interrupt Register (DSPI_DIPR)

Table 15-31. DSPI_DIPR Field Descriptions

Field	Description
0–31 DP[0:31]	Data Polarity. The DP bits define what value of the received deserialization data sets the DSPI_SR[DDIF] bit. 0 if received bit is 0 the DSPI_SR[DDIF] bit is set. 1 if received bit is 1 the DSPI_SR[DDIF] bit is set.

15.9 Functional description

The Deserial Serial Peripheral Interface (DSPI) module supports full-duplex, synchronous serial communications between MCUs and peripheral devices. The DSPI can also be used to reduce the number of pins required for I/O by serializing and deserializing up to 32 Parallel Input/Output signals. All communications are done with SPI-like protocol.

The DSPI has three configurations:

- SPI configuration in which the DSPI operates as a basic SPI or a queued SPI.
- DSI configuration in which the DSPI serializes and deserializes Parallel Input/Output signals or bits from memory mapped register.
- CSI configuration in which the DSPI combines the functionality of the SPI and DSI configurations.

Field DSPI_MCR[DCONF] determines the DSPI configuration. See Table 15-5 for the DSPI configuration values.

Registers DSPI_CTAR0 – DSPI_CTAR7 hold clock and transfer attributes. The SPI configuration allows to select which DSPI_CTAR to use on a frame by frame basis by setting a field in the SPI command. The DSI configuration statically selects which DSPI_CTAR to use. In CSI configuration priority logic determines if SPI data or DSI data is transferred and dictates what DSPI_CTAR is used for the data transfer. See Section 15.8.2.3, “DSPI Clock and Transfer Attributes Registers 0–7 (DSPI_CTAR0–DSPI_CTAR7),” for information on the fields of the DSPI_CTAR registers.

Typical master to slave connections are shown in Figure 15-29. When a data transfer operation is performed, data is serially shifted a predetermined number of bit positions. Because the modules are linked, data is exchanged between the master and the slave. The data that was in the master shift register is now in the shift register of the slave, and vice versa. At the end of a transfer, bit DSPI_SR[TCF] is set to indicate a completed transfer.

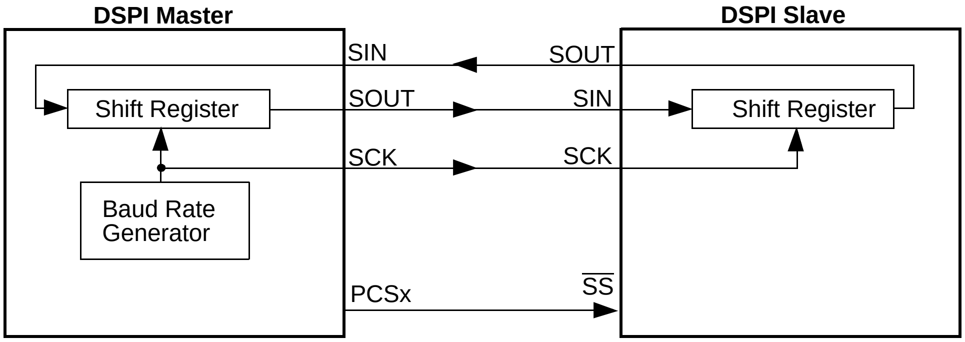


Figure 15-29. SPI and DSI Serial Protocol Overview

Generally more than one slave device can be connected to the DSPI master. Eight Peripheral Chip Select (PCS) signals of the DSPI masters can be used to select which of the slaves to communicate with.

The three DSPI configurations share transfer protocol and timing properties which are described independently of the configuration in Section 15.9.6, “Transfer formats”. The transfer rate and delay settings are described in Section 15.9.5, “DSPI baud rate and clock delay generation.”

15.9.1 Start and stop of DSPI transfers

The DSPI has two operating states: STOPPED and RUNNING. The states are independent of DSPI configuration. The default state of the DSPI is STOPPED. In the STOPPED state no serial transfers are initiated in master mode and no transfers are responded to in slave mode. The STOPPED state is also a safe state for writing the various configuration registers of the DSPI without causing undetermined results. In the RUNNING state serial transfers take place.

Bit DSPI_SR[TXRXS] indicates the DSPI’s operating state. The bit is set if the module is in RUNNING state.

The DSPI is started (DSPI transitions to RUNNING) when all of the following conditions are true:

- DSPI_SR[EOQF] bit is clear
- Device is not in the debug mode is or the DSPI_MCR[FRZ] bit is clear
- DSPI_MCR[HALT] bit is clear

The DSPI stops (transitions from RUNNING to STOPPED) after the current frame when any one of the following conditions exist:

- DSPI_SR[EOQF] bit is set
- Device in the debug mode and the DSPI_MCR[FRZ] bit is set
- DSPI_MCR[HALT] bit is set

State transitions from RUNNING to STOPPED occur on the next frame boundary if a transfer is in progress, or immediately if no transfers are in progress.

15.9.2 Serial peripheral interface (SPI) configuration

The SPI configuration transfers data serially using a shift register and a selection of programmable transfer attributes. The DSPI is in SPI configuration when field DSPI_MCR[DCONF] is 0b00. The SPI frames can be from 4 to 16 bits long. Host CPU or a DMA controller transfer the SPI data from the external to DSPI RAM queues to a transmit First-In First-Out (TX FIFO) buffer. The received data is stored in entries in the Receive FIFO (RX FIFO) buffer. Host CPU or the DMA controller transfer the received data from the RX FIFO to memory external to the DSPI. The FIFO buffers operation is described in Section 15.9.2.4, “Transmit first-in first-out (TX FIFO) buffering mechanism,” and Section 15.9.2.5, “Receive first-in first-out (RX FIFO) buffering mechanism.” The interrupt and DMA request conditions are described in Section 15.9.10, “Interrupts/DMA requests.”

The SPI configuration supports two module-specific modes: master mode and slave mode. The FIFO operations are similar for both modes. The main difference is that in master mode the DSPI initiates and controls the transfer according to the fields in the SPI command field of the TX FIFO entry. In slave mode the DSPI only responds to transfers initiated by a bus master external to the DSPI and the SPI command field space is used for the 16 most significant bitS of the transmit data.

15.9.2.1 Master mode

In SPI master mode the DSPI initiates the serial transfers by controlling the Serial Communications Clock (SCK) and the Peripheral Chip Select (PCS) signals. The SPI command field in the executing TX FIFO entry determines which of the DSPI_CTAR registers will be used to set the transfer attributes and which PCS signal to assert. The command field also contains various bits that help with queue management and transfer protocol. See Section 15.8.2.6, “DSPI PUSH TX FIFO Register (DSPI_PUSHR)” for details on the SPI command fields. The data field in the executing TX FIFO entry is loaded into the shift register and shifted out on the Serial Out (SOUT) pin. In SPI master mode, each SPI frame to be transmitted has a command associated with it allowing for transfer attribute control on a frame by frame basis.

15.9.2.2 Slave mode

In SPI slave mode the DSPI responds to transfers initiated by a SPI bus master. The DSPI does not initiate transfers. Certain transfer attributes such as clock polarity, clock phase and frame size must be set for successful communication with a SPI master. The SPI slave mode transfer attributes are set in the DSPI_CTAR0.

15.9.2.3 FIFO disable operation

The FIFO disable mechanisms allow SPI transfers without using the TX FIFO or RX FIFO. The DSPI operates as a double-buffered simplified SPI when the FIFOs are disabled. The FIFOs are disabled separately; setting the DSPI_MCR[DIS_TXF] bit disables the TX FIFO, and setting the DSPI_MCR[DIS_RXF] bit disables the RX FIFO.

The FIFO Disable mechanisms are transparent to the user and to host software; Transmit data and commands are written to the DSPI_PUSHR and received data is read from the DSPI_POPR.

When the TX FIFO is disabled the TFFF, TFUF and TXCTR fields in DSPI_SR behave as if there is a one-entry FIFO but the contents of the DSPI_TXFR registers and TXNXTPTR are undefined. Likewise, when the RX FIFO is disabled, the RFDF, RFOF and RXCTR fields in the DSPI_SR behave as if there is a one-entry FIFO, but the contents of the DSPI_RXFR registers and POPNXTPTR are undefined.

15.9.2.4 Transmit first-in first-out (TX FIFO) buffering mechanism

The TX FIFO functions as a buffer of SPI data and SPI commands for transmission. The TX FIFO holds from 1 to 16 words, each consisting of a command field and a data field. In MPC5676R, the number of entries in the TX FIFO is 4 words deep. SPI commands and data are added to the TX FIFO by writing to the DSPI PUSH TX FIFO Register (DSPI_PUSHR). TX FIFO entries can only be removed from the TX FIFO by being shifted out or by flushing the TX FIFO.

The TX FIFO Counter field DSPI_SR[TXCTR] indicates the number of valid entries in the TX FIFO. Field DSPI_SR[TXCTR] is updated every time the DSPI _PUSHR is written or SPI data is transferred into the shift register from the TX FIFO.

Field DSPI_SR[TXNXTPTR] indicates which TX FIFO Entry will be transmitted during the next transfer. Field DSPI_SR[TXNXTPTR] contains the positive offset from DSPI_TXFR0 in number of 32-bit registers. For example, TXNXTPTR equal to two means that the DSPI_TXFR2 contains the SPI data and command for the next transfer. Field DSPI_SR[TXNXTPTR] is incremented every time SPI data is transferred from the TX FIFO to the shift register. The maximum value of the field is equal to DSPI_HCR[TXFR] and it rolls over after reaching the maximum.

15.9.2.4.1 Filling the TX FIFO

Host software or other intelligent blocks can add (push) entries to the TX FIFO by writing to the DSPI_PUSHR. When the TX FIFO is not full, the TX FIFO Fill Flag (TFFF) in the DSPI_SR is set. The TFFF bit is cleared when TX FIFO is full and the DMA controller indicates that a write to DSPI_PUSHR is complete. Writing a ‘1’ to the TFFF bit also clears it. The TFFF can generate a DMA request or an interrupt request. See Section 15.9.10.2, “Transmit FIFO fill interrupt or DMA request,” for details.

The DSPI ignores attempts to push data to a full TX FIFO, the state of the TX FIFO does not change and no error condition is indicated.

15.9.2.4.2 Draining the TX FIFO

The TX FIFO entries are removed (drained) by shifting SPI data out through the shift register. Entries are transferred from the TX FIFO to the shift register and shifted out as long as there are valid entries in the TX FIFO. Every time an entry is transferred from the TX FIFO to the shift register, the TX FIFO Counter decrements by one. At the end of a transfer, bit DSPI_SR[TCF] is set to indicate the completion of a transfer. The TX FIFO is flushed by writing a ‘1’ to bit DSPI_MCR[CLR_TXF].

If an external bus master initiates a transfer with a DSPI slave while the slave’s DSPI TX FIFO is empty, the Transmit FIFO Underflow Flag (TFUF) in the slave’s DSPI_SR is set. See Section 15.9.10.4, “Transmit FIFO underflow interrupt request,” for details.

15.9.2.5 Receive first-in first-out (RX FIFO) buffering mechanism

The RX FIFO functions as a buffer for data received on the SIN pin. The RX FIFO holds from 1 to 16 received SPI data frames. In MPC5676R, the number of entries in the RX FIFO is 4 words deep. SPI data is added to the RX FIFO at the completion of a transfer when the received data in the shift register is transferred into the RX FIFO. SPI data are removed (popped) from the RX FIFO by reading the DSPI POP RX FIFO Register (DSPI_POPR). RX FIFO entries can only be removed from the RX FIFO by reading the DSPI_POPR or by flushing the RX FIFO.

The RX FIFO Counter field DSPI_SR[RXCTR] indicates the number of valid entries in the RX FIFO. Field DSPI_SR[RXCTR] is updated every time the DSPI _POPR is read or SPI data is copied from the shift register to the RX FIFO.

Field DSPI_SR[POPNXTPTR] points to the RX FIFO entry that is returned when the DSPI_POPR is read. Field DSPI_SR[POPNXTPTR] contains the positive offset from DSPI_RXFR0 in number of 32-bit registers. For example, POPNXTPTR equal to two means that the DSPI_RXFR2 contains the received SPI data that will be returned when DSPI_POPR is read. Field DSPI_SR[POPNXTPTR] is incremented every time the DSPI_POPR is read. The maximum value of the field is equal to DSPI_HCR[RXFR] and it rolls over after reaching the maximum.

15.9.2.5.1 Filling the RX FIFO

The RX FIFO is filled with the received SPI data from the shift register. While the RX FIFO is not full, SPI frames from the shift register are transferred to the RX FIFO. Every time a SPI frame is transferred to the RX FIFO the RX FIFO Counter is incremented by one.

If the RX FIFO and shift register are full and a transfer is initiated, the RFOF bit in the DSPI_SR is set indicating an overflow condition. Depending on the state of the ROOE bit in the DSPI_MCR, the data from the transfer that generated the overflow is either ignored or shifted in to the shift register. If the ROOE bit is set, the incoming data is shifted in to the shift register. If the ROOE bit is cleared, the incoming data is ignored.

15.9.2.5.2 Draining the RX FIFO

Host CPU or a DMA can remove (pop) entries from the RX FIFO by reading the DSPI POP RX FIFO Register (DSPI_POPR). A read of the DSPI_POPR decrements the RX FIFO Counter by one. Attempts to pop data from an empty RX FIFO are ignored and the RX FIFO Counter remains unchanged. The data, read from the empty RX FIFO, is undetermined.

When the RX FIFO is not empty, the RX FIFO Drain Flag (RFDF) in the DSPI_SR is set. The RFDF bit is cleared when the RX_FIFO is empty and the DMA controller indicates that a read from DSPI_POPR is complete or by writing a ‘1’ to it.

15.9.3 Deserial serial interface (DSI) configuration

The DSI configuration supports pin count reduction by serializing Parallel Input signals or register bits and shifting them out in a SPI-like protocol. The timing and transfer protocol is described in Section 15.9.6, “Transfer formats.” The received serial frames are converted to a parallel form (deserialized) and placed on the Parallel Output signals or in the DSPI_DDR. The various features of the DSI configuration are set in the DSPI DSI Configuration Register (DSPI_DSICR).

The DSI frames can be from 4 to 32 bits. With Multiple Transfer Operation (MTO) the DSPI supports serial chaining of DSPI modules within a device to create DSI frames up to 64 bits, consisting of concatenated bits from multiple DSPIs. The DSPI also supports parallel chaining allowing several DSPIs and off-chip SPI devices to share the same Serial Communications Clock (SCK) and Peripheral Chip Select (PCS) signals. See Section 15.9.3.6, “Multiple transfer operation (MTO),” for details on the serial and parallel chaining support.

15.9.3.1 DSI Master mode

In DSI master mode the DSPI initiates and controls the DSI transfers. The DSI master has four different conditions that can initiate a transfer:

- Continuous
- Change in data
- Trigger signal
- Trigger signal combined with a change in data

The four transfer initiation conditions are described in Section 15.9.3.5, “DSI transfer initiation control.” Transfer attributes are set during initialization. Field DSPI_DSICR[DSICTAS] determines which of the DSPI_CTAR registers will control the transfer attributes.

15.9.3.2 Slave mode

In DSI slave mode the DSPI responds to transfers initiated by a SPI or DSI bus master. In this mode the DSPI does not initiate DSI transfers. Certain transfer attributes such as clock polarity and phase must be set for successful communication with a DSI master. The DSI slave mode Transfer attributes are set in the DSPI_CTAR1.

If the CID bit in the DSPI_DSICR is set and the data in the DSPI_COMPR differs from the selected source of the serialized data, the slave DSPI will assert the $\overline{\text{MTRIG}}$ signal. If the slave’s HT signal is asserted and the TRRE is set, the slave DSPI asserts $\overline{\text{MTRIG}}$. These features are included to support chaining of several DSPI. Details about the $\overline{\text{MTRIG}}$ signal is found in Section 15.9.3.6, “Multiple transfer operation (MTO).”

15.9.3.3 DSI serialization

In the DSI configuration from 4 to 16 bits can be serialized using 2 different sources. The TXSS bit in the DSPI_DSICR selects between the DSPI DSI Serialization Data Register (DSPI_SDR) and the DSPI DSI Alternate Serialization Data Register (DSPI_ASDR) as the source of the serialized data. The DSPI_SDR holds the latest Parallel Input signal values which is sampled at every rising edge of the system clock. The DSPI_ASDR is written by host software and used as an alternate source of serialized data.

The DSPI_PISR0–3 registers allow to change relative position of the Parallel input pins in the transmitted frame. Each transmitted frame bit can be selected from 16 adjacent Parallel Inputs by writing IPSn fields. The IPSn field is treated as a 4-bit integer number, representing numbers from –8 to 7. The Parallel Input pin number, selected by IPSn field is defined by the difference between sum IPSn field number (n) and the IPSn field value. If the operation result is negative the number 32 should be added. If the result is higher than 32, 32 should be subtracted from the result.

For example, IPS0, set to minus 1 (binary 1111), preselects Parallel Input 1 to 0 position in the transmitted frame.

IPS6, set to 3 (binary 0011), preselects Parallel Input 3 to be bit number 6 in the transmitted frame, while the value minus 2 (1110) preselects Parallel Input 8.

IPS31, set to minus 8 (binary 1000), preselects Parallel Input 7 to be bit number 31 in the transmitted frame.

(Of course, the Parallel Input pin state, to be transmitted, should be selected by TXSS and the frame size should be higher than the bit position in the preselected frame.)

The DSPI_SSR provides additional way to create the frame for transmission. Each bit from this register is OR’d with the TXSS bit and controls individual transmitted bit source. This way, the transmitted frame can have any combination of the DSPI_SDR and DSPI_ASDR bits. This feature allows control SPI based devices, requiring control and data fields in the frame. Control field may come from DSPI_ASDR, set by the device’s CPU, while data field can be generated by device peripheral modules, such as PWM timers.

A copy of the last 32-bit DSI frame shifted out of the Shift Register is stored in the DSPI DSI Transmit Comparison Register (DSPI_COMPR). This register provides added visibility for debugging and it serves as a reference for transfer initiation control. Figure 15-30 shows the DSI Serialization logic.

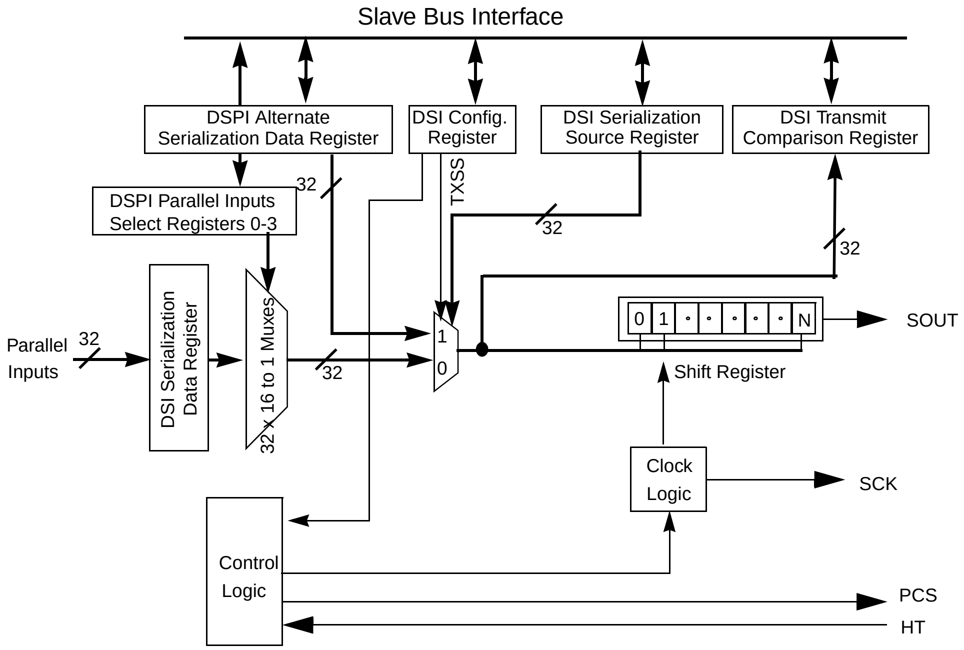


Figure 15-30. DSI serialization diagram

15.9.3.4 DSI deserialization

When all bits in a DSI frame have been shifted in, the frame is copied to the DSPI DSI Deserialization Data Register (DSPI_DDR). This register presents the deserialized data as Parallel Output signal values. The DSPI_DDR is memory mapped to allow host software to read the deserialized data directly.

The received data is bit-wise compared to the value of the DSI Deserialized Data Polarity Interrupt Register, bit-wise AND’ed with DSI Deserialized Interrupt Mask Register and the results OR’ed to produce the DDIF flag in the DSPI_SR, which in turn can cause a DDI interrupt request if the DDIFRE bit of DSPI_RSER is set.

Figure 15-31 shows the DSI deserialization logic.

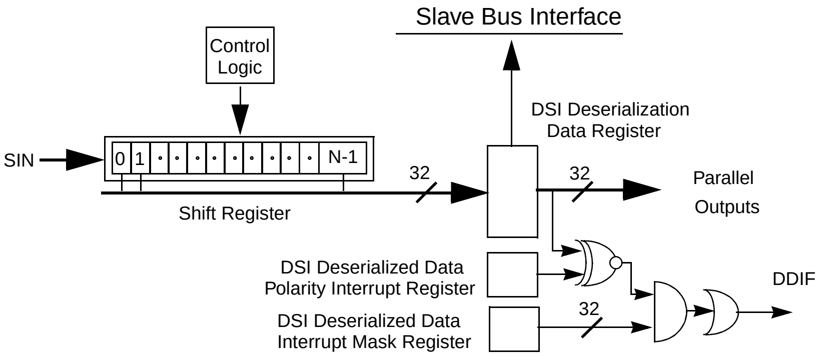


Figure 15-31. DSI deserialization diagram

15.9.3.5 DSI transfer initiation control

Data transfers for a master DSPI in DSI configuration are initiated by a condition. The transfer initiation conditions are selected by the TRRE and CID bits in the DSPI_DSICR. Table 15-32 lists the four transfer initiation conditions.

Table 15-32. DSI data transfer initiation control

DSPI_DSICR bits		Transfer initiation control
TRRE	**CID**	
0	0	Continuous
0	1	Change in Data
1	0	Triggered
1	1	Triggered or Change in Data

15.9.3.5.1 Continuous control

For Continuous Control a new DSI frame shifts out when the previous transfer cycle has completed and the Delay after Transfer (t_{DT}) has elapsed.

15.9.3.5.2 Change in data control

For Change in Data Control a transfer is initiated when the data to be serialized has changed since the transfer of the last DSI frame. A copy of the previously transferred DSI data is stored in the DSPI_COMPR. When the data, selected for the transfer from the DSPI_SDR and DSPI_ASDR registers is different from the data in the DSPI_COMPR a new DSI frame is transmitted. The $\overline{\mathrm{MTRIG}}$ output signal is asserted every time a change in data is detected.

15.9.3.5.3 Triggered control

For Triggered Control initiation of a transfer is controlled by the Hardware Trigger signal (HT). The TPOL bit in the DSPI_DSICR selects the active edge of HT. For HT to have any affect, the TRRE bit in the DSPI_DSICR must be set.

15.9.3.5.4 Triggered or change in data control

For Triggered or Change in Data Control initiation of a transfer is controlled by the HT signal or by the detection of a change in data to be serialized.

15.9.3.6 Multiple transfer operation (MTO)

In DSI configuration the MTO feature allows for multiple DSPIs within a device to be chained together in a parallel or serial configuration. The parallel chaining allows multiple DSPIs internal to a device and multiple SPI devices external to a device to share SCK and PCS signals thereby helping to minimize device pin count. The serial chaining allows bits from multiple DSPIs to be concatenated into a single DSI frame. MTO is enabled by setting the MTOE bit in the DSPI_DSICR.

In parallel and serial chaining there is one bus master and multiple bus slaves. The bus master initiates and controls the transfers, but the DSPI slaves generate trigger signals for the bus DSPI master when an internal condition in the slave warrants a transfer. The DSPI slaves also propagate triggers from other slaves to the master. When a DSPI slave detects a trigger signal on its HT input, the slave generates a trigger signal on the $\overline{\mathrm{MTRIG}}$ output.

Serial and parallel chaining require multiplexing of signals external to the DSPI.

NOTE

TSB operation is not available in MTO mode. TSBC and MTOE bits of DSPI_DSICR should not be set simultaneously.

15.9.3.6.1 Parallel chaining

NOTE

When using the DSPI in DSI mode, with MTO enabled, and clock phase set to leading edge capture (DSPIx_CTARn[CPHA]=0) the first bit shifted out of the master DSPI into the slave DSPI is read as “1”, regardless of the actual value. To account for this behavior, the following options are recommended:

- Select CPHA=1 (following edge capture), if suitable for external slave devices.
- Set the first bit of the transferred data to “1”, or ignore the first bit.
- Externally connect master SOUT to SIN of the first slave, rather than connecting via internal signals. This requires setting SIU_DISR_SINSEL*x* bits of the first slave DSPI to “00” and configuring the first slave's SIN pin and master SOUT pin as DSPI SIN and DSPI SOUT, respectively.

Parallel chaining allows the PCS and SCK signals from a Master DSPI to be shared by internal Slave DSPIs and external Slave SPI devices, thus reducing pin utilization of the MPC5676R MCU. Signal sharing reduces DSPI pin utilization. An example of a parallel chain is shown in Figure 15-32. In this example, the SOUT and SIN of the three DSPIs connect to separate external SPI devices, which share a common PCS and SCK.

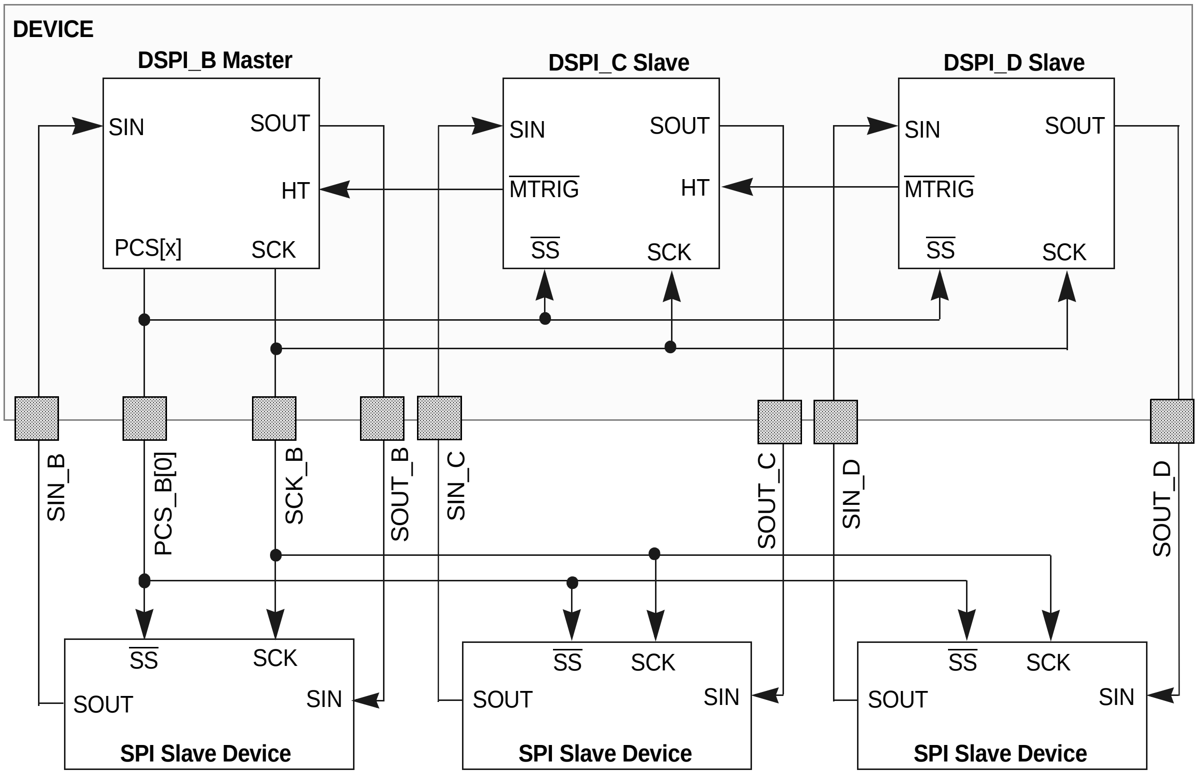


Figure 15-32. DSPI parallel chaining example

In the parallel chaining example, the SOUT and SIN of the three DSPIs connect to separate external SPI devices. All internal and external SPI blocks share PCS and SCK signals. DSPI_B controls and initiates all transfers, but the DSPI slaves each have a trigger output signal $\overline{\text{MTRIG}}$ that indicates to DSPI_B that a trigger condition has occurred in the DSPI slaves. When the slave DSPI has a change in data to be serialized, it asserts the $\overline{\text{MTRIG}}$ signal that propagates to DSPI_B which initiates the transfer.

15.9.3.7 Serial chaining

Serial chaining allows SPI operation with an external device that has more bits than one DSPI module. In a serial chain, one DSPI module operates as a master, the other DSPI modules operate as slaves.

The data output (SOUT) of the master is connected to the data input (SIN) of the slave. The SOUT of a slave is connected to the SIN of subsequent slaves until the last block in the chain, where the SOUT is connected to an external pin, which connects to the input of an external SPI device. The slave DSPI and external SPI device use the master peripheral chip select (PCS) and clock (SCK).

The Trigger input of the master allows a slave DSPI to trigger a transfer when a data change occurs in the slave DSPI and the slave DSPI is operating in Change in Data mode. The Trigger input of the master is connected to $\overline{\text{MTRIG}}$ output of the slave.

The concatenated frames can be from 8 to 64 bits long. Figure 15-33 shows an example of how the blocks can be connected in the MPC5676R.

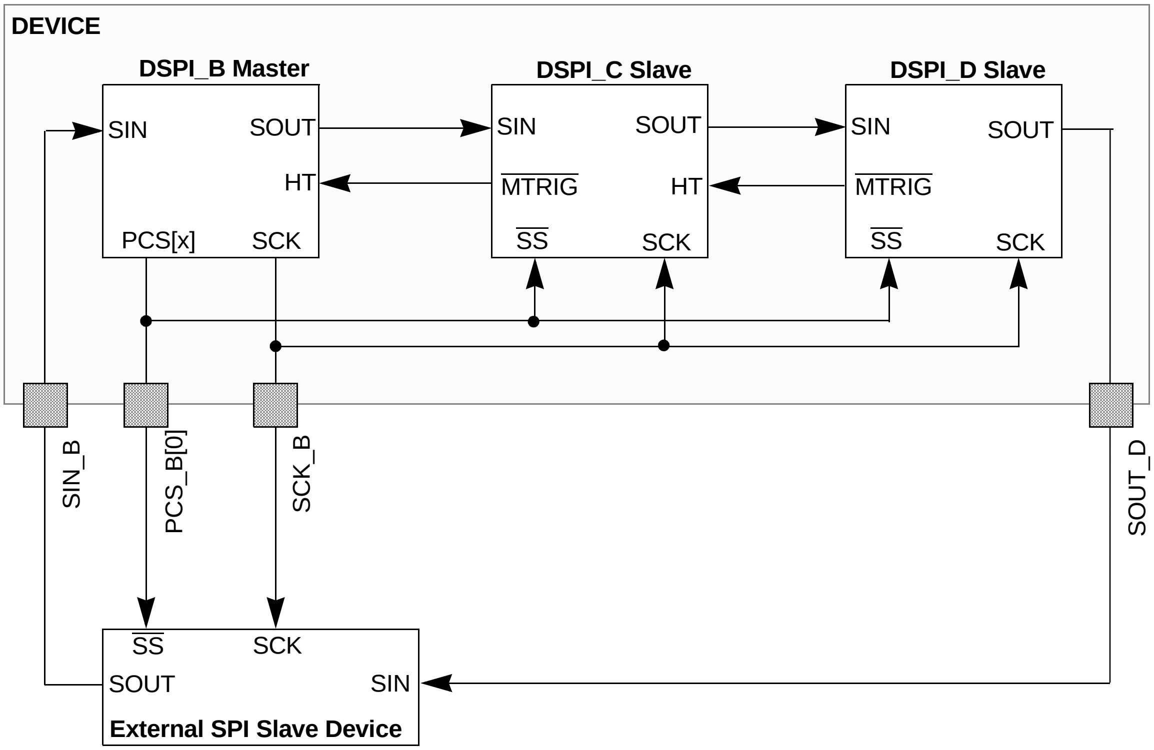


Figure 15-33. DSPI serial chaining example

The SOUT of DSPI_B is connected to the SIN of DSPI_C, the SOUT of DSPI_C is connected to the SIN of DSPI_D and the SOUT of the DSPI_D is connected to the SIN of the external SPI slave. The SOUT of the external SPI slave is connected to the SIN of DSPI_B.

DSPI_B controls and initiates all transfers, but the slave DSPIs use the trigger output signal $\overline{\text{MTRIG}}$ to indicate to DSPI_B that a trigger condition has occurred. When an on-chip DSPI slave has a change in data to be serialized it can assert the $\overline{\text{MTRIG}}$ signal to the DSPI master which initiates the transfer. The DSPI slaves also propagate trigger signals from other slaves to the DSPI master.

Field DSPI_DSICR[MTOCNT] in DSPI_B must be written with the total number of bits to be transferred. Field DSPI_DSICR[MTOCNT] must equal the sum of all FMSZ fields in the selected DSPI_CTAR registers for DSPI_B and all on-chip DSPI slaves. For example, if one 16-bit DSI frame is created by concatenating 8 bits from DSPI_B and 4 bits from DSPI_C and DSPI_D each, then DSPI_B's frame size must be set to 8, and the DSPI slaves' frame size must be set to 4 each. Field DSPI_DSICR[MTOCNT] in DSPI_B must be set to 16.

15.9.3.8 IMUX/SIU support for serial and parallel chaining

To support MTO, each DSPI in the MPC5676R has multiplexers on the SIN, $\overline{\text{SS}}$, SCK, and HT inputs. The Internal Multiplexers (IMUX) reside in the SIU module on the MPC5676R.

15.9.4 Combined serial interface (CSI) configuration

The CSI configuration of the DSPI is used to support SPI and DSI functions on a frame by frame basis. CSI configuration allows interleaving of DSI data frames from the Parallel Input signals with SPI commands and data from the TX FIFO. The data returned from the bus slave is either used to drive the Parallel Output signals or it is stored in the RX FIFO. The CSI configuration allows serialized data and configuration or diagnostic data to be transferred to a slave device using only one serial link. The DSPI is in CSI configuration when field DSPI_MCR[DCONF] is 0b10. Figure 15-34 shows an example of how a DSPI can be used with a deserializing peripheral that supports SPI control for control and diagnostic frames.

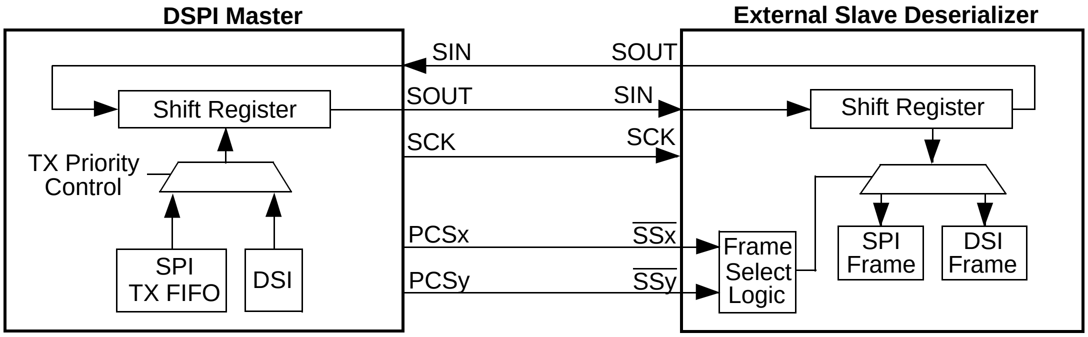


Figure 15-34. Example of system using DSPI in CSI configuration

In CSI configuration the DSPI transfers DSI data based on DSI transfer initiation control. When there are SPI commands in the TX FIFO, the SPI data has priority over the DSI frames. When the TX FIFO is empty, DSI transfer resumes.

Two peripheral chip select signals indicate whether DSI data or SPI data is transmitted. The user must configure the DSPI so that the two DSPI_CTAR registers associated with DSI data and SPI data assert different peripheral chip select signals denoted in the figure as PCSx and PCSy. The CSI configuration is only supported in master mode.

Data returned from the external slave while a DSI frame is transferred is placed on the Parallel Output signals. Data returned from the external slave while a SPI frame is transferred is moved to the RX FIFO. The TX FIFO and RX FIFO are fully functional in CSI mode.

15.9.4.1 CSI serialization

Serialization in the CSI configuration is similar to serialization in DSI configuration. The transfer attributes for SPI frames are determined by the DSPI_CTAR selected by the CTAS field in the SPI command halfword. The transfer attributes for the DSI frames are determined by the DSPI_CTAR selected by field DSPI_DSICR[DSICTAS].

The Parallel Inputs signal states are latched into the DSPI DSI Serialization Data Register (DSPI_SDR) on the rising edge of every system clock and serialized based on the transfer initiation control settings in the DSPI_DSICR. When SPI frames are written to the TX FIFO they have priority over DSI data from the DSPI_SDR and are transferred at the next frame boundary. A copy of the most recently transferred DSI frame is stored in the DSPI_COMPR. The Transfer Priority Logic selects the source of the serialized data and asserts the appropriate PCS signal.

15.9.4.2 CSI deserialization

The deserialized frames in CSI configuration goes into the DSPI_DDR or the RX FIFO based on the transfer priority logic. When DSI frames are transferred the returned frames are deserialized and latched into the DSPI_DDR. When SPI frames are transferred the returned frames are deserialized and written to the RX FIFO.

15.9.5 DSPI baud rate and clock delay generation

The SCK frequency and the delay values for serial transfer are generated by dividing the system clock frequency by a prescaler and a scaler with the option for doubling the baud rate. Figure 15-35 shows conceptually how the SCK signal is generated.

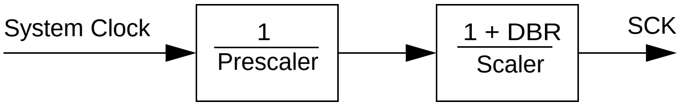


Figure 15-35. Communications clock prescalers and scalers

15.9.5.1 Baud rate generator

The baud rate is the frequency of the Serial Communication Clock (SCK). The system clock is divided by a prescaler (PBR) and scaler (BR) to produce SCK with the possibility of halving the scaler division. The DBR, PBR and BR fields in the DSPI_CTAR registers select the frequency of SCK by the formula in the BR field description. Table 15-33 shows an example of how to compute the baud rate.

Table 15-33. Baud rate computation example

f_{sys}	PBR	Prescaler	BR	Scaler	DBR	Baud rate
100 MHz	0b00	2	0b0000	2	0	25 Mb/s
20 MHz	0b00	2	0b0000	2	1	10 Mb/s

15.9.5.2 PCS to SCK delay (t_{CSC})

The PCS to SCK delay is the length of time from assertion of the PCS signal to the first SCK edge. See Figure 15-37 for an illustration of the PCS to SCK delay. The PCSSCK and CSSCK fields in the DSPI_CTAR*x* registers select the PCS to SCK delay by the formula in the CSSCK field description. Table 15-34 shows an example of how to compute the PCS to SCK delay.

Table 15-34. PCS to SCK delay computation example

f_{sys}	PCSSCK	Prescaler	CSSCK	Scaler	PCS to SCK delay
100 MHz	0b01	3	0b0100	32	0.96 µs

PCSCSK and CSSCK fields have no effect in TSB configuration.

15.9.5.3 After SCK delay (t_{ASC})

The After SCK Delay is the length of time between the last edge of SCK and the negation of PCS. See Figure 15-37 and Figure 15-38 for illustrations of the After SCK delay. The PASC and ASC fields in the DSPI_CTAR*x* registers select the After SCK Delay by the formula in the ASC field description. Table 15-35 shows an example of how to compute the After SCK delay.

Table 15-35. After SCK delay computation example

f_{sys}	PASC	Prescaler	ASC	Scaler	After SCK delay
100 MHz	0b01	3	0b0100	32	0.96 µs

PCASC and ASC fields have no effect in TSB configuration.

15.9.5.4 Delay after transfer (t_{DT})

The Delay after Transfer is the minimum time between negation of the PCS signal for a frame and the assertion of the PCS signal for the next frame. See Figure 15-37 for an illustration of the Delay after Transfer. The PDT and DT fields in the DSPI_CTAR*x* registers select the Delay after Transfer by the formula in the DT field description. Table 15-36 shows an example of how to compute the Delay after Transfer.

Table 15-36. Delay after transfer computation example

f_{sys}	PDT	Prescaler	DT	Scaler	Delay after transfer
100 MHz	0b01	3	0b1110	32768	0.98 ms

When in non-continuous clock mode the t_{DT} delay is configured according Equation 15-3. When in continuous clock mode and TSB is not enabled the delay is fixed at 1 SCK period.

In TSB mode the Delay after Transfer is equal to a number formed by concatenation of PDT and DT fields plus 1 of the SCK clock periods. See detailed information in Section 15.9.8, “Timed serial bus (TSB)”.

15.9.5.5 Peripheral chip select strobe enable ($\overline{\text{PCSS}}$)

The $\overline{\text{PCSS}}$ signal provides a delay to allow the PCS signals to settle after a transition occurs thereby avoiding glitches. When the DSPI is in master mode and PCSSE bit is set in the DSPI_MCR, $\overline{\text{PCSS}}$ provides a signal for an external demultiplexer to decode the DSPI_x_PCS[0] – PCS[4] and DSPI_x_PCS[6] – PCS[7] signals into as many as 128 glitch-free PCS signals. Figure 15-36 shows the timing of the $\overline{\text{PCSS}}$ signal relative to PCS signals.

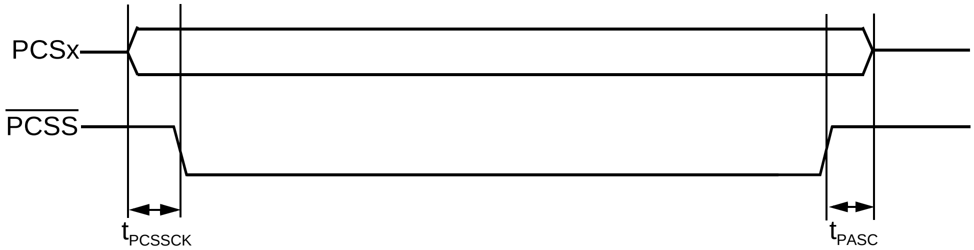


Figure 15-36. Peripheral chip select strobe timing

The delay between the assertion of the PCS signals and the assertion of $\overline{\text{PCSS}}$ is selected by field DSPI_CTAR[PCSSCK] based on the following formula:

$$t_{PCSSCK} = \frac{1}{f_{SYS}} \times PCSSCK \qquad \textbf{\textit{Eqn. 15-5}}$$

At the end of the transfer the delay between $\overline{\text{PCSS}}$ negation and PCS negation is selected by field DSPI_CTAR[PASC] based on the following formula:

$$t_{PASC} = \frac{1}{f_{SYS}} \times PASC \qquad \textbf{\textit{Eqn. 15-6}}$$

Table 15-37 shows an example of how to compute the t_{pcssck} delay.

Table 15-37. Peripheral chip select strobe assert computation example

f_{sys}	PCSSCK	Prescaler	Delay before transfer
100 MHz	0b11	7	70.0 ns

Table 15-38 shows an example of how to compute the t_{pasc} delay.

Table 15-38. Peripheral chip select strobe negate computation example

f_{sys}	PASC	Prescaler	Delay after transfer
100 MHz	0b11	7	70.0 ns

The $\overline{\text{PCSS}}$ signal is not supported when Continuous Serial Communication SCK or TSB mode are enabled.

15.9.6 Transfer formats

The SPI serial communication is controlled by the Serial Communications Clock (SCK) signal and the PCS signals. The SCK signal provided by the master device synchronizes shifting and sampling of the data on the SIN and SOUT pins. The PCS signals serve as enable signals for the slave devices.

When the DSPI is the bus master, the CPOL and CPHA bits in the DSPI Clock and Transfer Attributes Registers (DSPI_CTARx) select the polarity and phase of the serial clock, SCK. The polarity bit selects the idle state of the SCK. The clock phase bit selects if the data on SOUT is valid before or on the first SCK edge.

When the DSPI is the bus slave, CPOL and CPHA bits in the DSPI_CTAR0 (SPI) or DSPI_CTAR1 (DSI) select the polarity and phase of the serial clock. Even though the bus slave does not control the SCK signal, clock polarity, clock phase and number of bits to transfer must be identical for the master and the slave devices to ensure proper transmission.

The DSPI supports four different transfer formats:

- Classic SPI with CPHA = 0
- Classic SPI with CPHA = 1
- Modified Transfer format with CPHA = 0
- Modified Transfer format with CPHA = 1

A modified transfer format is supported to allow for high-speed communication with peripherals that require longer setup times. The DSPI can sample the incoming data later than halfway through the cycle to give the peripheral more setup time. The MTFE bit in the DSPI_MCR selects between Classic SPI Format and Modified Transfer Format.

In the SPI and DSI configurations, the DSPI provides the option of keeping the PCS signals asserted between frames. See Section 15.9.6.5, “Continuous selection format,” for details.

15.9.6.1 Classic SPI transfer format (CPHA = 0)

The transfer format shown in Figure 15-37 is used to communicate with peripheral SPI slave devices where the first data bit is available on the first clock edge. In this format, the master and slave sample their SIN pins on the odd-numbered SCK edges and change the data on their SOUT pins on the even-numbered SCK edges.

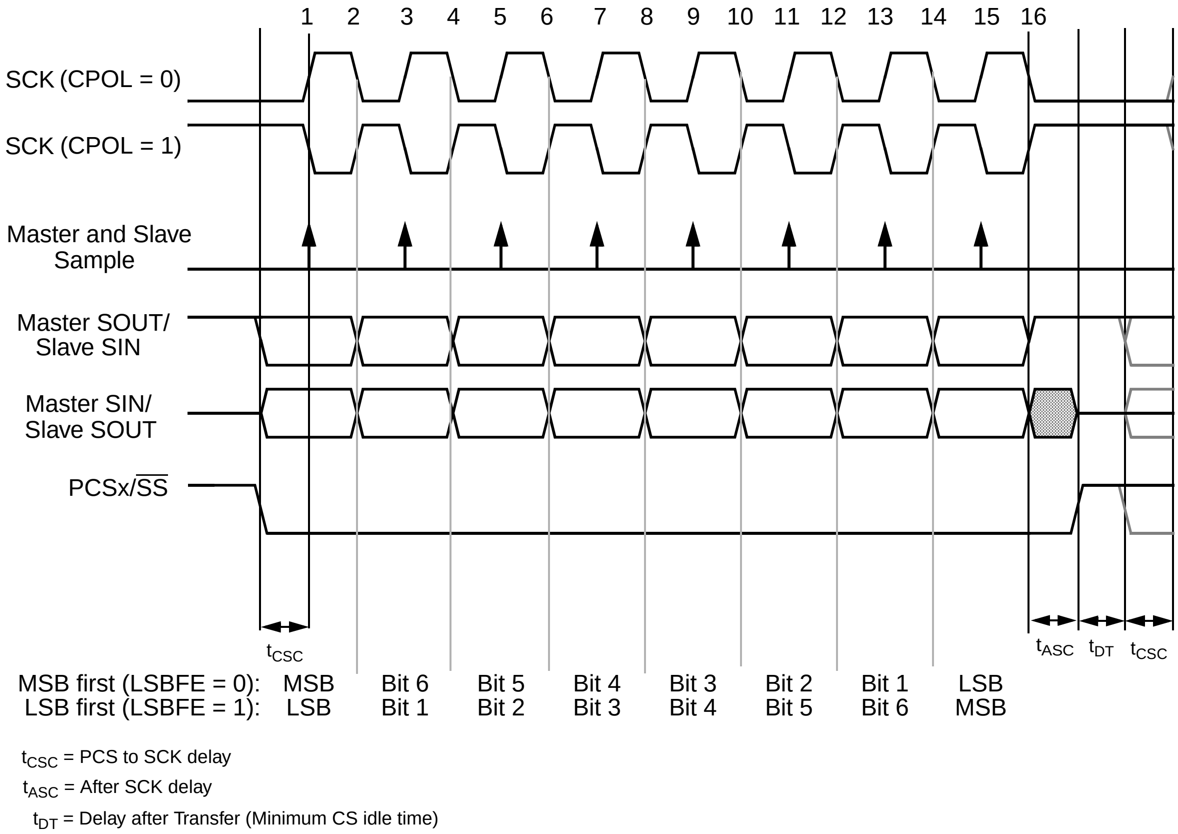
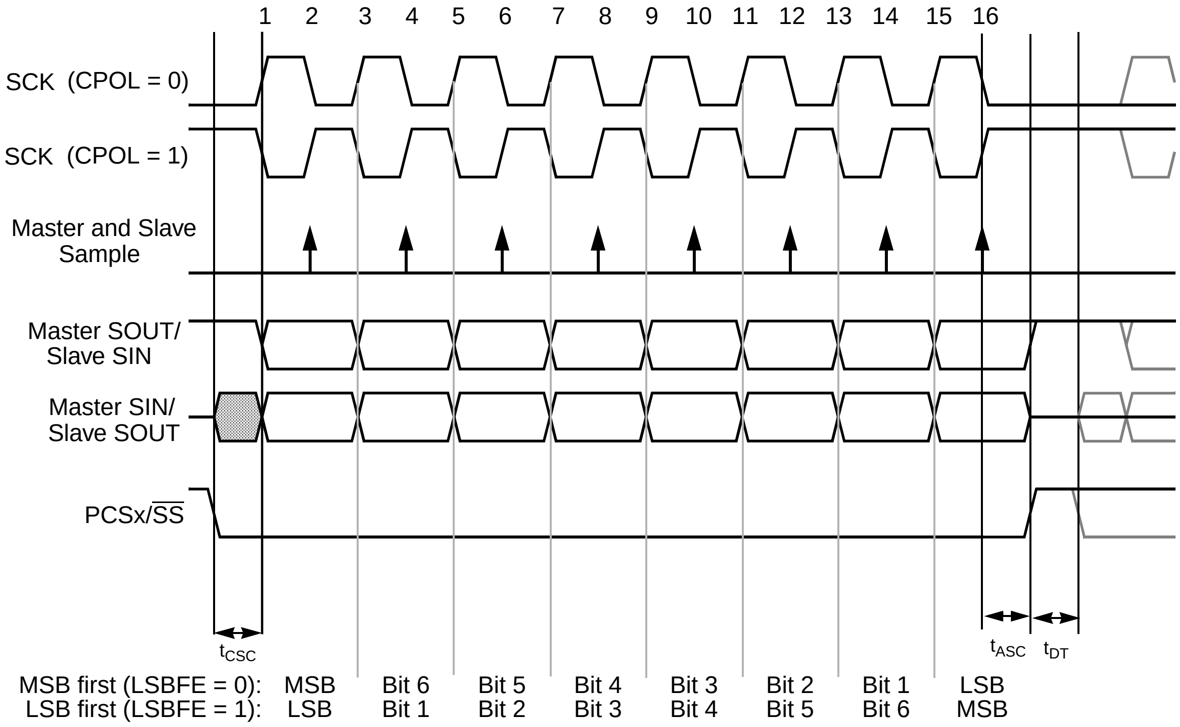


Figure 15-37. DSPI transfer timing diagram (MTFE = 0, CPHA = 0, FMSZ = 8)

The master initiates the transfer by placing its first data bit on the SOUT pin and asserting the appropriate peripheral chip select signals to the slave device. The slave responds by placing its first data bit on its SOUT pin. After the t_{CSC} delay elapses, the master outputs the first edge of SCK. The master and slave devices use this edge to sample the first input data bit on their serial data input signals. At the second edge of the SCK the master and slave devices place their second data bit on their serial data output signals. For the rest of the frame the master and the slave sample their SIN pins on the odd-numbered clock edges and changes the data on their SOUT pins on the even-numbered clock edges. After the last clock edge occurs a delay of t_{ASC} is inserted before the master negates the PCS signals. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

15.9.6.2 Classic SPI transfer format (CPHA = 1)

This transfer format shown in Figure 15-38 is used to communicate with peripheral SPI slave devices that require the first SCK edge before the first data bit becomes available on the slave SOUT pin. In this format the master and slave devices change the data on their SOUT pins on the odd-numbered SCK edges and sample the data on their SIN pins on the even-numbered SCK edges.



t_{CSC} = PCS to SCK delay
t_{ASC} = After SCK delay
t_{DT} = Delay after Transfer (minimum CS negation time)

Figure 15-38. DSPI Transfer Timing Diagram (MTFE = 0, CPHA = 1, FMSZ = 8)

The master initiates the transfer by asserting the PCS signal to the slave. After the t_{CSC} delay has elapsed, the master generates the first SCK edge and at the same time places valid data on the master SOUT pin. The slave responds to the first SCK edge by placing its first data bit on its slave SOUT pin.

At the second edge of the SCK the master and slave sample their SIN pins. For the rest of the frame the master and the slave change the data on their SOUT pins on the odd-numbered clock edges and sample their SIN pins on the even-numbered clock edges. After the last clock edge occurs a delay of t_{ASC} is inserted before the master negates the PCS signal. A delay of t_{DT} is inserted before a new frame transfer can be initiated by the master.

15.9.6.3 Modified SPI/DSI transfer format (MTFE = 1, CPHA = 0)

In this Modified Transfer Format both the master and the slave sample later in the SCK period than in Classic SPI mode to allow tolerate more delays in device pads and board traces. These delays become a more significant fraction of the SCK period as the SCK period decreases with increasing baud rates.

The master and the slave place data on the SOUT pins at the assertion of the PCS signal. After the PCS to SCK delay has elapsed the first SCK edge is generated. The slave samples the master SOUT signal on every odd numbered SCK edge. The DSPI in the slave mode when the MTFE bit is set also places new data on the slave SOUT on every odd numbered clock edge. Regular external slave, configured with CPHA = 0 format drives its SOUT output at every even numbered SCK clock edge.

The DSPI master places its second data bit on the SOUT line one system clock after odd numbered SCK edge if the system frequency to SCK frequency ratio is higher than three. If this ratio is below four the master changes SOUT at odd numbered SCK edge. The point where the master samples the SIN is selected by field DSPI_MCR[SMPL_PT]. Table 15-5 lists the number of system clock cycles between the active edge of SCK and the master Sample point. The master sample point can be delayed by one or two system clock cycles. Field DSPI_MCR[SMPL_PT] should be set to ‘0’ if the system to SCK frequency ratio is less than 4.

The following timing diagrams illustrate the DSPI operation with MTFE = 1. Timing delays shown are:

- T_{csc} = PCS to SCK assertion delay
- T_{acs} = After SCK PCS negation delay
- T_{su_ms} = Master SIN setup time
- T_{hd_ms} = Master SIN hold time
- T_{vd_sl} = Slave data output valid time, time between slave data output SCK driving edge and data becomes valid.
- T_{su_sl} = Data setup time on slave data input
- T_{hd_sl} = Data hold time on slave data input
- T_{sys} = System clock period

Figure 15-39 shows the modified transfer format for CPHA = 0 and f_{sys}/f_{sck} = 4. Only the condition where CPOL = 0 is illustrated. Solid triangles show the data sampling clock edges. The two possible slave behaviors are shown.

- Signal, marked “SOUT of Ext Slave”, presents regular SPI slave serial output.
- Signal, marked “SOUT of DSPI Slave”, presents DSPI in the slave mode with MTFE bit set.

Other MTFE = 1 diagrams show DSPI SIN input as being driven by a regular external SPI slave, configured according DSPI master CPHA programming.

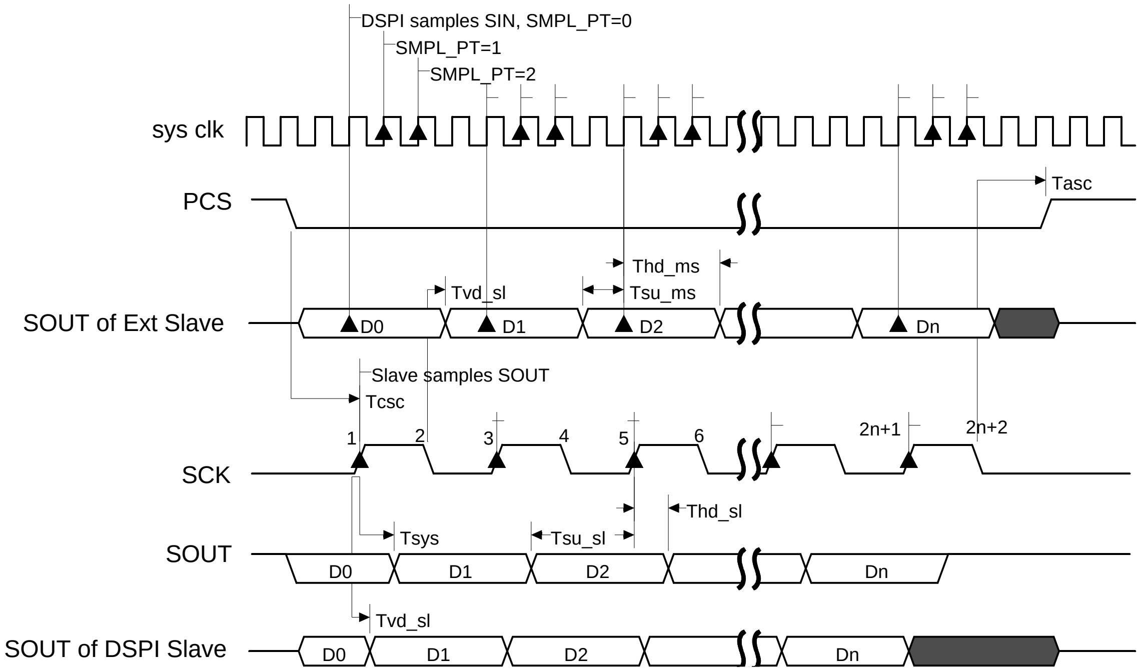


Figure 15-39. DSPI Modified Transfer Format (MTFE = 1, CPHA = 0, $f_{sck} = f_{sys}/4$)

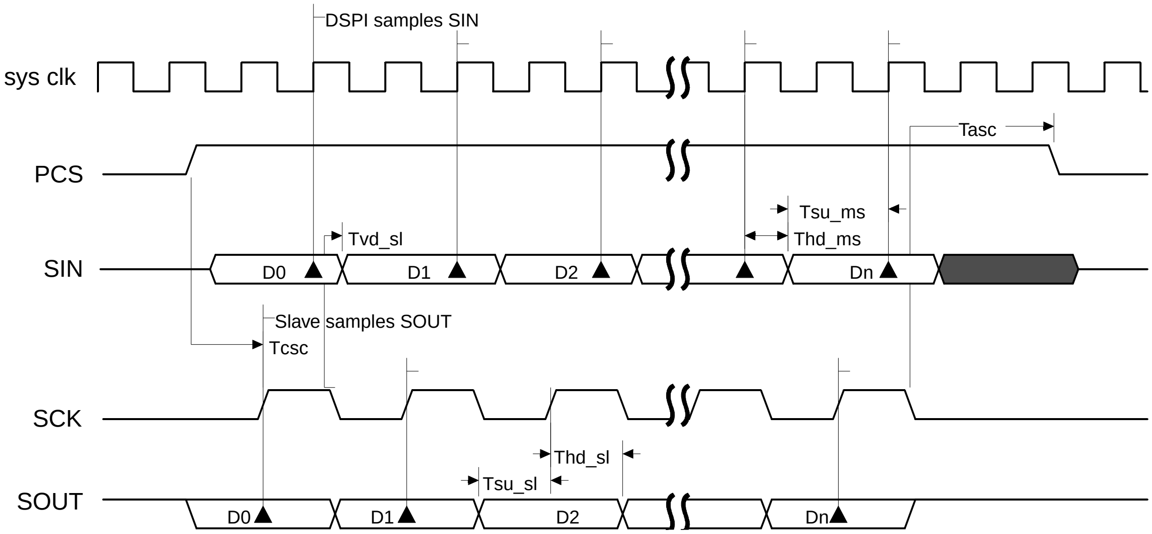


Figure 15-40. DSPI Modified Transfer Format (MTFE = 1, CPHA = 0, $f_{sck} = f_{sys}/2$)

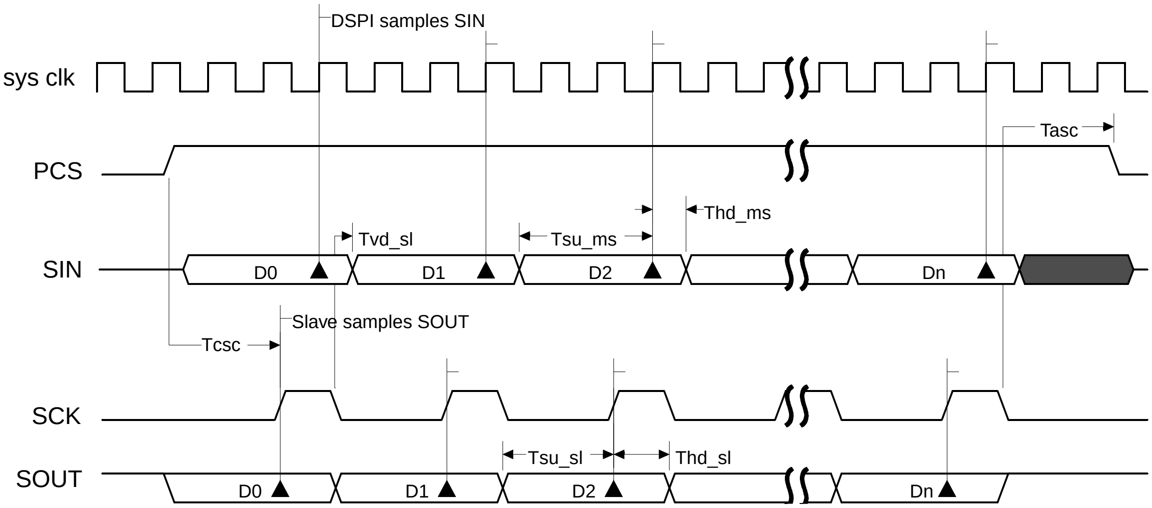


Figure 15-41. DSPI modified transfer format (MTFE = 1, CPHA = 0, $f_{sck} = f_{sys}/3$)

15.9.6.4 Modified SPI/DSI transfer format (MTFE = 1, CPHA = 1)

Figure 15-42 – Figure 15-44 show the Modified Transfer Format for CPHA = 1. Only the condition, where CPOL = 0 is shown. At the start of a transfer the DSPI asserts the PCS signal to the slave device. After the PCS to SCK delay has elapsed the master and the slave put data on their SOUT pins at the first edge of SCK. The slave samples the master SOUT signal on the even numbered edges of SCK. The master samples the slave SOUT signal on the odd numbered SCK edges starting with the third SCK edge. The slave samples the last bit on the last edge of the SCK. The master samples the last slave SOUT bit one half SCK cycle after the last edge of SCK. No clock edge will be visible on the master SCK pin during the sampling of the last bit. **The SCK to PCS delay must be greater or equal to half of the SCK period.**

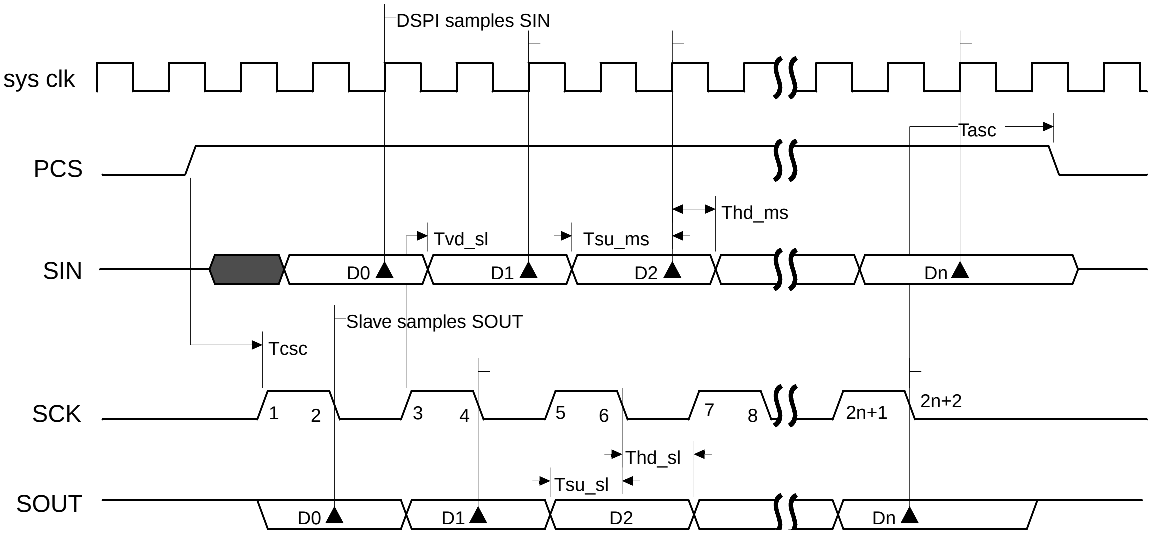


Figure 15-42. DSPI modified transfer format (MTFE = 1, CPHA = 1, $f_{sck} = f_{sys}/2$)

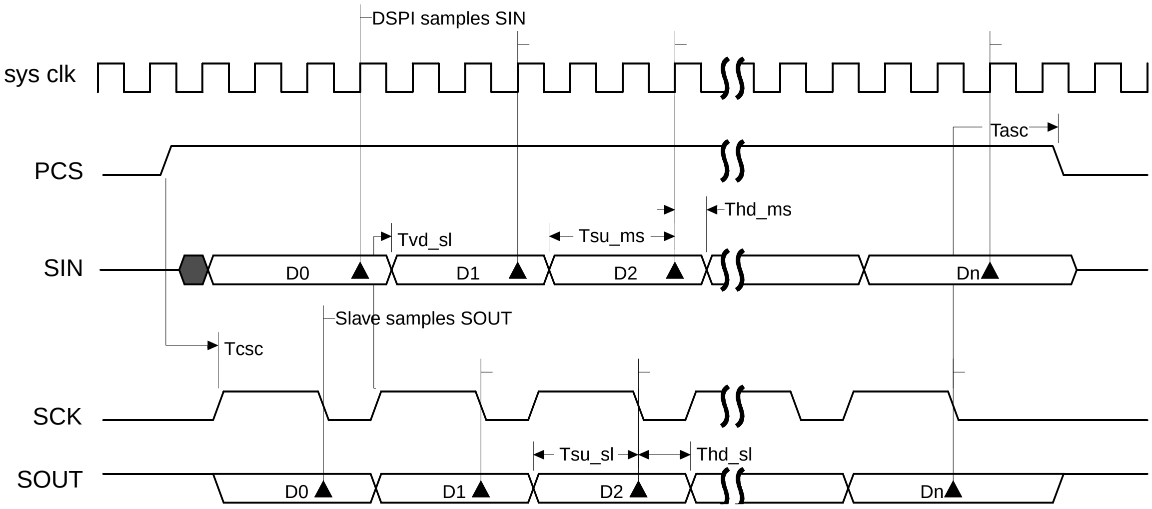


Figure 15-43. DSPI modified transfer format (MTFE = 1, CPHA = 1, $f_{sck} = f_{sys}/3$)

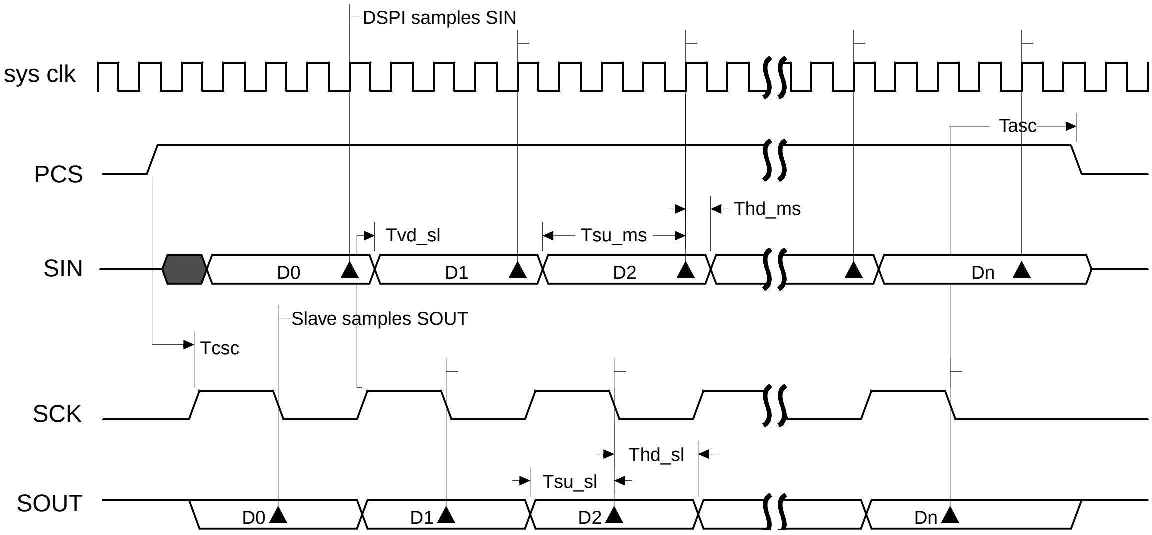


Figure 15-44. DSPI Modified transfer format (MTFE = 1, CPHA = 1, $f_{sck} = f_{sys}/4$)

15.9.6.5 Continuous selection format

Some peripherals must be deselected between every transfer. Other peripherals must remain selected between several sequential serial transfers. The Continuous Selection Format provides the flexibility to handle both cases. The Continuous Selection Format is enabled for the SPI configuration by setting the CONT bit in the SPI command. Continuous Selection is enabled for the DSI configuration by setting the DCONT bit in the DSPI_DSICR. The behavior of the PCS signals in the two configurations is identical so only SPI configuration will be described.

When the CONT bit = 0, the DSPI drives the asserted Chip Select signals to their idle states in between frames. The idle states of the Chip Select signals are selected by the PCSISn bits in the DSPI_MCR. Figure 15-45 shows the timing diagram for two 4-bit transfers with CPHA = 1 and CONT = 0.

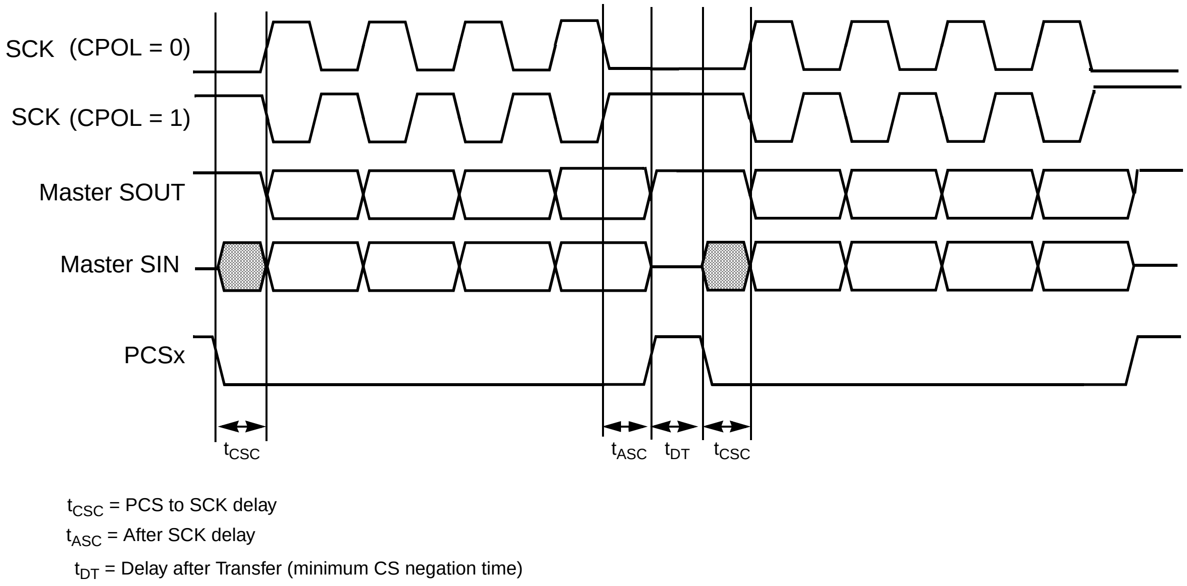
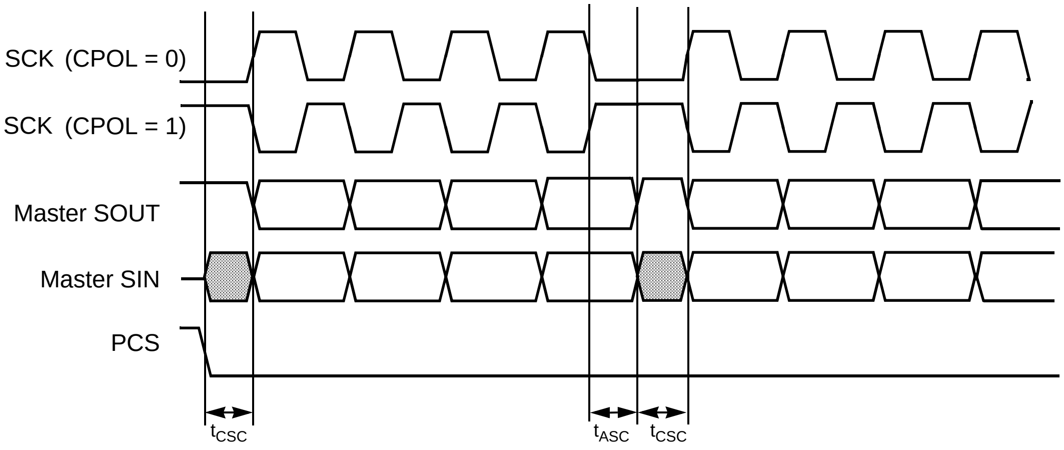


Figure 15-45. Example of non-continuous format (CPHA = 1, CONT = 0)

When the CONT bit = 1, the PCS signal remains asserted for the duration of the two transfers. The Delay between Transfers (t_{DT}) is not inserted between the transfers. Figure 15-46 shows the timing diagram for two 4-bit transfers with CPHA = 1 and CONT = 1.



t_{CSC} = PCS to SCK delay

t_{ASC} = After SCK delay

Figure 15-46. Example of continuous transfer (CPHA = 1, CONT = 1)

When using DSPI with continuous selection follow these rules:

- all transmit commands must have the same PCSn bits programming
- the DSPI_CTARs, selected by transmit commands, must be programmed with the same transfer attributes. Only field FMSZ can be programmed differently in these DSPI_CTARs.

WARNING

During continuous selection mode, to avoid generation of erroneous data, do not change the DSPI*x*_CTAR values between frames in the following conditions:

- If DSPI*x*_CTARn[CPHA=1] AND DSPI*x*_MCR[CONT_SCKE]=0, do not change DSPI*x*_CTARn[CPOL, CPHA, PCSSCK or PBR] between frames.
- If DSPI*x*_CTARn[CPHA]=0 OR DSPI*x*_MCR[CONT_SCKE]=1, do not change any bit field of DSPI*x*_CTARn, except [PBR], between frames.

NOTE

It is mandatory to fill the TXFIFO with the number of entries that will be concatenated under one PCS assertion for both master and slave before the TXFIFO becomes empty. For example, while transmitting in master mode, it should be ensured that the last entry in the TXFIFO, after which TXFIFO becomes empty, must have the CONT bit in command frame as deasserted (i.e. CONT bit = 0).While operating in slave mode, it should be ensured that when the last-entry in the TXFIFO is completely transmitted (that is, the corresponding TCF flag is asserted and TXFIFO is empty) the slave should be deselected for any further serial communication; else an underflow error occurs.

15.9.7 Continuous serial communications clock

The DSPI provides the option of generating a continuous SCK signal for slave peripherals that require a continuous clock.

Continuous SCK is enabled by setting bit DSPI_MCR[CONT_SCKE]. Continuous SCK is valid in all configurations.

Continuous SCK is only supported for CPHA = 1. Clearing CPHA is ignored if bit DSPI_MCR[CONT_SCKE] is set. Continuous SCK is supported for Modified Transfer Format.

Clock and transfer attributes for the Continuous SCK mode are set according to the following rules:

- The TX FIFO must be cleared before initiating any SPI configuration transfer.
- When the DSPI is in SPI configuration, CTAR0 shall be used initially. At the start of each SPI frame transfer, the CTAR specified by the CTAS for the frame should be CTAR0.
- When the DSPI is in DSI configuration, the CTAR specified by the DSICTAS field shall be used at all times.
- When the DSPI is in CSI configuration, the CTAR selected by the DSICTAS field shall be used initially. At the start of an SPI frame transfer, the CTAR specified by the CTAS value (which is CTAR0) for the frame shall be used. At the start of a DSI frame transfer, the CTAR specified by the DSICTAS field shall be used.
- In all configurations, the currently selected DSPI_CTAR remains in use until the start of a frame with a different DSPI_CTAR specified, or the Continuous SCK mode is terminated.

It is recommended to keep the baud rate the same while using the Continuous SCK. Switching clock polarity between frames while using Continuous SCK can cause errors in the transfer. Continuous SCK operation is not guaranteed if the DSPI is put into the External Stop mode or Module Disable mode.

Enabling Continuous SCK disables the PCS to SCK delay and the Delay after Transfer (t_{DT}) is fixed to one SCK cycle. When TSB configuration is enabled the t_{DT} is programmable from 1 to 65 SCK cycles. Figure 15-47 shows timing diagram for Continuous SCK format with Continuous Selection disabled.

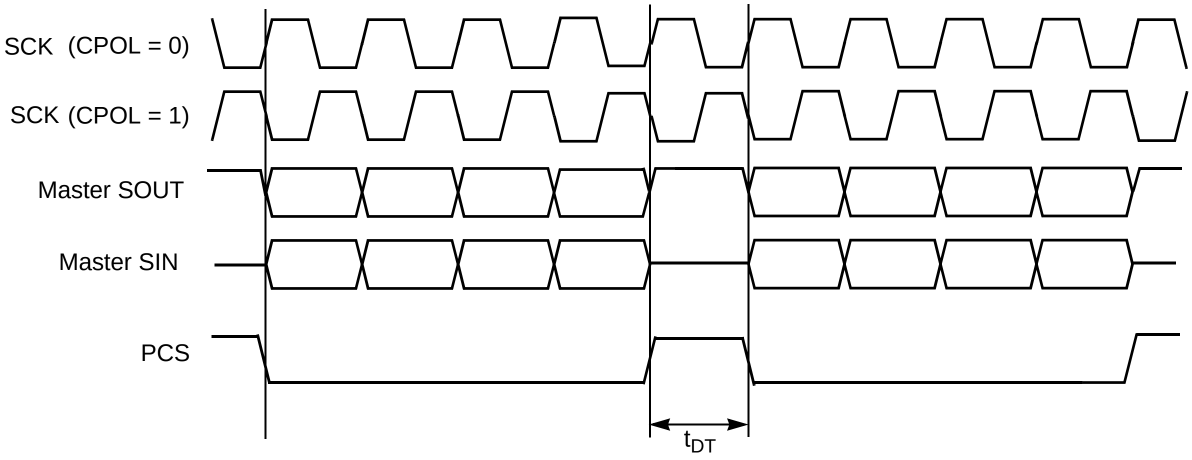


Figure 15-47. Continuous SCK timing diagram (CONT = 0)

If the CONT bit in the TX FIFO entry is set or the DCONT in the DSPI_DSICR is set, PCS remains asserted between the transfers. Under certain conditions, SCK can continue with PCS asserted, but with no data being shifted out of SOUT (SOUT pulled high). This can cause the slave to receive incorrect data. Those conditions include:

- Continuous SCK with CONT bit set, but no data in the transmit FIFO.
- Continuous SCK with CONT bit set and entering STOPPED state (refer to Section 15.9.1, “Start and stop of DSPI transfers”).
- Continuous SCK with CONT bit set and entering Stop mode or Module Disable mode.

Figure 15-48 shows timing diagram for Continuous SCK format with Continuous Selection enabled.

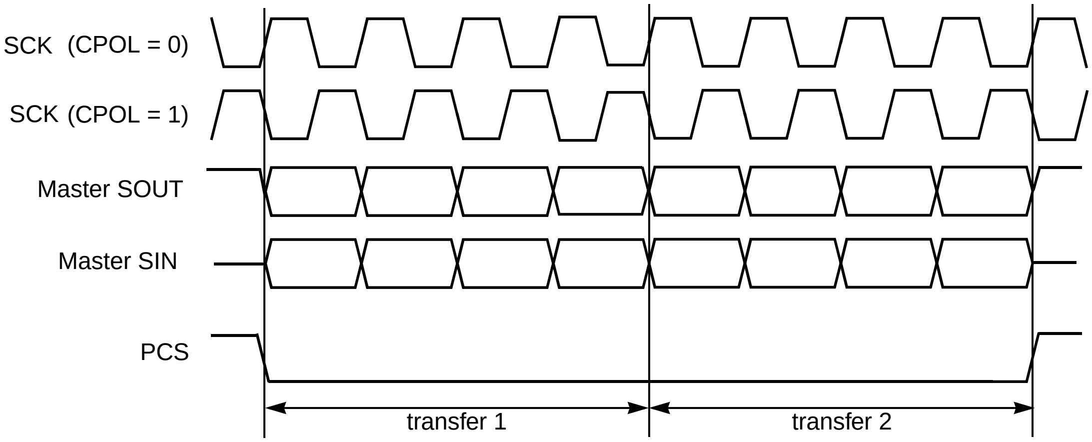


Figure 15-48. Continuous SCK timing diagram (CONT = 1)

15.9.8 Timed serial bus (TSB)

The DSPI can be programmed in Timed Serial Bus configuration by setting the TSBC bit in the DSPI_DSICR. See Section 15.8.2.10, “DSPI DSI Configuration Register (DSPI_DSICR)” for details.

TSB configuration provides the Micro Second Channel (MSC) downstream channel support.

The MSC upstream channel is not supported by the DSPI, but can be supported by any available Serial Communication Controller (SCI or UART) in the device.

To work in TSB mode the DSPI must be in master mode and in DSI (DCONF = 0b01) or CSI (DCONF = 0b10) configuration. Both Continuous and Non Continuous Serial Communication Clock (controlled by bit DSPI_MCR[CONT_SCKE]) are supported in the TSB mode.

Figure 15-49 shows the signals used in the TSB interface.

In the TSB configuration the DSPI is able to send from 4 to 34 bits MSC data frames (4 to 32 serialized data bits and up to 2 Data Selection zero bits). The serialized data bits source can be either:

- the DSPI DSI Alternate Serialization Data Register (DSPI_ASDR), written by the host software,
- Parallel Input pin states latched into the DSPI DSI Serialization Data Register (DSPI_SDR).

DSPI_DSICR TXSS bit or DSPI_SSR bits define the source of the data.

The Least Significant Bits of the DSPI_ASDR or DSPI_SDR registers are selected to be serialized if the data frame is set to less than 32 bits.

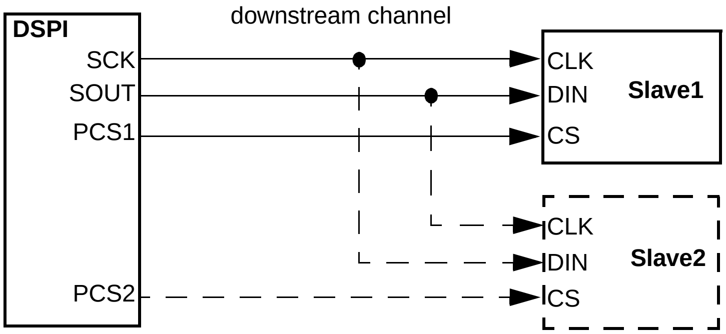


Figure 15-49. DSPI usage in the TSB configuration

The PCS signals are driven together with SOUT. The t_{CSC} and t_{ASC} delays are not available. Delay after Transfer (DT) is set in SCK clock periods as a binary number formed by concatenation of the DSPI_CTARn PDT and DT fields plus one, allowing to set DT from 1 to 64 serial clock periods. DT field provides least significant bits and PDT field provides most significant bits of the Delay after Transfer.

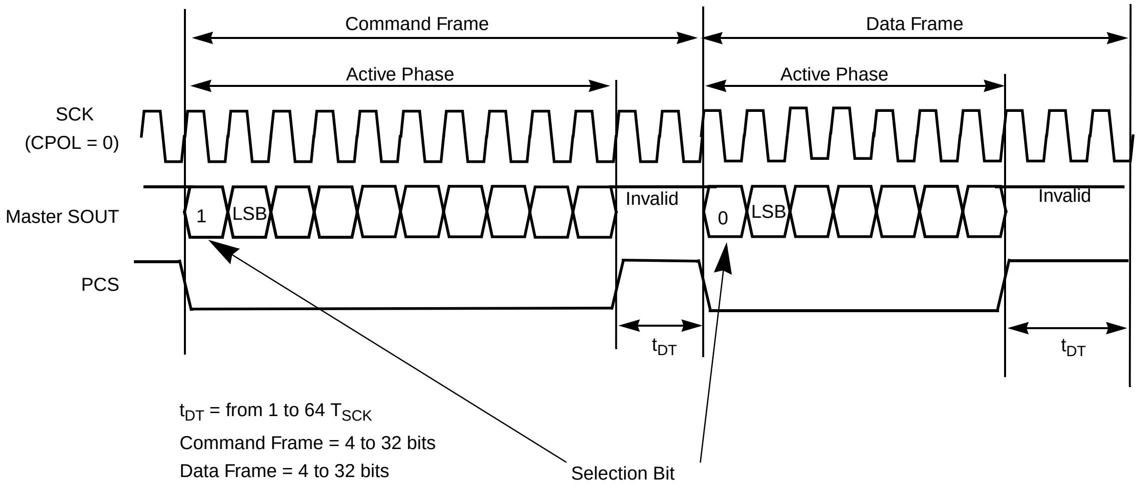


Figure 15-50. TSB Downstream frames

Figure 15-50 shows the two types of MSC downstream frames: command frame and data frame.

The first transmitted bit, called the selection bit, determines the frame type:

- The selection bit “0” indicates a data frame
- The selection bit “1” indicates a command frame

Data frame may contain up to two selection bits to support two external slave devices, (so called dual receiver configuration) or no selection bits at all.

The command frame can be written by software, through SPI TX FIFO, using one or two FIFO entries with help of the CONT bit. The data frame consists of up to 32 bits from the DSPI_SDR or DSPI_ASDR registers and up to two zero selection bits. The number of data bits in the data frame is defined by field DSPI_DSCICR1[TSBCNT].

The selection bit of the MSC command frames (1) can be implemented by software.

To comply with MSC specification, set DSPI_CTARn[LSBFE] to transmit the least significant bit first.

Regardless of the LSBFE bit setting, the Data Frame Selection Bits, if enabled, are always transmitted first, before the corresponding data subframes.

15.9.8.1 MSC dual receiver support with PCS switchover

When in TSB mode it is possible to switch the set of PCS signals that are driven during the first part of the frame to a different set of PCS signals during the second part of the frame. The bit, at which this switchover occurs, is defined by field FMSZ of the DSPI_CTARn register, which is selected by field DSPI_DSICR[DSICTAS].

Number of the bits, not including the Data Selection Bit, in the first part of the frame is equal to value of the FMSZ field plus one. During this part of the frame the PCS signal levels are controlled by DSPI_DSICR DPCSn bits, after that by DSPI_DSICR1 DPCS1_n bits.

The PCS switchover occurs at driving edge of the SCK clock output.

The second Data Selection Bit is inserted after the PCS switchover if enabled.

Data Frame with PCS switchover is shown in Figure 15-51.

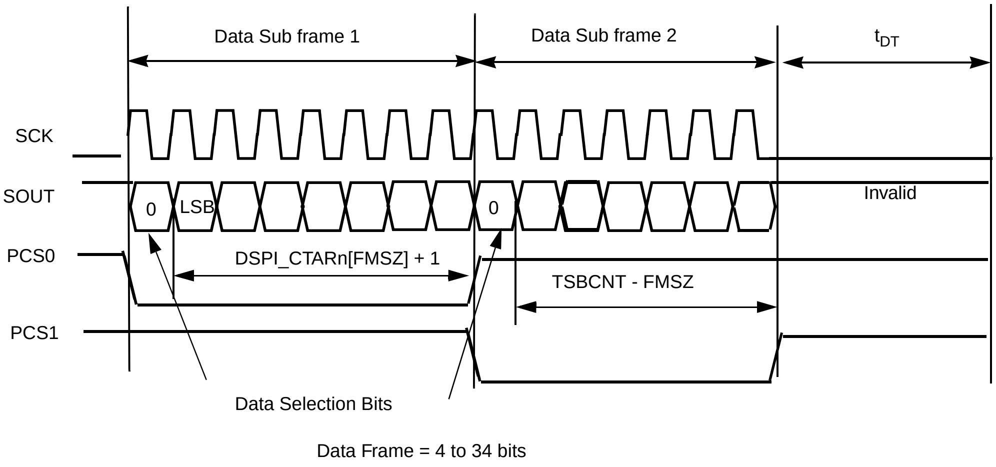


Figure 15-51. TSB data frame format for MSC dual receiver operation

15.9.9 Parity generation and check

The DSPI module can generate and check parity in the serial frame. The parity bit replaces the last transmitted bit in the frame. The parity is calculated for all transmitted data bits in frame, not including the last, would be transmitted, data bit. The parity generation/control is done on frame basis. The registers fields, setting frame size defines the total number of bits in the frame, including the parity bit. Thus, to transmit/receive the same number of data bits with parity check, increase the frame size by one versus the same data size frame without the parity check.

Parity can be selected as odd or even. Parity Errors in the received frame set Parity Error flags in the Status register. The Parity Error Interrupt Requests are generated if enabled. The DSPI module can be programmed to stop SPI or/and DSI frame transmission in case of a frame reception with parity error.

15.9.9.1 Parity for SPI frames

When the DSPI is in the master mode the parity generation is controlled by PE and PP bits of the TX FIFO entries (DSPI_PUSHR). Setting the PE bit enables parity generation for transmitted SPI frames and parity check for received frames. PP bit defines polarity of the parity bit.

When continuous PCS selection is used to transmit SPI data, two parity generation scenarios are available:

- Generate/check parity for the whole frame
- Generate/check parity for each subframe separately.

To generate/check parity for the whole frame set PE bit only in the last command/TX FIFO entry, forming this frame (with the DSPI_PUSHR).

To generate/check parity for each subframe set PE bit in each command/TX FIFO entry, forming this frame.

If the parity error occurs for received SPI frame, the DSPI_SR[SPEF] bit is set. If DSPI_MCR[PES] bit is set, the DSPI stops SPI frames transmission. To resume SPI operation clear the DSPI_SR[SPEF] or the DSPI_MCR[PES] bits.

In slave mode the parity is controlled by the PE and PP bits of the DSPI_CTAR0 register similar to the master mode parity generation without continuous PCS selection.

15.9.9.2 Parity for DSI frames

Parity generation is controlled by PE and PP bits of the DSPI_DSICR similar to the SPI frames. The parity is calculated and checked for each DSI frame. (DSPI_DSICR[DCONT] bit has no effect on parity generation.)

If the parity error occurs for received DSI frame, the DSPI_SR[DPEF] bit is set. To resume DSI operation clear the DSPI_SR[DPEF] bit.

15.9.10 Interrupts/DMA requests

The DSPI has several conditions that can generate interrupt requests and two conditions that can generate both interrupt or DMA requests. Table 15-39 lists these conditions. The ‘x’ in the request type columns indicates which signals are connected on the MPC5676R.

Table 15-39. Interrupt and DMA request conditions

Condition	Flag	Request type	
		Interrupt	DMA
End of Queue (EOQ)	EOQF	x	
TX FIFO Fill	TFFF	x	x
Transfer Complete	TCF	x	
TX FIFO Underflow	TFUF	x	
RX FIFO Drain	RFDF	x	x
RX FIFO Overflow	RFOF	x	
SPI Parity Error	SPEF	x	
DSI Parity Error	DPEF	x	
DSI Deserialized Data Match	DDIF	x	

Each condition has a flag bit in the DSPI Status Register (DSPI_SR) and an Request Enable bit in the DSPI DMA/Interrupt Request Select and Enable Register (DSPI_RSER). The TX FIFO Fill Flag (TFFF) and RX FIFO Drain Flag (RFDF) generate interrupt requests or DMA requests depending on the TFFFDIRS and RFDFDIRS bits in the DSPI_RSER.

The DSPI module also provides a global interrupt request line, which is asserted when any of individual interrupt requests lines is asserted.

15.9.10.1 End of queue interrupt request

The End of Queue Request indicates that the end of a transmit queue is reached. The End of Queue Request is generated when the EOQ bit in the executing SPI command is set and bit DSPI_RSER[EOQFRE] is set.

15.9.10.2 Transmit FIFO fill interrupt or DMA request

The Transmit FIFO Fill Request indicates that the TX FIFO is not full. The Transmit FIFO Fill Request is generated when the number of entries in the TX FIFO is less than the maximum number of possible entries, and the TFFFRE bit in the DSPI_RSER is set. The TFFFDIRS bit in the DSPI_RSER selects whether a DMA request or an interrupt request is generated.

15.9.10.3 Transfer complete interrupt request

The Transfer Complete Request indicates the end of the transfer of a serial frame. The Transfer Complete Request is generated at the end of each frame transfer when the TCF_RE bit is set in the DSPI_RSER.

15.9.10.4 Transmit FIFO underflow interrupt request

The Transmit FIFO Underflow Request indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for the DSPI, operating in slave mode and SPI configuration. The TFUF bit is set when the TX FIFO of a DSPI is empty, and a transfer is initiated from an external SPI master. If the TFUF bit is set while the TFUFRE bit in the DSPI_RSER is set, an interrupt request is generated.

15.9.10.5 Receive FIFO drain interrupt or DMA request

The Receive FIFO Drain Request indicates that the RX FIFO is not empty. The Receive FIFO Drain Request is generated when the number of entries in the RX FIFO is not zero, and the RFDFRE bit in the DSPI_RSER is set. The RFDFDIRS bit in the DSPI_RSER selects whether a DMA request or an interrupt request is generated.

15.9.10.6 Receive FIFO overflow interrupt request

The Receive FIFO Overflow Request indicates that an overflow condition in the RX FIFO has occurred. A Receive FIFO Overflow request is generated when RX FIFO and shift register are full and a transfer is initiated. The RFOFRE bit in the DSPI_RSER must be set for the interrupt request to be generated.

Depending on the state of the ROOE bit in the DSPI_MCR, the data from the transfer that generated the overflow is either ignored or shifted in to the shift register. If the ROOE bit is set, the incoming data is shifted in to the shift register. If the ROOE bit is cleared, the incoming data is ignored.

15.9.10.7 SPI frame parity error interrupt request

The SPI Frame Parity Error Flag indicates that a SPI frame with parity error had been received. The SPEFRE bit in the DSPI_RSER must be set for the interrupt request to be generated.

15.9.10.8 DSI frame parity error interrupt request

The DSI Frame Parity Error Flag indicates that a DSI frame with parity error has been received. The DPEFRE bit in the DSPI_RSER must be set for the interrupt request to be generated.

15.9.10.9 Deserialized data match interrupt request

The Deserialized Data Match Flag (DDIF) indicates that a DSI frame with data matches DSPI_DPIR data, masked with DSPI_DIMR, had been received. The DDIFRE bit in the DSPI_RSER must be set for the interrupt request to be generated.

15.9.11 Buffered SPI operation

The DSPI can use a FIFO buffering mechanism to transmit and receive commands and data to and from external devices. The Transmit FIFO buffers SPI commands and data to be transferred. The Receive FIFO buffers incoming serial data. Both FIFOs are four entries deep. The TX FIFO stores 32-bit words when the DSPIs are configured for Master Mode. The 32-bit words are composed of 16-bit command fields and data fields up to 16 bits wide. The RX FIFOs store 16-bit words of received data from external devices. When the DSPI is configured for Slave Mode, the DSPI ignores the SPI command in the TX FIFO.

For queued operations, the SPI queues reside in system memory external to the DSPI. Data transfers between the memory and the DSPI FIFOs are accomplished through the use of the eDMA controller or through Host software. See Figure 15-52 for a conceptual diagram of the queue data transfer control in the MPC5676R MCU.

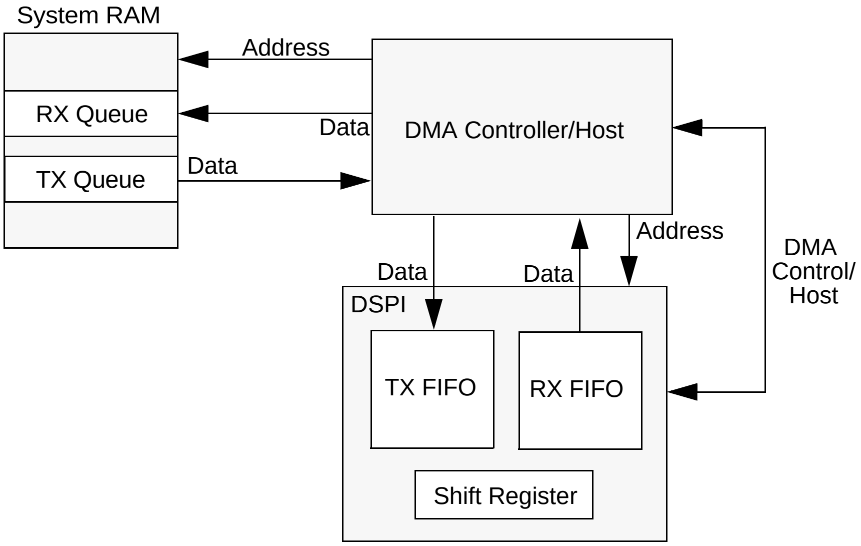


Figure 15-52. DSPI queue transfer control in the MPC5676R

15.9.12 Continuous peripheral chip select

For peripherals that must remain selected between sequential serial transfers, the DSPI provides the option of having the PCS signals asserted between transfers. For SPI transfers, the CONT bit in the SPI command fields selects the continuous PCS feature. For DSI and CSI transfers, the DCONT bit in the DSPI_DSICR selects the continuous PCS feature.

15.9.13 Peripheral chip select expansion and deglitching

The DSPI supports up to 256 Peripheral Chip Select Signals with the use of an external demultiplexer. Up to 128 Peripheral Chip Select Signals can be used if deglitching is desired. The $\overline{\text{PCSS}}$ signal provides the appropriate timing to enable and disable the demultiplexer for the DSPI_x_PCS[0:7] signals.

Figure 15-53 shows how an external 8-to-256 demultiplexer (on-board decoder) can be connected to the DSPI.

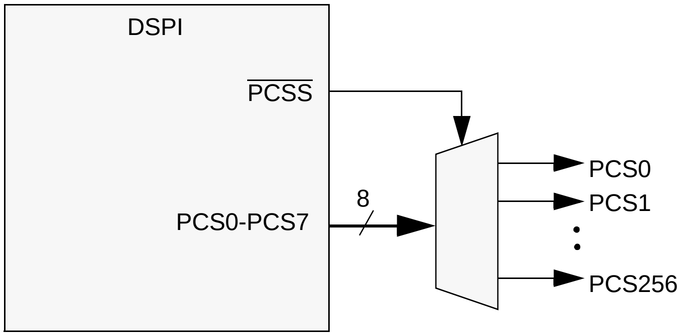


Figure 15-53. DSPI PCS expansion and deglitching

15.9.14 DMA and interrupt conditions

The DSPI has six conditions that can generate interrupt requests and two conditions that can generate both interrupt or DMA requests. Table 15-40 lists the conditions. The ‘x’ in the request type columns indicates which signals are connected on the MPC5676R.

Table 15-40. DSPI interrupt and DMA request conditions

Condition	Flag	Request type	
		Interrupt	DMA
End of queue reached	EOQF	x	
TX FIFO is not full	TFFF	x	x
Transfer of current frame complete	TCF	x	
Attempt to transmit with an empty Transmit FIFO	TFUF	x	
RX FIFO is not empty	RFDF	x	x
Frame received while Receive FIFO is full	RFOF	x	

All request conditions are detected in the SPI configuration and in the CSI configuration. In DSI configuration only the transfer of current frame complete condition is detected.

15.9.14.1 Transmit FIFO underflow flag (TFUF)

The Transmit FIFO Underflow Flag indicates that an underflow condition in the TX FIFO has occurred. The transmit underflow condition is detected only for DSPI modules operating in slave mode and SPI configuration. The transmit underflow condition is detected when the TX FIFO of a DSPI operating as a SPI slave is empty, and a transfer is initiated from an external SPI master.

15.9.14.2 Receive FIFO overflow flag (RFOF)

The Receive FIFO Overflow Flag indicates that an overflow condition in the RX FIFO has occurred, and that data may be lost. The Receive FIFO Overflow Flag is asserted when the RX FIFO is full, a new frame has been received in the shift register, and a transfer is initiated.

15.9.15 Modified SPI transfer format

In Modified Transfer Format, the slave peripheral has more time to place data on the SOUT pin before the DSPI samples the data. In the Modified Transfer Format, the Master samples the incoming data towards the end of the transfer cycle. For correct operation of the Modified Transfer Format, the user must thoroughly analyze the SPI link timing budget.

15.9.16 LVDS pad usage

The differential transmitter pad driver LVDS support data rate up to 40 MHz. Figure 15-54 describes the pad signals interface.

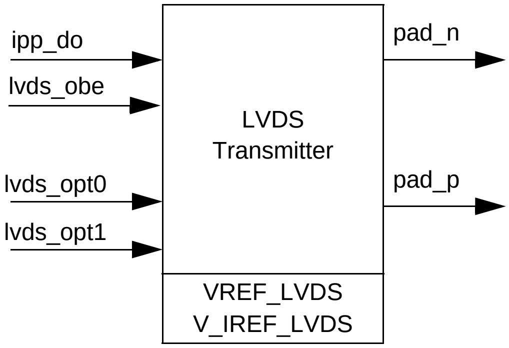


Figure 15-54. LVDS transmitter pad block diagram

Signals lvds_opt0 and lvds_opt1 control the voltage swing on the LVDS pad. These two signals are controlled by bits SRC[1:0] of the respective PCR. Table 15-41 gives the configuration for these bits.

Table 15-41. LVDS pads voltage swing

SRC[0]	SRC[1]	Current flowing in the driver	Differential voltage across pad_p and pad_n
0	0	normal	default
0	1	increased	increased
1	0	decreased	decreased
1	1	normal	same as default

15.9.17 DSPI connections to eTPU_A, eMIOS and SIU

The three DSPI modules connect to the input and output channels of the eTPUs and the eMIOS. Some of the DSPI outputs connect to the External Interrupt Input Multiplexing sub-block in the SIU. See Section 3.4.1, "External Interrupts" and Section 3.2.1.16, "External IRQ Input Select Register (SIU_EIISR)" for details on how the DSPI deserialized outputs can be used to trigger external interrupt requests.

15.9.17.1 DSPI_B connectivity

The DSPI_B connects to the eMIOS, eTPU_A and SIU as shown in Figure 15-55.

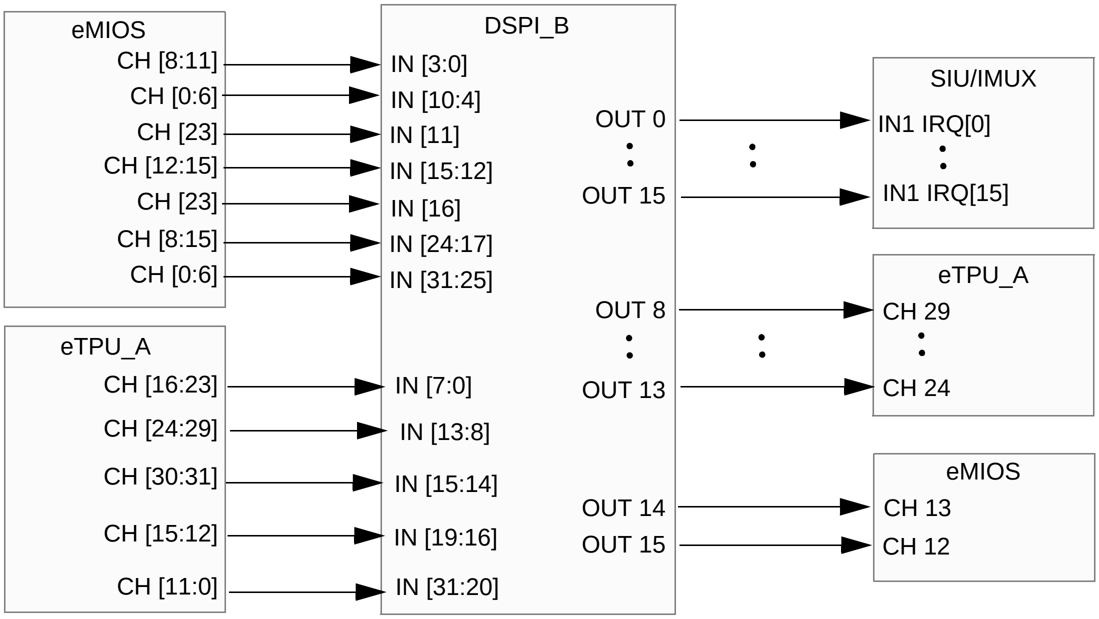


Figure 15-55. DSPI_B connectivity

Table 15-42 lists the DSPI_B connections.

Table 15-42. DSPI_B connectivity table

DSPI_B input	Connected to:	DSPI_B output	Connected to:
0	eMIOS Output Channel 11 eTPU_A Output Channel 23 GPDO350	0	Input 1 on IMUX for External IRQ[0]
1	eMIOS Output Channel 10 eTPU_A Output Channel 22 GPDO351	1	Input 1 on IMUX for External IRQ[1]
2	eMIOS Output Channel 9 eTPU_A Output Channel 21 GPDO352	2	Input 1 on IMUX for External IRQ[2]

Table 15-42. DSPI_B connectivity table (continued)

DSPI_B input	Connected to:
3	eMIOS Output Channel 8 eTPU_A Output Channel 20 GPDO353
4	eMIOS Output Channel 6 eTPU_A Output Channel 19 GPDO354
5	eMIOS Output Channel 5 eTPU_A Output Channel 18 GPDO355
6	eMIOS Output Channel 4 eTPU_A Output Channel 17 GPDO356
7	eMIOS Output Channel 3 eTPU_A Output Channel 16 GPDO357
8	eMIOS Output Channel 2 eTPU_A Output Channel 29 GPDO358
9	eMIOS Output Channel 1 eTPU_A Output Channel 28 GPDO359
10	eMIOS Output Channel 0 eTPU_A Output Channel 27 GPDO360
11	eMIOS Output Channel 23 eTPU_A Output Channel 26 GPDO361
12	eMIOS Output Channel 15 eTPU_A Output Channel 25 GPDO362
13	eMIOS Output Channel 14 eTPU_A Output Channel 24 GPDO363
14	eMIOS Output Channel 13 eTPU_A Output Channel 31 GPDO364
15	eMIOS Output Channel 12 eTPU_A Output Channel 30 GPDO365
16	eMIOS Output Channel 23 eTPU_A Output Channel 12 GPDO366

DSPI_B output	Connected to:
3	Input 1 on IMUX for External IRQ[3]
4	Input 1 on IMUX for External IRQ[4]
5	Input 1 on IMUX for External IRQ[5]
6	Input 1 on IMUX for External IRQ[6]
7	Input 1 on IMUX for External IRQ[7]
8	eTPU_A Input Channel 29, Input 1 on IMUX for External IRQ[8]
9	eTPU_A Input Channel 28, Input 1 on IMUX for External IRQ[9]
10	eTPU_A Input Channel 27, Input 1 on IMUX for External IRQ[10]
11	eTPU_A Input Channel 26, Input 1 on IMUX for External IRQ[11]
12	eTPU_A Input Channel 25, Input 1 on IMUX for External IRQ[12]
13	eTPU_A Input Channel 24, Input 1 on IMUX for External IRQ[13]
14	eMIOS Input Channel 13, Input 1 on IMUX for External IRQ[14]
15	eMIOS Input Channel 12, Input 1 on IMUX for External IRQ[15]
16	NC

Table 15-42. DSPI_B connectivity table (continued)

DSPI_B input	Connected to:
17	eMIOS Output Channel 15 eTPU_A Output Channel 13 GPDO367
18	eMIOS Output Channel 14 eTPU_A Output Channel 14 GPDO368
19	eMIOS Output Channel 13 eTPU_A Output Channel 15 GPDO369
20	eMIOS Output Channel 12 eTPU_A Output Channel 0 GPDO370
21	eMIOS Output Channel 11 eTPU_A Output Channel 1 GPDO371
22	eMIOS Output Channel 10 eTPU_A Output Channel 2 GPDO372
23	eMIOS Output Channel 9 eTPU_A Output Channel 3 GPDO373
24	eMIOS Output Channel 8 eTPU_A Output Channel 4 GPDO374
25	eMIOS Output Channel 6 eTPU_A Output Channel 5 GPDO375
26	eMIOS Output Channel 5 eTPU_A Output Channel 6 GPDO376
27	eMIOS Output Channel 4 eTPU_A Output Channel 7 GPDO377
28	eMIOS Output Channel 3 eTPU_A Output Channel 8 GPDO378
29	eMIOS Output Channel 2 eTPU_A Output Channel 9 GPDO379
30	eMIOS Output Channel 1 eTPU_A Output Channel 10 GPDO380

DSPI_B output	Connected to:
17	NC
18	NC
19	NC
20	NC
21	NC
22	NC
23	NC
24	NC
25	NC
26	NC
27	NC
28	NC
29	NC
30	NC

Table 15-42. DSPI_B connectivity table (continued)

DSPI_B input	Connected to:
31	eMIOS Output Channel 0 eTPU_A Output Channel 11 GPDO381

DSPI_B output	Connected to:
31	NC

15.9.17.2 DSPI_C connectivity

The DSPI_C connects to eTPU_A and SIU as shown in Figure 15-57.

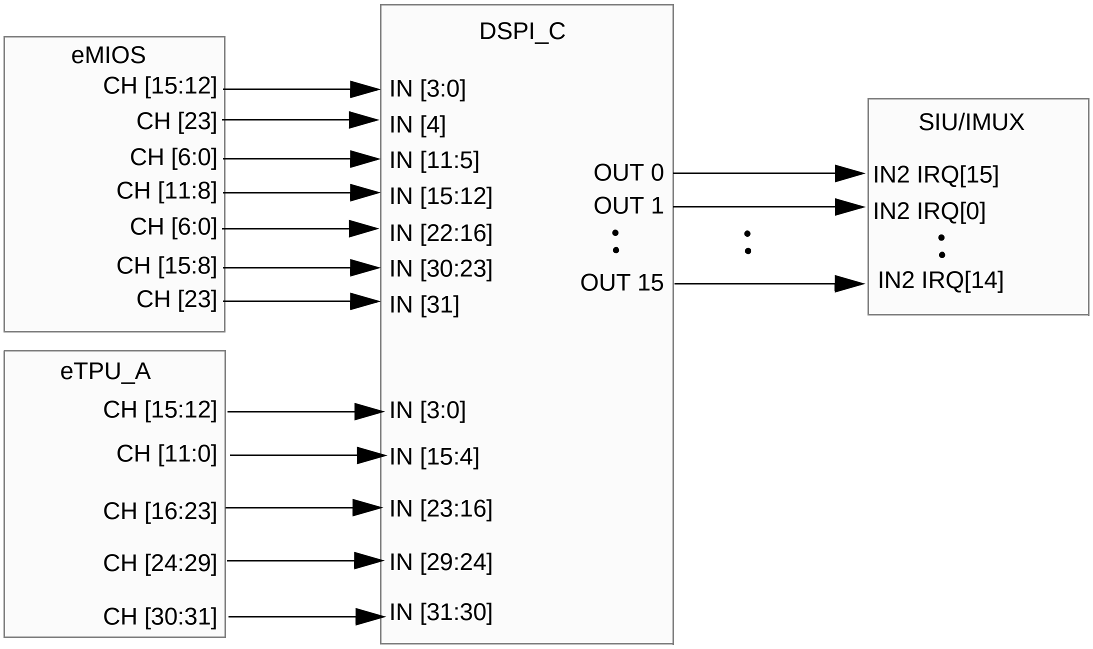


Figure 15-56. DSPI_C connectivity

Table 15-44 lists the DSPI_C connections.

Table 15-43. DSPI_C connectivity table

DSPI_C input	Connected to:
0	eMIOS Output Channel 7 eMIOS Output Channel 12 eTPU_A Output Channel 12 GPDO382
1	eMIOS Output Channel 16 eMIOS Output Channel 13 eTPU_A Output Channel 13 GPDO383

DSPI_C output	Connected to:
0	Input 2 on IMUX for External IRQ[15]
1	Input 2 on IMUX for External IRQ[0]

Table 15-43. DSPI_C connectivity table (continued)

DSPI_C input	Connected to:
2	eMIOS Output Channel 17 eMIOS Output Channel 14 eTPU_A Output Channel 14 GPDO384
3	eMIOS Output Channel 18 eMIOS Output Channel 15 eTPU_A Output Channel 15 GPDO385
4	eMIOS Output Channel 19 eMIOS Output Channel 23 eTPU_A Output Channel 0 GPDO386
5	eMIOS Output Channel 20 eMIOS Output Channel 0 eTPU_A Output Channel 1 GPDO387
6	eMIOS Output Channel 21 eMIOS Output Channel 1 eTPU_A Output Channel 2 GPDO388
7	eMIOS Output Channel 22 eMIOS Output Channel 2 eTPU_A Output Channel 3 GPDO389
8	eMIOS Output Channel 3 eTPU_A Output Channel 4 GPDO390
9	eMIOS Output Channel 4 eTPU_A Output Channel 5 GPDO391
10	eMIOS Output Channel 5 eTPU_A Output Channel 6 GPDO392
11	eMIOS Output Channel 6 eTPU_A Output Channel 7 GPDO393
12	eMIOS Output Channel 8 eTPU_A Output Channel 8 GPDO394
13	eMIOS Output Channel 9 eTPU_A Output Channel 9 GPDO395
14	eMIOS Output Channel 10 eTPU_A Output Channel 10 GPDO396

DSPI_C output	Connected to:
2	Input 2 on IMUX for External IRQ[1]
3	Input 2 on IMUX for External IRQ[2]
4	Input 2 on IMUX for External IRQ[3]
5	Input 2 on IMUX for External IRQ[4]
6	Input 2 on IMUX for External IRQ[5]
7	Input 2 on IMUX for External IRQ[6]
8	Input 2 on IMUX for External IRQ[7]
9	Input 2 on IMUX for External IRQ[8]
10	Input 2 on IMUX for External IRQ[9]
11	Input 2 on IMUX for External IRQ[10]
12	Input 2 on IMUX for External IRQ[11]
13	Input 2 on IMUX for External IRQ[12]
14	Input 2 on IMUX for External IRQ[13]

Table 15-43. DSPI_C connectivity table (continued)

DSPI_C input	Connected to:	DSPI_C output	Connected to:
15	eMIOS Output Channel 11 eTPU_A Output Channel 11 GPDO397	15	Input 2 on IMUX for External IRQ[14]
16	eMIOS Output Channel 0 eTPU_A Output Channel 23 GPDO398	16	NC
17	eMIOS Output Channel 1 eTPU_A Output Channel 22 GPDO399	17	NC
18	eMIOS Output Channel 2 eTPU_A Output Channel 21 GPDO400	18	NC
19	eMIOS Output Channel 3 eTPU_A Output Channel 20 GPDO401	19	NC
20	eMIOS Output Channel 4 eTPU_A Output Channel19 GPDO402	20	NC
21	eMIOS Output Channel 5 eTPU_A Output Channel 18 GPDO403	21	NC
22	eMIOS Output Channel 6 eTPU_A Output Channel 17 GPDO404	22	NC
23	eMIOS Output Channel 8 eTPU_A Output Channel 16 GPDO405	23	NC
24	eMIOS Output Channel 9 eTPU_A Output Channel 29 GPDO406	24	NC
25	eMIOS Output Channel 10 eTPU_A Output Channel 28 GPDO407	25	NC
26	eMIOS Output Channel 11 eTPU_A Output Channel 27 GPDO408	26	NC
27	eMIOS Output Channel 12 eTPU_A Output Channel 26 GPDO409	27	NC
28	eMIOS Output Channel 13 eTPU_A Output Channel 25 GPDO410	28	NC

Table 15-43. DSPI_C connectivity table (continued)

DSPI_C input	Connected to:	DSPI_C output	Connected to:
29	eMIOS Output Channel 14 eTPU_A Output Channel 24 GPDO411	29	NC
30	eMIOS Output Channel 15 eTPU_A Output Channel 31 GPDO412	30	NC
31	eMIOS Output Channel 23 eTPU_A Output Channel 30 GPDO413	31	NC

15.9.17.3 DSPI_D connectivity

The DSPI_D connects to SIU as shown in Figure 15-57.

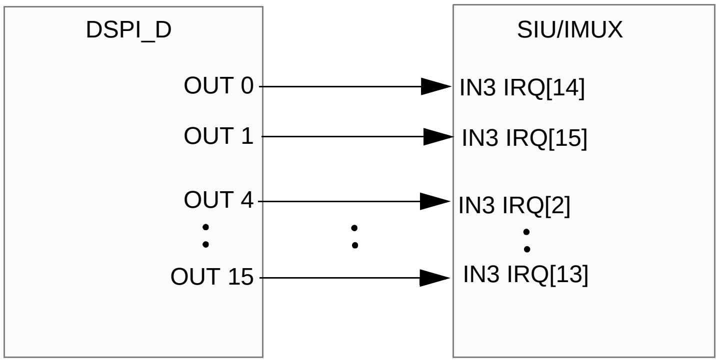


Figure 15-57. DSPI_D connectivity

Table 15-44 lists the DSPI_D connections.

Table 15-44. DSPI_D connectivity table

DSPI_D input	Connected to:	DSPI_D output	Connected to:
0	NC	0	Input 3 on IMUX for External IRQ[14]
1	NC	1	Input 3 on IMUX for External IRQ[15]
2	NC	2	NC
3	NC	3	NC
4	NC	4	Input 3 on IMUX for External IRQ[2]
5	NC	5	Input 3 on IMUX for External IRQ[3]
6	NC	6	Input 3 on IMUX for External IRQ[4]

Table 15-44. DSPI_D connectivity table

DSPI_D input	Connected to:
7	NC
8	NC
9	NC
10	NC
11	NC
12	NC
13	NC
14	NC
15	NC
16–31	NC

DSPI_D output	Connected to:
7	Input 3 on IMUX for External IRQ[5]
8	Input 3 on IMUX for External IRQ[6]
9	Input 3 on IMUX for External IRQ[7]
10	Input 3 on IMUX for External IRQ[8]
11	Input 3 on IMUX for External IRQ[9]
12	Input 3 on IMUX for External IRQ[10]
13	Input 3 on IMUX for External IRQ[11]
14	Input 3 on IMUX for External IRQ[12]
15	Input 3 on IMUX for External IRQ[13]
16–31	NC

15.9.18 Power saving features

The DSPI supports two power-saving strategies:

- External Stop mode
- Module Disable mode—Clock gating of non-memory mapped logic

15.9.18.1 Stop mode (External Stop mode)

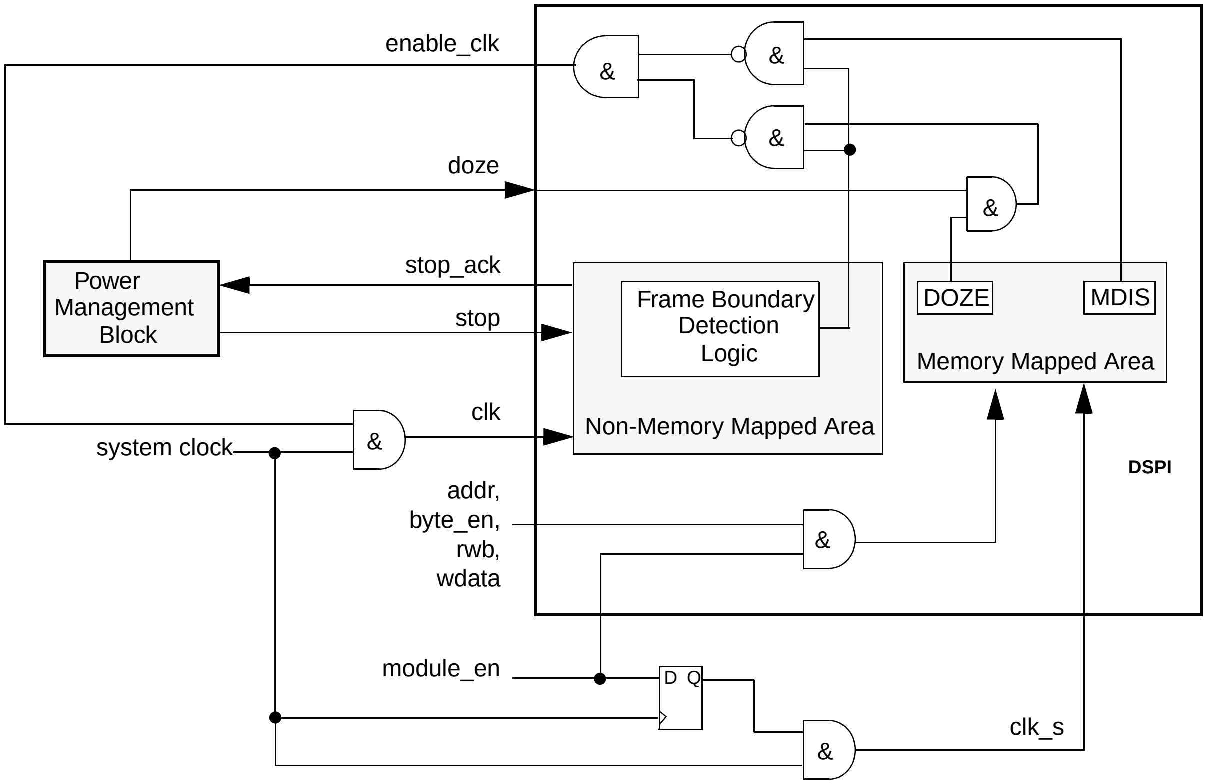


Figure 15-58.

The DSPI supports the stop mode protocol. When a request is made to enter external stop mode, the DSPI module acknowledges the request. If a serial transfer is in progress, the DSPI waits until it reaches the frame boundary before it is ready to have its clocks shut off. While the clocks are shut off, the DSPI memory-mapped logic is not accessible. The states of the interrupt and DMA request signals cannot be changed while in External Stop mode.

15.9.18.2 Module Disable mode

Module disable mode is a module-specific mode that the DSPI can enter to save power. Host CPU can initiate the module disable mode by setting bit DSPI_MCR[MDIS]. The module disable mode can also be initiated by hardware. A power management block can initiate the module disable mode by asserting the Doze mode signal while bit DSPI_MCR[DOZE] is set.

When the MDIS bit is set or the Doze mode signal is asserted while the DOZE bit is set, the DSPI negates Clock Enable signal at the next frame boundary. If implemented, the Clock Enable signal can stop the clock to the non-memory mapped logic. When Clock Enable is negated, the DSPI is in a dormant state, but the memory mapped registers are still accessible. Certain read or write operations have a different effect when the DSPI is in the module disable mode. Reading the RX FIFO Pop Register does not change the state of the RX FIFO. Likewise, writing to the TX FIFO Push Register does not change the state of the TX FIFO. Clearing either of the FIFOs has no effect in the module disable mode. Changes to the DIS_TXF and DIS_RXF fields of the DSPI_MCR have no effect in the module disable mode. In the module disable mode, all status bits and register flags in the DSPI return the correct values when read, but writing to them has no effect. Writing to the DSPI_TCR during module disable mode has no effect. Interrupt and DMA request signals cannot be cleared while in the module disable mode.

15.10 Initialization/Application information

15.10.1 How to manage DSPI queues

The queues are not part of the DSPI, but the DSPI includes features in support of queue management. Queues are primarily supported in SPI configuration.

1. When DSPI executes last command word from a queue, the EOQ bit in the command word is set to indicate to the DSPI that this is the last entry in the queue.
2. At the end of the transfer, corresponding to the command word with EOQ set is sampled, the EOQ flag DSPI_SR[EOQF] is set.
3. The setting of the EOQ flag disables serial transmission and reception of data, putting the DSPI in the STOPPED state. The TXRXS bit is cleared to indicate the STOPPED state.
4. The DMA can continue to fill TX FIFO until it is full or step 5 occurs.
5. Disable DSPI DMA transfers by disabling the DMA enable request for the DMA channel assigned to TX FIFO and RX FIFO. This is done by clearing the corresponding DMA enable request bits in the DMA controller.
6. Ensure all received data in RX FIFO has been transferred to memory receive queue by reading DSPI_SR[RXCNT] or by checking DSPI_SR[RFDF] after each read operation of the DSPI_POPR.
7. Modify DMA descriptor of TX and RX channels for new queues
8. Flush TX FIFO by writing a ‘1’ to bit DSPI_MCR[CLR_TXF]. Flush RX FIFO by writing a ‘1’ to bit DSPI_MCR[CLR_RXF].
9. Clear transfer count either by setting CTCNT bit in the command word of the first entry in the new queue or via CPU writing directly to field DSPI_TCR[TCNT].
10. Enable DMA channel by enabling the DMA enable request for the DMA channel assigned to the DSPI TX FIFO, and RX FIFO by setting the corresponding DMA set enable request bit.
11. Enable serial transmission and serial reception of data by clearing the EOQF bit.

15.10.2 Switching Master and Slave mode

When changing modes in the DSPI, follow the steps below to guarantee proper operation.

1. Halt the DSPI by setting DSPI_MCR[HALT].
2. Clear the transmit and receive FIFOs by writing a 1 to the CLR_TXF and CLR_RXF bits in DSPI_MCR.
3. Set the appropriate mode in DSPI_MCR[MSTR] and enable the DSPI by clearing DSPI_MCR[HALT].

15.10.3 Baud rate settings

Table 15-45 shows the baud rate that is generated based on the combination of the baud rate prescaler PBR and the baud rate scaler BR in the DSPI_CTAR registers. The values calculated assume a 100 MHz system frequency and the double baud rate DBR bit is clear.

Table 15-45. Baud rate values (bps)

		Baud rate divider prescaler values			
		2	**3**	**5**	**7**
Baud rate scaler values	2	25.0M	16.7M	10.0M	7.14M
	4	12.5M	8.33M	5.00M	3.57M
	6	8.33M	5.56M	3.33M	2.38M
	8	6.25M	4.17M	2.50M	1.79M
	16	3.12M	2.08M	1.25M	893k
	32	1.56M	1.04M	625k	446k
	64	781k	521k	312k	223k
	128	391k	260k	156k	112k
	256	195k	130k	78.1k	55.8k
	512	97.7k	65.1k	39.1k	27.9k
	1024	48.8k	32.6k	19.5k	14.0k
	2048	24.4k	16.3k	9.77k	6.98k
	4096	12.2k	8.14k	4.88k	3.49k
	8192	6.10k	4.07k	2.44k	1.74k
	16384	3.05k	2.04k	1.22k	872
	32768	1.53k	1.02k	610	436

15.10.4 Delay settings

Table 15-46 shows the values for the Delay after Transfer (t_{DT}) and CS to SCK Delay (T_{CSC}) that can be generated based on the prescaler values and the scaler values set in the DSPI_CTAR registers. The values calculated assume a 100 MHz system frequency.

Table 15-46 does not apply for TSB Continuous mode.

Table 15-46. Delay values

		Delay prescaler values			
		1	**3**	**5**	**7**
Delay scaler values	2	20.0 ns	60.0 ns	100.0 ns	140.0 ns
	4	40.0 ns	120.0 ns	200.0 ns	280.0 ns
	8	80.0 ns	240.0 ns	400.0 ns	560.0 ns
	16	160.0 ns	480.0 ns	800.0 ns	1.1 µs
	32	320.0 ns	960.0 ns	1.6 µs	2.2 µs
	64	640.0 ns	1.9 µs	3.2 µs	4.5 µs
	128	1.3 µs	3.8 µs	6.4 µs	9.0 µs
	256	2.6 µs	7.7 µs	12.8 µs	17.9 µs
	512	5.1 µs	15.4 µs	25.6 µs	35.8 µs
	1024	10.2 µs	30.7 µs	51.2 µs	71.7 µs
	2048	20.5 µs	61.4 µs	102.4 µs	143.4 µs
	4096	41.0 µs	122.9 µs	204.8 µs	286.7 µs
	8192	81.9 µs	245.8 µs	409.6 µs	573.4 µs
	16384	163.8 µs	491.5 µs	819.2 µs	1.1 ms
	32768	327.7 µs	983.0 µs	1.6 ms	2.3 ms
	65536	655.4 µs	2.0 ms	3.3 ms	4.6 ms

15.10.5 DSPI Compatibility with the QSPI of the MPC500 MCUs

Table 15-47 shows the translation of commands written to the TX FIFO command halfword with commands written to the Command Ram of the MPC500 family MCUs QSPI. The table illustrates how to configure the DSPI_CTAR registers to match the default cases for the possible combinations of the MPC500 Family QSPI Control Bits in its Command RAM. The defaults for QSPI are based on a system clock of 40MHz. All delay variables below will generate the same delay, or as close a possible, from the DSPI 100MHz system clock that an QSPI would generate from its 40MHz system clock. For other system clock frequencies, the customer can recompute the values using Section 15.10.4, “Delay settings”.

- For BITSE = 0 → 8 bits per transfer
- For DT = 0 → 0.425µs delay: For this value, the closest value in the DSPI is 0.480 µs

- For DSCK = 0 → 1/2 SCK period: For this value, the value for the DSPI is 20 ns

Table 15-47. DSPI Compatibility with MPC500 family QSPI

MPC500 family control bits DSPI corresponding control bits						Corresponding DSPI_CTAR configuration					
BITSE	CTAS[0]	DT	CTAS[1]	DSCK	CTAS[2]	DSPI_CTAR*x*	FMSZ	PDT	DT	PCSSCK	CSSCK
0		0		0		0	1111	10	0011	00	0000
0		0		1		1	1111	10	0011	user	user
0		1		0		2	1111	user[1]	user	00	0000
0		1		1		3	1111	user	user	user	user
1		0		0		4	user	10	0011	00	0000
1		0		1		5	user	10	0011	user	user
1		1		0		6	user	user	user	00	0000
1		1		1		7	user	user	user	user	user

[1] Selected by user

15.10.6 Calculation of FIFO pointer addresses

Complete visibility of the TX and RX FIFO contents is available through the FIFO registers, and valid entries can be identified through a memory mapped pointer and a memory mapped counter for each FIFO. The pointer to the first-in entry in each FIFO is memory mapped. For the TX FIFO the first-in pointer is the Transmit Next Pointer (TXNXTPTR). For the RX FIFO the first-in pointer is the Pop Next Pointer (POPNXTPTR). Figure 15-59 illustrates the concept of first-in and last-in FIFO entries along with the FIFO Counter. The TX FIFO is chosen for the illustration, but the concepts carry over to the RX FIFO. See Section 15.9.2.4, “Transmit first-in first-out (TX FIFO) buffering mechanism,” and Section 15.9.2.5, “Receive first-in first-out (RX FIFO) buffering mechanism,” for details on the FIFO operation.

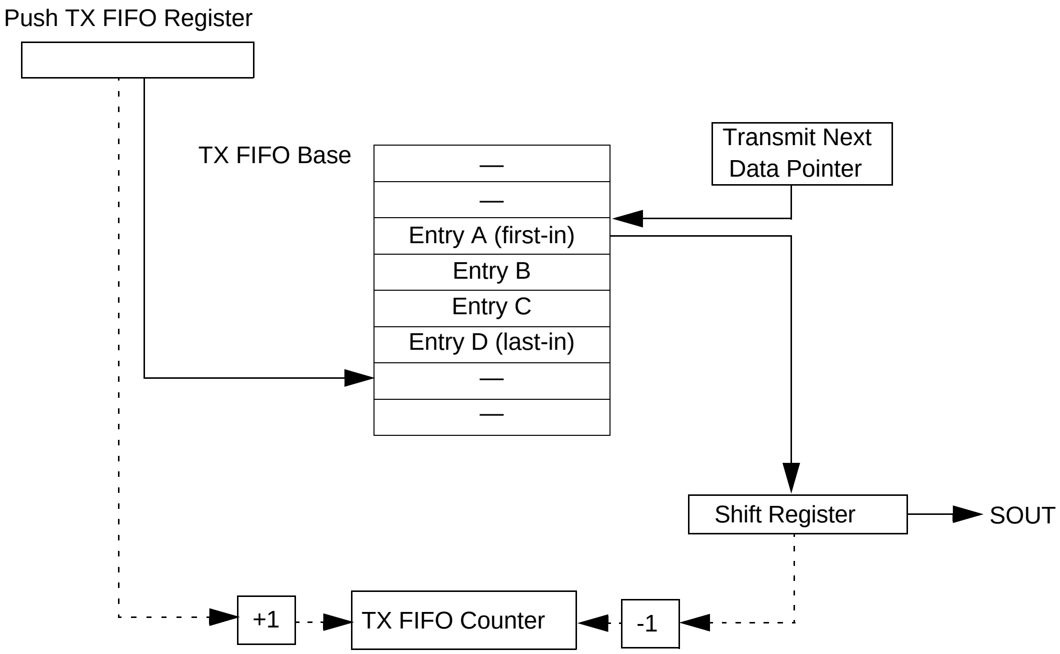


Figure 15-59. TX FIFO pointers and counter

15.10.6.1 Address calculation for the first-in entry and last-in entry in the TX FIFO

The memory address of the first-in entry in the TX FIFO is computed by the following equation:

$$\text{First-in Entry Address} = \text{TX FIFO Base} + (4 \times \text{TXNXTPTR}) \qquad \textit{Eqn. 15-7}$$

The memory address of the last-in entry in the TX FIFO is computed by the following equation:

$$\text{Last-in Entry address} = \text{TX FIFO Base} + 4 \times (\text{TXCTR} + \text{TXNXTPTR} - 1)\text{mod}(\text{TXFIFOdepth}) \qquad \textit{Eqn. 15-8}$$

TX FIFO Base: Base address of TX FIFO
TXCTR: TX FIFO Counter
TXNXTPTR: Transmit Next Pointer
TX FIFO Depth: Transmit FIFO depth, implementation-specific

15.10.6.2 Address calculation for the first-in entry and last-in entry in the RX FIFO

The memory address of the first-in entry in the RX FIFO is computed by the following equation:

$$\text{First-in Entry Address} = \text{RX FIFO Base} + (4 \times \text{POPNXTPTR}) \qquad \textit{Eqn. 15-9}$$

The memory address of the last-in entry in the RX FIFO is computed by the following equation:

$$\text{Last-in Entry address} = \text{RX FIFO Base} + 4 \times (\text{RXCTR} + \text{POPNXTPTR} - 1)\text{mod }(\text{RXFIFOdepth}) \qquad \textit{Eqn. 15-10}$$

RX FIFO Base: Base address of RX FIFO
RXCTR: RX FIFO counter
POPNXTPTR: Pop Next Pointer
RX FIFO Depth: Receive FIFO depth, implementation specific

Chapter 16
Development Trigger Semaphore (DTS)

16.1 Introduction

Devices in the MPC5676R family include a system development feature, the Development Trigger Semaphore (DTS) module, that enables software to signal an external tool by driving a persistent (affected only by reset or an external tool) signal on an external device pin. There are a variety of ways this module can be used, including as a component of an external real-time data acquisition system[1].

16.2 Overview

The Development Trigger Semaphore (DTS) module consists of three registers and a small amount of combinational logic to generate an output signal—DTS Trigger Out ($\overline{\text{DTO}}$). The registers are as follows.

- The DTS_SEMAPHORE register. Any bit in this 32-bit register, when set to a value of logic “1”, causes the DTS module output signal to be asserted, enabling an external tool to detect up to 32 signals from the application software. In an application, each bit is generally associated with a specific data set.

  Only the processor core and DMA module can set bits in this register. The bits can only be cleared by a tool access via Nexus Read/Write Access over the JTAG port.
- The DTS_STARTUP register. This register provides a mechanism for the external tool to notify software running on the CPU that the tool is connected and can provide information about either the type of tool or options that can be used by the software.
- The DTS_ENABLE register provides an enable/disable capability for the DTS feature.

The architecture is shown in Figure 16-1.

---

[1] When used as a component of a triggered data acquisition system, Nexus read/write access is via the JTAG interface of the Nexus debug port and is different than the data acquisition protocol defined in the IEEE-ISTO 5001-2003 or IEEE-ISTO 5001-2010 Nexus standards, which use the Nexus Auxiliary port.

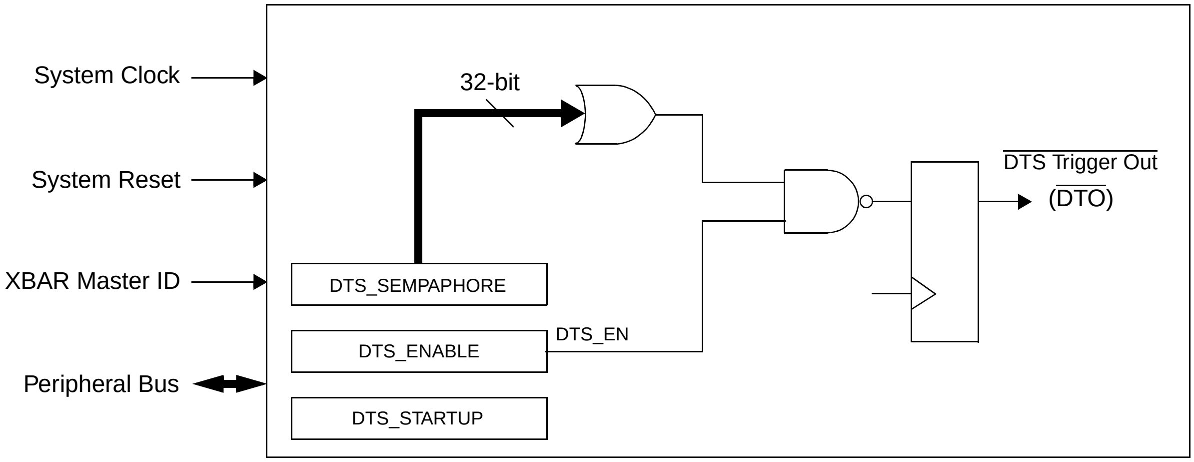


Figure 16-1. DTS block diagram

The DTS Trigger Out (DTO) signal is connected to one of the EVTO inputs of the Nexus Port Controller (NPC). The other EVTO inputs to the NPC are connected to the other Nexus modules in the device. DTO is asserted when any bit in the DTS_SEMAPHORE register is set.

NOTE

When the DTS module is enabled (DTS_ENABLE[DTS_EN]=0b1), the Nexus EVTO function of the EVTO pin is disabled and EVTO becomes the DTO. Unlike the EVTO function that only asserts for one clock, the DTO function remains asserted until the tool reads the DTS_SEMAPHORE register, clearing the register’s contents.

Figure 16-2, shows the chain of events that begins with setting of a bit in the DTS_SEMAPHORE register and the clearing of the register caused by a Nexus read.

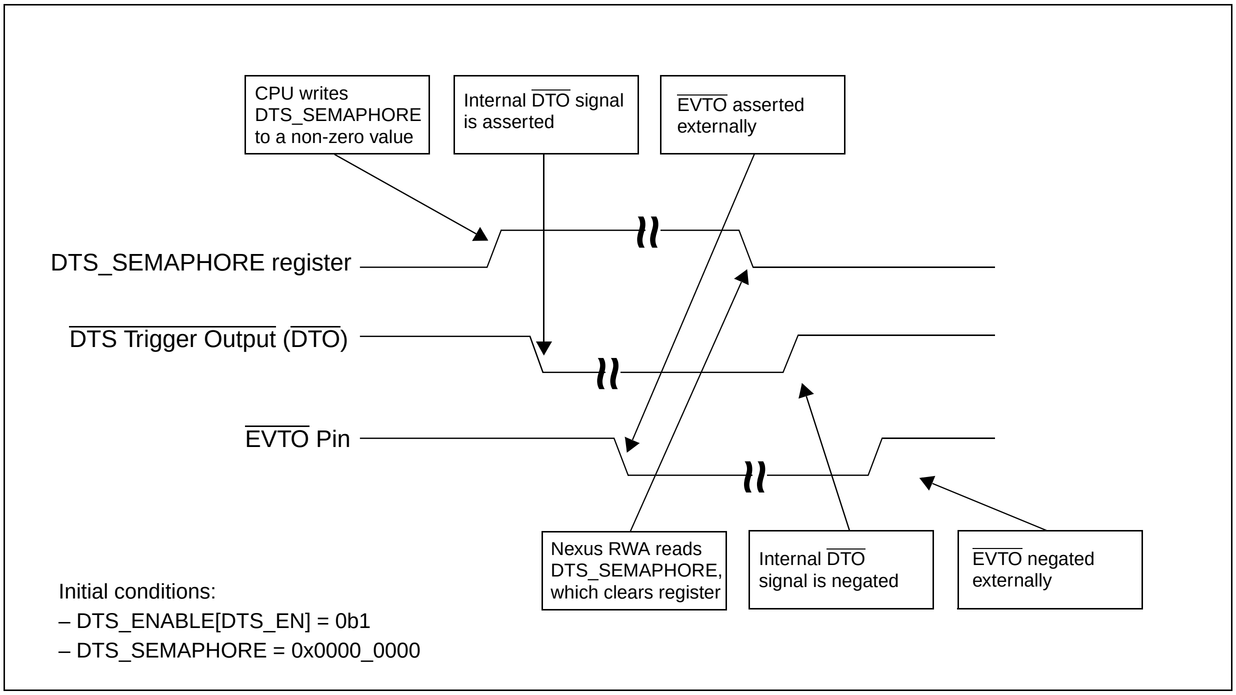


Figure 16-2. DTO event sequence

16.3 DTS device connections

The DTS module connects to the Peripheral Bridge (PBRIDGE) for access to the registers. The PBRIDGE is connected to a slave port of the Crossbar bus interface (XBAR). Connected to the XBAR master ports are the core, the eDMA module, the FlexRay module, and an External Bus interface[1].

The registers have limited access as described in Section 16.3.1, “DTS register access.” Access is based on the XBAR Master ID of the accessing module. Access to the DTS_SEMAPHORE register is limited to the core and the eDMA module and is restricted to only setting bits. Only an access via a Nexus Read/Write Access from an external tool through the Nexus/JTAG port of the device can clear bits in the DTS_SEMAPHORE register[2]. Similarly, the DTS_ENABLE and DTS_STARTUP registers can only be written via a Nexus Read/Write Access.

NOTE

Nexus Read/Write Accesses use the load/store bus of the core to perform accesses, but Nexus accesses have a different Master ID than normal core load/stores.

---

[1] The External Bus Interface XBAR master port is used for internal test of the device and is not accessible to the user.

[2] DTS_SEMAPHORE bits are cleared automatically when read through the Nexus/JTAG port.

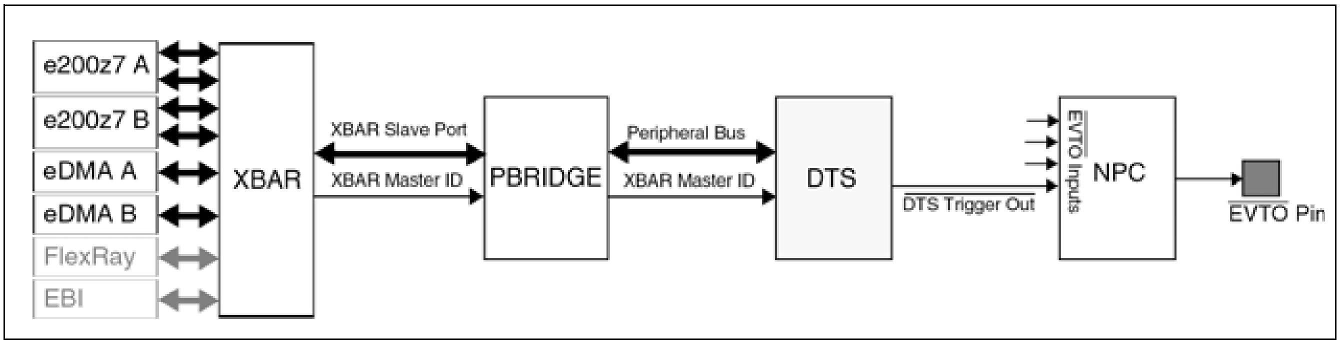


Figure 16-3. DTS device connections

16.3.1 DTS register access

A summary of accesses to all DTS registers by bus masters is provided in the table below. Note that only proper 32 bit accesses are valid. The effects of write accesses that are not 32 bits are not defined.

Table 16-1. DTS register access effects

Register	32-bit Read				32-bit Write			
	RWA[1]	Core	eDMA	FlexRay	RWA[1]	Core	eDMA	FlexRay
DTS_ENABLE	Data	Data	Data	Data	Data	No effect	No effect	No effect
DTS_STARTUP	Data	Data	Data	Data	Data	No effect	No effect	No effect
DTS_SEMAPHORE	Data and Clear[2]	Data	Data	Data	No effect	Bit OR	Bit OR	No effect

NOTES:

[1] Nexus Read/Write access via an external tool

[2] A read of the DTS_SEMAPHORE register by either Nexus Read/Write Access module is destructive and clears all bits in the register

Access to DTS module registers is controlled based on the XBAR Master ID of the accessing module. The table below shows the XBAR Master IDs for each of port.

NOTE

The XBAR Master ID should not be confused with the Master Port number of the XBAR. See Chapter 11, “AMBA Crossbar Switch (XBAR)” for details.

Tools must access the DTS registers (DTS_ENABLE, DTS_STARTUP, and DTS_SEMAPHORE) through the Nexus Read/Write Access mechanism of the core. JTAG accesses through the core appear as if the access is via the core and therefore will not have the same level of access as a Nexus R/W access.

16.4 Memory map

The table below shows the memory map of the Development Trigger Semaphore module registers. Three 32-bit registers are implemented. The rest of the memory map (0xC3F9_C00C through 0xC3F9_FFFF) is reserved.

Table 16-2. DTS Module

Address	Register	Description	Size (KB)	Access
DTS_BASE (0xC3F9_C000)	DTS_ENABLE	DTS output enable register	32	Restricted R/W[1]
DTS_BASE + 0x0004	DTS_STARTUP	DTS startup register	32	Restricted R/W[1]
DTS_BASE + 0x0008	DTS_SEMAPHORE	DTS semaphore register	32	Restricted R/W[1]
DTS_BASE + 0x000C – DTS_BASE + 0xFFFF	Reserved			

NOTES:

[1] Only certain types of accesses are allowed. See separate description.

16.5 Register descriptions

16.5.1 DTS Enable Register (DTS_ENABLE)

This DTS_ENABLE register controls the DTS Trigger Output ($\overline{\text{DTO}}$) and whether $\overline{\text{DTO}}$ is active on the $\overline{\text{EVTO}}$ output pin of the device. Figure 16-4 shows the format of the DTS_ENABLE register.

NOTE

Access to the DTS_SEMAPHORE and DTS_STARTUP registers are unaffected by the state of this register.

Address: DTS_BASE+0x0000 Access: Restricted R/W[1]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DTS_EN
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 16-4. DTS_ENABLE register

NOTES:

[1] The DTS_ENABLE register can be read by the core, but can only be written by a Nexus Read Write Access (RWA).

Table 16-3. DTS_ENABLE Field Descriptions

Name	Description
31 DTS_EN	DTS Enable. Controls whether the DTO signal is routed to the EVTO pin. 0: DTS output is disabled. 1: DTS output is enabled. Any bit set in the DTS_SEMAPHORE register will assert the DTS Trigger Output signal (DTO). **Note:** The DTS Enable bit is cleared by a device reset (either the assertion of the external RESET or by an internally generated reset). A JTAG reset does not change the state of this register.

16.5.2 DTS Startup Register (DTS_STARTUP)

The DTS_STARTUP register is used for tool detection and startup information exchange between the tool and software running on the microcontroller.

Address: DTS_BASE+0x0004

Access: Restricted R/W[1]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	AD31	AD30	AD29	AD28	AD27	AD26	AD25	AD24	AD23	AD22	AD21	AD20	AD19	AD18	AD17	AD16
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-5. DTS_STARTUP register

NOTES:

[1] The DTS_STARTUP register can be read by the core, the eDMA module and Nexus but can only be updated by a Nexus Read Write Access (RWA).

Table 16-4. DTS_STARTUP Field Descriptions

Name	Description
AD[31:0]	Application Dependent register bits. The bits have no defined meaning to the microcontroller. They are used to by an external tool to pass information, e.g., application options and status, to application software running on target microcontroller at startup time. Use a Nexus RWA 32-bit write access to update the contents of this register. **Note:** • A device reset (either from the RESET pin or an internally generated reset) clears all bits in the register. A JTAG reset does not change the contents of the register.

16.5.3 DTS Semaphore Register (DTS_SEMAPHORE)

The DTS_SEMAPHORE register is used by software to assert the $\overline{\text{DTO}}$ signal on the device $\overline{\text{EVTO}}$ pin. A 0b1 in any bit of this register causes the $\overline{\text{DTO}}$ signal on the $\overline{\text{EVTO}}$ pin to be driven low. The intended use of this register is for the $\overline{\text{DTO}}$ signal to notify tools that data is available. Individual bits are used to identify the specific data.

Address: DTS_BASE+0x0008

Access: Restricted R/W[1]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ST31	ST30	ST29	ST28	ST27	ST26	ST25	ST24	ST23	ST22	ST21	ST20	ST19	ST18	ST17	ST16
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ST15	ST14	ST13	ST12	ST11	ST10	ST9	ST8	ST7	ST06	ST5	ST04	ST3	ST02	ST1	ST0
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-6. DTS_SEMAPHORE register

NOTES:

[1] The core and eDMA modules can set bits in the DTS_SEMAPHORE register but cannot clear them—writes by the core and eDMA are bitwise ORed to the contents of the register. Nexus can only read this register but all bits are cleared after the read operation.

Table 16-5. DTS_SEMAPHORE Field Descriptions

Name	Description
ST[31:0]	Semaphore Trigger. When a core or eDMA writes a logical '1' to a bit, the bit is set. A write of '0' by the core or DMA does not change the state of the bit. • All register bits are set to ‘1’ by a device reset. • A JTAG reset does not change the state of this register. • The register can be accessed, with restrictions, by any core, DMA or any Nexus RWA. • For the core or DMA, only 32-bit write or read accesses are valid. • A core or DMA valid read access returns the current value of the register and leaves the register unchanged. 0: No flag. 1: Flag is set.

16.6 Example application

The calibration process of a new engine requires real-time access to calibration tables and the ability to update the tables in real-time[1]. The DTS module enables this capability by enabling software to assert a signal to an external device pin to notify an external tool that data is available. The tool can then retrieve the data.

In this type of application the DTS_SEMAPHORE register and $\overline{\text{DTS Trigger Output}}$ ($\overline{\text{DTO}}$) signal provide a mechanism to notify the calibration tool that the calibration variable or variables (or sets of measurements), up to 32, have been updated with new values and are available for the tool to access.

NOTE

> It is the user’s responsibility to ensure that the tool has time to retrieve the data prior to that particular trigger being set a second time. It is also permissible to have multiple triggers active at the same time or for a second trigger to be set before a previous trigger has been serviced, as long as it is not the same trigger (unless it is acceptable to the tool to not receive every data set).

Figure 16-7 shows an example DTS startup sequence for an external real-time data acquisition system. The startup and synchronization sequence can be as simple or as complicated as the need requires. However, a typical startup sequence is as follows:

1. The DTS_STARTUP register is cleared by a power on reset or any CPU reset.
2. The tool writes a non-zero value to the DTS_STARTUP register.
3. The CPU (user application software) then reads the value of the DTS_STARTUP register. Based on this value, different initialization options can be selected. The bits can be used for any application specific definitions.
4. Since the DTS_SEMAPHORE register is cleared when the tool reads the current value. The tool should perform all necessary initialization before reading this register. The application software can then check that the DTS_SEMAPHORE register was cleared by the tool, to determine that it is safe to start using it for its intended raster trigger semaphore function.
5. An optional hand shake from the CPU can be used to inform the tool that the user software has detected that the tool is attached and the CPU has performed the proper initialization for the tool by writing a predefined value to the DTS_SEMAPHORE register (the example shown in the figure above uses 0xAAAA_AAAA—all A's was used since it is unrealistic that 16 channels could be enabled very quickly after start up after a reset).

---

[1] MPC5676R devices also include an MMU modification feature, which enables real-time switching of calibration tables.

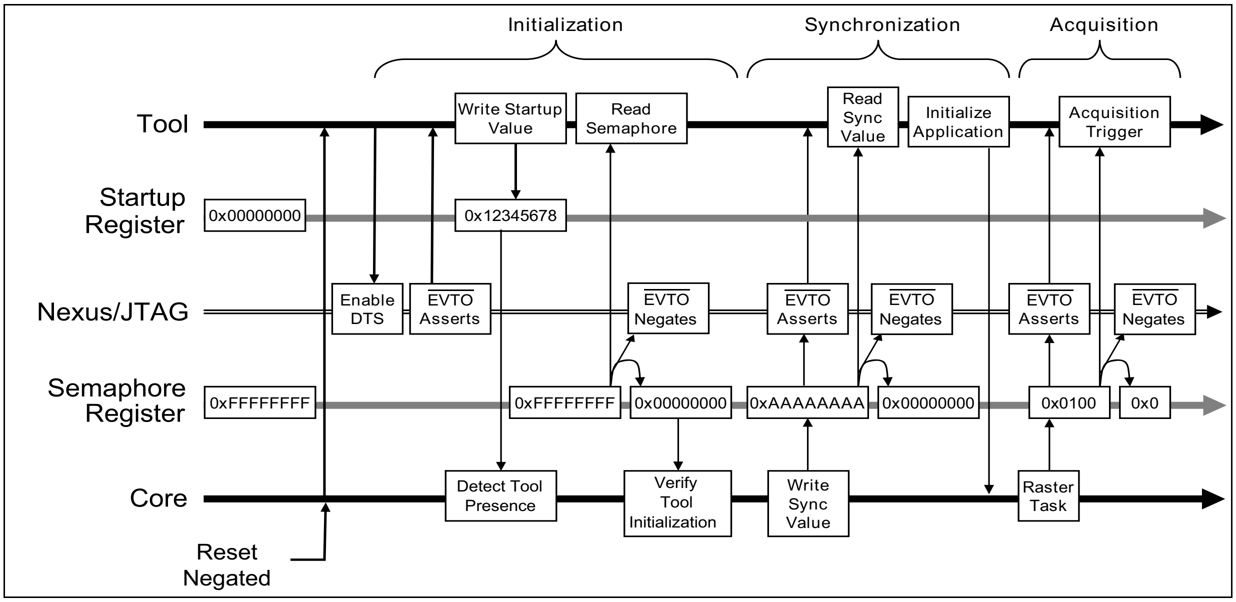


Figure 16-7. DTS startup sequence example

Chapter 17
Enhanced Direct Memory Access Controller (eDMA)

17.1 Introduction

This device includes two enhanced direct memory access controller (eDMA) blocks. The eDMA is a second-generation platform block capable of performing complex data movements through *n* programmable channels (*n*=64 for eDMA_A, *n*=64 for eDMA_B), with minimal intervention from the host processor. The hardware microarchitecture includes a DMA engine that performs source and destination address calculations, and the actual data movement operations, along with an SRAM-based memory containing the transfer control descriptors (TCD) for the channels.

17.1.1 Block Diagram

Figure 17-1 shows a simplified block diagram of each eDMA.

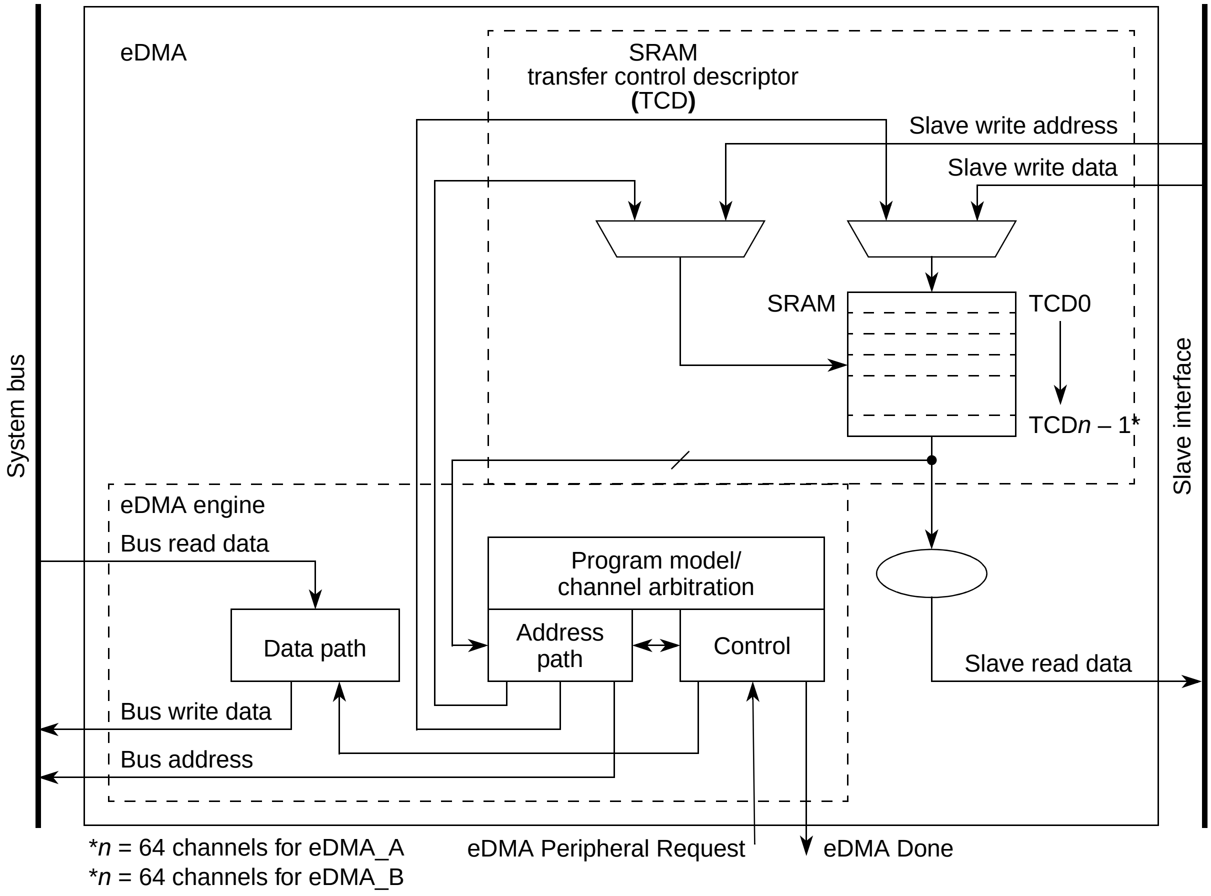


Figure 17-1. eDMA Block Diagram

17.1.2 Features

Each eDMA has these major features:

- All data movement via dual-address transfers: read from source, write to destination
  - — Programmable source, destination addresses, transfer size, and support for enhanced addressing modes
- Performs complex data transfers with minimal intervention from a host processor
  - — 32 bytes of data registers, used as temporary storage to support burst transfers (refer to SSIZE bit)
  - — Connections to the crossbar switch for bus mastering the data movement
- Transfer control descriptor organized to support two-deep, nested transfer operations
  - — An inner data transfer loop defined by a minor byte transfer count
  - — An outer data transfer loop defined by a major iteration count

- Channel activation via one of three methods:
  - Explicit software initiation
  - Initiation via a channel-to-channel linking mechanism for continuous transfers
  - Peripheral-paced hardware requests (one per channel)

    All three methods require one activation per execution of the minor loop
- Support for fixed-priority and round-robin channel arbitration
- Support for complex data structures
- Support to cancel transfers via software
- Channel completion reported via optional interrupt requests
  - Interrupt assignments are listed in the Chapter 27, "Interrupts and Interrupt Controller (INTC).
- Support for scatter-gather DMA processing
- Support for complex data structures
- Any channel can be programmed to be suspended by a higher priority channel's activation, before completion of a minor loop.

17.1.3 Modes of Operation

There are two main operating modes of eDMA: normal mode and debug mode. These modes are briefly described in this section.

17.1.3.1 Normal Mode

In normal mode, the eDMA is used to transfer data between a source and a destination. The source and destination can be a memory block or an I/O block capable of operation with the eDMA.

17.1.3.2 Debug Mode

In debug mode, the eDMA does not accept new transfer requests when its debug input signal is asserted. If the signal is asserted during transfer of a block of data described by a minor loop in the current active channel's TCD, the eDMA continues operation until completion of the minor loop.

17.2 External Signal Description

The eDMA has no direct external MCU signals. However, there are four external pins that can be configured in the SIU as external eDMA requests to four eDMA channels. Refer to Section 3.2.1.6, "DMA/Interrupt Request Select Register (SIU_DIRSR)" for details.

17.3 Memory Map and Registers

This section provides a detailed description of all eDMA registers.

17.3.1 Module Memory Map

The eDMA memory map is shown in Table 17-1. The address of each register is given as an offset to the eDMA base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed. In register names, an "*x*" is used to indicate A or B, depending on which eDMA's register you are using. If a register only exists in one of the eDMAs, the register description will state that.

The eDMA's programming model is partitioned into two regions: the first region defines a number of registers providing control functions; however, the second region corresponds to the local transfer control descriptor memory.

Some registers are implemented as two 32-bit registers, and include H and L suffixes, signaling the high and low portions of the control function.

Base addresses of eDMA_*x*:

- EDMA_A_BASE = 0xFFF4_4000
- EDMA_B_BASE = 0xFFF5_4000

Table 17-1. eDMA Memory Map

Offset from EDMA_*x*_BASE	Register	Bits	Access	Reset Value	Section/Page
0x0000	EDMA_*x*_MCR—eDMA module control register	32	R/W	0x0000_E400	17.3.2.1/17-9
0x0004	EDMA_*x*_ESR—eDMA error status register	32	R	0x0000_0000	17.3.2.2/17-12
0x0008	EDMA_*x*_ERQRH—eDMA_A enable request high register (channels 63–32) Reserved (eDMA_B)	32	R/W	0000_0000	17.3.2.3/17-15
0x000C	EDMA_*x*_ERQRL—eDMA enable request low register (channels 31–00)	32	R/W	0x0000_0000	17.3.2.3/17-15
0x0010	EDMA_*x*_EEIRLH—eDMA_A enable error interrupt register (channels 63–32) Reserved (eDMA_B)	32	R/W	0x0000_0000	17.3.2.4/17-17
0x0014	EDMA_*x*_EEIRL—eDMA enable error interrupt register (channels 31–00)	32	R/W	0x0000_0000	17.3.2.4/17-17
0x0018	EDMA_*x*_SERQR—eDMA set enable request register	8	W	0x00	17.3.2.5/17-19
0x0019	EDMA_*x*_CERQR—eDMA clear enable request register	8	W	0x00	17.3.2.6/17-20
0x001A	EDMA_*x*_SEEIR—eDMA set enable error interrupt register	8	W	0x00	17.3.2.7/17-21
0x001B	EDMA_*x*_CEEIR—eDMA clear enable error interrupt register	8	W	0x00	17.3.2.8/17-22
0x001C	EDMA_*x*_CIRQR—eDMA clear interrupt request register	8	W	0x00	17.3.2.9/17-23
0x001D	EDMA_*x*_CER—eDMA clear error register	8	W	0x00	17.3.2.10/17-23

Table 17-1. eDMA Memory Map (continued)

Offset from EDMA_*x*_BASE	Register	Bits	Access	Reset Value	Section/Page
0x001E	EDMA_*x*_SSBR—eDMA set start bit register	8	W	0x00	17.3.2.11/17-24
0x001F	EDMA_*x*_CDSBR—eDMA clear done status bit register	8	W	0x00	17.3.2.12/17-24
0x0020	EDMA_*x*_IRQRH—eDMA_A interrupt request register (channels 63–32) Reserved (eDMA_B)	32	R/W	0x0000_0000	17.3.2.13/17-26
0x0024	EDMA_*x*_IRQRL—eDMA interrupt request register (channels 31–00)	32	R/W	0x0000_0000	17.3.2.13/17-26
0x0028	EDMA_*x*_ERH—eDMA_A error register (channels 63–32) Reserved (eDMA_B)	32	R/W	0x0000_0000	17.3.2.14/17-27
0x002C	EDMA_*x*_ERL—eDMA error register (channels 31–00)	32	R/W	0x0000_0000	17.3.2.14/17-27
0x0030	EDMA_*x*_HRSH—eDMA_A hardware request status register (channels 63–32) Reserved (eDMA_B)	32	R/W	0x0000_0000	17.3.2.15/17-28
0x0034	EDMA_*x*_HRSL—eDMA hardware request status register (channels 31–00)	32	R/W	0x0000_0000	17.3.2.15/17-28
0x0038	EDMA_*x*_GWRH—eDMA Global Write Register High	32	R/W	0x0000_0000	17.3.2.16/17-29
0x003C	EDMA_*x*_GWRL—eDMA Global Write Register Low	32	R/W	0x0000_0000	17.3.2.16/17-29
0x0040–0x00FF	Reserved				
0x0100	EDMA_*x*_CPR0—eDMA channel 0 priority register	8	R/W	0x00	17.3.2.17/17-29
0x0101	EDMA_*x*_CPR1—eDMA channel 1 priority register	8	R/W	0x01	17.3.2.17/17-29
0x0102	EDMA_*x*_CPR2—eDMA channel 2 priority register	8	R/W	0x02	17.3.2.17/17-29
0x0103	EDMA_*x*_CPR3—eDMA channel 3 priority register	8	R/W	0x03	17.3.2.17/17-29
0x0104	EDMA_*x*_CPR4—eDMA channel 4 priority register	8	R/W	0x04	17.3.2.17/17-29
0x0105	EDMA_*x*_CPR5—eDMA channel 5 priority register	8	R/W	0x05	17.3.2.17/17-29
0x0106	EDMA_*x*_CPR6—eDMA channel 6 priority register	8	R/W	0x06	17.3.2.17/17-29
0x0107	EDMA_*x*_CPR7—eDMA channel 7 priority register	8	R/W	0x07	17.3.2.17/17-29
0x0108	EDMA_*x*_CPR8—eDMA channel 8 priority register	8	R/W	0x08	17.3.2.17/17-29
0x0109	EDMA_*x*_CPR9—eDMA channel 9 priority register	8	R/W	0x09	17.3.2.17/17-29
0x010A	EDMA_*x*_CPR10—eDMA channel 10 priority register	8	R/W	0x0A	17.3.2.17/17-29
0x010B	EDMA_*x*_CPR11—eDMA channel 11 priority register	8	R/W	0x0B	17.3.2.17/17-29
0x010C	EDMA_*x*_CPR12—eDMA channel 12 priority register	8	R/W	0x0C	17.3.2.17/17-29
0x010D	EDMA_*x*_CPR13—eDMA channel 13 priority register	8	R/W	0x0D	17.3.2.17/17-29
0x010E	EDMA_*x*_CPR14—eDMA channel 14 priority register	8	R/W	0x0E	17.3.2.17/17-29
0x010F	EDMA_*x*_CPR15—eDMA channel 15 priority register	8	R/W	0x0F	17.3.2.17/17-29

Table 17-1. eDMA Memory Map (continued)

Offset from EDMA_*x*_BASE	Register	Bits	Access	Reset Value	Section/Page
0x0110	EDMA_*x*_CPR16—eDMA channel 16 priority register	8	R/W	0x10	17.3.2.17/17-29
0x0111	EDMA_*x*_CPR17—eDMA channel 17 priority register	8	R/W	0x11	17.3.2.17/17-29
0x0112	EDMA_*x*_CPR18—eDMA channel 18 priority register	8	R/W	0x12	17.3.2.17/17-29
0x0113	EDMA_*x*_CPR19—eDMA channel 19 priority register	8	R/W	0x13	17.3.2.17/17-29
0x0114	EDMA_*x*_CPR20—eDMA channel 20 priority register	8	R/W	0x14	17.3.2.17/17-29
0x0115	EDMA_*x*_CPR21—eDMA channel 21 priority register	8	R/W	0x15	17.3.2.17/17-29
0x0116	EDMA_*x*_CPR22—eDMA channel 22 priority register	8	R/W	0x16	17.3.2.17/17-29
0x0117	EDMA_*x*_CPR23—eDMA channel 23 priority register	8	R/W	0x17	17.3.2.17/17-29
0x0118	EDMA_*x*_CPR24—eDMA channel 24 priority register	8	R/W	0x18	17.3.2.17/17-29
0x0119	EDMA_*x*_CPR25—eDMA channel 25 priority register	8	R/W	0x19	17.3.2.17/17-29
0x011A	EDMA_*x*_CPR26—eDMA channel 26 priority register	8	R/W	0x1A	17.3.2.17/17-29
0x011B	EDMA_*x*_CPR27—eDMA channel 27 priority register	8	R/W	0x1B	17.3.2.17/17-29
0x011C	EDMA_*x*_CPR28—eDMA channel 28 priority register	8	R/W	0x1C	17.3.2.17/17-29
0x011D	EDMA_*x*_CPR29—eDMA channel 29 priority register	8	R/W	0x1D	17.3.2.17/17-29
0x011E	EDMA_*x*_CPR30—eDMA channel 30 priority register	8	R/W	0x1E	17.3.2.17/17-29
0x011F	EDMA_*x*_CPR31—eDMA channel 31 priority register	8	R/W	0x1F	17.3.2.17/17-29
0x0120	EDMA_*x*_CPR32—eDMA channel 32 priority register	8	R/W	0x20	17.3.2.17/17-29
0x0121	EDMA_*x*_CPR33—eDMA channel 33 priority register	8	R/W	0x21	17.3.2.17/17-29
0x0122	EDMA_*x*_CPR34—eDMA channel 34 priority register	8	R/W	0x22	17.3.2.17/17-29
0x0123	EDMA_*x*_CPR35—eDMA channel 35 priority register	8	R/W	0x23	17.3.2.17/17-29
0x0124	EDMA_*x*_CPR36—eDMA channel 36 priority register	8	R/W	0x24	17.3.2.17/17-29
0x0125	EDMA_*x*_CPR37—eDMA channel 37 priority register	8	R/W	0x25	17.3.2.17/17-29
0x0126	EDMA_*x*_CPR38—eDMA channel 38 priority register	8	R/W	0x26	17.3.2.17/17-29
0x0127	EDMA_*x*_CPR39—eDMA channel 39 priority register	8	R/W	0x27	17.3.2.17/17-29
0x0128	EDMA_*x*_CPR40—eDMA channel 40 priority register	8	R/W	0x28	17.3.2.17/17-29
0x0129	EDMA_*x*_CPR41—eDMA channel 41 priority register	8	R/W	0x29	17.3.2.17/17-29
0x012A	EDMA_*x*_CPR42—eDMA channel 42 priority register	8	R/W	0x2A	17.3.2.17/17-29
0x012B	EDMA_*x*_CPR43—eDMA channel 43 priority register	8	R/W	0x2B	17.3.2.17/17-29
0x012C	EDMA_*x*_CPR44—eDMA channel 44 priority register	8	R/W	0x2C	17.3.2.17/17-29
0x012D	EDMA_*x*_CPR45—eDMA channel 45 priority register	8	R/W	0x2D	17.3.2.17/17-29
0x012E	EDMA_*x*_CPR46—eDMA channel 46 priority register	8	R/W	0x2E	17.3.2.17/17-29
0x012F	EDMA_*x*_CPR47—eDMA channel 47 priority register	8	R/W	0x2F	17.3.2.17/17-29

Table 17-1. eDMA Memory Map (continued)

Offset from EDMA_*x*_BASE	Register	Bits	Access	Reset Value	Section/Page
0x0130	EDMA_*x*_CPR48—eDMA channel 48 priority register	8	R/W	0x30	17.3.2.17/17-29
0x0131	EDMA_*x*_CPR49—eDMA channel 49 priority register	8	R/W	0x31	17.3.2.17/17-29
0x0132	EDMA_*x*_CPR50—eDMA channel 50 priority register	8	R/W	0x32	17.3.2.17/17-29
0x0133	EDMA_*x*_CPR51—eDMA channel 51 priority register	8	R/W	0x33	17.3.2.17/17-29
0x0134	EDMA_*x*_CPR52—eDMA channel 52 priority register	8	R/W	0x34	17.3.2.17/17-29
0x0135	EDMA_*x*_CPR53—eDMA channel 53 priority register	8	R/W	0x35	17.3.2.17/17-29
0x0136	EDMA_*x*_CPR54—eDMA channel 54 priority register	8	R/W	0x36	17.3.2.17/17-29
0x0137	EDMA_*x*_CPR55—eDMA channel 55 priority register	8	R/W	0x37	17.3.2.17/17-29
0x0138	EDMA_*x*_CPR56—eDMA channel 56 priority register	8	R/W	0x38	17.3.2.17/17-29
0x0139	EDMA_*x*_CPR57—eDMA channel 57 priority register	8	R/W	0x39	17.3.2.17/17-29
0x013A	EDMA_*x*_CPR58—eDMA channel 58 priority register	8	R/W	0x3A	17.3.2.17/17-29
0x013B	EDMA_*x*_CPR59—eDMA channel 59 priority register	8	R/W	0x3B	17.3.2.17/17-29
0x013C	EDMA_*x*_CPR60—eDMA channel 60 priority register	8	R/W	0x3C	17.3.2.17/17-29
0x013D	EDMA_*x*_CPR61—eDMA channel 61 priority register	8	R/W	0x3D	17.3.2.17/17-29
0x013E	EDMA_*x*_CPR62—eDMA channel 62 priority register	8	R/W	0x3E	17.3.2.17/17-29
0x013F	EDMA_*x*_CPR63—eDMA channel 63 priority register	8	R/W	0x3F	17.3.2.17/17-29
0x0140–0x0FFF	Reserved				
0x1000	EDMA_*x*_TCD00—eDMA transfer control descriptor 00	256	R/W	—[1]	17.3.2.18/17-30
0x1020	EDMA_*x*_TCD01—eDMA transfer control descriptor 01	256	R/W	—[1]	17.3.2.18/17-30
0x1040	EDMA_*x*_TCD02—eDMA transfer control descriptor 02	256	R/W	—[1]	17.3.2.18/17-30
0x1060	EDMA_*x*_TCD03—eDMA transfer control descriptor 03	256	R/W	—[1]	17.3.2.18/17-30
0x1080	EDMA_*x*_TCD04—eDMA transfer control descriptor 04	256	R/W	—[1]	17.3.2.18/17-30
0x10A0	EDMA_*x*_TCD05—eDMA transfer control descriptor 05	256	R/W	—[1]	17.3.2.18/17-30
0x10C0	EDMA_*x*_TCD06—eDMA transfer control descriptor 06	256	R/W	—[1]	17.3.2.18/17-30
0x10E0	EDMA_*x*_TCD07—eDMA transfer control descriptor 07	256	R/W	—[1]	17.3.2.18/17-30
0x1100	EDMA_*x*_TCD08—eDMA transfer control descriptor 08	256	R/W	—[1]	17.3.2.18/17-30
0x1120	EDMA_*x*_TCD09—eDMA transfer control descriptor 09	256	R/W	—[1]	17.3.2.18/17-30
0x1140	EDMA_*x*_TCD10—eDMA transfer control descriptor 10	256	R/W	—[1]	17.3.2.18/17-30
0x1160	EDMA_*x*_TCD11—eDMA transfer control descriptor 11	256	R/W	—[1]	17.3.2.18/17-30
0x1180	EDMA_*x*_TCD12—eDMA transfer control descriptor 12	256	R/W	—[1]	17.3.2.18/17-30
0x11A0	EDMA_*x*_TCD13—eDMA transfer control descriptor 13	256	R/W	—[1]	17.3.2.18/17-30
0x11C0	EDMA_*x*_TCD14—eDMA transfer control descriptor 14	256	R/W	—[1]	17.3.2.18/17-30

Table 17-1. eDMA Memory Map (continued)

Offset from EDMA_*x*_BASE	Register	Bits	Access	Reset Value	Section/Page
0x11E0	EDMA_*x*_TCD15—eDMA transfer control descriptor 15	256	R/W	—[1]	17.3.2.18/17-30
0x1200	EDMA_*x*_TCD16—eDMA transfer control descriptor 16	256	R/W	—[1]	17.3.2.18/17-30
0x1220	EDMA_*x*_TCD17—eDMA transfer control descriptor 17	256	R/W	—[1]	17.3.2.18/17-30
0x1240	EDMA_*x*_TCD18—eDMA transfer control descriptor 18	256	R/W	—[1]	17.3.2.18/17-30
0x1260	EDMA_*x*_TCD19—eDMA transfer control descriptor 19	256	R/W	—[1]	17.3.2.18/17-30
0x1280	EDMA_*x*_TCD20—eDMA transfer control descriptor 20	256	R/W	—[1]	17.3.2.18/17-30
0x12A0	EDMA_*x*_TCD21—eDMA transfer control descriptor 21	256	R/W	—[1]	17.3.2.18/17-30
0x12C0	EDMA_*x*_TCD22—eDMA transfer control descriptor 22	256	R/W	—[1]	17.3.2.18/17-30
0x12E0	EDMA_*x*_TCD23—eDMA transfer control descriptor 23	256	R/W	—[1]	17.3.2.18/17-30
0x1300	EDMA_*x*_TCD24—eDMA transfer control descriptor 24	256	R/W	—[1]	17.3.2.18/17-30
0x1320	EDMA_*x*_TCD25—eDMA transfer control descriptor 25	256	R/W	—[1]	17.3.2.18/17-30
0x1340	EDMA_*x*_TCD26—eDMA transfer control descriptor 26	256	R/W	—[1]	17.3.2.18/17-30
0x1360	EDMA_*x*_TCD27—eDMA transfer control descriptor 27	256	R/W	—[1]	17.3.2.18/17-30
0x1380	EDMA_*x*_TCD28—eDMA transfer control descriptor 28	256	R/W	—[1]	17.3.2.18/17-30
0x13A0	EDMA_*x*_TCD29—eDMA transfer control descriptor 29	256	R/W	—[1]	17.3.2.18/17-30
0x13C0	EDMA_*x*_TCD30—eDMA transfer control descriptor 30	256	R/W	—[1]	17.3.2.18/17-30
0x13E0	EDMA_*x*_TCD31—eDMA transfer control descriptor 31	256	R/W	—[1]	17.3.2.18/17-30
0x1400	EDMA_*x*_TCD32—eDMA transfer control descriptor 32	256	R/W	—[1]	17.3.2.18/17-30
0x1420	EDMA_*x*_TCD33—eDMA transfer control descriptor 33	256	R/W	—[1]	17.3.2.18/17-30
0x1440	EDMA_*x*_TCD34—eDMA transfer control descriptor 34	256	R/W	—[1]	17.3.2.18/17-30
0x1460	EDMA_*x*_TCD35—eDMA transfer control descriptor 35	256	R/W	—[1]	17.3.2.18/17-30
0x1480	EDMA_*x*_TCD36—eDMA transfer control descriptor 36	256	R/W	—[1]	17.3.2.18/17-30
0x14A0	EDMA_*x*_TCD37—eDMA transfer control descriptor 37	256	R/W	—[1]	17.3.2.18/17-30
0x14C0	EDMA_*x*_TCD38—eDMA transfer control descriptor 38	256	R/W	—[1]	17.3.2.18/17-30
0x14E0	EDMA_*x*_TCD39—eDMA transfer control descriptor 39	256	R/W	—[1]	17.3.2.18/17-30
0x1500	EDMA_*x*_TCD40—eDMA transfer control descriptor 40	256	R/W	—[1]	17.3.2.18/17-30
0x1520	EDMA_*x*_TCD41—eDMA transfer control descriptor 41	256	R/W	—[1]	17.3.2.18/17-30
0x1540	EDMA_*x*_TCD42—eDMA transfer control descriptor 42	256	R/W	—[1]	17.3.2.18/17-30
0x1560	EDMA_*x*_TCD43—eDMA transfer control descriptor 43	256	R/W	—[1]	17.3.2.18/17-30
0x1580	EDMA_*x*_TCD44—eDMA transfer control descriptor 44	256	R/W	—[1]	17.3.2.18/17-30
0x15A0	EDMA_*x*_TCD45—eDMA transfer control descriptor 45	256	R/W	—[1]	17.3.2.18/17-30
0x15C0	EDMA_*x*_TCD46—eDMA transfer control descriptor 46	256	R/W	—[1]	17.3.2.18/17-30

Table 17-1. eDMA Memory Map (continued)

Offset from EDMA_*x*_BASE	Register	Bits	Access	Reset Value	Section/Page
0x15E0	EDMA_*x*_TCD47—eDMA transfer control descriptor 47	256	R/W	—[1]	17.3.2.18/17-30
0x1600	EDMA_*x*_TCD48—eDMA transfer control descriptor 48	256	R/W	—[1]	17.3.2.18/17-30
0x1620	EDMA_*x*_TCD49—eDMA transfer control descriptor 49	256	R/W	—[1]	17.3.2.18/17-30
0x1640	EDMA_*x*_TCD50—eDMA transfer control descriptor 50	256	R/W	—[1]	17.3.2.18/17-30
0x1660	EDMA_*x*_TCD51—eDMA transfer control descriptor 51	256	R/W	—[1]	17.3.2.18/17-30
0x1680	EDMA_*x*_TCD52—eDMA transfer control descriptor 52	256	R/W	—[1]	17.3.2.18/17-30
0x16A0	EDMA_*x*_TCD53—eDMA transfer control descriptor 53	256	R/W	—[1]	17.3.2.18/17-30
0x16C0	EDMA_*x*_TCD54—eDMA transfer control descriptor 54	256	R/W	—[1]	17.3.2.18/17-30
0x16E0	EDMA_*x*_TCD55—eDMA transfer control descriptor 55	256	R/W	—[1]	17.3.2.18/17-30
0x1700	EDMA_*x*_TCD56—eDMA transfer control descriptor 56	256	R/W	—[1]	17.3.2.18/17-30
0x1720	EDMA_*x*_TCD57—eDMA transfer control descriptor 57	256	R/W	—[1]	17.3.2.18/17-30
0x1740	EDMA_*x*_TCD58—eDMA transfer control descriptor 58	256	R/W	—[1]	17.3.2.18/17-30
0x1760	EDMA_*x*_TCD59—eDMA transfer control descriptor 59	256	R/W	—[1]	17.3.2.18/17-30
0x1780	EDMA_*x*_TCD60—eDMA transfer control descriptor 60	256	R/W	—[1]	17.3.2.18/17-30
0x17A0	EDMA_*x*_TCD61—eDMA transfer control descriptor 61	256	R/W	—[1]	17.3.2.18/17-30
0x17C0	EDMA_*x*_TCD62—eDMA transfer control descriptor 62	256	R/W	—[1]	17.3.2.18/17-30
0x17E0	EDMA_*x*_TCD63—eDMA transfer control descriptor 63	256	R/W	—[1]	17.3.2.18/17-30

[1] See specific register description.

17.3.2 Register Descriptions

This section lists the eDMA registers in address order and describes the registers and their bit fields.

Reading reserved bits in a register returns the value of zero. Writes to reserved bits in a register are ignored. Reading or writing to a reserved memory location generates a bus error.

17.3.2.1 eDMA Control Register (EDMA_*x*_MCR)

The 32-bit EDMA_*x*_MCR defines the basic operating configuration of the eDMA.

The eDMA arbitrates channel service requests in four groups (0, 1) of 16 channels each:

- Group 0 contains channels 0–15
- Group 1 contains channels 16–31
- Group 2 contains channels 32-47
- Group 3 contains channels 48-63

Arbitration within a group can be configured to use a fixed priority or a round robin. In fixed-priority arbitration, the highest priority channel requesting service is selected to execute. The priorities are assigned by the channel priority registers. See Section 17.3.2.17, "eDMA Channel n Priority Registers (EDMA_x_CPRn)". In round-robin arbitration mode, the channel priorities are ignored and the channels within each group are cycled through, from channel 15 down to channel 0,without regard to priority.

The group priorities operate in a similar fashion. In group fixed-priority arbitration mode, channel service requests in the highest priority group are executed first where priority level 3 is the highest and priority level 0 is the lowest. The group priorities are assigned in the GRP*n*PRI fields of the eDMA control register (EDMA_*x*_MCR). All group priorities must have unique values prior to any channel service requests occur, otherwise a configuration error is reported. In group round-robin mode, the group priorities are ignored and the groups are cycled through, from group 3 down to group 0, without regard to priority.

Minor loop offsets are address offset values added to the final source address (SADDR) or destination address (DADDR) upon minor loop completion. When minor loop offsets are enabled, the minor loop offset (MLOFF) is added to the final source address (SADDR) or to the final destination address (DADDR) or to both addresses prior to the addresses being written back into the TCD. If the major loop is complete, the minor loop offset is ignored and the major loop address offsets (SLAST and DLAST_SGA) are used to compute the next EDMA_*x*_TCD.SADDR and EDMA_*x*_TCD.DADDR values.

When minor loop mapping is enabled (EDMA_*x*_MCR[EMLM] = 1), TCD*n* word2 is redefined. A portion of TCD*n* word2 is used to specify multiple fields: a source enable bit (SMLOE) to specify that the minor loop offset should be applied to the source address (SADDR) upon minor loop completion, a destination enable bit (DMLOE) to specify the minor loop offset should be applied to the destination address (DADDR) upon minor loop completion, and the sign extended minor loop offset value (MLOFF). The same offset value (MLOFF) is used for both source and destination minor loop offsets.

When either of the minor loop offsets is enabled (SMLOE is set or DMLOE is set), the NBYTES field is reduced to 10 bits. When both minor loop offsets are disabled (SMLOE is cleared and DMLOE is cleared), the NBYTES field becomes a 30-bit vector.

When minor loop mapping is disabled (EDMA_*x*_MCR[EMLM] = 0), all 32 bits of TCD*n* word2 are assigned to the NBYTES field. See Section 17.3.2.18, "Transfer Control Descriptor (TCD)", for more details.

Offset: EDMA_*x*_BASE + 0x0000

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CXFR	ECX
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	GRP3PRI		GRP2PRI		GRP1PRI		GRP0PRI		EMLM	CLM	HALT	HOE	ERGA	ERCA	EDBG	0
W																
Reset	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0

Figure 17-2. eDMA Control Register (EDMA_*x*_CR)

Table 17-2. EDMA_*x*_MCR Field Descriptions

Field	Description
0–13	Reserved
14 CXFR	Cancel Transfer. 0 Normal operation. 1 Cancel the remaining data transfer. Stop the executing channel and force the minor loop to be finished. The cancel takes effect after the last write of the current read/write sequence. The CXFR bit clears itself after the cancel has been honored. This cancel retires the channel normally as if the minor loop was completed.
15 ECX	Error cancel transfer. 0 Normal operation. 1 Cancel the remaining data transfer in the same fashion as the CXFR cancel transfer. Stop the executing channel and force the minor loop to be finished. The cancel takes effect after the last write of the current read/write sequence. The ECX bit clears itself after the cancel has been honored. In addition to cancelling the transfer, the ECX treats the cancel as an error condition; thus updating the EDMA_*x*_ESR register and generating an optional error interrupt. See Section 17.3.2.2, “eDMA Error Status Register (EDMA_x_ESR)”.
16–17 GRP3PRI	Channel group 3 priority. Group 3 priority level when fixed priority group arbitration is enabled.
18–19 GRP2PRI	Channel group 2 priority. Group 2 priority level when fixed priority group arbitration is enabled.
20–21 GRP1PRI	Channel group 1 priority. Group 1 priority level when fixed priority group arbitration is enabled.
22–23 GRP0PRI	Channel group 0 priority. Group 0 priority level when fixed priority group arbitration is enabled.
24 EMLM	Enable minor loop mapping. 0 Minor loop mapping disabled. TCD Word 2 is defined as a 32-bit n-bytes field. 1 Minor loop mapping enabled. When set, TCD*n* Word 2 is redefined to include individual enable fields, an offset field and the NBYTES field. The individual enable fields allow the minor loop offset to be applied to the source address, the destination address, or both. The NBYTES field is reduced when either offset is enabled.

Table 17-2. EDMA_*x*_MCR Field Descriptions (continued)

Field	Description
25 CLM	Continuous link mode. 0 A minor loop channel link made to itself goes through channel arbitration before being activated again. 1 A minor loop channel link made to itself does not go through channel arbitration before being activated again. Upon minor loop completion, the channel is active again if that channel has a minor loop channel link enabled and the link channel is itself. This effectively applies the minor loop offsets and restarts the next minor loop.
26 HALT	Halt DMA operations. 0 Normal operation. 1 Stall the start of any new channels. Executing channels are allowed to complete. Channel execution resumes when the HALT bit is cleared.
27 HOE	Halt on error. 0 Normal operation. 1 Any error causes the HALT bit to be set. Subsequently, all service requests are ignored until the HALT bit is cleared.
28 ERGA	Enable round-robin group arbitration. 0 Fixed-priority arbitration is used for selection among the groups. 1 Round-robin arbitration is used for selection among the groups.
29 ERCA	Enable Round-Robin Channel Arbitration. 0 Fixed-priority arbitration is used for channel selection within each group. 1 Round-robin arbitration is used for channel selection within each group.
30 EDBG	Enable Debug. 0 The assertion of the system debug control input is ignored. 1 The assertion of the system debug control input causes the eDMA to stall the start of a new channel. Executing channels are allowed to complete. Channel execution resumes when either the system debug control input is negated or the EDBG bit is cleared.
31	Reserved

17.3.2.2 eDMA Error Status Register (EDMA_*x*_ESR)

The EDMA_*x*_ESR provides information about the last recorded channel error. Channel errors can be caused by a configuration error (an illegal setting in the transfer control descriptor or an illegal priority register setting in fixed-arbitration mode) or an error termination to a bus master read or write cycle.

A configuration error is caused when the starting source or destination address, source or destination offsets, minor loop byte count, and the transfer size represent an inconsistent state. The addresses and offsets must be aligned on 0-modulo-transfer_size boundaries, and the minor loop byte count must be a multiple of the source and destination transfer sizes. All source reads and destination writes must be configured to the natural boundary of the programmed transfer size respectively.

In fixed-arbitration mode, a configuration error is generated when any two channel priority levels are equal and any channel is activated. The ERRCHN field is undefined for this type of error. All channel priority levels must be unique before any service requests are made.

If a scatter-gather operation is enabled on channel completion, a configuration error is reported if the scatter-gather address (DLAST_SGA) is not aligned on a 32-byte boundary. If minor loop channel linking is enabled on channel completion, a configuration error is reported when the link is attempted if the

EDMA_*x*_TCD.CITER.E_LINK bit is not equal to the EDMA_*x*_TCD.BITER.E_LINK bit. All configuration error conditions except scatter-gather and minor loop link error are reported as the channel is activated and assert an error interrupt request if enabled. When properly enabled, a scatter-gather configuration error is reported when the scatter-gather operation begins at major loop completion. A minor loop channel link configuration error is reported when the link operation is serviced at minor loop completion.

If a system bus read or write is terminated with an error, the data transfer is immediately stopped and the appropriate bus error flag is set. In this case, the state of the channel's transfer control descriptor is updated by the DMA engine with the current source address, destination address, and minor loop byte count at the point of the fault. If a bus error occurs on the last read prior to beginning the write sequence, the write is executed using the data captured during the bus error. If a bus error occurs on the last write prior to switching to the next read sequence, the read sequence is executed before the channel is terminated due to the destination bus error.

A transfer may be cancelled by software via the EDMA_*x*_MCR[CX] bit. When a cancel transfer request is recognized, the eDMA engine stops processing the channel. The current read-write sequence is allowed to finish. If the cancel occurs on the last read-write sequence of a major or minor loop, the cancel request is discarded and the channel retires normally.

The error cancel transfer is the same as a cancel transfer except the DMAES register is updated with the cancelled channel number and error cancel bit is set. The TCD of a cancelled channel has the source address and destination address of the last transfer saved in the TCD. It is the responsibility of the user to initialize the TCD again should the channel need to be restarted because the aforementioned fields have been modified by the eDMA engine and no longer represent the original parameters. When a transfer is cancelled via the error cancel transfer mechanism (setting the EDMA_*x*_MCR[ECX]), the channel number is loaded into the EDMA_*x*_ESR[ERRCHN] field and the EDMA_*x*_ESR[ECX] and EDMA_*x*_ESR[VLD] bits are set. In addition, an error interrupt may be generated if enabled. Refer to Section 17.3.2.14, "eDMA Error Registers (EDMA_x_ERH, EDMA_x_ERL)".

The occurrence of any type of error causes the DMA engine to stop the active channel and the appropriate channel bit in the eDMA error register to be asserted. At the same time, the details of the error condition are loaded into the EDMA_*x*_ESR. The major loop complete indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are not affected when an error is detected. After the error status has been updated, the DMA engine continues to operate by servicing the next appropriate channel. A channel that experiences an error condition is not automatically disabled. If a channel is terminated by an error and then issues another service request before the error is fixed, that channel will execute and terminate with the same error condition.

Offset: EDMA_*x*_BASE + 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VLD	0	0	0	0	0	0	0	0	0	0	0	0	0	0	ECX
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	GPE	CPE	ERRCHN						SAE	SOE	DAE	DOE	NCE	SGE	SBE	DBE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-4. eDMA Error Status Register (EDMA_*x*_ESR)

Table 17-3. EDMA_*x*_ESR Field Descriptions

Field	Description
0 VLD	Valid Bit. Logical OR of all EDMA_*x*_ERL status bits. 0 No EDMA_*x*_ER bits are set. 1 At least one EDMA_*x*_ER bit is set indicating a valid error exists that has not been cleared.
1–14	Reserved
15 ECX	Transfer canceled. 0 No canceled transfers. 1 The last recorded entry was a canceled transfer via the error cancel transfer input.
16 GPE	Group-priority error. 0 No group-priority error. 1 The last recorded error was a configuration error among the group priorities indicating not all group priorities are unique.
17 CPE	Channel-Priority Error. 0 No channel-priority error. 1 The last recorded error was a configuration error in the channel priorities within a group, indicating not all channel priorities within a group are unique.
18–23 ERRCHN	Error Channel Number or Canceled Channel Number. Channel number of the last recorded error (excluding GPE and CPE errors) or last recorded transfer that was error cancelled. **Note:** Do not rely on the number in the ERRCHN field group for channel-priority errors. Group- and Channel-priority errors must be resolved by inspection. The application code must interrogate the priority registers to find groups or channels with duplicate priority level.
24 SAE	Source Address Error. 0 No source address configuration error. 1 The last recorded error was a configuration error detected in the EDMA_*x*_TCD.SADDR field, indicating EDMA_*x*_TCD.SADDR is inconsistent with EDMA_*x*_TCD.SSIZE.
25 SOE	Source Offset Error. 0 No source offset configuration error. 1 The last recorded error was a configuration error detected in the EDMA_*x*_TCD.SOFF field, indicating EDMA_*x*_TCD.SOFF is inconsistent with EDMA_*x*_TCD.SSIZE.

Table 17-3. EDMA_*x*_ESR Field Descriptions (continued)

Field	Description
26 DAE	Destination Address Error. 0 No destination address configuration error. 1 The last recorded error was a configuration error detected in the EDMA_*x*_TCD.DADDR field, indicating EDMA_*x*_TCD.DADDR is inconsistent with EDMA_*x*_TCD.DSIZE.
27 DOE	Destination Offset Error. 0 No destination offset configuration error. 1 The last recorded error was a configuration error detected in the EDMA_*x*_TCD.DOFF field, indicating EDMA_*x*_TCD.DOFF is inconsistent with EDMA_*x*_TCD.DSIZE.
28 NCE	NBYTES/CITER Configuration Error. 0 No NBYTES/CITER configuration error. 1 The last recorded error was a configuration error detected in the EDMA_*x*_TCD.NBYTES or EDMA_*x*_TCD.CITER fields, indicating the following conditions exist: • EDMA_*x*_TCD.NBYTES is not a multiple of EDMA_*x*_TCD.SSIZE and EDMA_*x*_TCD.DSIZE, or • EDMA_*x*_TCD.CITER is equal to zero, or • EDMA_*x*_TCD.CITER.E_LINK is not equal to EDMA_*x*_TCD.BITER.E_LINK.
29 SGE	Scatter-Gather Configuration Error. 0 No scatter-gather configuration error. 1 The last recorded error was a configuration error detected in the EDMA_*x*_TCD.DLAST_SGA field, indicating EDMA_*x*_TCD.DLAST_SGA is not on a 32-byte boundary. This field is checked at the beginning of a scatter-gather operation after major loop completion if EDMA_*x*_TCD.E_SG is enabled.
30 SBE	Source Bus Error. 0 No source bus error. 1 The last recorded error was a bus error on a source read.
31 DBE	Destination Bus Error. 0 No destination bus error. 1 The last recorded error was a bus error on a destination write.

17.3.2.3 eDMA Enable Request Registers (EDMA_*x*_ERQRH, EDMA_*x*_ERQRL)

The EDMA_*x*_ERQRH and EDMA_*x*_ERQRL provide a bit map for the 64 channels to enable the request signal for each channel. EDMA_*x*_ERQRH supports channels 63–32, while EDMA_*x*_ERQRL covers channels 31–0.

The state of any given channel enable is directly affected by writes to these registers; the state is also affected by writes to the EDMA_*x*_SERQR and EDMA_*x*_CERQR. The EDMA_A*x*_CERQR and EDMA_*x*_SERQR are provided so that the request enable for a single channel can be modified without performing a read-modify-write sequence to the EDMA_*x*_ERQRH and EDMA_*x*_ERQRL.

Both the eDMA request input signal and this enable request flag must be asserted before a channel’s hardware service request is accepted. The state of the eDMA enable request flag does not effect a channel service request made through software or a linked channel request.

Address: EDMA_*x*_BASE + 0x0008 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ERQ 63	ERQ 62	ERQ 61	ERQ 60	ERQ 59	ERQ 58	ERQ 57	ERQ 56	ERQ 55	ERQ 54	ERQ 53	ERQ 52	ERQ 51	ERQ 50	ERQ 49	ERQ 48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ERQ 47	ERQ 46	ERQ 45	ERQ 44	ERQ 43	ERQ 42	ERQ 41	ERQ 40	ERQ 39	ERQ 38	ERQ 37	ERQ 36	ERQ 35	ERQ 34	ERQ 33	ERQ 32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-5. eDMA Enable Request High Register (EDMA_*x*_ERQRH)

Offset: EDMA_*x*_BASE + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ERQ 31	ERQ 30	ERQ 29	ERQ 28	ERQ 27	ERQ 26	ERQ 25	ERQ 24	ERQ 23	ERQ 22	ERQ 21	ERQ 20	ERQ 19	ERQ 18	ERQ 17	ERQ 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	19	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ERQ 15	ERQ 14	ERQ 13	ERQ 12	ERQ 11	ERQ 10	ERQ 09	ERQ 08	ERQ 07	ERQ 06	ERQ 05	ERQ 04	ERQ 03	ERQ 02	ERQ 01	ERQ 00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-6. eDMA Enable Request Register (EDMA_*x*_ERQRL)

Table 17-4. EDMA_*x*_ERQRL Field Descriptions

Field	Description
0–31 ERQ*n*	Enable eDMA Hardware Service Request *n.* 0 The eDMA request signal for channel *n* is disabled. 1 The eDMA request signal for channel *n* is enabled.

As a given channel completes processing its major iteration count, there is a flag in the transfer control descriptor that may affect the ending state of the EDMA_*x*_ERQR bit for that channel. If the EDMA_*x*_TCD.D_REQ bit is set, then the corresponding EDMA_*x*_ERQR bit is cleared after the major loop is complete, disabling the eDMA hardware request. Otherwise if the D_REQ bit is cleared, the state of the EDMA_*x*_ERQR bit is unaffected.

17.3.2.4 eDMA Enable Error Interrupt Registers (EDMA_*x*_EEIRH, EDMA_*x*_EEIRL)

The EDMA_*x*_EEIRH and EDMA_*x*_EEIRL provide a bit map for the 64 channels to enable the error interrupt signal for each channel. EDMA_*x*_EEIRH supports channels 63–32, while EDMA_*x*_EEIRL supports channels 31–0.

The state of any given channel's error interrupt enable is directly affected by writes to these registers; it is also affected by writes to the EDMA_*x*_SEEIR and EDMA_*x*_CEEIR. The EDMA_*x*_SEEIR and EDMA_*x*_CEEIR are provided so that the error interrupt enable for a single channel can be modified without the performing a read-modify-write sequence to the EDMA_*x*_EEIRH and EDMA_*x*_EEIRL.

Both the eDMA error indicator and this error interrupt enable flag must be asserted before an error interrupt request for a given channel is asserted.

Address: EDMA_*x*_BASE + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	EEI 63	EEI 62	EEI 61	EEI 60	EEI 59	EEI 58	EEI 57	EEI 56	EEI 55	EEI 54	EEI 53	EEI 52	EEI 51	EEI 50	EEI 49	EEI 48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EEI 47	EEI 46	EEI 45	EEI 44	EEI 43	EEI 42	EEI 41	EEI 40	EEI 39	EEI 38	EEI 37	EEI 36	EEI 35	EEI 34	EEI 33	EEI 32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-7. eDMA Enable Error Interrupt High Register (EDMA_*x*_EEIRH)

Address: EDMA_*x*_BASE + 0x0014 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	EEI 31	EEI 30	EEI 29	EEI 28	EEI 27	EEI 26	EEI 25	EEI 24	EEI 23	EEI 22	EEI 21	EEI 20	EEI 19	EEI 18	EEI 17	EEI 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	EEI 15	EEI 14	EEI 13	EEI 12	EEI 11	EEI 10	EEI 09	EEI 08	EEI 07	EEI 06	EEI 05	EEI 04	EEI 03	EEI 02	EEI 01	EEI 00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-8. eDMA Enable Error Interrupt Low Register (EDMA_*x*_EEIRL)

Table 17-5. EDMA_*x*_EEIRL Field Descriptions

Field	Description
0–31 EEI*n*	Enable Error Interrupt *n.* 0 The error signal for channel *n* does not generate an error interrupt. 1 The assertion of the error signal for channel *n* generate an error interrupt request.

17.3.2.5 eDMA Set Enable Request Register (EDMA_*x*_SERQR)

The EDMA_*x*_SERQR provides a simple memory-mapped mechanism to set a given bit in the EDMA_*x*_ERQRH or EDMA_*x*_ERQRL to enable the eDMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_*x*_ERQRH or EDMA_*x*_ERQRL to be set. Setting bit 1 (SERQ[0]) provides a global set function, forcing the entire contents of EDMA_*x*_ERQRH and EDMA_*x*_ERQRL to be asserted. Reads of this register return all zeroes.

If bit 0 is set, the SERQ command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Offset: EDMA_*x*_BASE + 0x0018 Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	SERQ[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 17-9. eDMA Set Enable Request Register (EDMA_*x*_SERQR)

Table 17-6. EDMA_*x*_SERQR Field Descriptions

Field	Descriptions
0 NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 1–7.
1–7 SERQ	Set Enable Request. 0–63 Set corresponding bit in EDMA_*x*_ERQRH or EDMA_*x*_ERQRL. 64–127 Set all bits in EDMA_*x*_ERQRH and EDMA_*x*_ERQRL.

17.3.2.6 eDMA Clear Enable Request Register (EDMA_*x*_CERQR)

The EDMA_*x*_CERQR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_*x*_ERQRH or EDMA_*x*_ERQRL to disable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_*x*_ERQRH or EDMA_*x*_ERQRL to be cleared. Setting bit 1 (CERQ[0]) provides a global clear function, forcing the entire contents of EDMA_*x*_ERQRH and EDMA_*x*_ERQRL to be zeroed, disabling all eDMA request inputs. Reads of this register return all zeroes.

If bit 0 is set, the CERQ command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Offset: EDMA_*x*_BASE + 0x0019 Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	CERQ[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 17-10. eDMA Clear Enable Request Register (EDMA_*x*_CERQR)

Table 17-7. EDMA_*x*_CERQR Field Descriptions

Field	Description
0 NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 1–7.
1–7 CERQ	Clear Enable Request. 0–63 Clear corresponding bit in EDMA_*x*_ERQRH or EDMA_*x*_ERQRL. 64–127 Clear all bits in EDMA_*x*_ERQRH and EDMA_*x*_ERQRL.

17.3.2.7 eDMA Set Enable Error Interrupt Register (EDMA_*x*_SEEIR)

The EDMA_*x*_SEEIR provides a memory-mapped mechanism to set a given bit in the EDMA_*x*_EEIRH or EDMA_*x*_EEIRL to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_*x*_EEIRH or EDMA_*x*_EEIRL to be set. Setting bit 1 (SEEI[0]) provides a global set function, forcing the entire contents of EDMA_*x*_EEIRH or EDMA_*x*_EEIRL to be asserted. Reads of this register return all zeroes.

If bit 0 is set, the SEEI command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Offset: EDMA_*x*_BASE + 0x001A Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	SEEI[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 17-11. eDMA Set Enable Error Interrupt Register (EDMA_*x*_SEEIR)

Table 17-8. EDMA_*x*_SEEIR Field Descriptions

Field	Description
0 NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 1–7.
1–7 SEEI[0:6]	Set Enable Error Interrupt. 0–63 Set corresponding bit in EDMA_*x*_EIRRH or EDMA_*x*_EIRRL. 64–127 Set all bits in EDMA_*x*_EIRRH or EDMA_*x*_EEIRL.

17.3.2.8 eDMA Clear Enable Error Interrupt Register (EDMA_*x*_CEEIR)

The EDMA_*x*_CEEIR provides a memory-mapped mechanism to clear a given bit in the EDMA_*x*_EEIRH or EDMA_*x*_EEIRL to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_*x*_EEIRH or EDMA_*x*_EEIRL to be cleared. Setting bit 1 (CEEI[0]) provides a global clear function, forcing the entire contents of the EDMA_*x*_EEIRH or EDMA_*x*_EEIRL to be zeroed, disabling error interrupts for all channels. Reads of this register return all zeroes.

If bit 0 is set, the CEEI command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Offset: EDMA_*x*_BASE + 0x001B Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	CEEI[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 17-12. eDMA Clear Enable Error Interrupt Register (EDMA_*x*_CEEIR)

Table 17-9. EDMA_*x*_CEEIR Field Descriptions

Field	Description
0 NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 1-7.
1–7 CEEI	Clear Enable Error Interrupt. 0–63 Clear corresponding bit in EDMA_*x*_EEIRH or EDMA_*x*_EEIRL. 64–127 Clear all bits in EDMA_*x*_EEIRH or EDMA_*x*_EEIRL.

17.3.2.9 eDMA Clear Interrupt Request Register (EDMA_*x*_CIRQR)

The EDMA_*x*_CIRQR provides a memory-mapped mechanism to clear a given bit in the EDMA_*x*_IRQRH or EDMA_*x*_IRQRL to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the EDMA_*x*_IRQRH or EDMA_*x*_IRQRL to be cleared. Setting bit 1 (CINT[0]) provides a global clear function, forcing the entire contents of the EDMA_*x*_IRQRH or EDMA_*x*_IRQRL to be zeroed, disabling all eDMA interrupt requests. Reads of this register return all zeroes.

If bit 0 is set, the CINT command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Offset: EDMA_*x*_BASE + 0X001C Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	CINT[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 17-13. eDMA Clear Interrupt Request (EDMA_*x*_CIRQR)

Table 17-10. EDMA_*x*_CIRQR Field Descriptions

Field	Description
0 NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 1-7.
1–7 CINT	Clear Interrupt Request. 0–63 Clear corresponding bit in EDMA_*x*_IRQRH or EDMA_*x*_IRQRL. 64–127 Clear all bits in EDMA_*x*_IRQRH or EDMA_*x*_IRQRL.

17.3.2.10 eDMA Clear Error Register (EDMA_*x*_CER)

The EDMA_*x*_CER provides a memory-mapped mechanism to clear a given bit in the EDMA_*x*_ERH or EDMA_*x*_ERL to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in the EDMA_*x*_ERH or EDMA_*x*_ERL to be cleared. Setting bit 1 (CERR[0]) provides a global clear function, forcing the entire contents of the EDMA_*x*_ERH or EDMA_*x*_ERL to be zeroed, clearing all channel error indicators. Reads of this register return all zeroes.

If bit 0 is set, the CERR command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Offset: EDMA_*x*_BASE + 0x001D Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	CERR[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 17-14. eDMA Clear Error Register (EDMA_*x*_CER)

Table 17-11. EDMA_*x*_CER Field Descriptions

Field	Description
0 NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 1–7.
1–7 CERR	Clear Error Indicator. 0–63 Clear corresponding bit in EDMA_*x*_ERH or EDMA_*x*_ERL. 64–127 Clear all bits in EDMA_*x*_ERH or EDMA_*x*_ERL.

17.3.2.11 eDMA Set START Bit Register (EDMA_*x*_SSBR)

The EDMA_*x*_SSBR provides a memory-mapped mechanism to set the START bit in the TCD of the given channel. The data value on a register write causes the START bit in the corresponding transfer control descriptor to be set. Setting bit 1 (SSB[0]) provides a global set function, forcing all START bits to be set. Reads of this register return all zeroes.

If bit 0 is set, the SSB command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Offset: EDMA_*x*_BASE + 0x001E Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	SSB[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 17-15. eDMA Set START Bit Register (EDMA_*x*_SSBR)

Table 17-12. EDMA_*x*_SSBR Field Descriptions

Field	Description
0 NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 1–7.
1–7 SSB	Set START Bit (channel service request). 0–63 Set the corresponding channel’s TCD START bit. 64–127 Set all TCD START bits.

17.3.2.12 eDMA Clear DONE Status Bit Register (EDMA_*x*_CDSBR)

The EDMA_*x*_CDSBR provides a memory-mapped mechanism to clear the DONE bit in the TCD of the given channel. The data value on a register write causes the DONE bit in the corresponding transfer control descriptor to be cleared. Setting bit 1 (CDSB[0]) provides a global clear function, forcing all DONE bits to be cleared.

If bit 0 is set, the CDSB command is ignored. This allows multiple byte registers to be written as a 32-bit word. Reads of this register return all zeroes.

Offset: EDMA_*x*_BASE + 0x001F Access: User write-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	0	0
W	NOP	CDSB[0:6]						
Reset	0	0	0	0	0	0	0	0

Figure 17-16. eDMA Clear DONE Status Bit Register (EDMA_*x*_CDSBR)

Table 17-13. EDMA_*x*_CDSBR Field Descriptions

Field	Description
0 NOP	No operation. 0 Normal operation. 1 No operation, ignore bits 1–7.
1–7 CDSB	Clear DONE Status Bit. 0–63 Clear the corresponding channel’s DONE bit. 64–127 Clear all TCD DONE bits.

17.3.2.13 eDMA Interrupt Request Registers (EDMA_*x*_IRQRH, EDMA_*x*_IRQRL)

The EDMA_*x*_IRQRH and EDMA_*x*_IRQRL provide a bit map for the 64 channels signaling the presence of an interrupt request for each channel. EDMA_*x*_IRQRH maps to channels 63–32 and EDMA_*x*_IRQRL maps to channels 31–0.

The DMA engine signals the occurrence of a programmed interrupt on the completion of a data transfer as defined in the transfer control descriptor by setting the appropriate bit in this register. The outputs of this register are directly routed to the interrupt controller (INTC). During the execution of the interrupt service routine associated with any given channel, software must clear the appropriate bit, negating the interrupt request. Typically, a write to the EDMA_*x*_CIRQR in the interrupt service routine is used for this purpose.

The state of any given channel's interrupt request is directly affected by writes to this register; it is also affected by writes to the EDMA_*x*_CIRQR. On writes to the EDMA_*x*_IRQRH or EDMA_*x*_IRQRL, a 1 in any bit position clears the corresponding channel's interrupt request. A 0 in any bit position has no effect on the corresponding channel's current interrupt status. The EDMA_*x*_CIRQR is provided so the interrupt request for a single channel can be cleared.

Address: EDMA_*x*_BASE + 0x0020 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	INT 63	INT 62	INT 61	INT 60	INT 59	INT 58	INT 57	INT 56	INT 55	INT 54	INT 53	INT 52	INT 51	INT 50	INT 49	INT 48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	INT 47	INT 46	INT 45	INT 44	INT 43	INT 42	INT 41	INT 40	INT 39	INT 38	INT 37	INT 36	INT 35	INT 34	INT 33	INT 32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-17. eDMA Interrupt Request High Register (EDMA_*x*_IRQRH)

Address: EDMA_*x*_BASE + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	INT 31	INT 30	INT 29	INT 28	INT 27	INT 26	INT 25	INT 24	INT 23	INT 22	INT 21	INT 20	INT 19	INT 18	INT 17	INT 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	19	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	INT 15	INT 14	INT 13	INT 12	INT 11	INT 10	INT 09	INT 08	INT 07	INT 06	INT 05	INT 04	INT 03	INT 02	INT 01	INT 00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-18. eDMA Interrupt Request Register (EDMA_*x*_IRQRL)

Table 17-14. EDMA_*x*_IRQRL Field Descriptions

Field	Description
0–31 INT*n*	eDMA Interrupt Request *n.* 0 The interrupt request for channel *n* is cleared. 1 The interrupt request for channel *n* is active.

17.3.2.14 eDMA Error Registers (EDMA_*x*_ERH, EDMA_*x*_ERL)

The EDMA_*x*_ERH and EDMA_*x*_ERL provides a bit map for the 64 channels signaling the presence of an error for each channel. EDMA_*x*_ERH supports channels 63–32 and EDMA_*x*_ERL maps to channels 31-0.

The DMA engine signals the occurrence of a error condition by setting the appropriate bit in this register. The outputs of this register are enabled by the contents of the EDMA_*x*_EEIR, then logically summed across 64 channels to form an error interrupt request, which is then routed to the interrupt controller. During the execution of the interrupt service routine associated with any eDMA errors, it is software's responsibility to clear the appropriate bit, negating the error interrupt request. Typically, a write to the EDMA_*x*_CER in the interrupt service routine is used for this purpose. The normal eDMA channel completion indicators, setting the transfer control descriptor DONE flag and the possible assertion of an interrupt request, are not affected when an error is detected.

The contents of this register can also be polled and a non-zero value indicates the presence of a channel error, regardless of the state of the EDMA_*x*_EEIR. The EDMA_*x*_ESR[VLD] bit is a logical OR of all bits in this register and it provides a single bit indication of any errors. The state of any given channel's error indicators is affected by writes to this register; it is also affected by writes to the EDMA_*x*_CER. On writes to EDMA_*x*_ERH or EDMA_*x*_ERL, a 1 in any bit position clears the corresponding channel's error status. A 0 in any bit position has no effect on the corresponding channel's current error status. The EDMA_*x*_CER is provided so the error indicator for a single channel can be cleared.

Address: EDMA_*x*_BASE + 0x0028 Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ERR 63	ERR 62	ERR 61	ERR 60	ERR 59	ERR 58	ERR 57	ERR 56	ERR 55	ERR 54	ERR 53	ERR 52	ERR 51	ERR 50	ERR 49	ERR 48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ERR 47	ERR 46	ERR 45	ERR 44	ERR 43	ERR 42	ERR 41	ERR 40	ERR 39	ERR 38	ERR 37	ERR 36	ERR 35	ERR 34	ERR 33	ERR 32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-19. eDMA Error High Register (EDMA_*x*_ERH)

Address: EDMA_*x*_BASE + 0x002C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	ERR 31	ERR 30	ERR 29	ERR 28	ERR 27	ERR 26	ERR 25	ERR 24	ERR 23	ERR 22	ERR 21	ERR 20	ERR 19	ERR 18	ERR 17	ERR 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	ERR 15	ERR 14	ERR 13	ERR 12	ERR 11	ERR 10	ERR 09	ERR 08	ERR 07	ERR 06	ERR 05	ERR 04	ERR 03	ERR 02	ERR 01	ERR 00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-20. eDMA Error Register (EDMA_*x*_ERL)

Table 17-15. EDMA_*x*_ERL Field Descriptions

Field	Description
0–31 ERR*n*	eDMA Error *n*. 0 An error in channel *n* has not occurred. 1 An error in channel *n* has occurred.

17.3.2.15 DMA Hardware Request Status (EDMA_*x*_HRSH, EDMA_*x*_HRSL)

The EDMA_*x*_HRSH and EDMA_*x*_HRSL registers provide a bit map to show the current hardware request status for each channel. EDMA_*x*_HRSH maps to channels 64–32 and EDMA_*x*_HRSL maps to channels 31-00.

Address: EDMA_*x*_BASE + 0x0030

Access: User R/W

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	HRS 63	HRS 62	HRS 61	HRS 60	HRS 59	HRS 58	HRS 57	HRS 56	HRS 55	HRS 54	HRS 53	HRS 52	HRS 51	HRS 50	HRS 49	HRS 48
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	HRS 47	HRS 46	HRS 45	HRS 44	HRS 43	HRS 42	HRS 41	HRS 40	HRS 39	HRS 38	HRS 37	HRS 36	HRS 35	HRS 34	HRS 33	HRS 32
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-21. EDMA Hardware Request Status Register High (EDMA_*x*_HRSH)

Address: EDMA_*x*_BASE + 0x0034 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	HRS 31	HRS 30	HRS 29	HRS 28	HRS 27	HRS 26	HRS 25	HRS 24	HRS 23	HRS 22	HRS 21	HRS 20	HRS 19	HRS 18	HRS 17	HRS 16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	HRS 15	HRS 14	HRS 13	HRS 12	HRS 11	HRS 10	HRS 09	HRS 08	HRS 07	HRS 06	HRS 05	HRS 04	HRS 03	HRS 02	HRS 01	HRS 00
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17-22. EDMA Hardware Request Status Register Low (EDMA_*x*_HRSL)

Table 17-16. EDMA_*x*_HRSL Field Descriptions

Field	Description
0–31 HRS*n*	DMA Hardware Request Status 0 A hardware service request for channel *n* is not present. 1 A hardware service request for channel *n* is present. **Note:** The hardware request status reflects the state of the request as seen by the arbitration logic. Therefore, this status is affected by the EDMA_*x*_ERQRL[ERQ*n*] bit.

17.3.2.16 eDMA Global Write Registers (EDMA_*x*_GWRH and EDMA_*x*_GWRL)

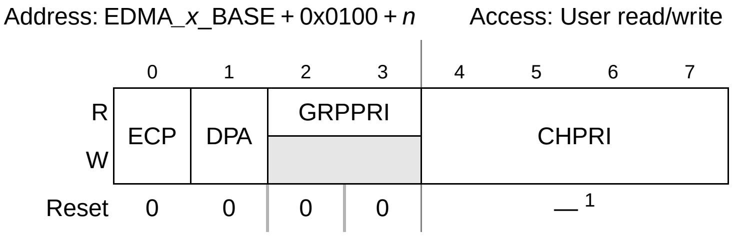
The EDMA_x_GWRH and EDMA_x_GWRL registers provide coherency controls to the Cache Coherency Unit (CCU). The EDMA_x_GWRH and EDMA_x_GWRL registers provide a bit map to enable the CCU to snoop data writes from any enabled channel. When the Global Write Enable (GWEN) bit for a channel is set, the eDMA will signal the CCU whenever that channel performs a write. The EDMA_x_GWRH and EDMA_x_GWRL registers perform no functions within the eDMA.

17.3.2.17 eDMA Channel *n* Priority Registers (EDMA_*x*_CPR*n*)

When the fixed-priority channel arbitration mode is enabled (EDMA_*x*_MCR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel. The channel priorities are evaluated by numeric value; that is, 0 is the lowest priority, 1 is the next higher priority, then 2, 3, etc. If software modifies channel priority values, then the software must ensure that the channel priorities contain unique values. Otherwise, a configuration error is reported. The range of the priority value is limited to the values of 0 through 15. When read, the GRPPRI bits of the EDMA_*x*_CPR*n* register reflect the current priority level of the group of channels in which the corresponding channel resides. GRPPRI bits are not affected by writes to the EDMA_*x*_CPR*n* registers. The group priority is assigned in the EDMA_*x*_MCR. See Figure 17-2 and Table 17-2 for the EDMA_*x*_MCR definition.

Channel preemption is enabled on a per-channel basis by setting the ECP bit in the EDMA_*x*_CPR*n* register. Channel preemption allows the executing channel's data transfers to be temporarily suspended in favor of starting a higher priority channel. After the preempting channel has completed all its minor loop data transfers, the preempted channel is restored and resumes execution. After the restored channel completes one read/write sequence, it is again eligible for preemption. If any higher priority channel requests service, the restored channel is suspended and the higher priority channel is serviced. Nested preemption (attempting to preempt a preempting channel) is not supported. After a preempting channel begins execution, it cannot be preempted. Preemption is available only when fixed arbitration is selected for both group and channel arbitration modes.

A channel's ability to pre-empt another channel can be disabled by setting EDMA_*x*_CPR[DPA]. When a channel's pre-empt ability is disabled, that channel cannot suspend a lower priority channel's data transfer; regardless of the lower priority channel's ECP setting. This allows for a pool of low priority, large data moving channels to be defined. These low priority channels can be configured to not preempt each other, thus preventing a low priority channel from consuming the preempt slot normally available a true, high priority channel.



[1] The reset value for the channel priority field, CHPRI[0–3], is equal to the corresponding channel number for each priority register for example, EDMA_*x*_CPRI0[CHPRI] = 0b0000 and EDMA_*x*_CPR15[CHPRI] = 0b1111.

Figure 17-23. eDMA Channel *n* Priority Register (EDMA_*x*_CPR*n*)

Table 17-17. EDMA_*x*_CPRn Field Descriptions

Field	Description
0 ECP	Enable Channel Preemption. 0 Channel *n* cannot be suspended by a higher priority channel's service request. 1 Channel *n* can be temporarily suspended by the service request of a higher priority channel.
1 DPA	Disable preempt ability. 0 Channel n can suspend a lower priority channel. 1 Channel n cannot suspend any channel, regardless of channel priority.
2–3 GRPPRI	Channel *n* current group priority. Group priority assigned to this channel group when fixed-priority arbitration is enabled. These two bits are read-only; writes are ignored. The reset value for the group priority fields, is equal to the corresponding channel number for each priority register; that is, EDMA_*x*_CPR31[GRPPRI] = 0b01.
4–7 CHPRI	Channel *n* Arbitration Priority. Channel priority when fixed-priority arbitration is enabled. The reset value for the channel priority fields CHPRI[0–3], is equal to the corresponding channel number for each priority register; for example, EDMA_*x*_CPR31[CHPRI] = 0b1111.

17.3.2.18 Transfer Control Descriptor (TCD)

Each channel requires a 32-byte transfer control descriptor for defining the desired data movement operation. The channel descriptors are stored in the local memory in sequential order: channel 0, channel

1,... channel 63. The definitions of the TCD are presented as eight 32-bit values. Table 17-18 is a field list of the basic TCD structure.

Table 17-18. TCD*n* 32-bit Memory Structure

eDMA Offset	TCD*n* Field	
0x1000+(32 x *n*)+0x0000	Source address (saddr)	
0x1000+(32 x *n*)+0x0004	Transfer attributes	Signed source address offset (soff)
0x1000+(32 x *n*)+0x0008	MLOFF-Minor Loop Offset	NBYTES-Minor Byte Count
0x1000+(32 x *n*)+0x000C	Last source address adjustment (slast)	
0x1000+(32 x *n*)+0x0010	Destination address (daddr)	
0x1000+(32 x *n*)+0x0014	Current major iteration count (citer)	Signed destination address offset (doff)
0x1000 (32 x *n*) 0x0018	Last destination address adjustment / scatter-gather address (dlast_sga)	
0x1000+(32 x *n*)+0x001c	Beginning major iteration count (biter)	Channel control/status

Figure 17-24 and Table 17-19 define the fields of the TCD*n* structure.

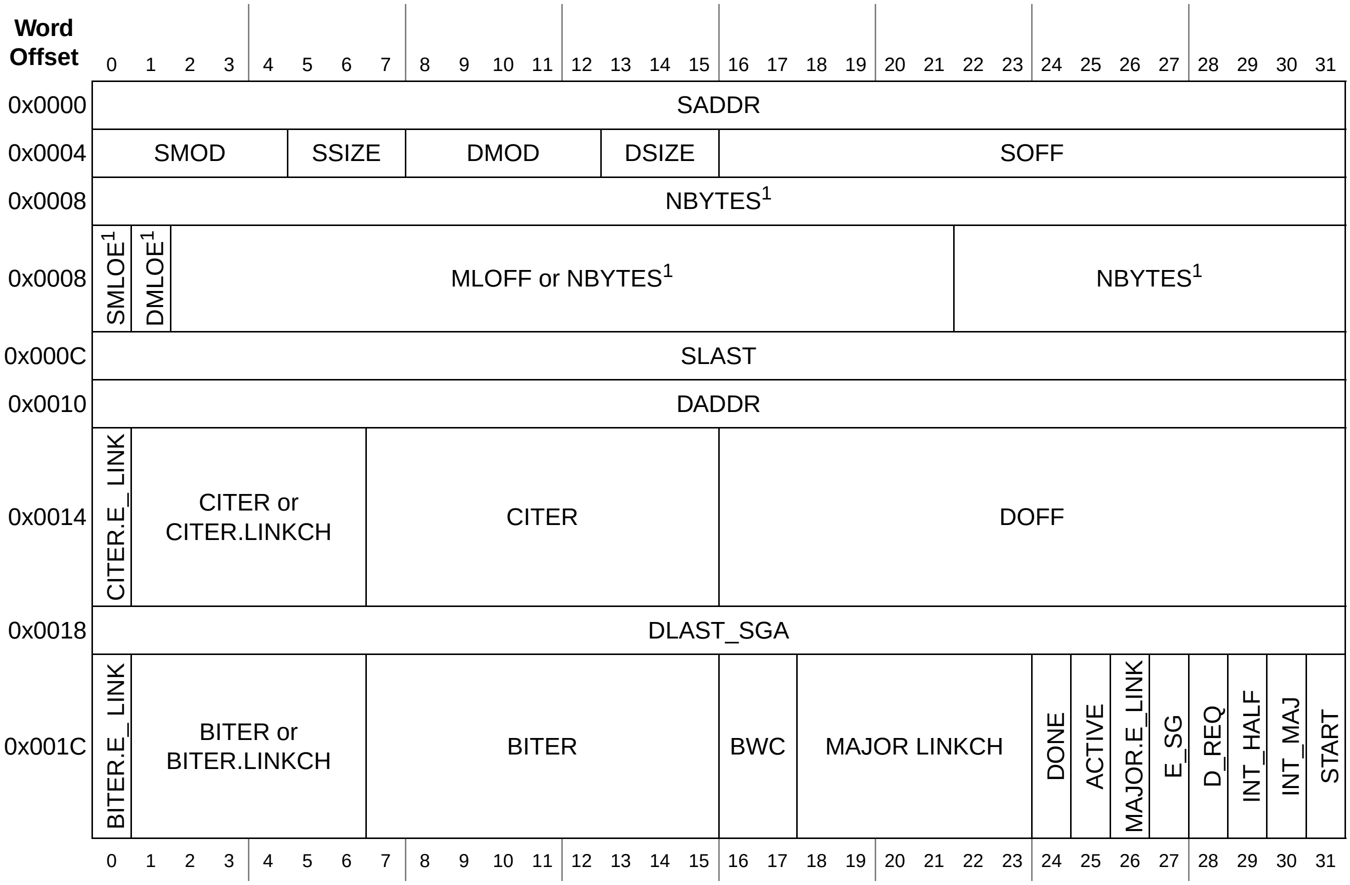


Figure 17-24. TCD Structure

[1] The fields implemented in Word 2 depend on whether EDMA_*x*_MCR(EMLM) is set to 0 or 1. Refer to Table 17-2.

NOTE

The TCD structures for the eDMA channels shown in Figure 17-24 are implemented in internal SRAM. These structures are not initialized at reset; therefore, all channel TCD parameters must be initialized by the application code before activating that channel.

Table 17-19. TCDn Field Descriptions

Field	Description
0x0 [0:31] SADDR	Source address. Memory address pointing to the source data. Word 0x0, bits 0–31.
0x4 [0:4] SMOD	Source address modulo. 0 Source address modulo feature is disabled. non-0 This value defines a specific address range that is specified to be the value after SADDR + SOFF calculation is performed or the original register value. The setting of this field provides the ability to easily implement a circular data queue. For data queues requiring power-of-2 size bytes, the queue should start at a 0-modulo-size address and the SMOD field should be set to the appropriate value for the queue, freezing the desired number of upper address bits. The value programmed into this field specifies the number of lower address bits that are allowed to change. For this circular queue application, the SOFF is typically set to the transfer size to implement post-increment addressing with the SMOD function constraining the addresses to a 0-modulo-size range.
0x4 [5:7] SSIZE	Source data transfer size. 000 8-bit 001 16-bit 010 32-bit 011 64-bit 100 Reserved 101 32-byte (64-bit, 4 beat, WRAP4 burst) 110 Reserved 111 Reserved The attempted specification of a reserved encoding causes a configuration error.
0x4 [8:12] DMOD	Destination address modulo. See the SMOD[0:5] definition.
0x4 [13:15] DSIZE	Destination data transfer size. See the SSIZE[0:2] definition.
0x4 [16:31] SOFF	Source address signed offset. Sign-extended offset applied to the current source address to form the next-state value as each source read is completed.
0x8 [0] SMLOE [1]	Source minor loop offset enable This flag selects whether the minor loop offset is applied to the source address upon minor loop completion. 0 The minor loop offset is not applied to the saddr. 1 The minor loop offset is applied to the saddr.

Table 17-19. TCDn Field Descriptions (continued)

Field	Description
0x8 [1] DMLOE [1]	Destination minor loop offset enable This flag selects whether the minor loop offset is applied to the destination address upon minor loop completion. 0 The minor loop offset is not applied to the daddr. 1 The minor loop offset is applied to the daddr.
0x8 [2-21] MLOFF or NBYTES [1]	Inner “minor” byte transfer count or Minor loop offset If both SMLOE and DMLOE are cleared, this field is part of the byte transfer count. If either SMLOE or DMLOE are set, this field represents a sign-extended offset applied to the source or destination address to form the next-state value after the minor loop is completed.
0x8 [22:31] NBYTES [1]	Inner “minor” byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel is activated, the contents of the appropriate TCD is loaded into the eDMA engine, and the appropriate reads and writes performed until the complete byte transfer count has been transferred. This is an indivisible operation and cannot be stalled or halted. Once the minor count is exhausted, the current values of the SADDR and DADDR are written back into the local memory, the major iteration count is decremented and restored to the local memory. If the major iteration count is completed, additional processing is performed. **Note:** The NBYTES value of 0x0000_0000 is interpreted as 0x1_0000_0000, thus specifying a 4 GB transfer.
0xC [0:31] SLAST	Last source address adjustment. Adjustment value added to the source address at the completion of the outer major iteration count. This value can be applied to “restore” the source address to the initial value, or adjust the address to reference the next data structure.
0x10 [0:31] DADDR	Destination address. Memory address pointing to the destination data.
0x14 [0] CITER.E_LINK	Enable channel-to-channel linking on minor loop completion. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by CITER.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the EDMA_*x*_TCD.START bit of the specified channel. If channel linking is disabled, the CITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled. **Note:** This bit must be equal to the BITER.E_LINK bit. Otherwise, a configuration error is reported.
0x14 [1:6] CITER or CITER.LINKCH	Current major iteration count or link channel number. If channel-to-channel linking is disabled (EDMA_*x*_TCD.CITER.E_LINK = 0), then • No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [161:175] are used to form a 15-bit CITER field. Otherwise, • After the minor loop is exhausted, the DMA engine initiates a channel service request at the channel defined by CITER.LINKCH[0:5] by setting that channel’s EDMA_*x*_TCD.START bit.

Table 17-19. TCDn Field Descriptions (continued)

Field	Description
0x14 [7:15] CITER	Current major iteration count. This 9 or 15-bit count represents the current major loop count for the channel. It is decremented each time the minor loop is completed and updated in the transfer control descriptor memory. After the major iteration count is exhausted, the channel performs a number of operations (for example, final source and destination address calculations), optionally generating an interrupt to signal channel completion before reloading the CITER field from the beginning iteration count (BITER) field. **Note:** When the CITER field is initially loaded by software, it must be set to the same value as that contained in the BITER field. **Note:** If the channel is configured to execute a single service request, the initial values of BITER and CITER should be 0x0001.
0x14 [16:31] DOFF	Destination address signed Offset. Sign-extended offset applied to the current destination address to form the next-state value as each destination write is completed.
0x18 [0:31] DLAST_SGA	Last destination address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter-gather). If scatter-gather processing for the channel is disabled (EDMA_*x*_TCD.E_SG = 0) then • Adjustment value added to the destination address at the completion of the outer major iteration count. This value can be applied to restore the destination address to the initial value, or adjust the address to reference the next data structure. Otherwise, • This address points to the beginning of a 0-modulo-32 byte region containing the next transfer control descriptor to be loaded into this channel. This channel reload is performed as the major iteration count completes. The scatter-gather address must be 0-modulo-32 byte, otherwise a configuration error is reported.
0x1C [0] BITER.E_LINK	Enables channel-to-channel linking on minor loop complete. As the channel completes the inner minor loop, this flag enables the linking to another channel, defined by BITER.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the EDMA_*x*_TCD.START bit of the specified channel. If channel linking is disabled, the BITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR.E_LINK channel linking. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled. **Note:** When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field. Otherwise, a configuration error is reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.
0x1C [1:6] BITER or BITER.LINKCH	Starting major iteration count or link channel number. If channel-to-channel linking is disabled (EDMA_*x*_TCD.BITER.E_LINK = 0), then • No channel-to-channel linking (or chaining) is performed after the inner minor loop is exhausted. TCD bits [225:239] are used to form a 15-bit BITER field. Otherwise, • After the minor loop is exhausted, the DMA engine initiates a channel service request at the channel, defined by BITER.LINKCH[0:5], by setting that channel's EDMA_*x*_TCD.START bit. **Note:** When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field. Otherwise, a configuration error is reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.

Table 17-19. TCDn Field Descriptions (continued)

Field	Description
0x1C [7:15] BITER	Starting major iteration count. As the transfer control descriptor is first loaded by software, this field must be equal to the value in the CITER field. As the major iteration count is exhausted, the contents of this field are reloaded into the CITER field. **Note:** If the channel is configured to execute a single service request, the initial values of BITER and CITER should be 0x0001.
0x1C [16:17] BWC	Bandwidth control. This two-bit field provides a mechanism to effectively throttle the amount of bus bandwidth consumed by the eDMA. In general, as the eDMA processes the inner minor loop, it continuously generates read/write sequences until the minor count is exhausted. This field forces the eDMA to stall after the completion of each read/write access to control the bus request bandwidth seen by the system bus crossbar switch (XBAR). 00 No DMA engine stalls 01 Reserved 10 DMA engine stalls for 4 cycles after each r/w 11 DMA engine stalls for 8 cycles after each r/w
0x1C [18:23] MAJOR.LINKCH	Link channel number. If channel-to-channel linking on major loop complete is disabled (EDMA_*x*_TCD.MAJOR.E_LINK = 0) then, • No channel-to-channel linking (or chaining) is performed after the outer major loop counter is exhausted. Otherwise • After the major loop counter is exhausted, the DMA engine initiates a channel service request at the channel defined by MAJOR.LINKCH[0:5] by setting that channel's EDMA_*x*_TCD.START bit.
0x1C [24] DONE	Channel done. This flag indicates the eDMA has completed the outer major loop. It is set by the DMA engine as the CITER count reaches zero; it is cleared by software or hardware when the channel is activated (when the DMA engine has begun processing the channel, not when the first data transfer occurs). **Note:** This bit must be cleared to write the MAJOR.E_LINK or E_SG bits.
0x1C [25] ACTIVE	Channel active. This flag signals the channel is currently in execution. It is set when channel service begins, and is cleared by the DMA engine as the inner minor loop completes or if any error condition is detected.
0x1C [26] MAJOR.E_LINK	Enable channel-to-channel linking on major loop completion. As the channel completes the outer major loop, this flag enables the linking to another channel, defined by MAJOR.LINKCH[0:5]. The link target channel initiates a channel service request via an internal mechanism that sets the EDMA_*x*_TCD.START bit of the specified channel. **Note:** To support the dynamic linking coherency model, this field is forced to zero when written to while the EDMA_*x*_TCD.DONE bit is set. 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
0x1C [27] E_SG	Enable scatter-gather processing. As the channel completes the outer major loop, this flag enables scatter-gather processing in the current channel. If enabled, the DMA engine uses DLAST_SGA as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure which is loaded as the transfer control descriptor into the local memory. **Note:** To support the dynamic scatter-gather coherency model, this field is forced to zero when written to while the EDMA_*x*_TCD.DONE bit is set. 0 The current channel's TCD is normal format. 1 The current channel's TCD specifies a scatter gather format. The DLAST_SGA field provides a memory pointer to the next TCD to be loaded into this channel after the outer major loop completes its execution.

Table 17-19. TCDn Field Descriptions (continued)

Field	Description
0x1C [28] D_REQ	Disable hardware request. If this flag is set, the eDMA hardware automatically clears the corresponding EDMA_*x*_ERQH or EDMA_*x*_ERQL bit when the current major iteration count reaches zero. 0 The channel's EDMA_*x*_ERQH or EDMA_*x*_ERQL bit is not affected. 1 The channel's EDMA_*x*_ERQH or EDMA_*x*_ERQL bit is cleared when the outer major loop is complete.
0x1C [29] INT_HALF	Enable an interrupt when major counter is half complete. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the EDMA_*x*_ERQH or EDMA_*x*_ERQL when the current major iteration count reaches the halfway point. Specifically, the comparison performed by the eDMA engine is (CITER == (BITER >> 1)). This halfway point interrupt request is provided to support double-buffered (aka ping-pong) schemes, or other types of data movement where the processor needs an early indication of the transfer's progress. CITER = BITER = 1 with INT_HALF enabled will generate an interrupt as it satisfies the equation (CITER == (BITER >> 1)) after a single activation. 0 The half-point interrupt is disabled. 1 The half-point interrupt is enabled.
0x1C [30] INT_MAJ	Enable an interrupt when major iteration count completes. If this flag is set, the channel generates an interrupt request by setting the appropriate bit in the EDMA_*x*_ERQH or EDMA_*x*_ERQL when the current major iteration count reaches zero. 0 The end-of-major loop interrupt is disabled. 1 The end-of-major loop interrupt is enabled.
0x1C [31] START	Channel start. If this flag is set the channel is requesting service. The eDMA hardware automatically clears this flag after the channel begins execution. 0 The channel is not explicitly started. 1 The channel is explicitly started via a software initiated service request.

17.4 Functional Description

This section provides an overview of the microarchitecture and functional operation of the eDMA block.

The eDMA module is partitioned into two major modules: the DMA engine and the transfer control descriptor local memory. The DMA engine is further partitioned into four submodules, which are detailed below.

- DMA engine
  - — Address path: This module implements registered versions of two channel transfer control descriptors: channel p and channel q, and is responsible for all the master bus address calculations. All the implemented channels provide the same functionality. This hardware structure allows the data transfers associated with one channel to be preempted after the completion of a read/write sequence if a higher priority channel service request is asserted while the first channel is active. After a channel is activated, it runs until the minor loop is completed unless preempted by a higher priority channel. This capability provides a mechanism (optionally enabled by EDMA_*x*_CPR*n*[ECP]) where a large data move operation can be preempted to minimize the time another channel is blocked from execution.
  - — When another channel is activated, the contents of its transfer control descriptor is read from the local memory and loaded into the registers of the other address path channel{p,q}. After the inner minor loop completes execution, the address path hardware writes the new values for

the TCD*n*.{SADDR, DADDR, CITER} back into the local memory. If the major iteration count is exhausted, additional processing is performed, including the final address pointer updates, reloading the TCDn.CITER field, and a possible fetch of the next TCD*n* from memory as part of a scatter-gather operation.

  - Data path: This module implements the actual bus master read/write datapath. It includes 32 bytes of register storage (matching the maximum transfer size) and the necessary mux logic to support any required data alignment. The system read data bus is the primary input, and the system write data bus is the primary output.
  - The address and data path modules directly support the two-stage pipelined system bus. The address path module represents the 1st stage of the bus pipeline (the address phase), while the data path module implements the second stage of the pipeline (the data phase).
  - Program model/channel arbitration: This module implements the first section of eDMA's programming model and also the channel arbitration logic. The programming model registers are connected to the slave bus (not shown). The eDMA peripheral request inputs and eDMA interrupt request outputs are also connected to this module (via the control logic).
  - Control: This module provides all the control functions for the DMA engine. For data transfers where the source and destination sizes are equal, the DMA engine performs a series of source read, destination write operations until the number of bytes specified in the inner minor loop byte count has been moved.

    A minor loop interaction is defined as the number of bytes to transfer (*n*bytes) divided by the transfer size. Transfer size is defined as:

    ```
    if (SSIZE < DSIZE)
    transfer size = destination transfer size (# of bytes)
    else
    transfer size = source transfer size (# of bytes)
    ```

    Minor loop TCD variables are SOFF, SMOD, DOFF, DMOD, NBYTES, SADDR, DADDR, BWC, ACTIVE, AND START. Major loop TCD variables are DLAST, SLAST, CITER, BITER, DONE, D_REQ, INT_MAJ, MAJOR_LNKCH, and INT_HALF.

    For descriptors where the sizes are not equal, multiple access of the smaller size data are required for each reference of the larger size. For example, if the source size references 16-bit data and the destination is 32-bit data, two reads are performed, then one 32-bit write.

- TCD local memory
  - Memory controller: This logic implements the required dual-ported controller, handling accesses from both the DMA engine as well as references from the slave bus. As noted earlier, in the event of simultaneous accesses, the DMA engine is given priority and the slave transaction is stalled.
  - Memory array: The TCD is implemented using a single-ported, synchronous compiled RAM memory array.

17.4.1 eDMA Basic Data Flow

The eDMA transfers data based on a two-deep, nested flow. The basic flow of a data transfer can be partitioned into three segments. As shown in Figure 17-25, the first segment involves the channel service request. In the diagram, this example uses the assertion of the eDMA peripheral request signal to request service for channel *n*. Channel service request via software and the TCDn.START bit follows the same basic flow as an eDMA peripheral request. The eDMA peripheral request input signal is registered internally and then routed through the DMA engine, first through the control module, then into the program model/channel arbitration module. In the next cycle, the channel arbitration is performed using the fixed-priority or round-robin algorithm. After the arbitration is complete, the activated channel number is sent through the address path and converted into the required address to access the TCD local memory. Next, the TCD memory is accessed and the required descriptor read from the local memory and loaded into the DMA engine address path channel {p,q} registers. The TCD memory is organized 64-bits in width to minimize the time needed to fetch the activated channel's descriptor and load it into the eDMA engine address path channel {p,q} registers.

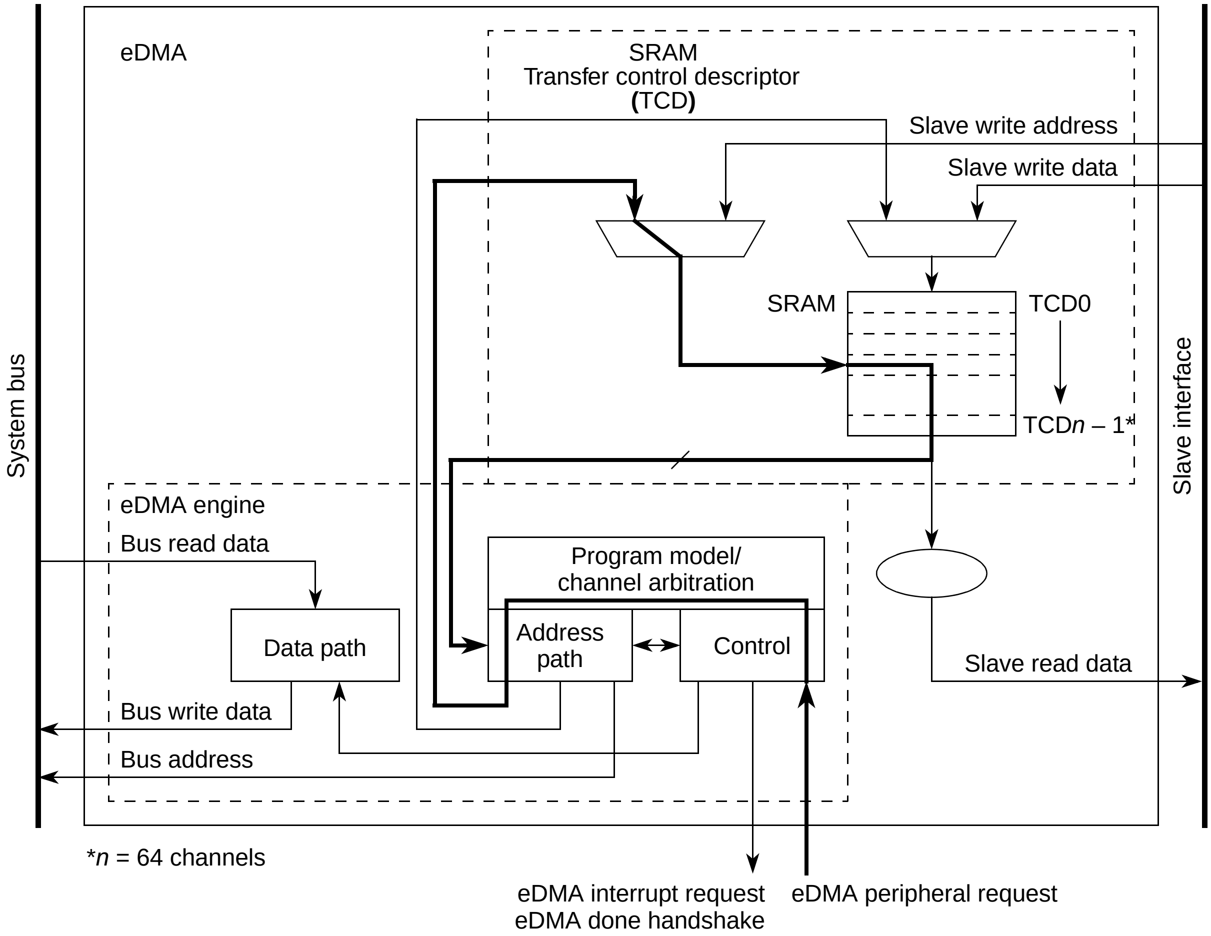


Figure 17-25. eDMA Operation, Part 1

In the second part of the basic data flow as shown in Figure 17-26, the modules associated with the data transfer (address path, data path, and control) sequence through the required source reads and destination writes to perform the actual data movement. The source reads are initiated and the fetched data is temporarily stored in the data path module until it is gated onto the system bus during the destination write. This source read/destination write processing continues until the inner minor byte count has been transferred. The eDMA done handshake signal is asserted at the end of the minor byte count transfer.

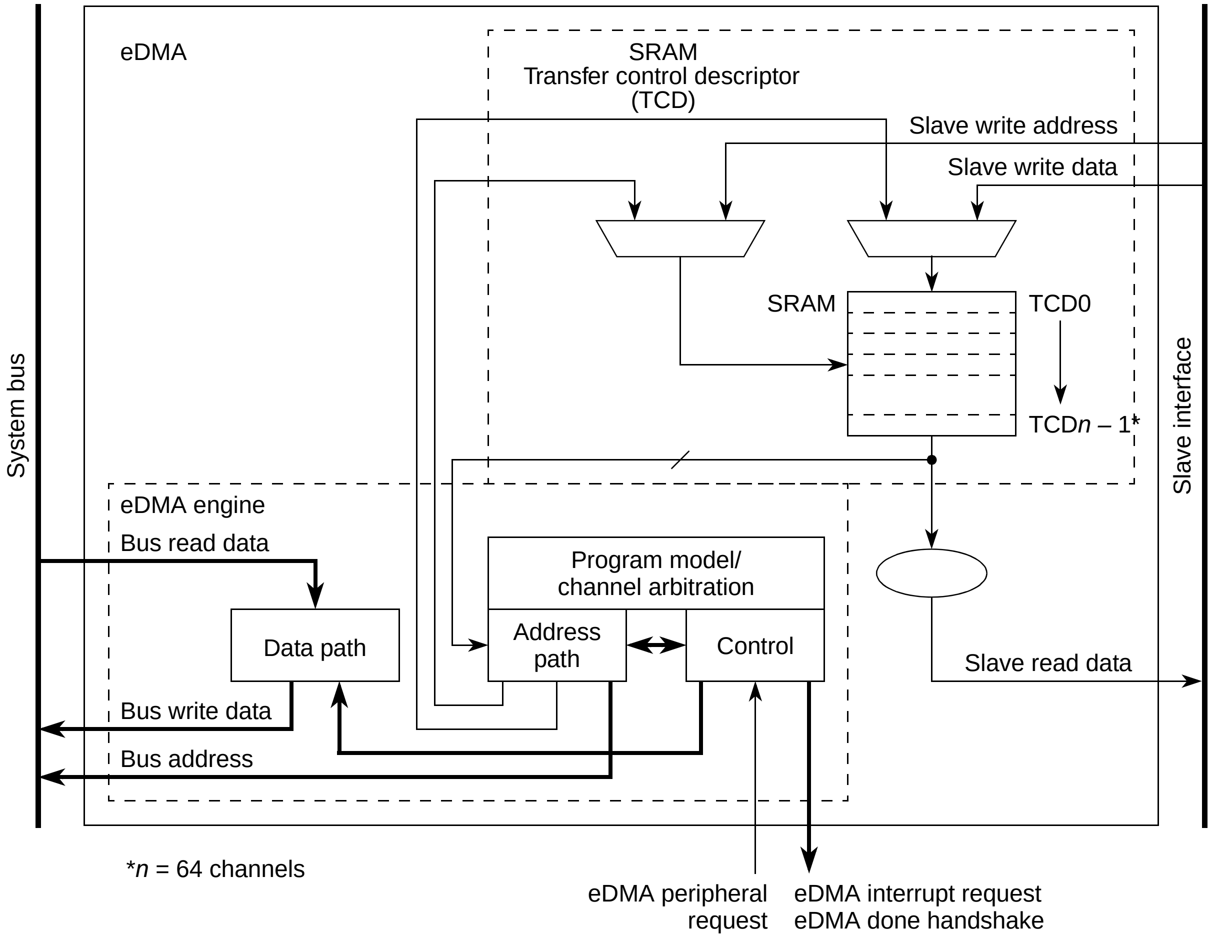


Figure 17-26. eDMA Operation, Part 2

After the inner minor byte count has been moved, the final phase of the basic data flow is performed. In this segment, the address path logic performs the required updates to certain fields in the channel’s TCD; for example, SADDR, DADDR, CITER. If the outer major iteration count is exhausted, then there are additional operations performed. These include the final address adjustments and reloading of the BITER field into the CITER. Additionally, assertion of an optional interrupt request occurs at this time, as does a possible fetch of a new TCD from memory using the scatter-gather address pointer included in the descriptor. The updates to the TCD memory and the assertion of an interrupt request are shown in Figure 17-27.

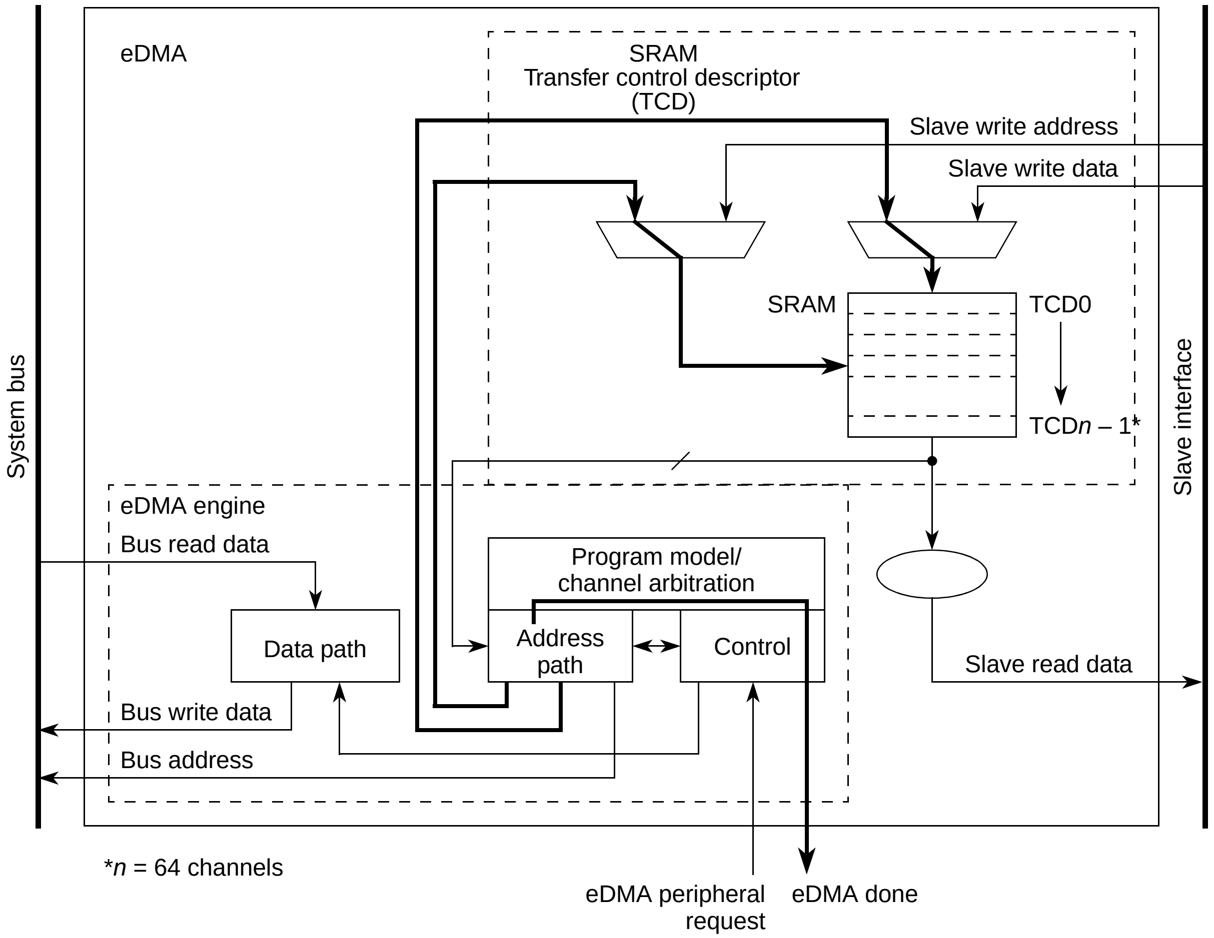


Figure 17-27. eDMA Operation, Part 3

17.5 Initialization / Application Information

17.5.1 eDMA Initialization

A typical initialization of the eDMA has the following sequence:

1. Write the EDMA_*x*_MCR if a configuration other than the default is desired.
2. Write the channel priority levels into the EDMA_*x*_CPR*n* registers if a configuration other than the default is desired.
3. Enable error interrupts in the EDMA_*x*_EEIRL and/or EDMA_*x*_EEIRH registers if desired.
4. Write the 32-byte TCD for each channel that may request service.
5. Enable any hardware service requests via the EDMA_*x*_ERQRH and/or EDMA_*x*_ERQRL registers.
6. Request channel service by software (setting the EDMA_*x*_TCD.START bit) or by hardware (slave device asserting its DMA peripheral request signal).

After any channel requests service, a channel is selected for execution based on the arbitration and priority levels written into the programmer's model. The DMA engine reads the entire TCD, including the primary transfer control parameter shown in Table 17-20, for the selected channel into its internal address path module. As the TCD is being read, the first transfer is initiated on the system bus unless a configuration error is detected. Transfers from the source (as defined by the source address, EDMA_*x*_TCD.SADDR) to the destination (as defined by the destination address, EDMA_*x*_TCD.DADDR) continue until the specified number of bytes (EDMA_*x*_TCD.NBYTES) have been transferred. When the transfer is complete, the DMA engine's local EDMA_*x*_TCD.SADDR, EDMA_*x*_TCD.DADDR, and EDMA_*x*_TCD.CITER are written back to the main TCD memory and any minor loop channel linking is performed, if enabled. If the major loop is exhausted, further post processing is executed; for example, interrupts, major loop channel linking, and scatter-gather operations, if enabled.

Table 17-20. TCD Primary Control and Status Fields

TCD Field Name	Description
START	Control bit to start channel when using a software initiated DMA service (Automatically cleared by hardware)
ACTIVE	Status bit indicating the channel is currently in execution
DONE	Status bit indicating major loop completion (cleared by software when using a software initiated DMA service)
D_REQ	Control bit to disable DMA request at end of major loop completion when using a hardware-initiated DMA service
BWC	Control bits for throttling bandwidth control of a channel
E_SG	Control bit to enable scatter-gather feature
INT_HALF	Control bit to enable interrupt when major loop is half complete
INT_MAJ	Control bit to enable interrupt when major loop completes

Figure 17-28 shows how each DMA request initiates one minor loop transfer (iteration) without CPU intervention. DMA arbitration can occur after each minor loop, and one level of minor loop DMA preemption is allowed. The number of minor loops in a major loop is specified by the beginning iteration count (biter).

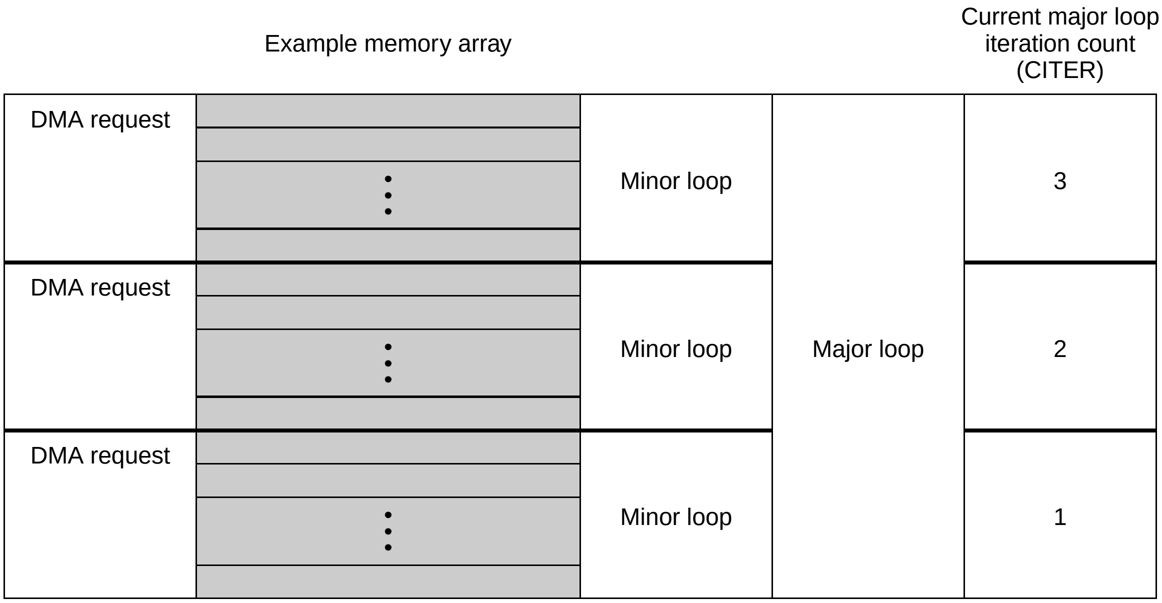


Figure 17-28. Example of Multiple Loop Iterations

Figure 17-29 lists the memory array terms and how the TCD settings interrelate.

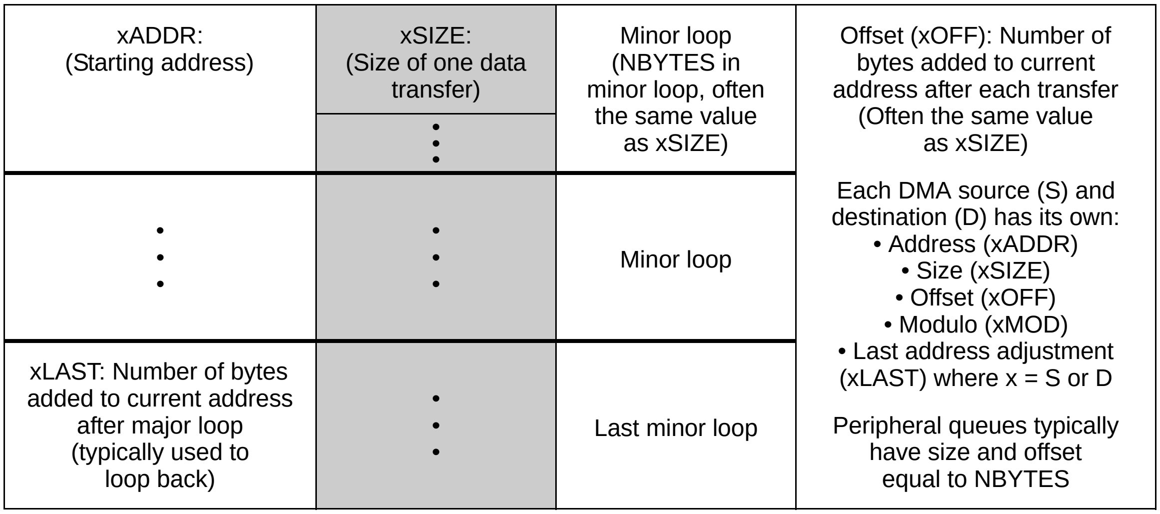


Figure 17-29. Memory Array Terms

17.5.2 DMA Programming Errors

The DMA performs various tests on the transfer control descriptor to verify consistency in the descriptor data. Most programming errors are reported on a per-channel basis with the exception of two errors: group-priority error and channel-priority error, or EDMA_*x*_ESR[GPE] and EDMA_*x*_ESR[CPE], respectively.

For all error types other than group- or channel-priority errors, the channel number causing the error is recorded in the EDMA_*x*_ESR. If the error source is not removed before the next activation of the problem channel, the error is detected and recorded again.

Channel-priority errors are identified within a group after that group has been selected as the active group. For the example, all of the channel priorities in group 1 are unique, but some of the channel priorities in group 0 are the same:

1. The DMA is configured for fixed-group and fixed-channel arbitration modes.
2. Group 1 is the highest priority and all channels are unique in that group.
3. Group 0 is the next highest priority and has two channels with the same priority level.
4. If group 1 has any service requests, those requests are executed.
5. After all of group 1 requests have completed, group 0 becomes the next active group.
6. If group 0 has a service request, then an undefined channel in group 0 is selected and a channel-priority error will occur.
7. This repeats until the all of group 0 requests have been removed or a higher priority group 1 request comes in.

In this sequence, for item 2, the DMA acknowledge lines assert only if the selected channel is requesting service via the DMA peripheral request signal. If interrupts are enabled for all channels, the user receives an error interrupt, but the channel number for the EDMA_*x*_ER and the error interrupt request line are undetermined because they reflect the undefined channel. A group-priority error is global and any request in any group causes a group-priority error.

If priority levels are not unique, the highest (channel/group) priority that has an active request is selected, but the lowest numbered (channel/group) with that priority is selected by arbitration and executed by the DMA engine. The hardware service request handshake signals, error interrupts, and error reporting are associated with the selected channel.

17.5.3 DMA Request Assignments

The assignments between the DMA requests from the modules to the channels of the two eDMAs are shown in Table 17-22 and Table 17-22. The source column is written in C language syntax. The syntax is module_instance.register[bit].

Table 17-21. DMA Request Summary for eDMA_A

DMA Request	Channel	Source	Description
EQADC_A_FISR0_CFFF0	0	EQADC_A.FISR0[CFFF0]	EQADC_A Command FIFO 0 Fill Flag
EQADC_A_FISR0_RFDF0	1	EQADC_A.FISR0[RFDF0]	EQADC_A Receive FIFO 0 Drain Flag
EQADC_A_FISR1_CFFF1	2	EQADC_A.FISR1[CFFF1]	EQADC_A Command FIFO 1 Fill Flag
EQADC_A_FISR1_RFDF1	3	EQADC_A.FISR1[RFDF1]	EQADC_A Receive FIFO 1 Drain Flag
EQADC_A_FISR2_CFFF2	4	EQADC_A.FISR2[CFFF2]	EQADC_A Command FIFO 2 Fill Flag
EQADC_A_FISR2_RFDF2	5	EQADC_A.FISR2[RFDF2]	EQADC_A Receive FIFO 2 Drain Flag
EQADC_A_FISR3_CFFF3	6	EQADC_A.FISR3[CFFF3]	EQADC_A Command FIFO 3 Fill Flag
EQADC_A_FISR3_RFDF3	7	EQADC_A.FISR3[RFDF3]	EQADC_A Receive FIFO 3 Drain Flag
EQADC_A_FISR4_CFFF4	8	EQADC_A.FISR4[CFFF4]	EQADC_A Command FIFO 4 Fill Flag
EQADC_A_FISR4_RFDF4	9	EQADC_A.FISR4[RFDF4]	EQADC_A Receive FIFO 4 Drain Flag
EQADC_A_FISR5_CFFF5	10	EQADC_A.FISR5[CFFF5]	EQADC_A Command FIFO 5 Fill Flag
EQADC_A_FISR5_RFDF5	11	EQADC_A.FISR5[RFDF5]	EQADC_A Receive FIFO 5 Drain Flag
DSPIB_SR_TFFF	12	DSPIB.SR[TFFF]	DSPIB Transmit FIFO Fill Flag
DSPIB_SR_RFDF	13	DSPIB.SR[RFDF]	DSPIB Receive FIFO Drain Flag
DSPIC_SR_TFFF	14	DSPIC.SR[TFFF]	DSPIC Transmit FIFO Fill Flag
DSPIC_SR_RFDF	15	DSPIC.SR[RFDF]	DSPIC Receive FIFO Drain Flag
DSPID_SR_TFFF	16	DSPID.SR[TFFF]	DSPID Transmit FIFO Fill Flag
DSPID_SR_RFDF	17	DSPID.SR[RFDF]	DSPID Receive FIFO Drain Flag
eSCIA_COMBTX	18	ESCIA.SR[TDRE] || ESCIA.SR[TC] || ESCIA.SR[TXRDY]	eSCIA combined DMA request of the Transmit Data Register Empty, Transmit Complete, and LIN Transmit Data Ready DMA requests
eSCIA_COMBRX	19	ESCIA.SR[RDRF] || ESCIA.SR[RXRDY]	eSCIA combined DMA request of the Receive Data Register Full and LIN Receive Data Ready DMA requests
eMIOS_GFR_F0	20	EMIOS.GFR[F0]	eMIOS channel 0 Flag
eMIOS_GFR_F1	21	EMIOS.GFR[F1]	eMIOS channel 1 Flag
eMIOS_GFR_F2	22	EMIOS.GFR[F2]	eMIOS channel 2 Flag
eMIOS_GFR_F3	23	EMIOS.GFR[F3]	eMIOS channel 3 Flag
eMIOS_GFR_F4	24	EMIOS.GFR[F4]	eMIOS channel 4 Flag

Table 17-21. DMA Request Summary for eDMA_A (continued)

DMA Request	Channel	Source	Description
eMIOS_GFR_F8	25	EMIOS.GFR[F8]	eMIOS channel 8 Flag
eMIOS_GFR_F9	26	EMIOS.GFR[F9]	eMIOS channel 9 Flag
eTPU_CDTRSR_A_DTRS0	27	ETPU.CDTRSR_A[DTRS0]	eTPUA Channel 0 Data Transfer Request Status
eTPU_CDTRSR_A_DTRS1	28	ETPU.CDTRSR_A[DTRS1]	eTPUA Channel 1 Data Transfer Request Status
eTPU_CDTRSR_A_DTRS2	29	ETPU.CDTRSR_A[DTRS2]	eTPUA Channel 2 Data Transfer Request Status
eTPU_CDTRSR_A_DTRS14	30	ETPU.CDTRSR_A[DTRS14]	eTPUA Channel 14 Data Transfer Request Status
eTPU_CDTRSR_A_DTRS15	31	ETPU.CDTRSR_A[DTRS15]	eTPUA Channel 15 Data Transfer Request Status
DSPIA_SR_TFFF	32	DSPIAISR[TFFF]	DSPIA Transmit FIFO Fill Flag
DSPIA_SR_RFDF	33	DSPIA.SR[RFDF]	DSPIA Receive FIFO Drain Flag
eSCIB_COMBTX	34	ESCIB.SR[TDRE] || ESCIB.SR[TC] || ESCIB.SR[TXRDY]	eSCIB combined DMA request of the Transmit Data Register Empty, Transmit Complete, and LIN Transmit Data Ready DMA requests
eSCIB_COMBRX	35	ESCIB.SR[RDRF] || ESCIB.SR[RXRDY]	eSCIB combined DMA request of the Receive Data Register Full and LIN Receive Data Ready DMA requests
eMIOS_GFR_F6	36	EMIOS.GFR[F6]	eMIOS channel 6 Flag
eMIOS_GFR_F7	37	EMIOS.GFR[F7]	eMIOS channel 7 Flag
eMIOS_GFR_F10	38	EMIOS.GFR[F10]	eMIOS channel 10 Flag
eMIOS_GFR_F11	39	EMIOS.GFR[F11]	eMIOS channel 11 Flag
eMIOS_GFR_F16	40	EMIOS.GFR[F16]	eMIOS channel 16 Flag
eMIOS_GFR_F17	41	EMIOS.GFR[F17]	eMIOS channel 17 Flag
eMIOS_GFR_F18	42	EMIOS.GFR[F18]	eMIOS channel 18 Flag
eMIOS_GFR_F19	43	EMIOS.GFR[F19]	eMIOS channel 19 Flag
eTPU_CDTRSR_A_DTRS12	44	ETPU.CDTRSR_A[DTRS12]	eTPUA Channel 12 Data Transfer Request Status
eTPU_CDTRSR_A_DTRS13	45	ETPU.CDTRSR_A[DTRS13]	eTPUA Channel 13 Data Transfer Request Status
eTPU_CDTRSR_A_DTRS28	46	ETPU.CDTRSR_A[DTRS28]	eTPUA Channel 28 Data Transfer Request Status
eTPU_CDTRSR_A_DTRS29	47	ETPU.CDTRSR_A[DTRS29]	eTPUA Channel 29 Data Transfer Request Status
SIU_EISR_EIF0	48	SIU.SIU_EISR[EIF0]	SIU External Interrupt Flag 0
SIU_EISR_EIF1	49	SIU.SIU_EISR[EIF1]	SIU External Interrupt Flag 1
SIU_EISR_EIF2	50	SIU.SIU_EISR[EIF2]	SIU External Interrupt Flag 2
SIU_EISR_EIF3	51	SIU.SIU_EISR[EIF3]	SIU External Interrupt Flag 3
eTPU_CDTRSR_B_DTRS0	52	ETPU.CDTRSR_B[DTRS0]	eTPUB Channel 0 Data Transfer Request Status
eTPU_CDTRSR_B_DTRS1	53	ETPU.CDTRSR_B[DTRS1]	eTPUB Channel 1 Data Transfer Request Status
eTPU_CDTRSR_B_DTRS2	54	ETPU.CDTRSR_B[DTRS2]	eTPUB Channel 2 Data Transfer Request Status

Table 17-21. DMA Request Summary for eDMA_A (continued)

DMA Request	Channel	Source	Description
eTPU_CDTRSR_B_DTRS3	55	ETPU.CDTRSR_B[DTRS3]	eTPUB Channel 3 Data Transfer Request Status
eTPU_CDTRSR_B_DTRS12	56	ETPU.CDTRSR_B[DTRS12]	eTPUB Channel 12 Data Transfer Request Status
eTPU_CDTRSR_B_DTRS13	57	ETPU.CDTRSR_B[DTRS13]	eTPUB Channel 13 Data Transfer Request Status
eTPU_CDTRSR_B_DTRS14	58	ETPU.CDTRSR_B[DTRS14]	eTPUB Channel 14 Data Transfer Request Status
eTPU_CDTRSR_B_DTRS15	59	ETPU.CDTRSR_B[DTRS15]	eTPUB Channel 15 Data Transfer Request Status
eTPU_CDTRSR_B_DTRS28	60	ETPU.CDTRSR_B[DTRS28]	eTPUB Channel 28 Data Transfer Request Status
eTPU_CDTRSR_B_DTRS29	61	ETPU.CDTRSR_B[DTRS29]	eTPUB Channel 29 Data Transfer Request Status
eTPU_CDTRSR_B_DTRS30	62	ETPU.CDTRSR_B[DTRS30]	eTPUB Channel 30 Data Transfer Request Status
eTPU_CDTRSR_B_DTRS31	63	ETPU.CDTRSR_B[DTRS31]	eTPUB Channel 31 Data Transfer Request Status

Table 17-22. DMA Request Summary for eDMA_B

DMA Request	Channel	Source	Description
EQADC_B_FISR0_CFFF0	0	EQADC_B.FISR0[CFFF0]	EQADC_B Command FIFO 0 Fill Flag
EQADC_B_FISR0_RFDF0	1	EQADC_B.FISR0[RFDF0]	EQADC_B Receive FIFO 0 Drain Flag
EQADC_B_FISR1_CFFF1	2	EQADC_B.FISR1[CFFF1]	EQADC_B Command FIFO 1 Fill Flag
EQADC_B_FISR1_RFDF1	3	EQADC_B.FISR1[RFDF1]	EQADC_B Receive FIFO 1 Drain Flag
EQADC_B_FISR2_CFFF2	4	EQADC_B.FISR2[CFFF2]	EQADC_B Command FIFO 2 Fill Flag
EQADC_B_FISR2_RFDF2	5	EQADC_B.FISR2[RFDF2]	EQADC_B Receive FIFO 2 Drain Flag
EQADC_B_FISR3_CFFF3	6	EQADC_B.FISR3[CFFF3]	EQADC_B Command FIFO 3 Fill Flag
EQADC_B_FISR3_RFDF3	7	EQADC_B.FISR3[RFDF3]	EQADC_B Receive FIFO 3 Drain Flag
EQADC_B_FISR4_CFFF4	8	EQADC_B.FISR4[CFFF4]	EQADC_B Command FIFO 4 Fill Flag
EQADC_B_FISR4_RFDF3	9	EQADC_B.FISR4[RFDF4]	EQADC_B Receive FIFO 4 Drain Flag
EQADC_B_FISR5_CFFF5	10	EQADC_B.FISR5[CFFF5]	EQADC_B Command FIFO 5 Fill Flag
EQADC_B_FISR5_RFDF5	11	EQADC_B.FISR5[RFDF5]	EQADC_B Receive FIFO 5 Drain Flag
DECFILTERA_IB	12	DECFILTERA.IB	Decimation Filter A Input Buffer Fill Flag
DECFILTERA_OB	13	DECFILTERA.OB	Decimation Filter A Output Buffer Drain Flag
DECFILTERB_IB	14	DECFILTERB.IB	Decimation Filter B Input Buffer Fill Flag
DECFILTERB_OB	15	DECFILTERB.OB	Decimation Filter B Output Buffer Drain Flag
DECFILTERC_IB	16	DECFILTERC.IB	Decimation Filter CInput Buffer Fill Flag
DECFILTERC_OB	17	DECFILTERC.OB	Decimation Filter C Output Buffer Drain Flag
DECFILTERD_IB	18	DECFILTERD.IB	Decimation Filter D Input Buffer Fill Flag
DECFILTERD_OB	19	DECFILTERD.OB	Decimation Filter D Output Buffer Drain Flag
DECFILTERE_IB	20	DECFILTERE.IB	Decimation Filter E Input Buffer Fill Flag

Table 17-22. DMA Request Summary for eDMA_B (continued)

DMA Request	Channel	Source	Description
DECFILTERE_OB	21	DECFILTERE.OB	Decimation Filter E Output Buffer Drain Flag
DECFILTERF_IB	22	DECFILTERF.IB	Decimation Filter F Input Buffer Fill Flag
DECFILTERF_OB	23	DECFILTERF.OB	Decimation Filter F Output Buffer Drain Flag
DECFILTERG_IB	24	DECFILTERG.IB	Decimation Filter G Input Buffer Fill Flag
DECFILTERG_OB	25	DECFILTERG.OB	Decimation Filter G Output Buffer Drain Flag
DECFILTERH_IB	26	DECFILTERH.IB	Decimation Filter H Input Buffer Fill Flag
DECFILTERH_OB	27	DECFILTERH.OB	Decimation Filter H Output Buffer Drain Flag
DECFILTERI_IB	28	DECFILTERI.IB	Decimation Filter I Input Buffer Fill Flag
DECFILTERI_OB	29	DECFILTERI.OB	Decimation Filter I Output Buffer Drain Flag
DECFILTERJ_IB	30	DECFILTERJ.IB	Decimation Filter J Input Buffer Fill Flag
DECFILTERJ_OB	31	DECFILTERJ.OB	Decimation Filter J Output Buffer Drain Flag
DECFILTERK_IB	32	DECFILTERK.IB	Decimation Filter K Input Buffer Fill Flag
DECFILTERK_OB	33	DECFILTERK.OB	Decimation Filter K Output Buffer Drain Flag
DECFILTERL_IB	34	DECFILTERL.IB	Decimation Filter L Input Buffer Fill Flag
DECFILTERL_OB	35	DECFILTERL.OB	Decimation Filter L Output Buffer Drain Flag
eTPU_CDTRSR_C_DTRS0	44	ETPU.CDTRSR_C[DTRS0]	eTPUC Channel 0 Data Transfer Request Status
eTPU_CDTRSR_C_DTRS1	45	ETPU.CDTRSR_C[DTRS1]	eTPUC Channel 1 Data Transfer Request Status
eTPU_CDTRSR_C_DTRS2	46	ETPU.CDTRSR_C[DTRS2]	eTPUC Channel 2Data Transfer Request Status
eTPU_CDTRSR_C_DTRS3	47	ETPU.CDTRSR_C[DTRS3]	eTPUC Channel 3 Data Transfer Request Status
eTPU_CDTRSR_C_DTRS4	48	ETPU.CDTRSR_C[DTRS4]	eTPUC Channel 4 Data Transfer Request Status
eTPU_CDTRSR_C_DTRS5	49	ETPU.CDTRSR_C[DTRS5]	eTPUC Channel 5 Data Transfer Request Status
eTPU_CDTRSR_C_DTRS6	50	ETPU.CDTRSR_C[DTRS6]	eTPUC Channel 6 Data Transfer Request Status
eTPU_CDTRSR_C_DTRS7	51	ETPU.CDTRSR_C[DTRS7]	eTPUC Channel 7 Data Transfer Request Status
DSPIE_SR_TFFF	61	DSPIEISR[TFFF]	DSPIE Transmit FIFO Fill Flag
DSPIE_SR_RFDF	62	DSPIE.SR[RFDF]	DSPIE Receive FIFO Drain Flag

17.5.4 DMA Arbitration Mode Considerations

17.5.4.1 Fixed-Group Arbitration, Fixed-Channel Arbitration

In this mode, the channel service request from the highest priority channel in the highest priority group is selected to execute. If the eDMA is programmed so the channels within one group use fixed priorities, and that group is assigned the highest fixed priority of all groups, it is possible for that group to take all the bandwidth of the eDMA controller. That is, no other groups can be serviced if there is always at least one DMA request pending on a channel in the highest priority group when the controller arbitrates the next DMA request. The advantage of this scenario is that latency can be small for channels that need to be serviced quickly. Preemption is available in this scenario only.

17.5.4.2 Round-Robin Group Arbitration, Fixed-Channel Arbitration

When one or more DMA requests arrive from one or more groups, the channel with the highest priority from a specific group is serviced first. Groups are serviced starting with the highest group number with a service request and rotating through to the lowest group number containing a service request.

After the channel request is serviced, the group round robin algorithm selects the highest pending request from the next group in the round-robin sequence. Servicing continues round robin, always servicing the highest priority channel in the next group in the sequence, or skipping a group if it has no pending requests.

If a channel requests service at a rate that equals or exceeds the round robin service rate, then that channel is always serviced before lower priority channels in the same group, and the lower priority channels are never serviced. The advantage of this scenario is that no one group can consume all the eDMA bandwidth. The highest priority channel selection latency is potentially greater than fixed/fixed arbitration. Excessive request rates on high-priority channels can prevent the servicing of lower priority channels in the same group.

17.5.4.3 Round-Robin Group Arbitration, Round-Robin Channel Arbitration

Groups are serviced as described in Section 17.5.4.2, “Round-Robin Group Arbitration, Fixed-Channel Arbitration”, but this time channels are serviced in channel number order. One channel only is serviced from each requesting group for each round robin pass through the groups.

Within each group, channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to channel priority levels.

Because channels are serviced in round-robin manner, any channel that generates DMA requests faster than a combination of the group round-robin service rate and the channel service rate for its group does not prevent the servicing of other channels in its group. Any DMA requests that are not serviced are simply lost, but at least one channel gets serviced.

This scenario ensures that all channels are guaranteed service at some point, regardless of the request rates. However, the potential latency could be high. All channels are treated equally. Priority levels are not used in round-robin/round-robin mode.

17.5.4.4 Fixed-Group Arbitration, Round-Robin Channel Arbitration

The highest priority group with a request is serviced. Lower priority groups are serviced if no pending requests exist in the higher priority groups.

Within each group, channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to the channel priority levels assigned within the group.

This scenario could cause the same bandwidth consumption problem as indicated in Section 17.5.4.1, “Fixed-Group Arbitration, Fixed-Channel Arbitration”, but all the channels in the highest priority group get serviced. Service latency is short on the highest priority group, but could potentially get longer and longer as the group priority decreases.

17.5.5 DMA Transfer

17.5.5.1 Single Request

To perform a simple transfer of *n* bytes of data with one activation, set the major loop to 1 (EDMA_*x*_TCD.CITER = EDMA_*x*_TCD.BITER = 1). The data transfer begins after the channel service request is acknowledged and the channel is selected to execute. After the transfer is complete, the EDMA_*x*_TCD.DONE bit is set and an interrupt is generated if properly enabled.

For example, the following TCD entry is configured to transfer 16 bytes of data. The eDMA is programmed for one iteration of the major loop transferring 16 bytes per iteration. The source memory has a byte wide memory port located at 0x1000. The destination memory has a word wide port located at 0x2000. The address offsets are programmed in increments to match the size of the transfer; one byte for the source and four bytes for the destination. The final source and destination addresses are adjusted to return to their beginning values.

EDMA_*x*_TCD.CITER = EDMA_*x*_TCD.BITER = 1

EDMA_*x*_TCD.NBYTES = 16

EDMA_*x*_TCD.SADDR = 0x1000

EDMA_*x*_TCD.SOFF = 1

EDMA_*x*_TCD.SSIZE = 0

EDMA_*x*_TCD.SLAST = –16

EDMA_*x*_TCD.DADDR = 0x2000

EDMA_*x*_TCD.DOFF = 4

EDMA_*x*_TCD.DSIZE = 2

EDMA_*x*_TCD.DLAST_SGA = –16

EDMA_*x*_TCD.INT_MAJ = 1

EDMA_*x*_TCD.START = 1 (Must be written last after all other fields have been initialized)

All other TCD fields = 0

This would generate the following sequence of events:

1. Slave write to the EDMA_*x*_TCD.START bit requests channel service.

2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: EDMA_*x*_TCD.DONE = 0, EDMA_*x*_TCD.START = 0, EDMA_*x*_TCD.ACTIVE = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
   a) read_byte(0x1000), read_byte(0x1001), read_byte(0x1002), read_byte(0x1003)
   b) write_word(0x2000) → first iteration of the minor loop
   c) read_byte(0x1004), read_byte(0x1005), read_byte(0x1006), read_byte(0x1007)
   d) write_word(0x2004) → second iteration of the minor loop
   e) read_byte(0x1008), read_byte(0x1009), read_byte(0x100a), read_byte(0x100b)
   f) write_word(0x2008) → third iteration of the minor loop
   g) read_byte(0x100c), read_byte(0x100d), read_byte(0x100e), read_byte(0x100f)
   h) write_word(0x200c) → last iteration of the minor loop → major loop complete
6. eDMA engine writes: EDMA_*x*_TCD.SADDR = 0x1000, EDMA_*x*_TCD.DADDR = 0x2000, EDMA_*x*_TCD.CITER = 1 (EDMA_*x*_TCD.BITER).
7. eDMA engine writes: EDMA_*x*_TCD.ACTIVE = 0, EDMA_*x*_TCD.DONE = 1, EDMA_*x*_IRQR*n* = 1.
8. The channel retires.

The eDMA goes idle or services the next channel.

17.5.5.2 Multiple Requests

The next example is the same as previous, excepting transferring 32 bytes via two hardware requests. The only fields that change are the major loop iteration count and the final address offsets. The eDMA is programmed for two iterations of the major loop transferring 16 bytes per iteration. After the channel's hardware requests are enabled in the EDMA_*x*_ERQR, channel service requests are initiated by the slave device (ERQR should be set after TCD). Note that EDMA_*x*_TCD.START = 0.

EDMA_*x*_TCD.CITER = EDMA_*x*_TCD.BITER = 2
EDMA_*x*_TCD.NBYTES = 16
EDMA_*x*_TCD.SADDR = 0x1000
EDMA_*x*_TCD.SOFF = 1
EDMA_*x*_TCD.SSIZE = 0
EDMA_*x*_TCD.SLAST = –32
EDMA_*x*_TCD.DADDR = 0x2000
EDMA_*x*_TCD.DOFF = 4
EDMA_*x*_TCD.DSIZE = 2
EDMA_*x*_TCD.DLAST_SGA = –32
EDMA_*x*_TCD.INT_MAJ = 1
EDMA_*x*_TCD.START = 0 (Must be written last after all other fields have been initialized)

All other TCD fields = 0

This generates the following sequence of events:

1. First hardware (eDMA peripheral request) request for channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: EDMA_*x*_TCD.DONE = 0, EDMA_*x*_TCD.START = 0, EDMA_*x*_TCD.ACTIVE = 1.
4. eDMA engine reads: channel TCD data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
    a) read_byte(0x1000), read_byte(0x1001), read_byte(0x1002), read_byte(0x1003)
    b) write_word(0x2000) → first iteration of the minor loop
    c) read_byte(0x1004), read_byte(0x1005), read_byte(0x1006), read_byte(0x1007)
    d) write_word(0x2004) → second iteration of the minor loop
    e) read_byte(0x1008), read_byte(0x1009), read_byte(0x100a), read_byte(0x100b)
    f) write_word(0x2008) → third iteration of the minor loop
    g) read_byte(0x100c), read_byte(0x100d), read_byte(0x100e), read_byte(0x100f)
    h) write_word(0x200c) → last iteration of the minor loop
6. eDMA engine writes: EDMA_*x*_TCD.SADDR = 0x1010, EDMA_*x*_TCD.DADDR = 0x2010, EDMA_*x*_TCD.CITER = 1.
7. eDMA engine writes: EDMA_*x*_TCD.ACTIVE = 0.
8. The channel retires → one iteration of the major loop.

The eDMA goes idle or services the next channel.

9. Second hardware (eDMA peripheral request) requests channel service.
10. The channel is selected by arbitration for servicing.
11. eDMA engine writes: EDMA_*x*_TCD.DONE = 0, EDMA_*x*_TCD.START = 0, EDMA_*x*_TCD.ACTIVE = 1.
12. eDMA engine reads: channel TCD data from local memory to internal register file.
13. The source to destination transfers are executed as follows:
    a) read_byte(0x1010), read_byte(0x1011), read_byte(0x1012), read_byte(0x1013)
    b) write_word(0x2010) → first iteration of the minor loop
    c) read_byte(0x1014), read_byte(0x1015), read_byte(0x1016), read_byte(0x1017)
    d) write_word(0x2014) → second iteration of the minor loop
    e) read_byte(0x1018), read_byte(0x1019), read_byte(0x101a), read_byte(0x101b)
    f) write_word(0x2018) → third iteration of the minor loop
    g) read_byte(0x101c), read_byte(0x101d), read_byte(0x101e), read_byte(0x101f)
    h) write_word(0x201c) → last iteration of the minor loop → major loop complete
14. eDMA engine writes: EDMA_*x*_TCD.SADDR = 0x1000, EDMA_*x*_TCD.DADDR = 0x2000, EDMA_*x*_TCD.CITER = 2 (EDMA_*x*_TCD.BITER).

15. eDMA engine writes: EDMA_*x*_TCD.ACTIVE = 0, EDMA_*x*_TCD.DONE = 1, EDMA_*x*_IRQR*n* = 1.
16. The channel retires → major loop complete.

The eDMA goes idle or services the next channel.

17.5.5.3 Modulo Feature

The modulo feature of the eDMA provides the ability to implement a circular data queue in which the size of the queue is a power of two. MOD is a 5-bit bit-field for both the source and destination in the TCD and specifies which lower address bits are allowed to increment from their original value after the address + offset calculation. All upper address bits remain the same as in the original value. A setting of 0 for this field disables the modulo feature.

Table 17-23 shows how the transfer addresses are specified based on the setting of the MOD field. Here a circular buffer is created where the address wraps to the original value while the 28 upper address bits (0x1234567x) retain their original value. In this example the source address is set to 0x12345670, the offset is set to 4 bytes and the mod field is set to 4, allowing for a 2^4 byte (16-byte) size queue.

Table 17-23. Modulo Feature Example

Transfer Number	Address
1	0x12345670
2	0x12345674
3	0x12345678
4	0x1234567C
5	0x12345670
6	0x12345674

17.5.6 TCD Status

17.5.6.1 Minor Loop Complete

There are two methods to test for minor loop completion when using software initiated service requests. The first method is to read the EDMA_*x*_TCD.CITER field and test for a change. Another method may be extracted from the sequence below. The second method is to test the EDMA_*x*_TCD.START bit AND the EDMA_*x*_TCD.ACTIVE bit. The minor loop complete condition is indicated by both bits reading zero after the EDMA_*x*_TCD.START was written to a 1. Polling the EDMA_*x*_TCD.ACTIVE bit may be inconclusive because the active status may be missed if the channel execution is short in duration.

The TCD status bits execute the following sequence for a software activated channel:

1. EDMA_*x*_TCD.START = 1, EDMA_*x*_TCD.ACTIVE = 0, EDMA_*x*_TCD.DONE = 0 (channel service request via software).

2. EDMA_*x*_TCD.START = 0, EDMA_*x*_TCD.ACTIVE = 1, EDMA_*x*_TCD.DONE = 0 (channel is executing).
3. EDMA_*x*_TCD.START = 0, EDMA_*x*_TCD.ACTIVE = 0, EDMA_*x*_TCD.DONE = 0 (channel has completed the minor loop and is idle), or
4. EDMA_*x*_TCD.START = 0, EDMA_*x*_TCD.ACTIVE = 0, EDMA_*x*_TCD.DONE = 1 (channel has completed the major loop and is idle).

The best method to test for minor loop completion when using hardware initiated service requests is to read the EDMA_*x*_TCD.CITER field and test for a change. The hardware request and acknowledge handshakes signals are not visible in the programmer's model.

The TCD status bits execute the following sequence for a hardware activated channel:

1. eDMA peripheral request asserts (channel service request via hardware).
2. EDMA_*x*_TCD.START = 0, EDMA_*x*_TCD.ACTIVE = 1, EDMA_*x*_TCD.DONE = 0 (channel is executing).
3. EDMA_*x*_TCD.START = 0, EDMA_*x*_TCD.ACTIVE = 0, EDMA_*x*_TCD.DONE = 0 (channel has completed the minor loop and is idle), or
4. EDMA_*x*_TCD.START = 0, EDMA_*x*_TCD.ACTIVE = 0, EDMA_*x*_TCD.DONE = 1 (channel has completed the major loop and is idle).

For both activation types, the major loop complete status is explicitly indicated via the EDMA_*x*_TCD.DONE bit.

The EDMA_*x*_TCD.START bit is cleared automatically when the channel begins execution, regardless of how the channel was activated.

17.5.6.2 Active Channel TCD Reads

The eDMA will read back the true EDMA_*x*_TCD.SADDR, EDMA_*x*_TCD.DADDR, and EDMA_*x*_TCD.NBYTES values if read while a channel is executing. The true values of the SADDR, DADDR, and NBYTES are the values the eDMA engine is currently using in its internal register file and not the values in the TCD local memory for that channel. The addresses (SADDR and DADDR) and NBYTES (decrements to zero as the transfer progresses) can give an indication of the progress of the transfer. All other values are read back from the TCD local memory.

17.5.6.3 Preemption Status

Preemption is available only when fixed arbitration is selected for both group- and channel-arbitration modes. A preempt-able situation is one in which a preempt-enabled channel is running and a higher priority request becomes active. When the eDMA engine is not operating in fixed group, fixed-channel arbitration mode, the determination of the relative priority of the actively running and the outstanding requests become undefined. Channel and group priorities are treated as equal (or more exactly, constantly rotating) when round-robin arbitration mode is selected.

The EDMA_*x*_TCD.ACTIVE bit for the preempted channel remains asserted throughout the preemption. The preempted channel is temporarily suspended while the preempting channel executes one iteration of the major loop. Two EDMA_*x*_TCD.ACTIVE bits set at the same time in the overall TCD map indicates a higher priority channel is actively preempting a lower priority channel.

17.5.7 Channel Linking

Channel linking (or chaining) is a mechanism in which one channel sets the EDMA_*x*_TCD.START bit of another channel (or itself), thus initiating a service request for that channel. This operation is automatically performed by the eDMA engine at the conclusion of the major or minor loop when properly enabled.

The minor loop channel linking occurs at the completion of the minor loop (or one iteration of the major loop). The EDMA_*x*_TCD.CITER.E_LINK field are used to determine whether a minor loop link is requested. When enabled, the channel link is made after each iteration of the minor loop except for the last. When the major loop is exhausted, only the major loop channel link fields are used to determine if a channel link should be made. For example, with the initial fields of:

EDMA_*x*_TCD.CITER.E_LINK = 1
EDMA_*x*_TCD.CITER.LINKCH = 0xC
EDMA_*x*_TCD.CITER value = 0x4
EDMA_*x*_TCD.MAJOR.E_LINK = 1
EDMA_*x*_TCD.MAJOR.LINKCH = 0x7

will execute as:

1. Minor loop done → set channel 12 EDMA_*x*_TCD.START bit
2. Minor loop done → set channel 12 EDMA_*x*_TCD.START bit
3. Minor loop done → set channel 12 EDMA_*x*_TCD.START bit
4. Minor loop done, major loop done → set channel 7 EDMA_*x*_TCD.START bit

When minor loop linking is enabled (EDMA_*x*_TCD.CITER.E_LINK = 1), the EDMA_*x*_TCD.CITER field uses a nine bit vector to form the current iteration count.

When minor loop linking is disabled (EDMA_*x*_TCD.CITER.E_LINK = 0), the EDMA_*x*_TCD.CITER field uses a 15-bit vector to form the current iteration count. The bits associated with the EDMA_*x*_TCD.CITER.LINKCH field are concatenated onto the CITER value to increase the range of the CITER.

NOTE

> After configuration, the EDMA_*x*_TCD.CITER.E_LINK bit and the EDMA_*x*_TCD.BITER.E_LINK bit must be equal or a configuration error is reported. The CITER and BITER vector widths must be equal to calculate the major loop, halfway done interrupt point.

Table 17-24 summarizes how a DMA channel can link to another DMA channel, i.e, use another channel’s TCD, at the end of a loop.

Table 17-24. Channel Linking Parameters

Desired Link Behavior	TCD Control Field Name	Description
Link at end of minor loop	citer.e_link	Enable channel-to-channel linking on minor loop completion (current iteration).
	citer.linkch	Link channel number when linking at end of minor loop (current iteration).
Link at end of major loop	major.e_link	Enable channel-to-channel linking on major loop completion.
	major.linkch	Link channel number when linking at end of major loop.

17.5.8 Dynamic Programming

This section provides recommended methods to change the programming model during channel execution.

17.5.8.1 Dynamic Channel Linking and Dynamic Scatter-Gather Operation

Dynamic channel linking and dynamic scatter-gather operation is the process of changing the EDMA_*x*_TCD.MAJOR.E_LINK or EDMA_*x*_TCD.E_SG bits during channel execution. These bits are read from the TCD local memory at the end of channel execution thus allowing the user to enable either feature during channel execution.

Because the user is allowed to change the configuration during execution, a coherency model is needed. Consider a scenario where the user attempts to execute a dynamic channel link by enabling the EDMA_*x*_TCD.MAJOR.E_LINK bit at the same time the eDMA engine is retiring the channel. The EDMA_*x*_TCD.MAJOR.E_LINK would be set in the programmer’s model, but it would be unclear whether the actual link was made before the channel retired.

The following coherency model is recommended when executing a dynamic channel link or dynamic scatter-gather request:

1. Set the EDMA_*x*_TCD.MAJOR.E_LINK bit.
2. Read back the EDMA_*x*_TCD.MAJOR.E_LINK bit
3. Test the EDMA_*x*_TCD.MAJOR.E_LINK request status:
    a) If the bit is set, the dynamic link attempt was successful.
    b) If the bit is cleared, the attempted dynamic link did not succeed, the channel was already retiring.

This same coherency model is true for dynamic scatter-gather operations. For both dynamic requests, the TCD local memory controller forces the EDMA_*x*_TCD.MAJOR.E_LINK and EDMA_*x*_TCD.E_SG bits to zero on any writes to a channel’s TCD after that channel’s EDMA_*x*_TCD.DONE bit is set indicating the major loop is complete.

NOTE

The user must clear the EDMA_*x*_TCD.DONE bit before writing the EDMA_*x*_TCD.MAJOR.E_LINK or EDMA_*x*_TCD.E_SG bits. The EDMA_*x*_TCD.DONE bit is cleared automatically by the eDMA engine after a channel begins execution.

Chapter 18
Enhanced Modular Input/Output Subsystem (eMIOS200)

18.1 Introduction

The eMIOS200 provides functionality to generate or measure events. The eMIOS200 is implemented with its own configuration of timer channels to suit the target applications needs, while providing a consistent user interface with previous eMIOS implementations. The MPC5676R has one eMIOS200 module that implements 24-bit counters.

18.1.1 Block Diagram

Figure 18-1 shows the eMIOS200 block diagram.

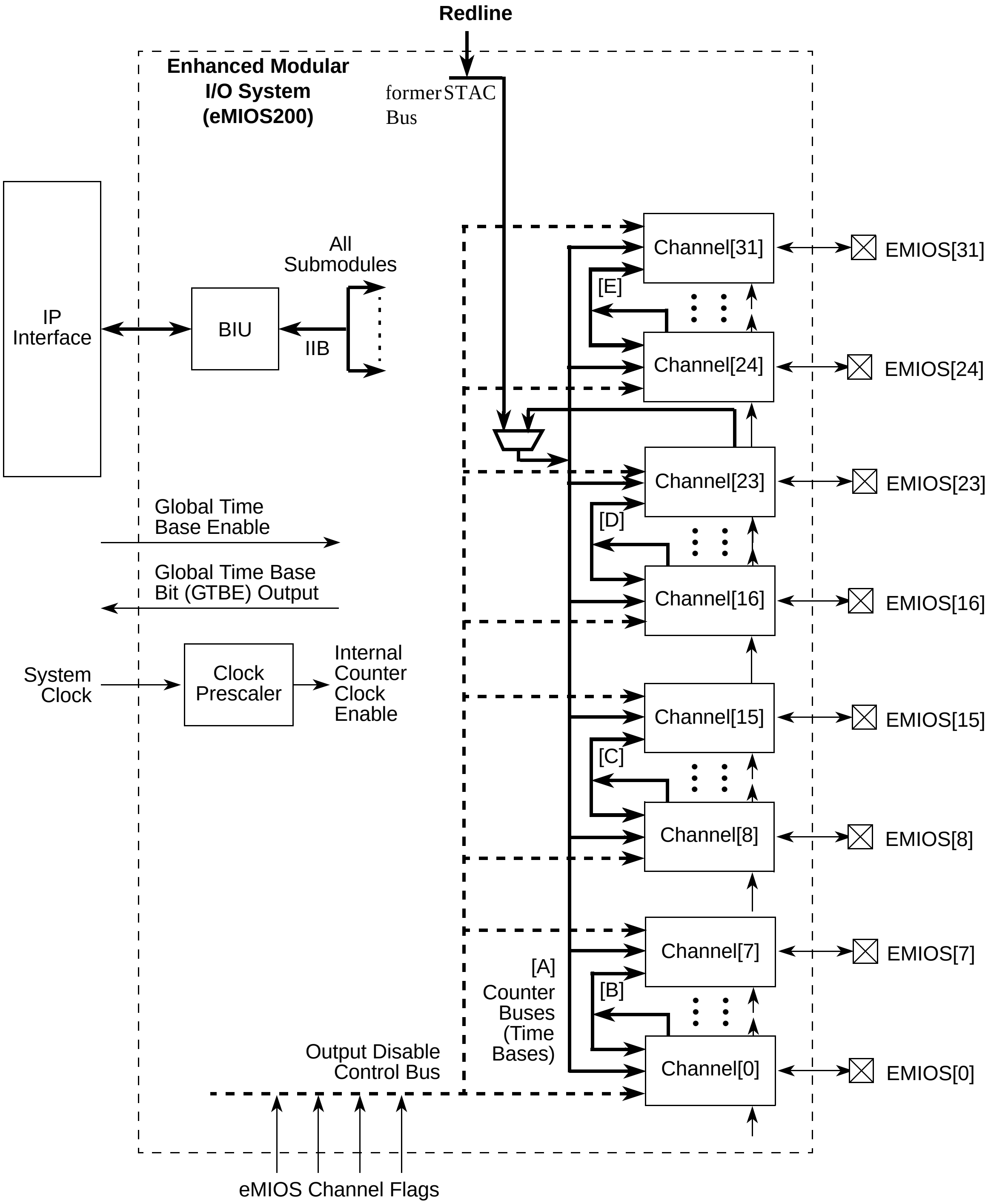


Figure 18-1. eMIOS200 Block Diagram

18.1.2 Features

- 32 unified channels (UC)
- Channel features:
    - 24-bit registers for captured/match values
    - 24-bit internal counter
    - Internal prescaler
    - Selectable time base
    - Can generate its own time base
- Five 24-bit-wide counter buses
    - Counter bus A can be driven by unified channel 23or by the STAC bus.
    - Counter buses B, C, D, and E are driven by unified channels 0, 8, 16, and 24, respectively
    - Counter bus A can be shared among all unified channels. UCs 0 to 7, 8 to 15, 16 to 23, and 24 to 31 can share counter buses B, C, D, and E, respectively
- One global prescaler
- The output signal from the module configuration register's global time base enable bit (EMIOS_MCR[GTBE]) is wrapped back into the global timebase enable input so that the timebase of each channel can be started simultaneously.
- Shared time bases through the counter buses
- Shadow FLAG register
- State of eMIOS200 can be frozen for debug purposes
- Debug mode is supported

18.1.3 Modes of Operation

The Unified Channels can be configured to operate in the following modes:

- General-Purpose Input/Output (GPIO) Mode"
- Single Action Input Capture (SAIC) Mode
- Single Action Output Compare (SAOC) Mode
- Input Pulse-Width Measurement (IPWM) Mode
- Input Period Measurement (IPM) Mode
- Double Action Output Compare (DAOC) Mode
- Pulse/Edge Accumulation (PEA) Mode
- Pulse/Edge Counting (PEC) Mode
- Quadrature Decode (QDEC) Mode
- Windowed Programmable Time Accumulation (WPTA) Mode
- Modulus Counter (MC) Mode
- Modulus Counter Buffered (MCB) Mode
- Output Pulse Width and Frequency Modulation (OPWFM) Mode

- Output Pulse-Width and Frequency Modulation Buffered (OPWFMB) Mode
- Center Aligned Output Pulse Width Modulation with Dead Time (OPWMC) Mode
- Center-Aligned Output PWM Buffered with Dead Time (OPWMCB) Mode
- Output Pulse Width Modulation (OPWM) Mode
- Output Pulse-Width Modulation Buffered (OPWMB) Mode
- Output Pulse-Width Modulation Buffered (OPWMB) Mode

18.1.4 eMIOS200 Channel Configurations

Table 18-1 lists the eMIOS modes supported on the unified channels for this device. All included modes are on all channels.

Table 18-1. eMIOS Modes

Description	Name	Section/Page
General Purpose Input / Output	GPIO	18.4.1.1.1/18-20
Single Action Input Capture	SAIC	18.4.1.1.2/18-21
Single Action Output Compare	SAOC	18.4.1.1.3/18-22
Input Pulse-Width Measurement	IPWM	18.4.1.1.4/18-24
Input Period Measurement	IPM	18.4.1.1.5/18-25
Double Action Output Compare	DAOC	18.4.1.1.6/18-27
Pulse Edge Accumulation	PEA	18.4.1.1.7/18-29
Pulse Edge Counting	PEC	18.4.1.1.8/18-31
Quadrature Decode	QDEC	18.4.1.1.9/18-33
Windowed Programmable Time Accumulation	WPTA	18.4.1.1.10/18-34
Modulus Counter	MC	18.4.1.1.11/18-35
Modulus Counter Buffered (Up / Down)	MCB	18.4.1.1.12/18-37
Output Pulse Width and Frequency Modulation	OPWFM	18.4.1.1.13/18-40
Output Pulse Width and Frequency Modulation Buffered	OPWFMB	18.4.1.1.14/18-41
Center-Aligned Output PWM with Dead Time	OPWMC	18.4.1.1.15/18-46
Center-Aligned Output PWM Buffered with Dead Time	OPWMCB	18.4.1.1.16/18-49
Output Pulse Width Modulation	OPWM	18.4.1.1.18/18-56
Output Pulse Width Modulation Buffered	OPWMB	18.4.1.1.18/18-56

18.2 External Signal Description

Refer to Section 2.2, “External Signal Descriptions, Pin Multiplexing, and Attributes”, and Section 2.3, “Detailed Signal Description”, for detailed signal descriptions.

18.2.1 eMIOS[*n*]

eMIOS[*n*] are the eMIOS channel pins. When used as an input, an eMIOS[*n*] signal is available to be read by the MCU through the UCIN bit in the EMIOS_CSR[*n*] register. When used as an output, the UCOUT bit of the EMIOS_CSR[*n*] register reflects the state of the output pin.

NOTE

> All eMIOS channels support both input and output functions, however some channels do not support input from the physical pins, but through internal routing logic. Refer to Table 2-4 for complete details of the I/O capability of each channel.

18.2.2 Output Disable Input — eMIOS200 Output Disable Input Signal

Each output pin may be disabled by the assertion of a selected eMIOS channel FLAG bit from the EMIOS_GFR register described in 18.3.2.2/18-9. The choice of FLAG is defined by the ODIS and ODISSL bit fields of the EMIOS_CCR[n] described in section 18.3.2.7/18-12.

18.3 Memory Map and Register Description

This section provides a detailed description of all eMIOS200 registers.

18.3.1 Memory Map

The eMIOS200 memory map is shown in Table 18-2. The address of each register is given as an offset to the eMIOS200 base address. Registers are listed in address order, identified by complete name and mnemonic, and lists the type of accesses allowed.

Table 18-2. eMIOS200 Memory Map

Offset from EMIOS_BASE (0xC3FA_0000)	Register	Bits	Access[1]	Reset Value	Section/Page
Global Registers					
0x0000	EMIOS_MCR—Module Configuration Register	32	R/W	0x0000_0000	18.3.2.1/18-8
0x0004	EMIOS_GFR—Global Flag Register	32	R	0x0000_0000	18.3.2.2/18-9
0x0008	EMIOS_OUDR—Output Update Disable Register	32	R/W	0x0000_0000	18.3.2.3/18-10
0x000C–0x001F	Reserved				
Unified Channel 0 Registers					
0x0020	EMIOS_CADR[0]—Channel A Data Register	32	R/W	0x0000_0000	18.3.2.4/18-10
0x0024	EMIOS_CBDR[0]—Channel B Data Register	32	R/W	0x0000_0000	18.3.2.5/18-11
0x0028	EMIOS_CCNTR[0]—Channel Counter Register	32	R	0x0000_0000	18.3.2.6/18-12
0x002C	EMIOS_CCR[0]—Channel Control Register	32	R/W	0x0000_0000	18.3.2.7/18-12
0x0030	EMIOS_CSR[0]—Channel Status Register	32	R	0x0000_0000	18.3.2.8/18-18
0x0034	EMIOS_ALTA[0][2]—Alternate A Register	32	R/W	0x0000_0000	18.3.2.9/18-19
0x0038–0x003F	Reserved				
Unified Channel 1 Registers					
0x0040	EMIOS_CADR[1]—A Register	32	R/W	0x0000_0000	18.3.2.4/18-10
0x0044	EMIOS_CBDR[1]—B Register	32	R/W	0x0000_0000	18.3.2.5/18-11
0x0048	EMIOS_CCNTR[1]—Counter Register	32	R	0x0000_0000	18.3.2.6/18-12
0x004C	EMIOS_CCR[1]—Control Register	32	R/W	0x0000_0000	18.3.2.7/18-12
0x0050	EMIOS_CSR[1]—Status Register	32	R	0x0000_0000	18.3.2.8/18-18
0x0054	EMIOS_ALTA[1][2]—Alternate A Register	32	R/W	0x0000_0000	18.3.2.9/18-19
0x0058–0x005F	Reserved				
Unified Channel 2 Registers					
0x0060	EMIOS_CADR[2]—A Register	32	R/W	0x0000_0000	18.3.2.4/18-10
0x0064	EMIOS_CBDR[2]—B Register	32	R/W	0x0000_0000	18.3.2.5/18-11

Table 18-2. eMIOS200 Memory Map (continued)

Offset from EMIOS_BASE (0xC3FA_0000)	Register	Bits	Access[1]	Reset Value	Section/Page
0x0068	EMIOS_CCNTR[2]—Counter Register	32	R	0x0000_0000	18.3.2.6/18-12
0x006C	EMIOS_CCR[2]—Control Register	32	R/W	0x0000_0000	18.3.2.7/18-12
0x0070	EMIOS_CSR[2]—Status Register	32	R	0x0000_0000	18.3.2.8/18-18
0x0074	EMIOS_ALTA[2][2]—Alternate A Register	32	R/W	0x0000_0000	18.3.2.9/18-19
0x0078–0x007F	Reserved				
Unified Channel 3–31 Registers					
0x0080–0x041F	Same as other Channel Registers (e.g. EMIOS_CADR[2], EMIOS_CBDR[2], etc.)	32	—	—	—
0x0420–0x3FFF	Reserved				

[1] Note that R/W registers may contain some read-only or write-only bits.

[2] The alternate address register provides and alternate read-only address to access A2 channel registers in pulse edge counting (PEC) and windowed programmable time accumulation (WPTA) modes. IF EMIOS_CADR[*n*] register is used with EMIOS_ALTA[*n*], both A1 and A2 registers can be accessed in these modes.

Table 18-3. Unified Channel Base Offsets

Unified Channel	Offset from EMIOS_BASE (0xC3FA_0000)	Unified Channel	Offset from EMIOS_BASE (0xC3FA_0000))
Unified Channel 0	0x0020	Unified Channel 16	0x0220
Unified Channel 1	0x0040	Unified Channel 17	0x0240
Unified Channel 2	0x0060	Unified Channel 18	0x0260
Unified Channel 3	0x0080	Unified Channel 19	0x0280
Unified Channel 4	0x00A0	Unified Channel 20	0x02A0
Unified Channel 5	0x00C0	Unified Channel 21	0x02C0
Unified Channel 6	0x00E0	Unified Channel 22	0x02E0
Unified Channel 7	0x0100	Unified Channel 23	0x0300
Unified Channel 8	0x0120	Unified Channel 24	0x0320
Unified Channel 9	0x0140	Unified Channel 25	0x0340
Unified Channel 10	0x0160	Unified Channel 26	0x0360
Unified Channel 11	0x0180	Unified Channel 27	0x0380
Unified Channel 12	0x01A0	Unified Channel 28	0x03A0
Unified Channel 13	0x01C0	Unified Channel 29	0x03C0
Unified Channel 14	0x01E0	Unified Channel 30	0x03E0
Unified Channel 15	0x0200	Unified Channel 31	0x0400

18.3.2 Register Descriptions

This section lists the eMIOS200 registers in address order and describes the registers and their bit fields.

18.3.2.1 eMIOS200 Module Configuration Register (EMIOS_MCR)

The EMIOS_MCR contains global control bits for the eMIOS200 block.

Offset: EMIOS_BASE + 0x0000

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R		MDIS	FRZ	GTBE	ETB	GPREN	0	0	0	0	0	0	SRV[0:3]			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	GPRE[0:7]								0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-2. eMIOS200 Module Configuration Register (EMIOS_MCR)

Table 18-4. EMIOS_MCR Field Descriptions

Field	Description
0	Reserved. **Note:** Writing to this bit updates the register value, and reading it returns the last value written, but the bit has no other effect.
1 MDIS	Module Disable Bit. Puts the eMIOS200 in low-power mode. The MDIS bit is used to stop the clock of the block, except the access to registers EMIOS_MCR and EMIOS_OUDR. 0 Clock is running. 1 Enter low-power mode.
2 FRZ	Freeze Bit. Enables the eMIOS200 to freeze the registers of the unified channels when debug mode is requested at MCU level. Each unified channel must have FREN bit set in order to enter freeze mode. While in freeze mode, the eMIOS200 continues to operate to allow the MCU access to the unified channel registers. The unified channel remains frozen until the FRZ bit is written to 0 or the MCU exits debug mode or the unified channel FREN bit is cleared. 0 Exit freeze mode. 1 Stops unified channel operation when in debug mode and the FREN bit is set in the EMIOS_CCR[*n*] register.
3 GTBE	Global Time Base Enable Bit. The GTBE bit is used to export a global time base enable from the module and provide a method to start time bases of several blocks simultaneously. 0 Global time base enable out signal negated. 1 Global time base enable out signal asserted. **Note:** The global time base enable input pin controls the internal counters. When asserted, internal counters are enabled. When negated, internal counters are disabled.
4 ETB	External Time Base Bit. The ETB bit selects the time base source that drives counter bus[A]. 0 Counter bus[A] assigned to Unified Channel 23 1 STAC drives counter bus [A] If ETB is set to select STAC as the counter bus[A] source, the GTBE must be set to enable the STAC to counter bus[A]. See the STAC bus configuration register (ETPU_REDCR) section of the eTPU chapter for more information about the STAC.

Table 18-4. EMIOS_MCR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>5
GPREN</td><td>Global Prescaler Enable Bit. The GPREN bit enables the prescaler counter.
0 Prescaler disabled and prescaler counter is cleared.
1 Prescaler enabled.</td></tr>
<tr><td>6–11</td><td>Reserved</td></tr>
<tr><td>12–15
SRV</td><td>Server time slot. Selects the address of a specific STAC server to which the STAC client submodule is assigned. See Section 18.4.3, “STAC Client Submodule”.
0000 eTPU engine A, TCR1
0001 eTPU engine B, TCR1
0010 eTPU engine A, TCR2
0011 eTPU engine B, TCR2
0100 eTPU engine C, TCR1
0101 eTPU engine C, TCR2
0110-1111 Reserved</td></tr>
<tr><td>16–23
GPRE</td><td>Global Prescaler Bits. The GPRE bits select the clock divider value for the global prescaler.
<table>
<tr><th>GPRE</th><th>Divide Ratio</th></tr>
<tr><td>0000_0000</td><td>1</td></tr>
<tr><td>0000_0001</td><td>2</td></tr>
<tr><td>0000_0010</td><td>3</td></tr>
<tr><td>0000_0011</td><td>4</td></tr>
<tr><td>.
.
.</td><td>.
.
.</td></tr>
<tr><td>1111_1110</td><td>255</td></tr>
<tr><td>1111_1111</td><td>256</td></tr>
</table>
</td></tr>
<tr><td>24–31</td><td>Reserved</td></tr>
</table>

18.3.2.2 eMIOS200 Global Flag Register (EMIOS_GFR)

Offset: EMIOS_BASE + 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	F31	F30	F29	F28	F27	F26	F25	F24	F23	F22	F21	F20	F19	F18	F17	F16
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	F15	F14	F13	F12	F11	F10	F9	F8	F7	F6	F5	F4	F3	F2	F1	F0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-3. eMIOS200 Global Flag Register (EMIOS_GFR)

Table 18-5. EMIOS_GFR Field Descriptions

Field	Description
0–31 F*n*	FLAG Bits. The EMIOS_GFR is a read-only register that groups the FLAG bits from all channels. These bits are mirrors of the FLAG bits of each channel register (EMIOS_CSR[*n*]).

18.3.2.3 eMIOS200 Output Update Disable Register (EMIOS_OUDR)

Offset: EMIOS_BASE + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	OU31	OU30	OU29	OU28	OU27	OU26	OU25	OU24	OU23	OU22	OU21	OU20	OU19	OU18	OU17	OU16
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	OU15	OU14	OU13	OU12	OU11	OU10	OU9	OU8	OU7	OU6	OU5	OU4	OU3	OU2	OU1	OU0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-4. eMIOS200 Output Update Disable Register (EMIOS_OUDR)

Table 18-6. EMIOS_OUDR Field Descriptions

Field	Description
0–31 OU*n*	Channel [*n*] Output Update Disable Bits. When running MC, MCB, or an output mode, values are written to registers A2 and B2. OU*n* bits are used to disable transfers from registers A2 to A1 and B2 to B1. Each bit controls one channel. 0 Transfer enabled. Depending on the operation mode, transfer may occur immediately or in the next period. Unless stated otherwise, transfer occurs immediately. 1 Transfers disabled.

18.3.2.4 eMIOS200 A Register (EMIOS_CADR[*n*])

Offset: UC[*n*] base address + 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	A[0:23]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	A[0:23]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-5. eMIOS200 A Register (EMIOS_CADR[*n*])

Depending on the mode of operation, internal registers A1 or A2, used for matches and captures, can be assigned to address EMIOS_CADR[*n*]. A1 and A2 are cleared by reset. Table 18-7 summarizes the EMIOS_CADR[*n*] writing and reading accesses for all operation modes. For more information see Section 18.4.1.1, “Unified Channel Modes of Operation”.

18.3.2.5 eMIOS200 B Register (EMIOS_CBDR[*n*])

Offset: UC[*n*] base address + 0x0004 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	B[0:23]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	B[0:23]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-6. eMIOS200 B Register (EMIOS_CBDR[*n*])

Depending on the mode of operation, internal registers B1 or B2 can be assigned to address EMIOS_CBDR[*n*]. Both B1 and B2 are cleared by reset. Table 18-7 summarizes the EMIOS_CBDR writing and reading accesses for all operation modes. For more information see section Section 18.4.1.1, "Unified Channel Modes of Operation".

Depending on the channel configuration, it may have the EMIOS_CBDR register or not. This means that if at least one mode that requires the register is implemented, then the register is present. Otherwise, it is absent. MPC5676R has register B (EMIOS_CBDR) in all channels.

Table 18-7. EMIOS_CADR[*n*], EMIOS_CBDR[*n*], and EMIOS_ALTA[*n*] Values Assignment

Operation Mode	Register Access					
	Write	Read	Write	Read	Alternate Write	Alternate Read
GPIO	A1, A2	A1	B1, B2	B1	A2	A2
SAIC[1]	—	A2	B2	B2	—	—
SAOC[1]	A2	A1	B2	B2	—	—
IPWM	—	A2	—	B1	—	—
IPM	—	A2	—	B1	—	—
DAOC	A2	A1	B2	B1	—	—
PEA	A1	A2	—	B1	—	—
PEC[1]	A1	A1	B1	B1	—	A2
QDEC[1]	A1	A1	B2	B2	—	—
WPTA	A1	A1	B1	B1	—	A2
MC[1]	A2	A1	B2	B2	—	—
OPWFM	A2	A1	B2	B1	—	—
OPWMC	A2	A1	B2	B1	—	—
OPWM	A2	A1	B2	B1	—	—
MCB[1]	A2	A1	B2	B2	—	—

Table 18-7. EMIOS_CADR[*n*], EMIOS_CBDR[*n*], and EMIOS_ALTA[*n*] Values Assignment (continued)

Operation Mode	Register Access					
	Write	Read	Write	Read	Alternate Write	Alternate Read
OPWFMB	A2	A1	B2	B1	—	—
OPWMCB	A2	A1	B2	B1	—	—
OPWMB	A2	A1	B2	B1	—	—

[1] In these modes, the register EMIOS_CBDR[*n*] is not used, but B2 can be accessed.

18.3.2.6 eMIOS200 Counter Register (EMIOS_CCNTR[*n*])

Offset: UC[*n*] base address + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	C[0:23]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	C[0:23]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] In GPIO mode or freeze action, this register is writable.

Figure 18-7. eMIOS200 Counter Register (EMIOS_CCNTR[*n*])

The EMIOS_CCNTR[*n*] register contains the value of the internal counter for eMIOS channel *n*. When GPIO mode is selected or the channel is frozen, the EMIOS_CCNTR[*n*] register is read/write. For all other modes, the EMIOS_CCNTR[*n*] is a read-only register. When entering some operation modes, this register is automatically cleared (refer to Section 18.4.1.1, “Unified Channel Modes of Operation”, for details).

Depending on the channel configuration it may have an internal counter or not. It means that if at least one mode that requires the counter is implemented, then the counter is present, otherwise it is absent.

18.3.2.7 eMIOS200 Control Register (EMIOS_CCR[*n*])

Offset: UC[*n*] base address + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FREN	ODIS	ODISSL		UCPRE		UC PREN	DMA	0	IF				FCK	FEN	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	FORC MA	FORC MB	0	BSL		ED SEL	ED POL	MODE						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-8. eMIOS200 Control Register (EMIOS_CCR[*n*])

The control register gathers bits reflecting the status of the unified channel input/output signals and the overflow condition of the internal counter, as well as several read/write control bits.

Table 18-8. EMIOS_CCR[*n*] Field Descriptions

Field	Description
0 FREN	Freeze Enable Bit. The FREN bit, if set and validated by FRZ bit in EMIOS_MCR register, freezes all registers' values when in debug mode, allowing the MCU to perform debug functions. 0 Normal operation. 1 Freeze unified channel registers' values.
1 ODIS	Output Disable Bit. The ODIS bit allows disabling the output pin when running any of the output modes with the exception of GPIO mode. 0 The output pin operates normally. 1 If the selected output disable input signal is asserted, the output pin goes to EDPOL for OPWFMB and OPWMB modes and to the complement of EDPOL for other modes, but the unified channel continues to operate normally, i.e., it continues to produce FLAG and matches. When the selected output disable input signal is negated, the output pin operates normally.
2–3 ODISSL	Output Disable Select Bits. The ODISSL bits select an eMIOS channel flag to disable an output when the flag asserts. ODISSL 00 = eMIOS Channel Flag Number 11 ODISSL 01 = eMIOS Channel Flag Number 10 ODISSL 10 = eMIOS Channel Flag Number 9 ODISSL 11 = eMIOS Channel Flag Number 8
4–5 UCPRE	Prescaler Bits. The UCPRE bits select the clock divider value for the internal prescaler of unified channel. UCPRE 00 = Divide Ratio 1 UCPRE 01 = Divide Ratio 2 UCPRE 10 = Divide Ratio 3 UCPRE 11 = Divide Ratio 4
6 UCPREN	Prescaler Enable Bit. The UCPREN bit enables the prescaler counter. 0 Prescaler disabled. 1 Prescaler enabled and the prescaler counter is loaded with UCCPRE value.
7 DMA	Direct Memory Access Bit. The DMA bit selects whether the FLAG generation is used as an interrupt or as a DMA request. 0 FLAG/overrun assigned to interrupt request. 1 FLAG/overrun assigned to DMA request.
8	Reserved

ODISSL sub-table (field 2–3):

ODISSL	eMIOS Channel Flag Number
00	11
01	10
10	9
11	8

UCPRE sub-table (field 4–5):

UCPRE	Divide Ratio
00	1
01	2
10	3
11	4

Table 18-8. EMIOS_CCR[*n*] Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>9–12
IF</td><td>Input Filter Bits. The IF bits control the programmable input filter, selecting the minimum input pulse width that can pass through the filter. For output modes, these bits have no meaning.
<table>
<tr><th>IF[1]</th><th>Minimum Input Pulse Width [FLT_CLK Periods]</th></tr>
<tr><td>0000</td><td>Bypassed[2]</td></tr>
<tr><td>0001</td><td>02</td></tr>
<tr><td>0010</td><td>04</td></tr>
<tr><td>0100</td><td>08</td></tr>
<tr><td>1000</td><td>16</td></tr>
<tr><td>All others</td><td>Reserved</td></tr>
</table>
[1] Filter latency is three clock edges.
[2] The input signal is synchronized before arriving to the digital filter.</td></tr>
<tr><td>13
FCK</td><td>Filter Clock Select Bit. The FCK bit selects the clock source for the programmable input filter.
0 Prescaled clock.
1 Main clock.</td></tr>
<tr><td>14
FEN</td><td>FLAG Enable Bit. The FEN bit allows the unified channel FLAG bit to generate an interrupt signal or a DMA request signal (the type of signal to be generated is defined by the DMA bit).
0 Disable (FLAG does not generate an interrupt or DMA request).
1 Enable (FLAG generates an interrupt or DMA request).</td></tr>
<tr><td>15–17</td><td>Reserved</td></tr>
<tr><td>18
FORCMA</td><td>Force Match A Bit. For output modes, the FORCMA bit is equivalent to a successful comparison on comparator A (except that the FLAG bit is not set). This bit is cleared by reset and is always read as 0. This bit is valid for every output operation mode which uses comparator A, otherwise it has no effect.
0 Has no effect.
1 Force a match at comparator A.
For input modes, the FORCMA bit is not used and writing to it has no effect.</td></tr>
<tr><td>19
FORCMB</td><td>Force Match B Bit. For output modes, the FORCMB bit is equivalent to a successful comparison on comparator B (except that the FLAG bit is not set). This bit is cleared by reset and is always read as 0. This bit is valid for every output operation mode which uses comparator B, otherwise it has no effect.
0 Has no effect.
1 Force a match at comparator B.
For input modes, the FORCMB bit is not used and writing to it has no effect.</td></tr>
<tr><td>20</td><td>Reserved</td></tr>
</table>

Table 18-8. EMIOS_CCR[*n*] Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>21–22
BSL</td><td>Bus Select Bits. The BSL bits are used to select either one of the counter buses or the internal counter to be used by the unified channel.
<table>
<tr><th>BSL</th><th>Selected Bus</th></tr>
<tr><td>00</td><td>All channels: counter bus[A]</td></tr>
<tr><td>01</td><td>Channels 0 to 7: counter bus[B]
Channels 8 to 15: counter bus[C]
Channels 16 to 23: counter bus[D]
Channels 24 to 31: counter bus[E]</td></tr>
<tr><td>10</td><td>Reserved</td></tr>
<tr><td>11</td><td>All channels: internal counter</td></tr>
</table></td></tr>
<tr><td>23
EDSEL</td><td>Edge Selection Bit. For input modes, the EDSEL bit selects if the internal counter is triggered by both edges of a pulse or by a single edge only as defined by the EDPOL bit. When not shown in the mode of operation description, this bit has no effect.
0 Single edge triggering defined by the EDPOL bit.
1 Both edges triggering.

For GPIO in mode, the EDSEL bit selects if a FLAG can be generated.
0 A FLAG is generated as defined by the EDPOL bit.
1 No FLAG is generated.

For SAOC mode, the EDSEL bit selects the behavior of the output flip-flop at each match.
0 The EDPOL value is transferred to the output flip-flop.
1 The output flip-flop is toggled.</td></tr>
</table>

Table 18-8. EMIOS_CCR[*n*] Field Descriptions (continued)

Field	Description
24 EDPOL	Edge Polarity Bit. For input modes, the EDPOL bit asserts which edge triggers either the internal counter or an input capture or a FLAG. When not shown in the mode of operation description, this bit has no effect. 0 Trigger on a falling edge. 1 Trigger on a rising edge. For QDEC (MODE[6] cleared), the EDPOL bit selects the count direction according to *direction* signal (UC[n] input). 0 counts down when UC[n] is asserted 1 counts up when UC[n] is asserted **Note:** UC[n-1] EDPOL bit selects which edge clocks the internal counter of UC[n] 0 Trigger on a falling edge 1 Trigger on a rising edge For QDEC (MODE[6] set), the EDPOL bit selects the count direction according to the phase difference. 0 internal counter decrements if phase_A is ahead phase_B signal 1 internal counter increments if phase_A is ahead phase_B signal **Note:** In order to operate properly, EDPOL bit must contain the same value in UC[n] and UC[n-1] For output modes, the EDPOL bit is used to select the logic level on the output pin. 0 A match on comparator A clears the output flip-flop, while a match on comparator B sets it. 1 A match on comparator A sets the output flip-flop, while a match on comparator B clears it.
25–31 MODE	Mode Selection Bits. The MODE bits select the mode of operation of the unified channel, as shown in Table 18-9. Refer to Table 18-1 for more information on the different modes. **Note:** If a reserved value is written to MODE, the results are unpredictable.

Table 18-9. MODE Bits

MODE	Mode	Description
000_0000	GPIO (input)	General-Purpose Input/Output mode (input)
000_0001	GPIO (output)	General-Purpose Input/Output mode (output)
000_0010	SAIC	Single Action Input Capture
000_0011	SAOC	Single Action Output Compare
000_0100	IPWM	Input Pulse Width Measurement
000_0101	IPM	Input Period Measurement
000_0110	DAOC	Double Action Output compare (with FLAG set on B match)
000_0111	DAOC	Double Action Output compare (with FLAG set on both match)
000_1000	PEA	Pulse/Edge Accumulation (continuous)
000_1001	PEA	Pulse/Edge Accumulation (single-shot)
000_1010	PEC	Pulse/Edge Counting (continuous)
000_1011	PEC	Pulse/Edge Counting (single-shot)
000_1100	QDEC	Quadrature Decode (for count & direction encoders type)

Table 18-9. MODE Bits (continued)

MODE	Mode	Description
000_1101	QDEC	Quadrature Decode (for phase_A & phase_B encoders type)
000_1110	WPTA	Windowed Programmable Time Accumulation
000_1111	Reserved	
001_0000	MC	Modulus Counter (Up counter with clear on match start, internal clock)
001_0001	MC	Modulus Counter (Up counter with clear on match start, external clock)
001_0010	MC	Modulus Counter (Up counter with clear on match end, internal clock)
001_0011	MC	Modulus Counter (Up counter with clear on match end, external clock)
001_0100	MC	Modulus Counter (Up/Down counter with flag on A match, internal clock)
001_0101	MC	Modulus Counter (Up/Down counter with flag on A match, external clock)
001_0110	MC	Modulus Counter (Up/Down counter with flag on A match and cycle boundary, internal clock)
001_0111	MC	Modulus Counter (Up/Down counter with flag on A match and cycle boundary, external clock)
001_1000	OPWFM	Output Pulse Width and Frequency Modulation (flag on B match, immediate update)
001_1001	OPWFM	Output Pulse Width and Frequency Modulation (flag on B match, next period update)
001_1010	OPWFM	Output Pulse Width and Frequency Modulation (flag on both A and B matches, immediate update)
001_1011	OPWFM	Output Pulse Width and Frequency Modulation (flag on both A and B matches, next period update)
001_1100	OPWMC	Center Aligned Output Pulse Width Modulation (flag in trailing edge, trailing edge dead time)
001_1101	OPWMC	Center Aligned Output Pulse Width Modulation (flag in trailing edge, leading edge dead time)
001_1110	OPWMC	Center Aligned Output Pulse Width Modulation (flag in both edges, trailing edge dead time)
001_1111	OPWMC	Center Aligned Output Pulse Width Modulation (flag in both edges, leading edge dead time)
010_0000	OPWM	Output Pulse Width Modulation (flag on B match, immediate update)
010_0001	OPWM	Output Pulse Width Modulation (flag on B match, next period update)
010_0010	OPWM	Output Pulse Width Modulation (flag on both matches, immediate update)
010_0011	OPWM	Output Pulse Width Modulation (flag on both matches, next period update)
010_0100 – 100_1111	Reserved	
101_0000	MCB	Modulus Counter Buffered (Up counter with clear on match start, internal clock)
101_0001	MCB	Modulus Counter Buffered (Up counter with clear on match start, external clock)
101_0010 – 101_0011	Reserved	
101_0100	MCB	Modulus Counter Buffered (Up/Down counter with flag on A match, internal clock)

Table 18-9. MODE Bits (continued)

MODE	Mode	Description
101_0101	MCB	Modulus Counter Buffered (Up/Down counter with flag on A match, external clock)
101_0110	MCB	Modulus Counter Buffered (Up/Down counter with flag on A match and cycle boundary, internal clock)
101_0111	MCB	Modulus Counter Buffered (Up/Down counter with flag on A match and cycle boundary, external clock)
101_1000	OPWFMB	Output Pulse Width and Frequency Modulation Buffered, (flag on B match)
101_1001	Reserved	
101_1010	OPWFMB	Output Pulse Width and Frequency Modulation Buffered, (flag on both A and B matches)
101_1011	Reserved	
101_1100	OPWMCB	Center Aligned Output Pulse Width Modulation Buffered (flag in trailing edge, trailing edge dead time)
101_1101	OPWMCB	Center Aligned Output Pulse Width Modulation Buffered (flag in trailing edge, leading edge dead time)
101_1110	OPWMCB	Center Aligned Output Pulse Width Modulation Buffered (flag in both edges, trailing edge dead time)
101_1111	OPWMCB	Center Aligned Output Pulse Width Modulation Buffered (flag in both edges, leading edge dead time)
110_0000	OPWMB	Output Pulse Width Modulation Buffered (flag on B match)
110_0001	Reserved	
110_0010	OPWMB	Output Pulse Width Modulation Buffered (flag on both A and B matches)
110_0011 – 111_1111	Reserved	

18.3.2.8 eMIOS200 Status Register (EMIOS_CSR[*n*])

Offset: UC[*n*] base address + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OVR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	OVFL	0	0	0	0	0	0	0	0	0	0	0	0	UCIN	UCOUT	FLAG
W	w1c															w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-9. eMIOS200 Status Register (EMIOS_CSR[*n*])

Table 18-10. EMIOS_CSR[*n*] Field Descriptions

Field	Description
0 OVR	Overrun Bit. The OVR bit indicates that FLAG generation occurred when the FLAG bit was already set. This bit can be cleared by clearing the FLAG bit or by software writing a 1. 0 Overrun has not occurred. 1 Overrun has occurred.
1–15	Reserved
16 OVFL	Overflow Bit. The OVFL bit indicates that an overflow has occurred in the internal counter. This bit must be cleared by software writing a 1. 0 An overflow has not occurred. 1 An overflow has occurred.
17–28	Reserved
29 UCIN	Unified Channel Input Pin Bit. The UCIN bit reflects the input pin state after being filtered and synchronized.
30 UCOUT	Unified Channel Output. The UCOUT bit reflects the output pin state.
31 FLAG	FLAG Bit. The FLAG bit is set when an input capture or a match event in the comparators occurred. This bit must be cleared by software writing a 1. 0 FLAG cleared. 1 FLAG set event has occurred. **Note:** emios_flag_out reflects the FLAG bit value. When the DMA bit is set, the FLAG bit can be cleared by the DMA controller.

18.3.2.9 eMIOS200 Alternate A Register (EMIOS_ALTA[*n*])

UC[*n*] base address + 0x0014

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	ALTA[0:23]							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ALTA[0:23]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-10. eMIOS200 Alternate A Register (EMIOS_ALTA[*n*])

The EMIOS_ALTA[*n*] register provides an alternate read-only address to access A2 channel registers in GPIO, PEC, WPTA modes. If the EMIOS_CADR[*n*] register is used with EMIOS_ALTA[*n*], both A1 and A2 registers can be accessed in these modes.

18.4 Functional Description

The eMIOS200 provides independently operating unified channels (UC) that can be configured and accessed by a host MCU. The five time bases can be shared by the channels through five counter buses and each unified channel can generate its own time base.

The eMIOS200 block is reset asynchronously. All registers are cleared on reset.

18.4.1 Unified Channel (UC)

.Each unified channel consists of the following:

- A programmable clock prescaler
- Two double-buffered data registers, A and B, that allow up to two input capture and/or output compare events to occur before software intervention is needed
- Two comparators (equal only), A and B, which compare the selected counter bus with the value in the data registers
- Internal counter, which can be used as a local time base or to count input events
- Programmable input filter, which ensures that only valid pin transitions are received by channel
- Programmable input edge detector, which detects the rising, falling or either edges
- An output flip-flop, which holds the logic level to be applied to the output pin
- eMIOS200 status and control register

An output disable input selector, which selects the output disable input signal to be used as output disable

18.4.1.1 Unified Channel Modes of Operation

The mode of operation of the unified channel is determined by the mode select bits MODE in the EMIOS_CCR[*n*] register (see Table 18-9 for details).

When entering an output mode (except for GPIO mode), the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR[*n*] register.

As the internal counter EMIOS_CCNTR[*n*] continues to run in all modes (except for GPIO mode), it is possible to use this as a time base if the resource is not used in the current mode.

In order to provide smooth waveform generation even if A and B registers are changed on the fly, the MCB, OPWFMB, OPWMB and OPWMCB modes are available. In these modes, the A and B registers are double-buffered. These modes are presented in separate sections since basic differences exist between these modes and the MC, OPWFM, OPWM, and OPWMC modes, respectively.

18.4.1.1.1 General-Purpose Input/Output (GPIO) Mode

In GPIO mode, all input capture and output compare functions of the unified channel are disabled, the internal counter (EMIOS_CCNTR[*n*] register) is cleared and disabled. All control bits remain accessible. In order to prepare the unified channel for a new operation mode, writing to registers EMIOS_CADR[*n*] or EMIOS_CBDR[*n*] stores the same value in registers A1/A2 or B1/B2, respectively.

The MODE[6] bit selects between input (MODE[6] = 0) and output (MODE[6] = 1) modes.

When changing the MODE bits, the application software must go to GPIO mode first to reset the unified channel's internal functions properly. Failure to do this could lead to invalid and unexpected output compare or input capture results or the FLAGs being set incorrectly.

In GPIO input mode (MODE = 000_0000), the FLAG generation is determined according to EDPOL and EDSEL bits and the input pin status can be determined by reading the UCIN bit.

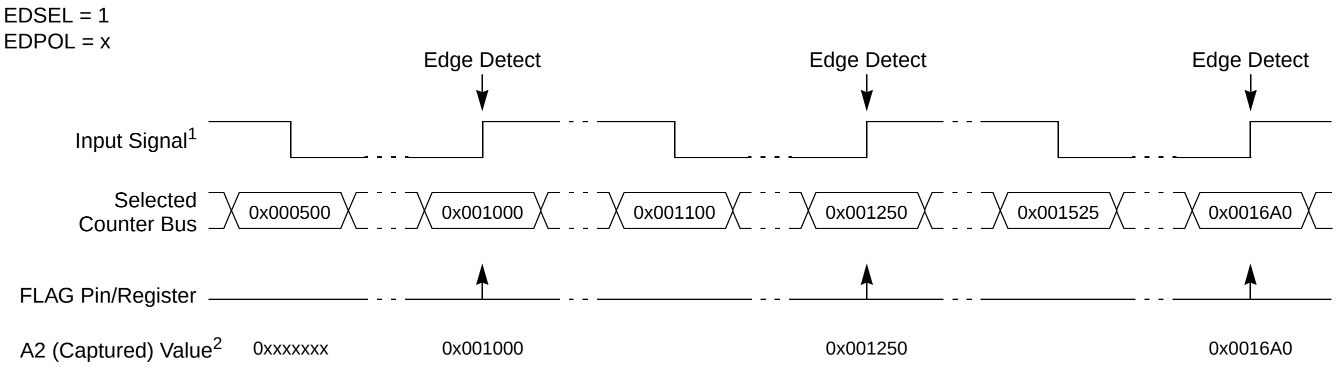
In GPIO output mode (MODE = 000_0001), the unified channel is used as a single output port pin and the value of the EDPOL bit is permanently transferred to the output flip-flop.

18.4.1.1.2 Single Action Input Capture (SAIC) Mode

In SAIC mode (MODE = 000_0010), when a triggering event occurs on the input pin, the value on the selected time base is captured into register A2. At the same time, the FLAG bit is set to indicate that an input capture has occurred. The EMIOS_CADR[*n*] register returns the value of register A2. The channel is ready to capture events as soon as SAIC mode is entered coming out from GPIO mode. The events are captured as soon as they occur, thus reading register A (EMIOS_CADR) always returns the value of the latest captured event. Subsequent captures are enabled with no need of further reads from the EMIOS_CADR[*n*] register. The FLAG bit is set at any time a new event is captured.

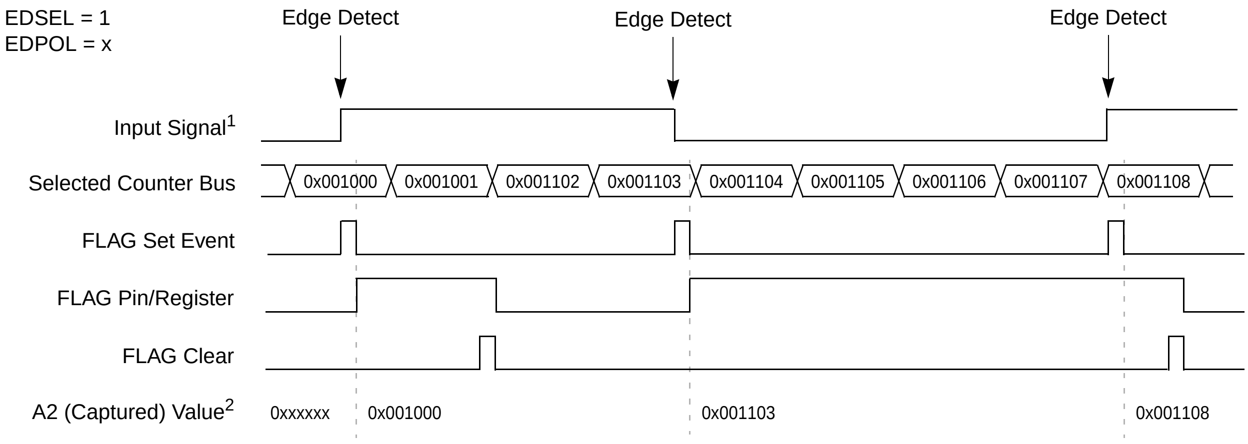
The input capture is triggered by a rising, falling, or either edge on the input pin, as configured by the EDPOL and EDSEL bits in EMIOS_CCR[*n*] register.

Figure 18-11 and Figure 18-12 show how the unified channel can be used for input capture.



Notes:
1. After input filter
2. EMIOS_CADR[*n*] ≤ A2

Figure 18-11. SAIC with Rising Edge Triggering Example



Notes: 1 After input filter
2 EMIOS_CADR[*n*] ≤ A2

Figure 18-12. SAIC with Both Edges Triggering Example

18.4.1.1.3 Single Action Output Compare (SAOC) Mode

In SAOC mode (MODE = 000_0011), a match value is loaded in register A2 and then transferred to register A1 to be compared with the selected time base. When a match occurs, the EDSEL bit selects whether the output flip-flop is toggled or the value in EDPOL is transferred to it. At the same time, the FLAG bit is set to indicate that the output compare match has occurred. Writing to the EMIOS_CADR[*n*] register stores the value in register A2 and reading to the EMIOS_CADR[*n*] register returns the value of register A1.

An output compare match can be simulated in software by setting the FORCMA bit in the EMIOS_CCR[*n*] register. In this case, the FLAG bit is not set.

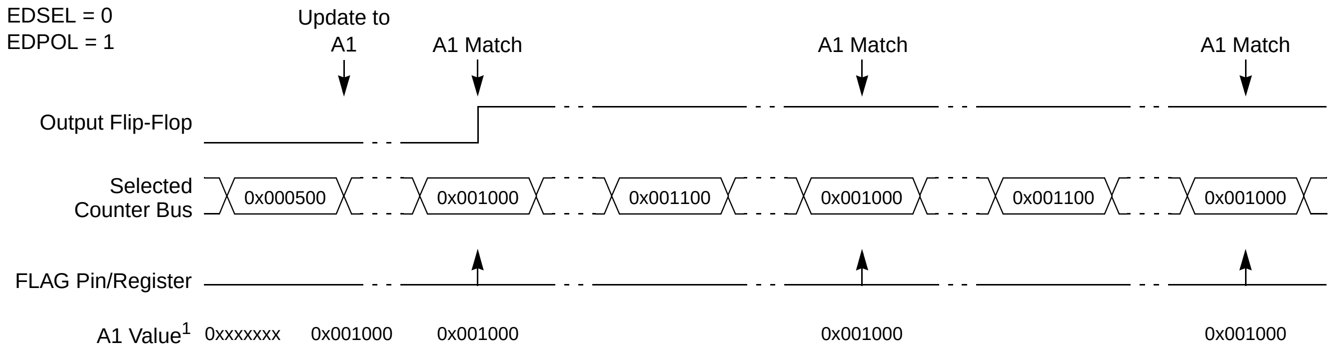
When SAOC mode is entered coming out of GPIO mode, the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR[*n*] register.

The counter bus can be either internal or external and is selected through the BSL bits.

Figure 18-13 and Figure 18-14 show how the unified channel can be used to perform a single output compare with the EDPOL value being transferred to the output flip-flop and toggling the output flip-flop at each match, respectively. Note that once in SAOC mode, the matches are enabled. Thus the desired match value on register A1 must be written before the mode is entered. Register A1 can be updated at any time, thus modifying the match value, which is reflected in the output signal generated by the channel. Subsequent matches are enabled with no need of further writes to EMIOS_CADR[*n*]. The FLAG bit is set at the same time a match occurs (see Figure 18-15).

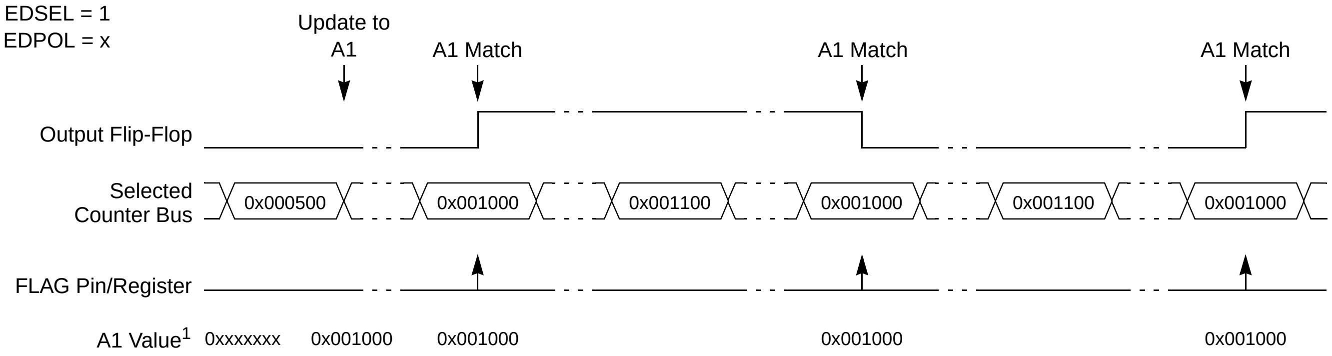
NOTE

In SAOC mode, the internal channel counter is free-running, and starts counting as soon as the SAOC mode is entered.



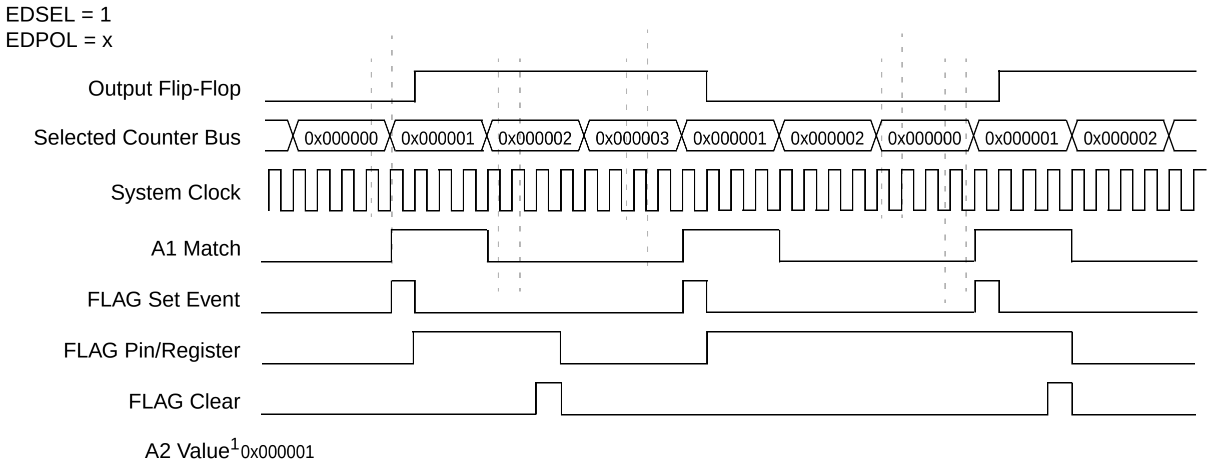
Notes: 1. EMIOS_CADR[n] = A2
A2 = A1 according to OU[*n*] bit

Figure 18-13. SAOC Example — EDPOL Value Being Transferred to the Output Flip-flop



Note: 1. EMIOS_CADR[*n*] = A2

Figure 18-14. SAOC Example — Toggling the Output Flip-Flop



Note: 1. EMIOS_CADR[*n*] ≤ A2

Figure 18-15. SAOC Example with Flag Behavior

18.4.1.1.4 Input Pulse-Width Measurement (IPWM) Mode

The IPWM mode (MODE = 000_0100) allows measuring the width of a positive or negative pulse by capturing the leading edge on register B1 and the trailing edge on register A2. Successive captures are done on consecutive edges of opposite polarity. The leading edge sensitivity (i.e., pulse polarity) is selected by the EDPOL bit in the EMIOS_CCR[*n*] register. Registers EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] return the values in register A2 and B1, respectively.

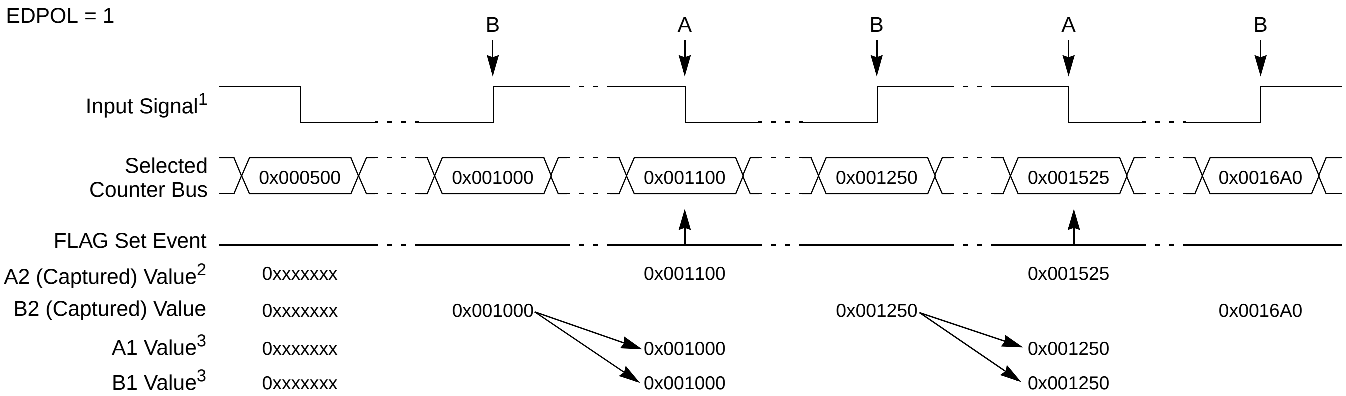
The capture function of register A2 remains disabled until the first leading edge triggers the first input capture on register B2. When this leading edge is detected, the count value of the selected time base is latched into register B2; the FLAG bit is not set. When the trailing edge is detected, the count value of the selected time base is latched into register A2. At the same time, the FLAG bit is set and the content of register B2 is transferred to register B1 and to register A1.

If subsequent input capture events occur while the corresponding FLAG bit is set, registers A2, B1, and A1 are updated with the latest captured values and the FLAG remains set. Registers EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] return the value in registers A2 and B1, respectively.

In order to guarantee coherent access, reading EMIOS_CADR[*n*] forces B1 to be updated with the content of register A1. At the same time, transfers between B2 and B1 are disabled until the next read of EMIOS_CBDR[*n*] register. Reading EMIOS_CBDR[*n*] register forces B1 be updated with A1 register content and re-enables transfers from B2 to B1, to take effect at the next trailing edge capture. Transfers from B2 to A1 are not blocked at any time.

The input pulse width is calculated by subtracting the value in B1 from A2.

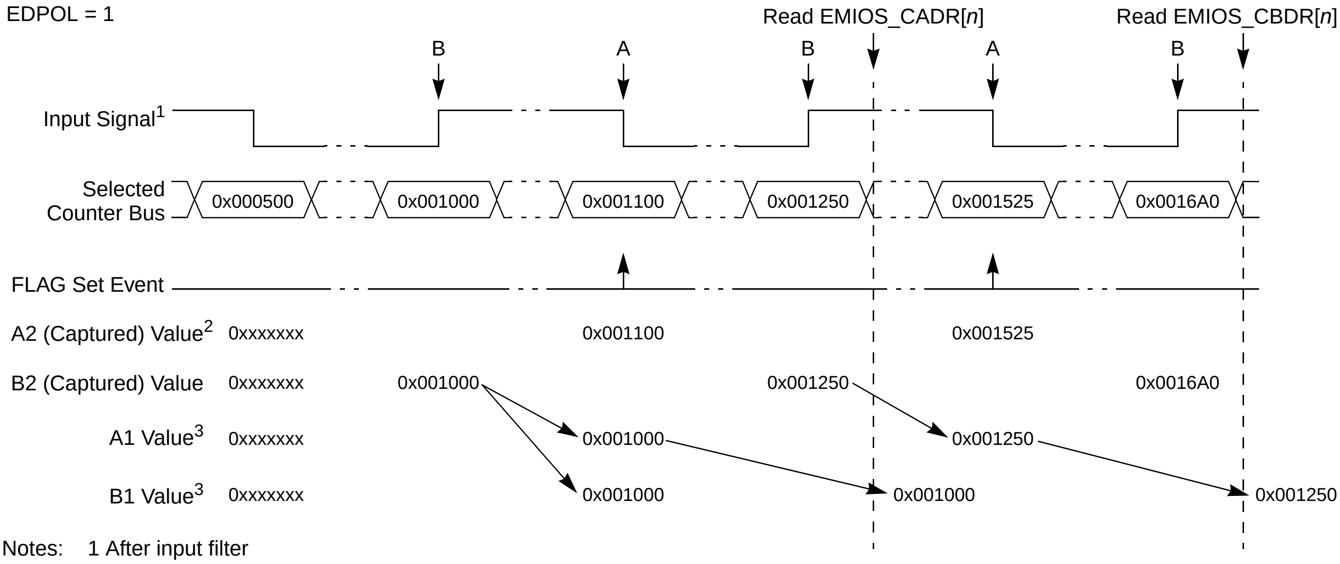
Figure 18-16 shows how the unified channel can be used for input pulse-width measurement.



Notes: 1. After input filter
2. EMIOS_CADR[*n*] = A2
3. EMIOS_CBDR[*n*] = B1

Figure 18-16. IPWM Example

Figure 18-17 shows the A1 and B1 updates when EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] register reads occur. The A1 register has always coherent data related to the A2 register. When a EMIOS_CADR[*n*] read is performed, the B1 register is loaded with the A1 register content. This guarantees that the data in register B1 always has the coherent data related to the last EMIOS_CADR[*n*] read. The B1 register updates remain locked until a EMIOS_CBDR[*n*] read occurs. If a read of EMIOS_CADR[*n*] is performed, B1 is updated with contents of A1 even if the B1 update is locked by a previous EMIOS_CADR[*n*] read operation.



Notes: 1 After input filter
2 EMIOS_CADR[*n*] = A2
3 EMIOS_CBDR[*n*] = B1

Figure 18-17. B1 and A1 Updates at EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] Reads

Reading EMIOS_CADR[*n*] followed by EMIOS_CBDR[*n*] always provides coherent data. If coherent data is not required, the sequence of reads should be inverted, and EMIOS_CBDR[*n*] should be read before EMIOS_CADR[*n*]. Even in this case, register B1 updates are blocked after EMIOS_CADR[*n*] is read. Therefore, a second EMIOS_CBDR[*n*] read is required to release the B1 register updates.

18.4.1.1.5 Input Period Measurement (IPM) Mode

The IPM mode (MODE = 000_0101) allows the measurement of the period of an input signal by capturing two consecutive rising edges or two consecutive falling edges. Successive input captures are done on consecutive edges of the same polarity. The edge polarity is defined by the EDPOL bit in the EMIOS_CCR[*n*] register.

When the first edge of selected polarity is detected, the selected time base is latched into the registers A2 and B2, and the data previously held in register B2 is transferred to register B1. On this first capture the FLAG line is not set and the values in registers B1 are meaningless. On the second and subsequent captures, the FLAG line is set and data in register B2 is transferred to register B1.

When the second edge of the same polarity is detected, the counter bus value is latched into registers A2 and B2, and the data previously held in register B2 is transferred to data register B1 and to register A1. The FLAG bit is set to indicate that the start and end points of a complete period have been captured. This sequence of events is repeated for each subsequent capture. The EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] registers return the values in the A2 and B1 registers, respectively.

To allow coherent data, reading EMIOS_CADR[*n*] forces A1 content be transferred to B1 register and disables transfers between B2 and B1. These transfers are disabled until the next read of the

EMIOS_CBDR[*n*] register. Reading EMIOS_CBDR[*n*] register forces A1 content to be transferred to B1 and re-enables transfers from B2 to B1, to take effect at the next edge capture.

The input pulse period is calculated by subtracting the value in B1 from A2.

Figure 18-18 shows how the unified channel can be used for input period measurement.

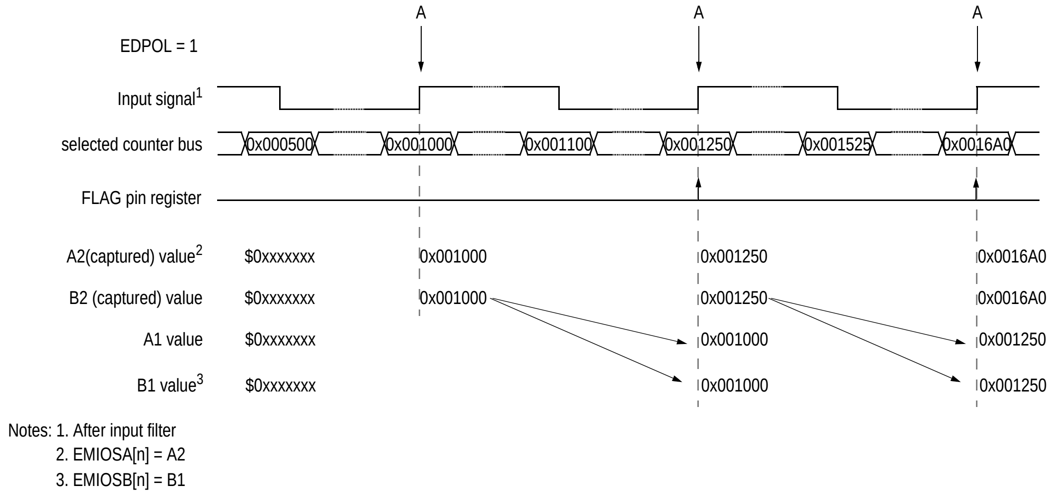


Figure 18-18. IPM Example

Figure 18-19 shows the A1 and B1 register updates when EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] read operations are performed. When a EMIOS_CADR[*n*] read occurs, the contents of A1 are transferred to B1, thus providing coherent data in the A2 and B1 registers. Transfers from B2 to B1 are then blocked until EMIOS_CBDR[*n*] is read. After EMIOS_CBDR[*n*] is read, the contents of register A1 are transferred to register B1 and the transfers from B2 to B1 are re-enabled to occur at the transfer edges, which is the leading edge in the Figure 18-19 example.

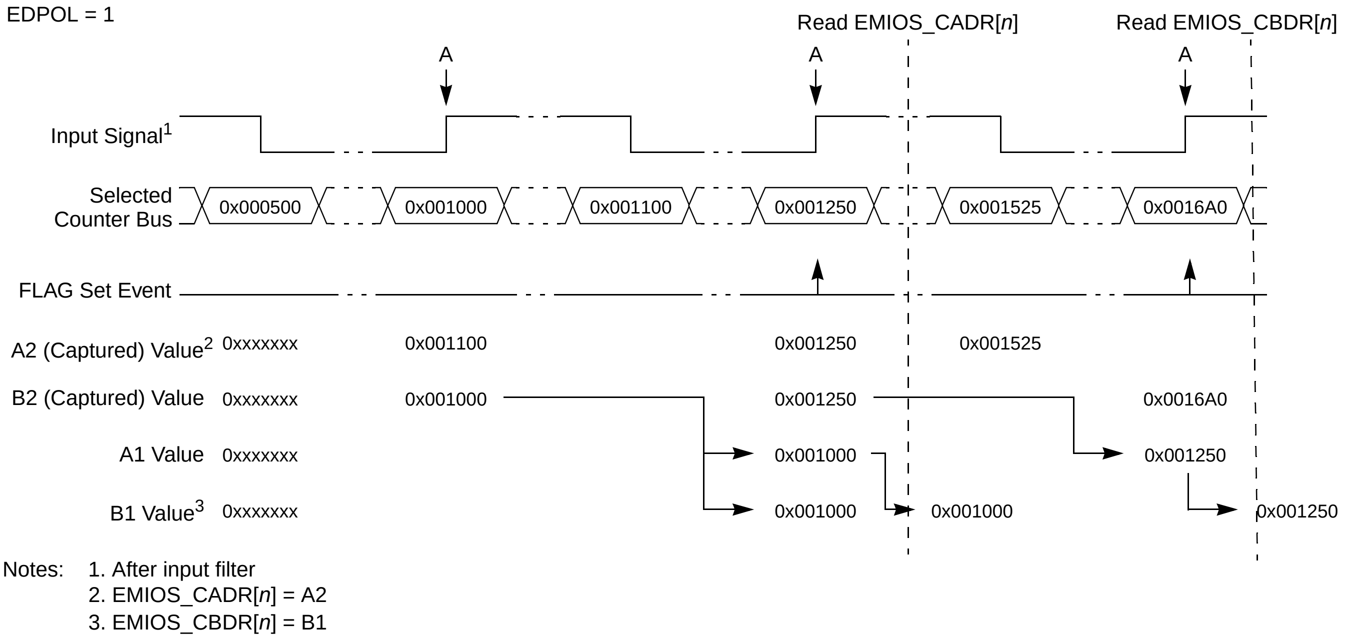


Figure 18-19. A1 and B1 Updates at EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] Reads

18.4.1.1.6 Double Action Output Compare (DAOC) Mode

In the DAOC mode the leading and trailing edges of the variable pulse-width output are generated by matches occurring on comparators A and B, respectively.

When the DAOC mode is entered (coming out of GPIO mode), both comparators are disabled and the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR[*n*] register

Data written to A2 and B2 are transferred to A1 and B1, respectively, on the next system clock cycle if the OU[*n*] bit in EMIOS_OUDR register is cleared (see Figure 18-22). The transfer is blocked if the OU[*n*] bit is set. Comparator A is enabled only after the transfer to A1 register occurs, and is disabled on the next A match. Comparator B is enabled only after the transfer to B1 register occurs, and is disabled on the next B match. Comparators A and B are enabled and disabled independently.

The output flip-flop is set to the value of EDPOL when a match occurs on comparator A and to the complement of EDPOL when a match occurs on comparator B.

MODE[6] controls if the FLAG bit is set on both matches or on the second match only (see Table 18-9 for details). FLAG bit assertion depends on comparator enabling.

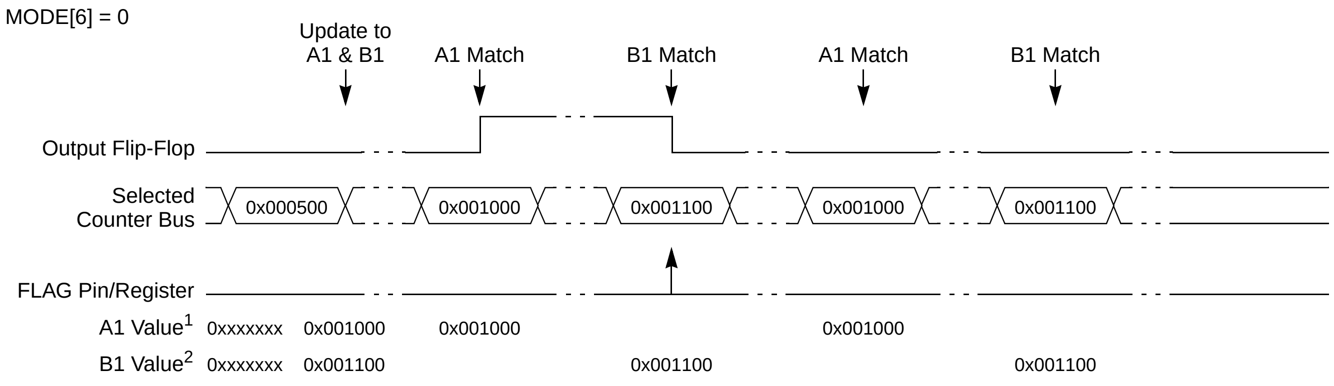
If subsequent enabled output compares occur on registers A1 and B1, pulses continue to be generated, regardless of the state of the FLAG bit.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a comparison event in comparator A or B, respectively. The FLAG bit is not affected by these forced operations.

NOTE

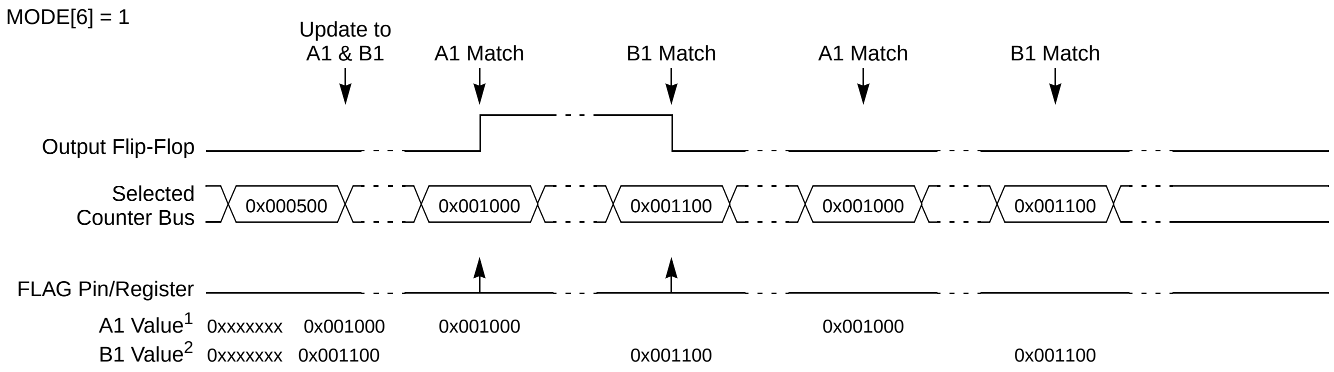
> If both A1 and B1 registers are loaded with the same value, the B match prevails concerning the output pin state. The output flip-flop is set to the complement of EDPOL, the FLAG bit is set, and both comparators are disabled.

Figure 18-20 and Figure 18-21 show how the unified channel can be used to generate a single output pulse with FLAG bit being set on the second match or on both matches, respectively.



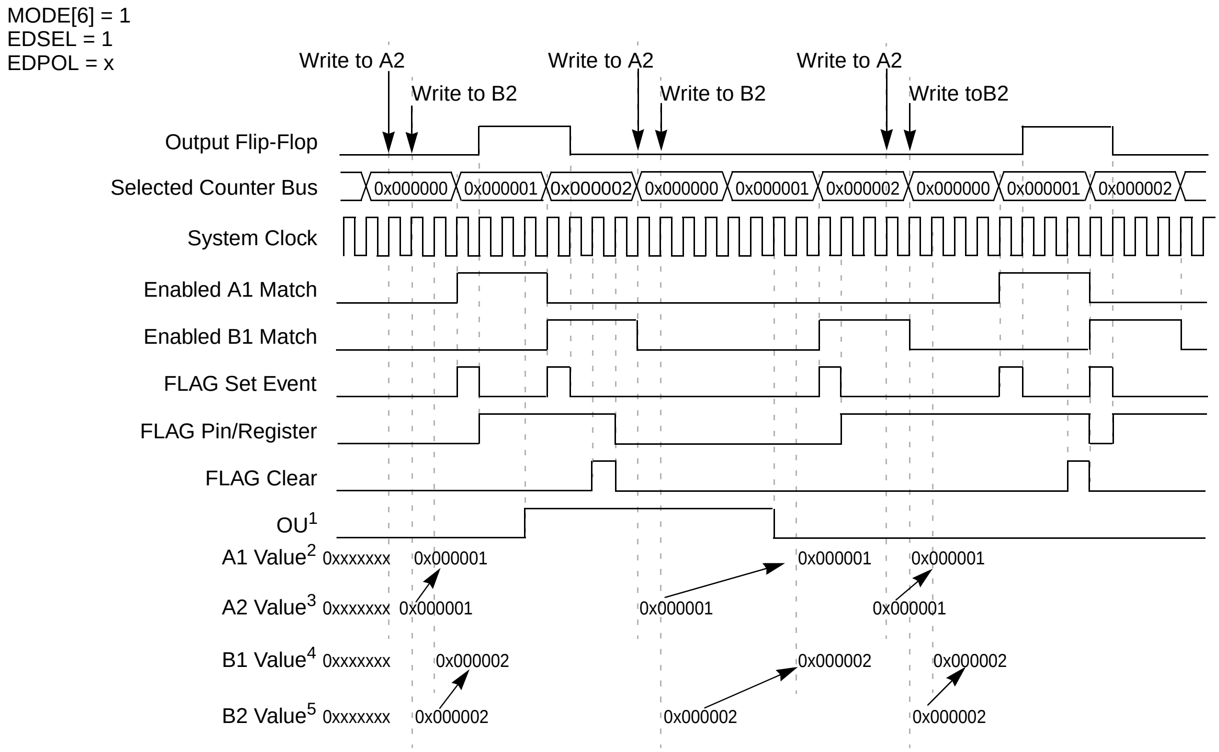
Notes: 1. EMIOS_CADR[*n*] = A1
2. EMIOS_CBDR[*n*] = B1
A2 = A1 according to OU[*n*] bit
B2 = B1 according to OU[*n*] bit

Figure 18-20. DAOC with FLAG Set on the Second Match



Notes: 1. EMIOS_CADR[*n*] = A1
2. EMIOS_CBDR[*n*] = B1
A2 = A1 according to OU[*n*] bit
B2 = B1 according to OU[*n*] bit

Figure 18-21. DAOC with FLAG Set on Both Matches



Note: 1. OU[*n*] bit of EMIOS_OUDR register

2. EMIOS_CADR[*n*] = A1 (when reading)
3. EMIOS_CADR[*n*] = A2 (when writing)
4. EMIOS_CBDR[*n*] = B1 (when reading)
5. EMIOS_CBDR[*n*] = B2 (when writing)

Figure 18-22. DAOC with Transfer Disabling Example

18.4.1.1.7 Pulse/Edge Accumulation (PEA) Mode

The PEA mode returns the time taken to detect a desired number of input events. The MODE[6] bit selects between continuous or single-shot operation.

After writing to register A1, the internal counter is cleared on the first input event, ready to start counting input events, and the selected timebase is latched into register B2. On the match between the internal counter and register A1, a counter bus capture is triggered to register A2 and B2. The data previously held in register B2 is transferred to register B1 and the FLAG bit is set to indicate that an event has occurred. The desired time interval can be determined by subtracting register B1 from A2. Registers EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] return the values in register A2 and B1, respectively.

As part of the coherency mechanism, reading EMIOS_CADR[*n*] disables transfers from B2 to B1. These transfers are disabled until the next read of the EMIOS_CBDR[*n*] register. Reading the EMIOS_CBDR[*n*] register re-enables transfers from B2 to B1, to take effect at the next transfer event, as previously described.[1]

---

[1] If B1 was not updated due to B2 to B1 transfer being disabled after reading register EMIOS_CADR[*n*], further EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] reads will not return coherent data until a new bus capture is triggered to registers A2 and B2. This capture event is indicated by the channel FLAG being asserted. If enabled, the FLAG also generates an interrupt.

In order to have coherent data in continuous operation mode, the following steps should be performed, assuming the FLAG bit is initially cleared:

1. Wait for FLAG assertion.
2. Read the EMIOS_CADR[*n*] register.
3. Read the EMIOS_CBDR[*n*] register.
4. Clear the FLAG bit.
5. Return to step #1.

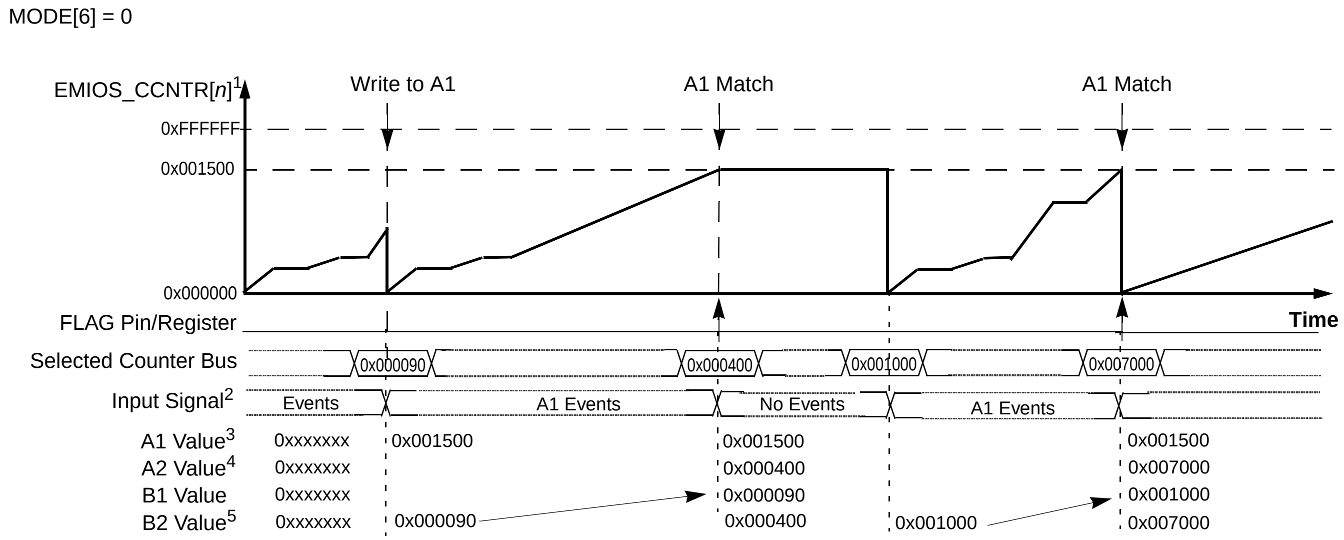
Accumulation cycles may be lost if the read is not performed in a timely manner. Whenever the Overrun bit is asserted it means that one or more cycles have been lost.

Triggering of the counter clock (input event) is done by a rising or falling edge or both edges on the input pin. The polarity of the triggering edge is selected by the EDSEL and EDPOL bits in EMIOS_CCR[*n*] register.

For continuous mode operation (MODE[6] cleared, MODE[0:6] = 000_1000), the counter is cleared on the next input event after a FLAG generation and continues to operate as previously described.

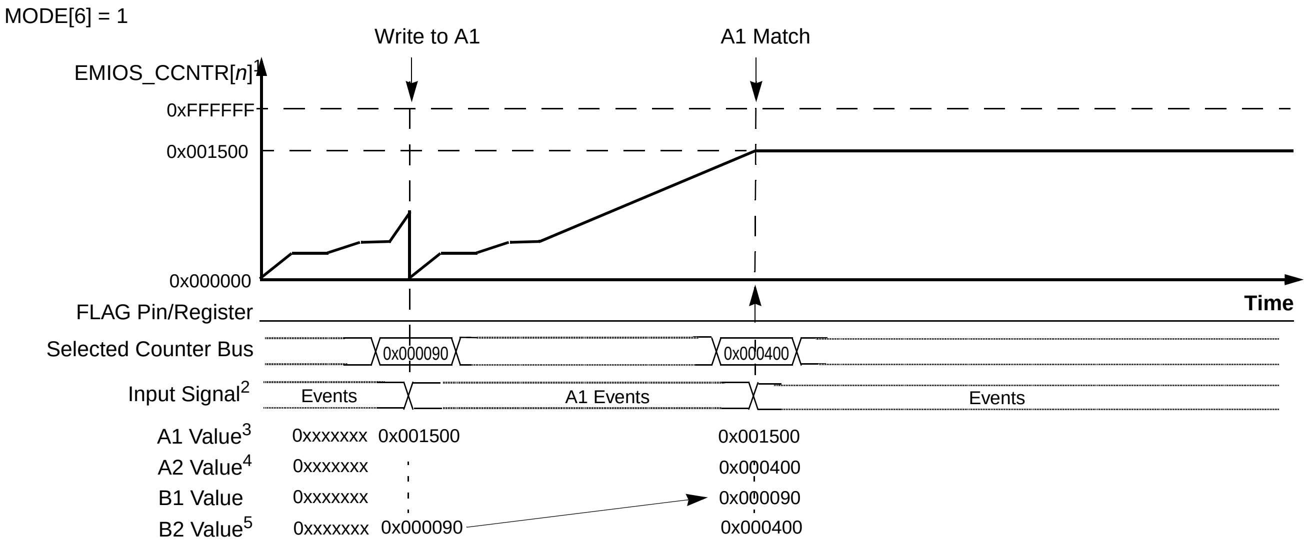
For single-shot operation (MODE[6] set, MODE[0:6] = 000_1001), the counter is not cleared or incremented after a FLAG generation until a new writing operation to register A (EMIOS_CADR) is performed.

Figure 18-23 and Figure 18-24 show how the unified channel can be used for continuous and single-shot pulse/edge accumulation mode.



Notes: 1. Cleared on the first input event after writing to register A1
2. After input filter
3. EMIOS_CADR[*n*] = A1 (when writing)
4. EMIOS_CADR[*n*] = A2 (when reading)
5. EMIOS_CBDR[*n*] = B1

Figure 18-23. PEA Continuous Mode Example



Notes: 1. Cleared on the first input event after writing to register A1
2. After input filter
3. EMIOS_CADR[*n*] = A1 (when writing)
4. EMIOS_CADR[*n*] = A2 (when reading)
5. EMIOS_CBDR[*n*] = B1

Figure 18-24. PEA Single-Shot Mode Example

18.4.1.1.8 Pulse/Edge Counting (PEC) Mode

The PEC mode returns the amount of pulses or edges detected on the input for a desired time window. The MODE[6] bit selects between continuous or single-shot operation.

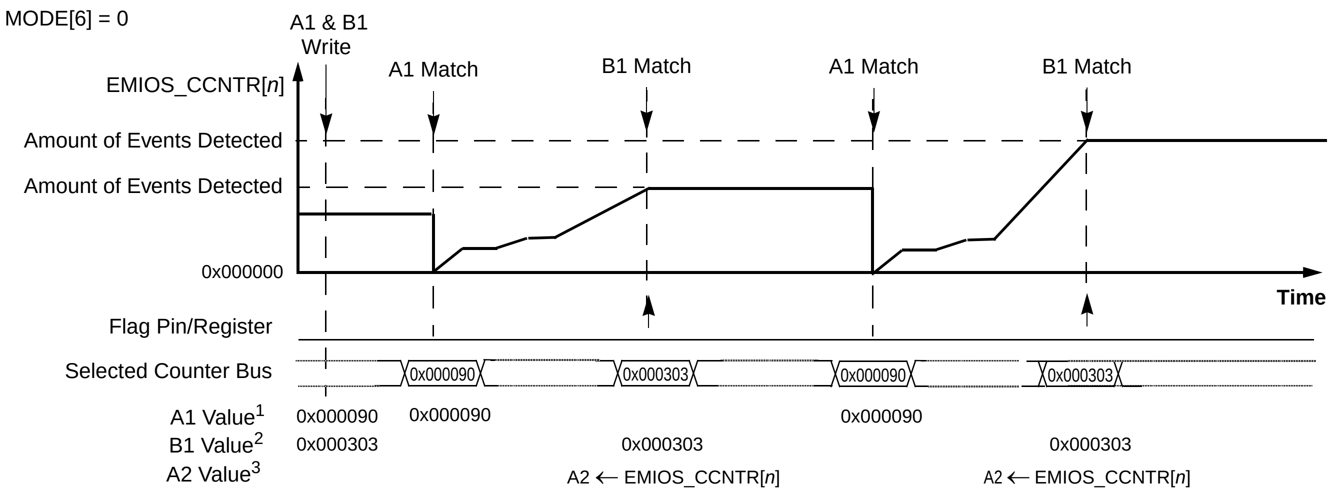
Triggering of the internal counter is done by a rising or falling edge or both edges on the input signal. The polarity and the triggering edge is selected by EDSEL and EDPOL bits in the EMIOS_CCR[*n*] register.

Register A1 holds the start time and register B1 holds the stop time for the time window. After writing to register A1, when a match occur between comparator A and the selected timebase, the internal counter is cleared and it is ready to start counting input events. When the time base matches comparator B, the internal counter is disabled and its content is transferred to register A2. At the same time the FLAG bit is set. Reading registers EMIOS_CCNTR[*n*] or A2 returns the amount of detected pulses.

For continuous operation (MODE[6] cleared, MODE[0:6] = 000_1010), the next match between comparator A and the selected time base clears the internal counter and counting is enabled again. In order to guarantee coherent measurements when reading EMIOS_CCNTR[*n*] after the FLAG is set, the software must check if the time base value is out of the time interval defined by registers A1 and B1. Alternatively register A2 always holds the latest available measurement providing coherent data at any time after the first FLAG had occurred. This register is addressed by the alternate address EMIOS_ALTA[*n*].

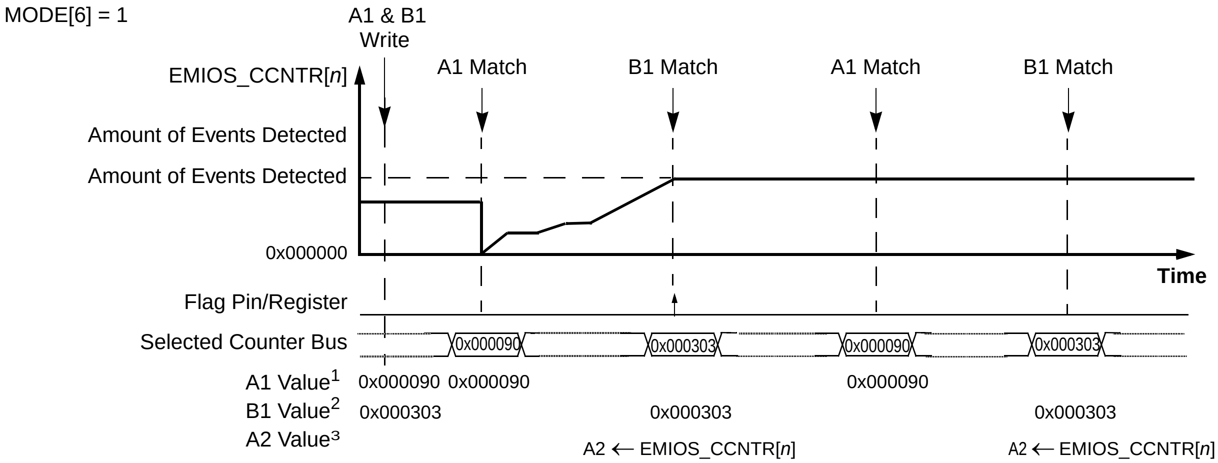
For single-shot operation (MODE[6] set, MODE[0:6] = 000_1011), the next match between comparator A and the selected time base has no effect, until a new write to register A (EMIOS_CADR) is performed. The EMIOS_CCNTR content is also transferred to register A2 when a match in the B comparator occurs.

Figure 18-25 and Figure 18-26 show how the unified channel can be used for continuous or single-shot pulse/edge counting mode.



Notes: 1. EMIOS_CADR[n] = A1
2. EMIOS_CBDR[n] = B1
3. EMIOS_ALTA[n] = A2

Figure 18-25. PEC Continuous Mode Example



Notes: 1. EMIOS_CADR[n] = A1
2. EMIOS_CBDR[n] = B1
3. EMIOS_ALTA[n] = A2

Figure 18-26. PEC Single-Shot Mode Example

18.4.1.1.9 Quadrature Decode (QDEC) Mode

QDEC mode uses UC[*n*] operating in QDEC mode and the input programmable filter (IPF) from UC[*n* – 1]. Note that UC[*n* – 1] can be configured, at the same time, to an operation mode that does not use I/O pins, such as modulus counter mode (MC). The connection among the unified channels is circular, i.e., when UC[0] is running in QDEC mode, the input programmable filter from UC[31] is being used.

This mode generates a FLAG every time the internal counter matches A1 register. The internal counter is automatically selected and is not cleared when entering this mode.

MODE[6] bit selects which type of encoder is used: *count & direction* encoder or *phase_A & phase_B* encoder.

When operating with *count & direction* encoder (MODE[6] cleared), the UC[*n*] input pin must be connected to the *direction* signal and the UC[*n* – 1] input pin must be connected to the count signal of the quadrature encoder. The EDPOL bit for UC[*n*] selects the count direction according to the *direction* signal and the EDPOL bit for UC[*n* – 1] selects whether the internal counter is clocked by the rising or falling edge of the *count* signal.

When operating with *phase_A & phase_B* encoder (MODE[6] set), the UC[*n*] input pin must be connected to the *phase_A* signal and the UC[*n* – 1] input pin must be connected to the *phase_B* signal of the quadrature encoder. The EDPOL bit selects the count direction according to the phase difference between *phase_A & phase_B* signals.

Figure 18-27 and Figure 18-28 show two unified channels configured to quadrature decode mode for *count & direction* encoder and *phase_A & phase_B* encoders, respectively.

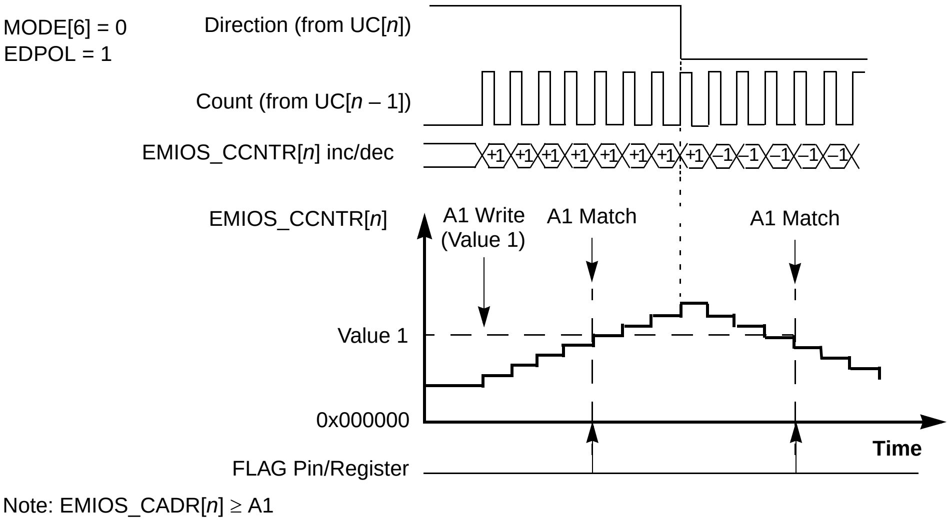


Figure 18-27. QDEC Mode Example with *Count & Direction* Encoder

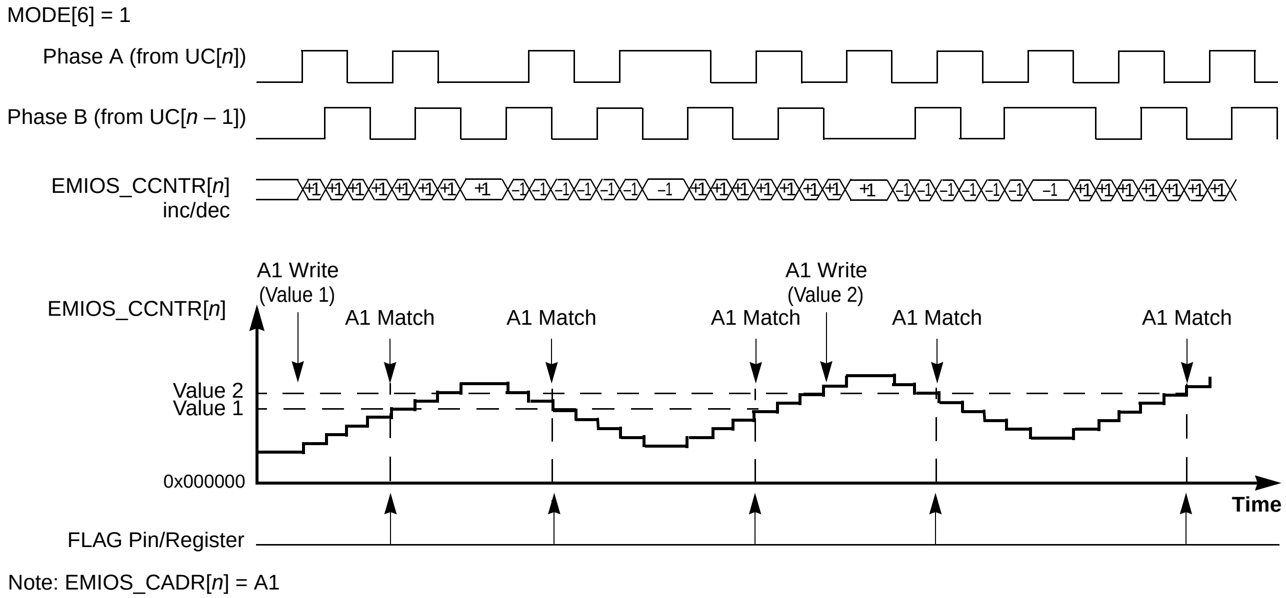


Figure 18-28. QDEC Example with *Phase_A & Phase_B* Encoder

18.4.1.1.10 Windowed Programmable Time Accumulation (WPTA) Mode

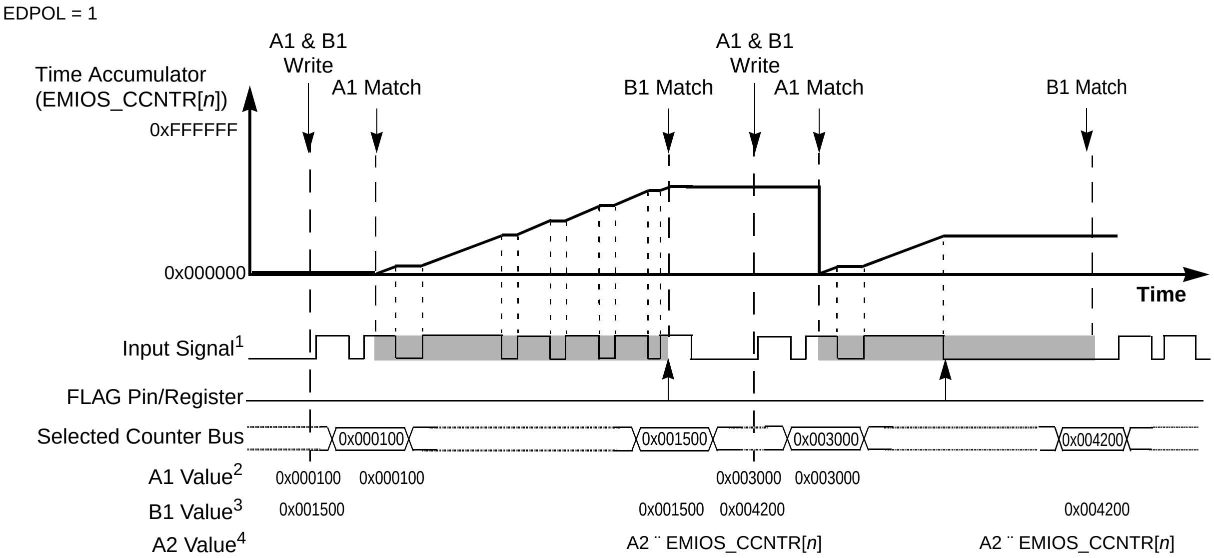
The WPTA mode (MODE = 000_1110) accumulates the sum of the total high time or low time of an input signal over a programmable interval (time window).

The UCPRE[1:0] prescaler bits in the EMIOS_CCR[*n*] register define the increment rate of the internal counter.

Register A1 holds the start time and register B1 holds the stop time of the programmable time interval. When a match occurs between register A (EMIOS_CADR) and the selected timebase, the internal counter is cleared and it is ready to start counting. The internal counter is used as a time accumulator, i.e., it counts up when the input signal has the same polarity of the EDPOL bit in the EMIOS_CCR[*n*] register and does not count otherwise. When a match occurs in comparator B, the internal counter is disabled regardless of the input signal polarity and the FLAG bit is set. At the same time, the contents of the EMIOS_CCNTR[*n*] register is transferred to register A2. Reading registers EMIOS_CCNTR[*n*] or A2 returns the high or low time of the input signal.

Note that EMIOS_CCNTR[*n*] is stable only outside the time window defined from A1 to B1 matches. Otherwise, its contents reflect a count in progress and not the final value. Alternatively to EMIOS_CCNTR[*n*], register A2 returns the latest available measurement. Since this register is updated only at comparator B matches, it always contains stable and up-to-date data. In this mode, this register is accessible through the alternate register address EMIOS_ALTA[*n*].

Figure 18-29 shows how the unified channel can be used to accumulate high time.



Notes: 1. After the input filter
2. EMIOS_CADR[*n*] = A1
3. EMIOS_CBDR[*n*] = B1
4. EMIOS_ALTA[*n*] = A2

Figure 18-29. WTPA Example

18.4.1.1.11 Modulus Counter (MC) Mode

The MC mode can be used to provide a time base for a counter bus or as a general purpose timer.

The MODE[6] bit selects the internal or external clock source when cleared or set, respectively. When the external clock is selected, the input signal pin is used as the source and the triggering polarity edge is selected by the EDPOL and EDSEL bits in the EMIOS_CCR[*n*] register.

The internal counter counts up from the current value until it matches the value in register A1. Register B1 is cleared and is not accessible to the MCU. The MODE[4] bit selects up mode or up/down mode, when cleared or set, respectively.

When in up count mode, a match between the internal counter and register A1 sets the FLAG and clears the internal counter. The timing of those events varies according to the MC mode setup as follows:

- Internal counter clearing on match start (MODE = 001_000b)
  - When MODE[6] is set, the external clock is selected. In this case, the internal counter clears as soon as the match signal occurs. The channel FLAG bit is set at the same time the match occurs. Note that by having the internal counter cleared as soon as the match occurs and incremented at the next input event, a shorter zero count is generated. See Figure 18-59 and Figure 18-61.
  - When MODE[6] is cleared, the internal clock source is selected. In this case, the counter clears as soon as the match signal occurs. The channel FLAG bit is set at the same time the match occurs. At the next prescaler tick after the match, the internal counter remains at 0 and only resumes counting on the following tick. See Figure 18-59 and Figure 18-62.
- Internal counter clearing on match end (MODE = 001_001b)

- When MODE[6] is set, the external clock is selected. In this case, the internal counter clears when the match signal is asserted and the input event occurs. The channel FLAG bit is set at the same time the counter is cleared. See Figure 18-59 and Figure 18-63.
- When MODE[6] is cleared, the internal clock source is selected. In this case, the internal counter clears when the match signal is asserted and the prescaler tick occurs. The channel FLAG bit is set at the same time the counter is cleared. See Figure 18-59 and Figure 18-63.

NOTE

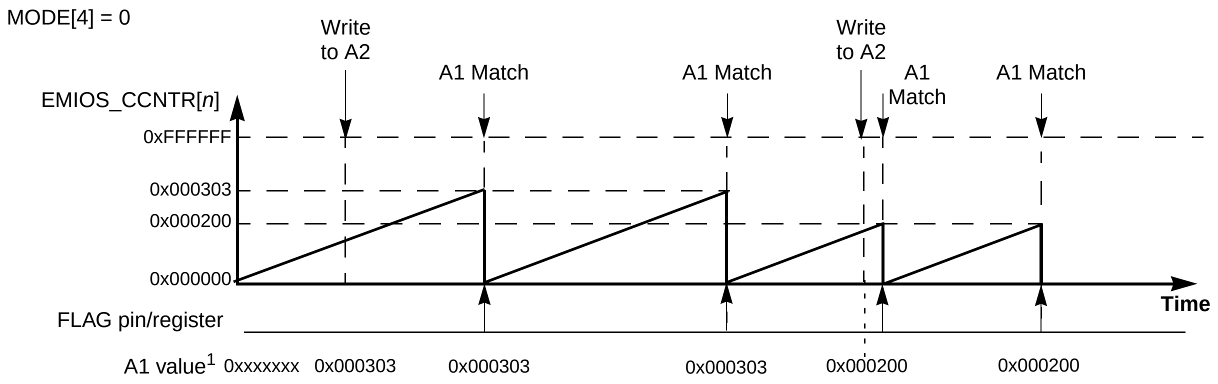
> If the internal clock source is selected and the prescaler of the internal counter is set to 1, the MC mode behaves the same way even in Clear on Match Start or Clear on Match End sub-modes.

When in up/down count mode (MODE = 001_01bb), a match between the internal counter and register A1 sets the FLAG and changes the counter direction from increment to decrement. A match between register B1 and the internal counter changes the counter direction from decrement to increment and sets the FLAG only if MODE[5] bit is set.

Only values other than 0x00_0000 must be written into register A (EMIOS_CADR). Loading 0x00_0000 leads to unpredictable results.

Updates on register A (EMIOS_CADR) or the counter in MC mode may cause a loss of match in the current cycle if the transfer occurs near the match. In this case, the counter may roll over and resume operation in the next cycle.

Figure 18-30 and Figure 18-31 show how the unified channel can be used as a modulus counter in up mode and up/down mode, respectively.



Notes: 1. EMIOS_CADR[*n*] = A1
A2 = A1 according to OU[*n*] bit

Figure 18-30. MC Up Mode Example

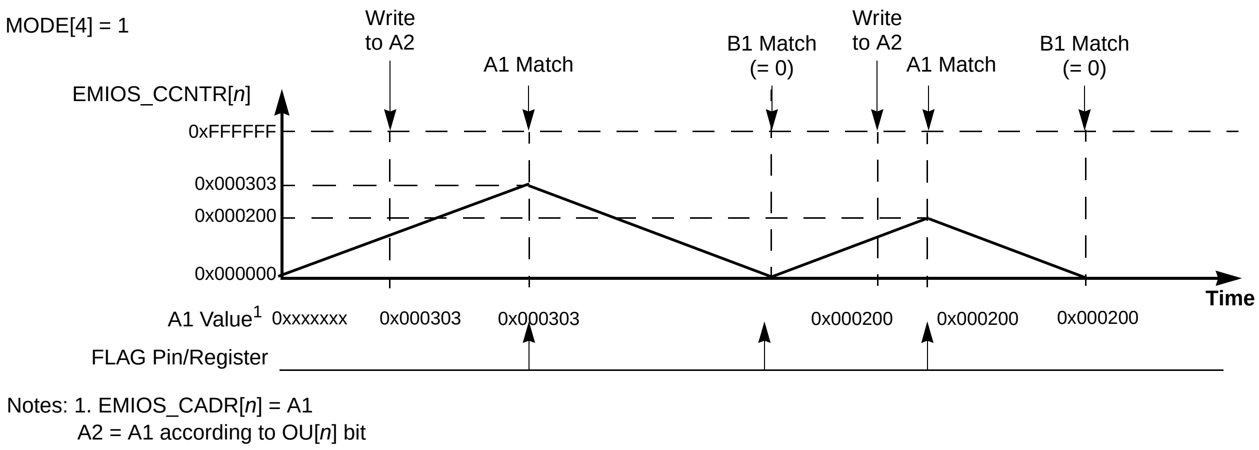


Figure 18-31. MC Up/Down Mode Example

18.4.1.1.12 Modulus Counter Buffered (MCB) Mode

The MCB mode provides a time base that can be shared with other channels through the internal counter buses. Register A1 is double-buffered, thus allowing smooth transitions between cycles when changing the A2 register value on the fly. Register A1 is updated at the cycle boundary, which is defined as when the internal counter reaches the value 0x00_0001.

The internal counter values operate within a range from 0x00_0001 up to the value of register A1 in MCB mode. When entering MCB mode (coming out of GPIO mode), the internal counter value must be within that range. Otherwise, the first A match will not occur, causing the channel internal counter to wrap at the maximum counter value of 0xFF_FFFF. After the counter wrap occurs, it returns to 0x00_0001 and resumes normal MCB mode operation. To avoid this counter wrap condition, make sure the internal counter value is within the 0x00_0001 to A1 register value range when entering the MCB mode.

MODE[6] bit selects the internal clock source if set to 0, or external if set to 1. When the external clock is selected, the input channel pin is used as the channel clock source. The active edge of this clock is defined by the EDPOL and EDSEL bits in the EMIOS_CCR[*n*] channel register.

When entering MCB mode, if the up counter is selected by MODE[4] = 0, the internal counter starts counting up from its current value until the A1 match occurs. On the next system clock cycle after the A1 match, the internal counter is set to 0x00_0001 and the FLAG bit is set to '1'.

If the up/down counter is selected by setting MODE[4] = 1, the counter changes direction at the A1 match and counts down until it reaches 0x00_0001. After it reaches 0x00_0001, it counts up again. Register B1 is set to 0x00_0001 on entering MCB mode cannot be changed while this mode is selected. B1 is used to generate a match to set the internal counter in up-count direction if up/down mode is selected.

The MCB mode counts between 0x00_0001 and the value in the A1 register. Only values greater than 0x00_0001 are allowed to be written to the A1 register. Loading values other than those leads to unpredictable results. The counter cycle period is equal to A1 value in up counter mode. If in up/down counter mode the period is defined by the expression: (2 × A1) – 2.

Figure 18-32 shows the counter cycle for several A1 values. Register A1 is loaded with the value in A2 at the cycle boundary. Any value written to the A2 register within cycle (*n*) is updated to A1 at the next cycle boundary and therefore is used on cycle (*n* + 1). The cycle boundary between cycle (*n*) and cycle (*n* + 1) is defined as the first system clock cycle of cycle (*n* + 1). The flags are generated as soon as the A1 match occurs.

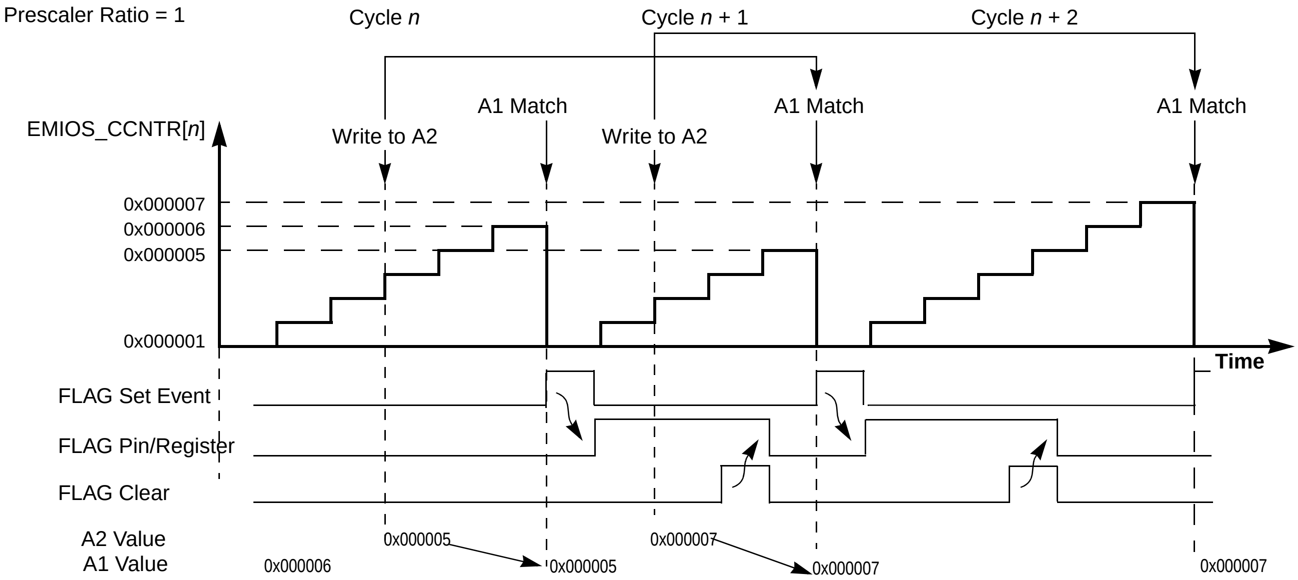


Figure 18-32. Modulus Counter Buffered (MCB) Up Count Mode

Figure 18-33 shows the MCB in up/down counter mode. Register A1 is updated at the cycle boundary. If A2 is written in cycle (*n*), this new value is used in cycle (*n* + 1) for an A1 match. When MODE[5] is cleared, flags are generated only on an A1 match. If MODE[5] is set to 1, flags are also generated at the cycle boundary.

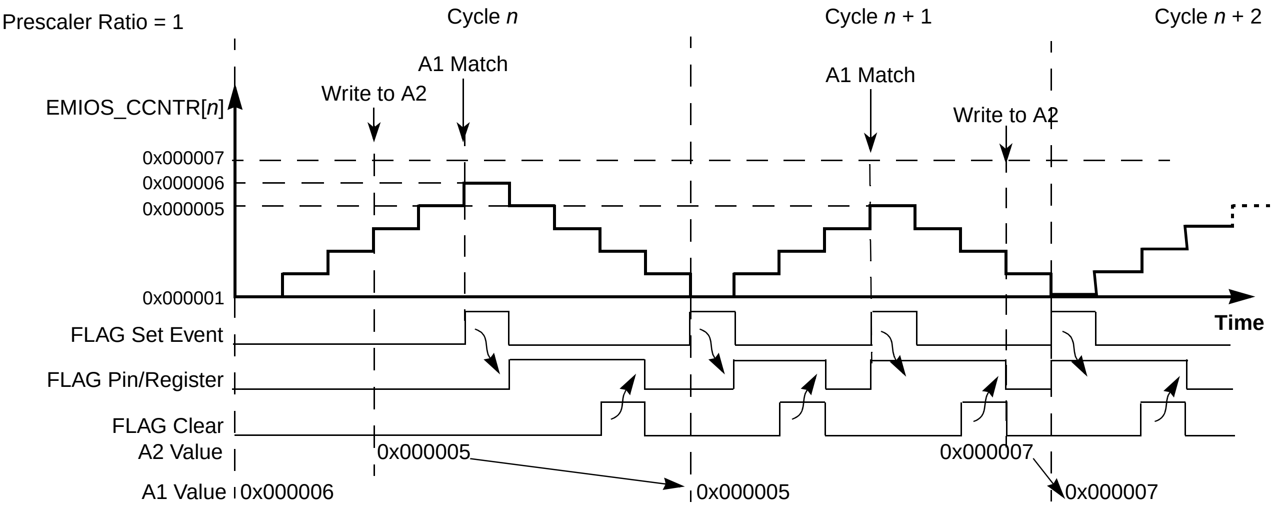


Figure 18-33. MCB Up/Down Mode

Figure 18-34 shows the A1 register update process in up counter mode. The A1 load signal is generated at the last system clock period of a counter cycle.

The A1 load signal is generated based on the detection of the internal counter reaching 0x00_0001 and has the duration of one system clock cycle. During the load pulse, A1 still holds its previous value. It is updated at the second system clock cycle only. Thus, A1 is updated with A2 value at the same time that the counter (EMIOS_CCNTR[*n*]) is loaded with 0x00_0001. The load signal pulse has the duration of one system clock period. If A2 is written within cycle (*n*), its value is available at A1 at the first clock of cycle (*n* + 1) and the new value is used for match at cycle (*n* + 1). The update disable bits OU[*n*] of the EMIOS_OUDR register can be used to control the update of this register, thus allowing the A1 register update to be delayed for synchronization purposes.

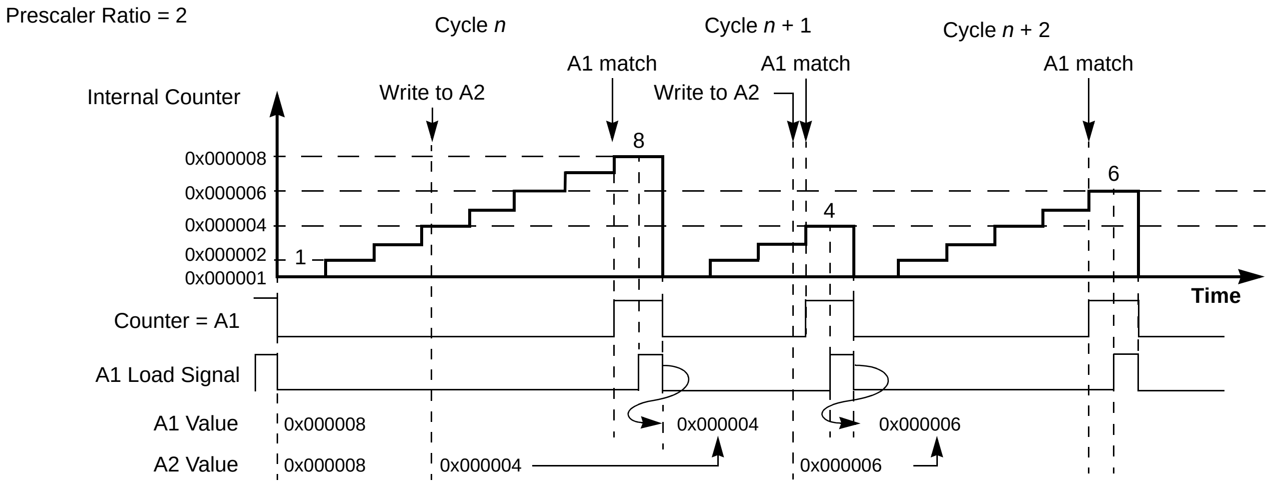


Figure 18-34. MCB Mode A1 Register Update in Up Counter Mode

Figure 18-35 shows the A1 register update in up/down counter mode. Note that A2 can be written at any time within cycle (*n*) in order to be used in cycle (*n* + 1). Thus A1 receives this new value at the next cycle boundary. The update disable bits (OU[*n*] in EMIOS_OUDR) can be used to disable the update of A1 register.

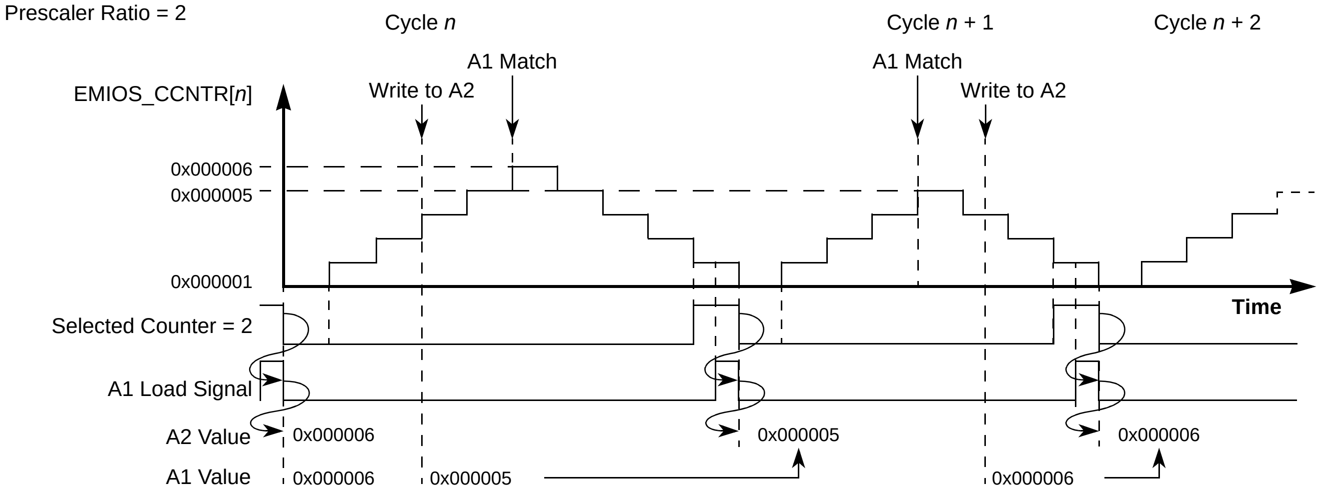


Figure 18-35. MCB Mode A1 Register Update in Up/Down Counter Mode

18.4.1.1.13 Output Pulse Width and Frequency Modulation (OPWFM) Mode

In the OPWFM mode, the duty cycle of the output signal is the value defined in register A1 plus 0x1, and the period is the value defined in register B1 plus 0x1. MODE[6] bit controls the transfer from register B2 to B1, which can be done either immediately (MODE[6] cleared, MODE = 00110b0), providing the fastest change in the duty cycle, or at every match of register A1 (MODE[6] set, MODE = 00110b1).

When OPWFM mode is entered, coming out from GPIO mode, the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR[*n*] register.

The internal counter is automatically selected as a time base, therefore the BSL[1:0] bits in register EMIOS_CCR[*n*] have no meaning. When a match on comparator A occurs, the output flip-flop is set to the value of the EDPOL bit. When a match occurs on comparator B, the output flip-flop is set to the complement of the EDPOL bit and the internal counter is cleared.

FLAG can be generated at match B, when MODE[5] is cleared, or in both matches, when MODE[5] is set.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A or B respectively. Also, FORCMB clears the internal counter. Note that the FLAG bit is not set by the FORCMA or FORCMB operations.

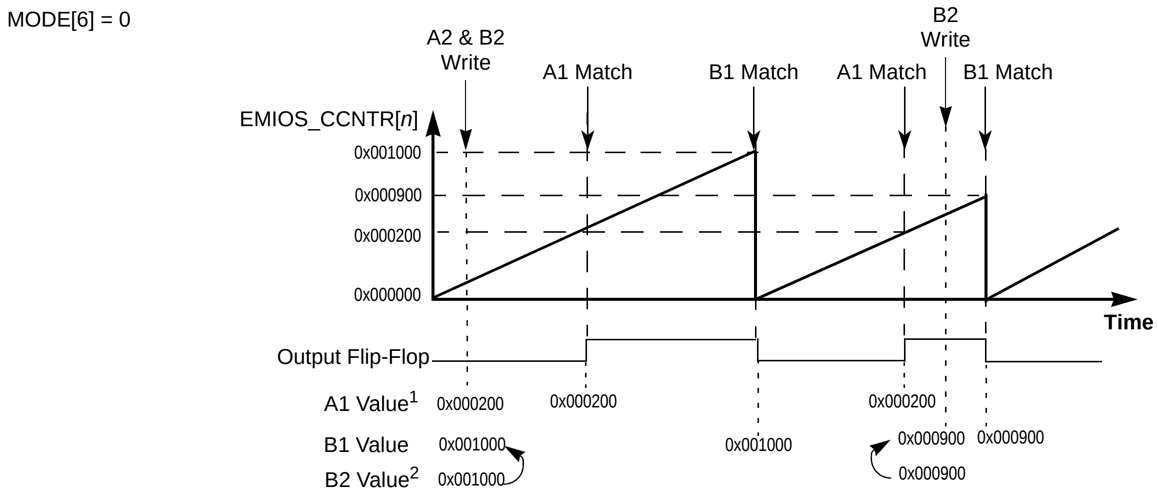
If subsequent comparisons occur on comparators A and B, the PWFM pulses continue to be output, regardless of the state of the FLAG bit.

In order to achieve 100% duty cycle, both registers A1 and B1 must be set to the same value. When a simultaneous match occurs on comparators A and B, the output flip-flop is set to the value of EDPOL bit. 0% duty cycle is possible by writing 0x0 to register A (EMIOS_CADR). When a match occurs, the output flip-flop is set at every period to the complement of EDPOL bit. The transfer from register B2 to B1 is still controlled by MODE[6] bit.

NOTE

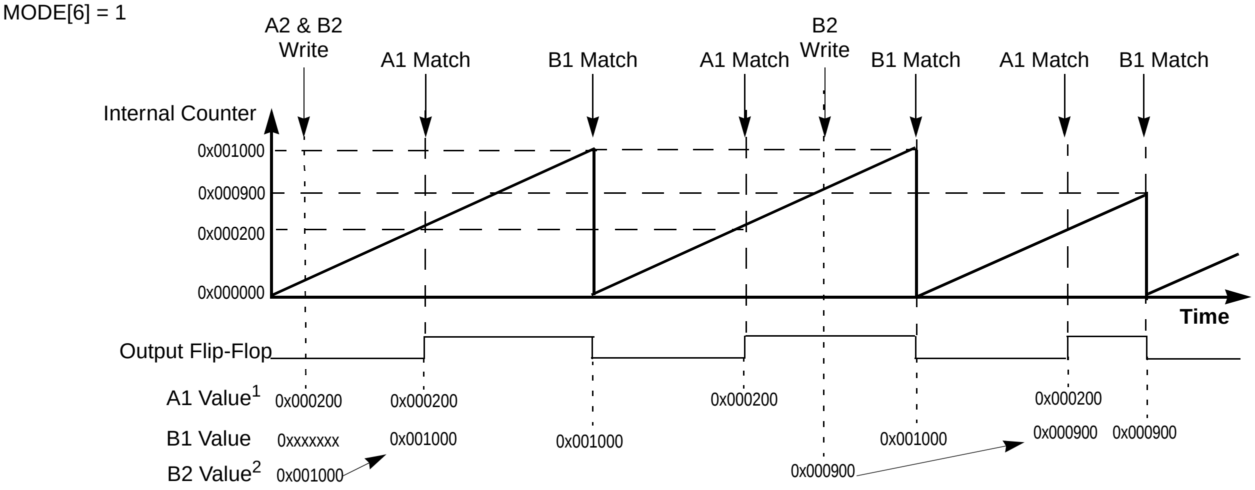
Writing 0x0 to A1 and B1 produces a duty cycle of 0%.

Figure 18-36 shows the unified channel running in OPWFM mode with immediate register update and Figure 18-37 shows the unified channel running in OPWFM mode with next period update.



Notes: 1. EMIOS_CADR[*n*] = A1
2. EMIOS_CBDR[*n*] = B2
A2 = A1according to OU[*n*] bit
B2 = B1according to OU[*n*] bit

Figure 18-36. OPWFM with Immediate Update



Notes: 1. EMIOS_CADR[*n*] = A1
2. EMIOS_CBDR[*n*] = B2
A2 = A1according to OU[*n*] bit
B2 = B1according to OU[*n*] bit

Figure 18-37. OPWFM with Next Period Update

18.4.1.1.14 Output Pulse-Width and Frequency Modulation Buffered (OPWFMB) Mode

The OPWFMB mode provides waveforms with variable duty cycle and frequency. The internal channel counter is automatically selected as the time base when this mode is selected. The A1 register indicates the duty cycle and the B1 register indicates the frequency. Both A1 and B1 registers are double-buffered to allow smooth signal generation when changing the registers values on the fly. This mode supports 0% and 100% duty cycles.

At OPWFMB mode entry, the output flip-flop is set to the value of the EDPOL bit in the EMIOS_CCR[*n*] register.

To provide smooth and consistent channel operation, this mode differs substantially from the OPWFM mode. The main differences reside in the A1 and B1 registers update, on the delay from the A1 match to the output pin transition, and on the internal counter values, which range from 0x00_0001 up to the value in register B1.

When entering OPWFMB mode (coming out of GPIO mode), if the internal counter value is not within that range, then the B match will not occur, causing the channel internal counter to wrap at the maximum counter value which is 0xFF_FFFF. After the counter wrap occurs, the value returns to 0x00_0001 and the counter resumes normal OPWFMB mode operation. Thus in order to avoid the counter wrap condition, make sure its initial value is within the range between 0x00_0001 and the B1 register value the OPWFMB mode is entered.

When a match on comparator A occurs, the output register is set to the value of EDPOL. When a match on comparator B occurs, the output register is set to the complement of EDPOL. B1 match also causes the internal counter to transition to 0x00_0001, thus restarting the counter cycle.

Only values greater than 0x00_0001 are allowed to be written to the B1 register. Loading values other than those leads to unpredictable results.

Figure 18-38 shows the operation of the OPWFMB mode regarding output pin transitions and A1/B1 registers match events. The output pin transition occurs when the A1 or B1 match signal is deasserted, which is indicated by the A1 match negative edge detection signal. If register A1 is set to 0x00_0004, the output pin transitions four counter periods after the cycle has started, plus one system clock cycle. In the example shown in Figure 18-38 the internal counter prescaler has a ratio of two.

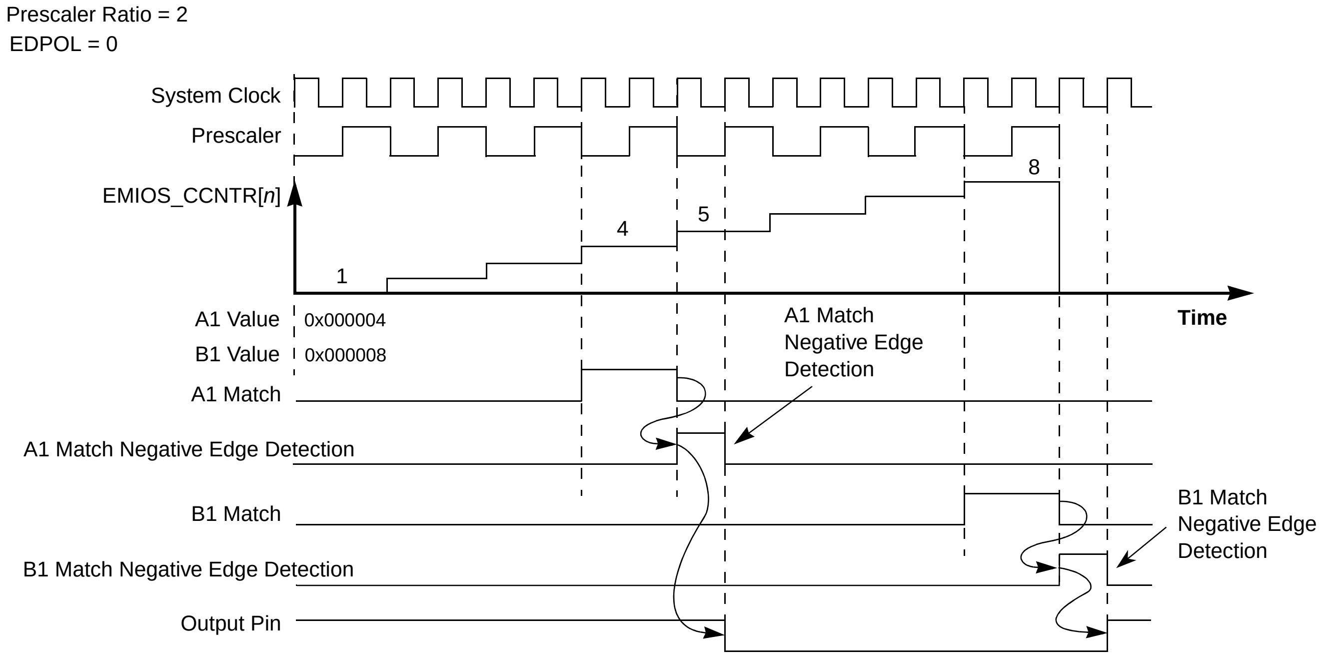


Figure 18-38. OPWFMB A1 and B1 Match to Output Register Delay

Figure 18-39 shows the generated output signal if A1 is set to 0x0. Because the counter does not reach 0 in this mode, the channel internal logic infers a match as if A1 = 0x00_0001 with the difference that in this case, the positive edge of the match signal is used to trigger the output pin transition instead of the negative edge used when A1 = 0x00_0001. An A1 positive edge match signal from cycle (*n* + 1) occurs at the same time as B1 negative edge match signal from cycle (*n*). This allows using the A1 positive edge match to mask the B1 negative edge match when they occur at the same time. The result is that no transition occurs on the output flip-flop and a 0% duty cycle is generated.

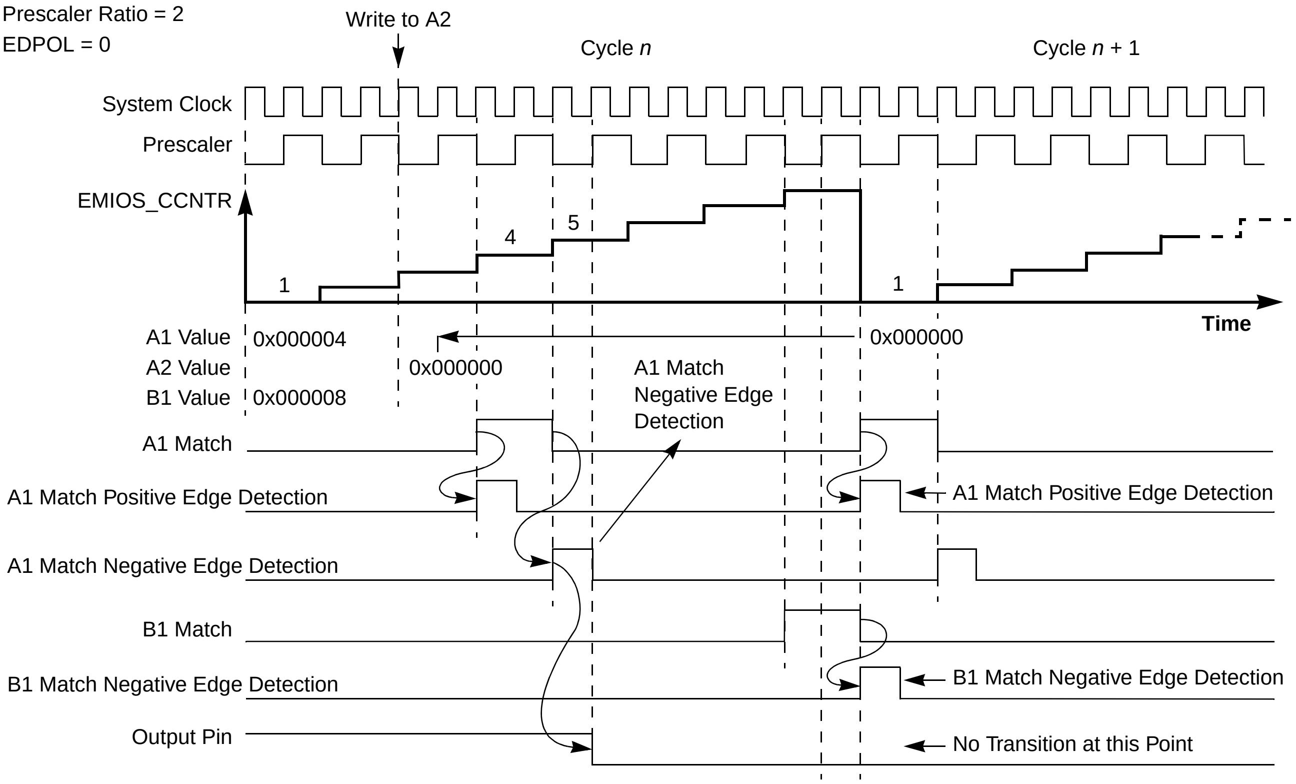


Figure 18-39. OPWFMB Mode with A1 = 0 (0% duty cycle)

Figure 18-40 shows the timing for the A1 and B1 registers load. The A1 and B1 load use the same signal which is generated at the last system clock period of a counter cycle. Thus, A1 and B1 are updated respectively with A2 and B2 values at the same time that the EMIOS_CCNTR[*n*] counter is loaded with 0x00_0001. This event is defined as the cycle boundary. The load signal pulse has the duration of one system clock period. If A2 and B2 are written within cycle (*n*), their values are loaded into A1 and B1, respectively, at the first clock of cycle (*n* + 1) and the new values are used for matches at cycle (*n* + 1). The update disable bits (OU[*n*] in EMIOS_OUDR) can be used to control the update of these registers, thus allowing to delay the A1 and B1 registers update for synchronization purposes.

In Figure 18-40, it is assumed that both the channel and global prescalers are set to 0x00_0001 (each divide ratio is two), meaning that the channel internal counter transitions at every four system clock cycles. FLAGs can be generated only on B1 matches when MODE[5] is cleared, or on either A1 or B1 matches when MODE[5] is set. Because the B1 FLAG occurs at the cycle boundary, this flag can be used to indicate that A2 or B2 data written on cycle (*n*) were loaded to A1 or B1, respectively, thus generating matches in cycle (*n* + 1). Note that the FLAG has a synchronous operation, meaning that it is asserted one system clock cycle after the FLAG set event.

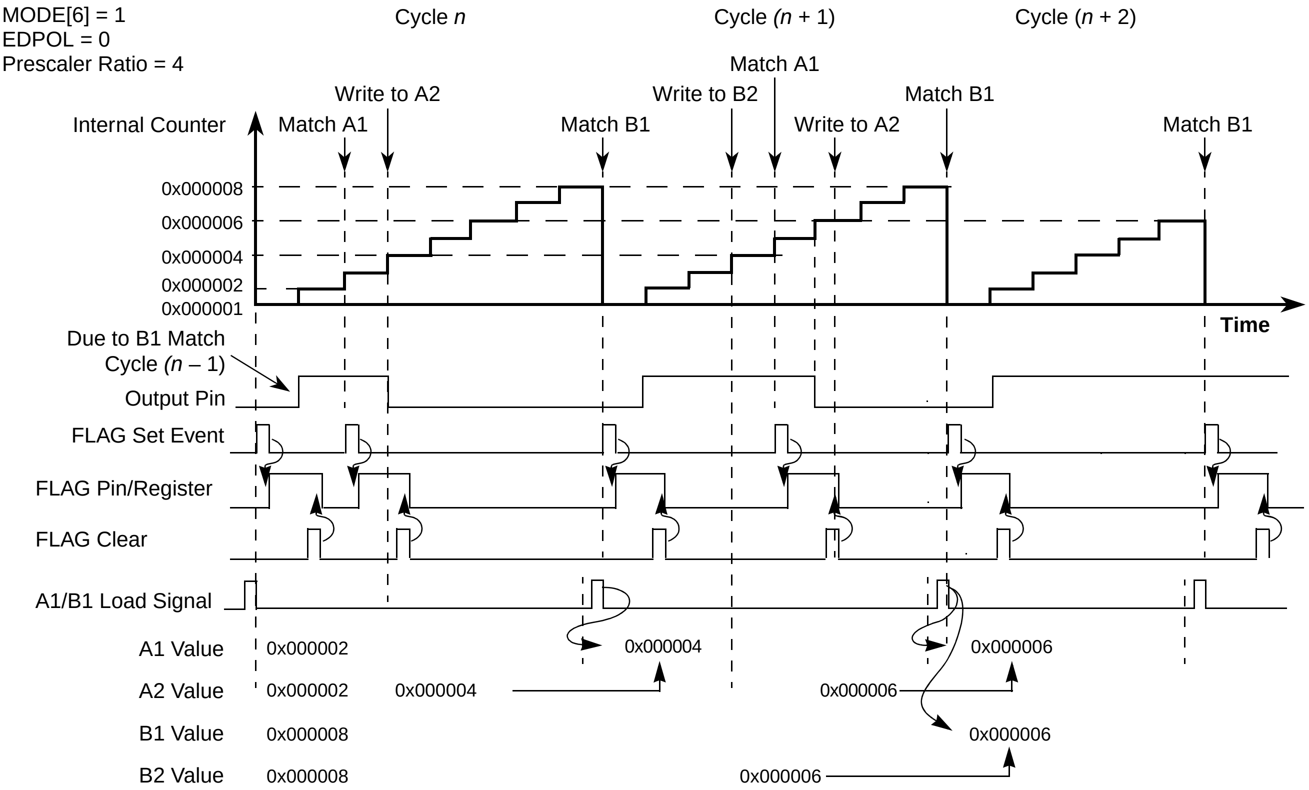


Figure 18-40. OPWFMB A1 and B1 Registers Update and Flags

Figure 18-41 shows the operation of the output disable feature in OPWFMB mode. In contrast to the OPWFM mode, the output disable forces the channel output flip-flop to the value of the EDPOL bit. This functionality targets applications that use active-high signals and a high-to-low transition at A1 match. In this case, EDPOL should be set to 0. Note that both the channel and global prescalers are set to 0x00_0000 (each divide ratio is one), meaning that the channel internal counter transitions at every system clock cycle.

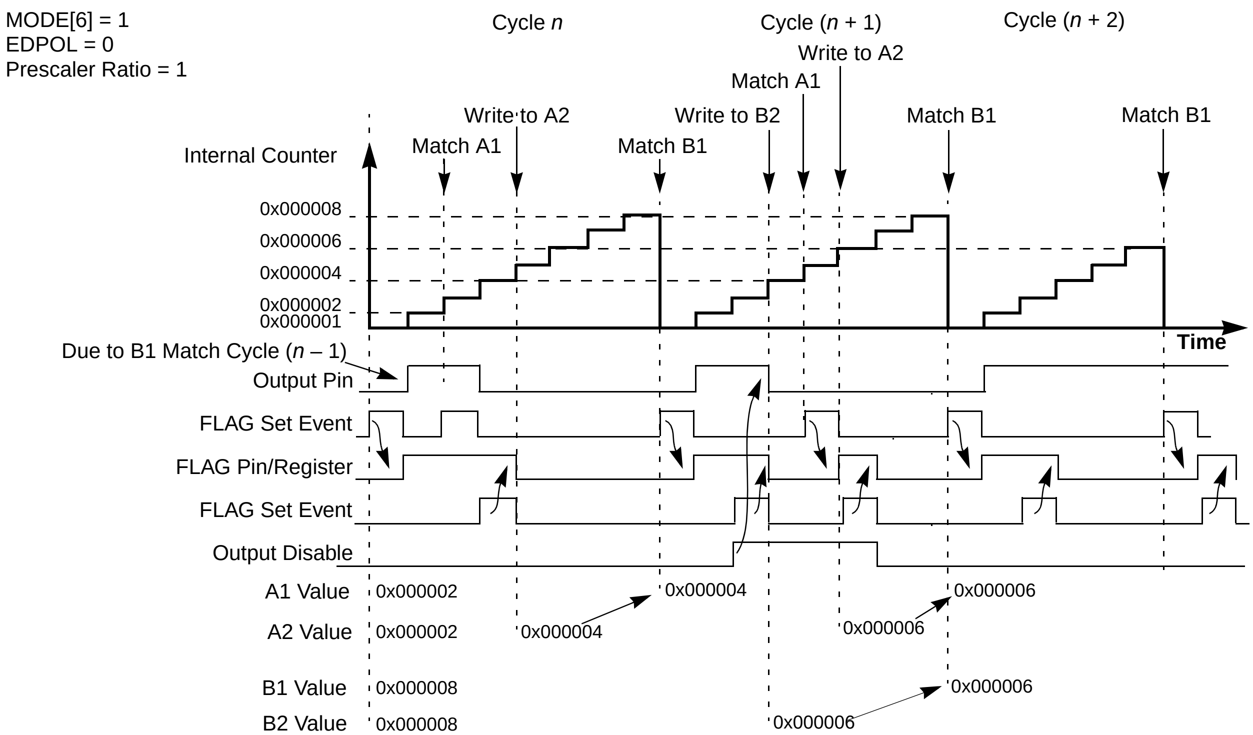


Figure 18-41. OPWFMB Mode with Active Output Disable

The output disable has a synchronous operation, meaning that the assertion of the output disable input pin causes the channel output flip-flop to transition to EDPOL at the next system clock cycle. If the output disable input is deasserted the output pin transition at the following A1 or B1 match.

In Figure 18-41 it is assumed that the output disable input is enabled and selected for the channel. Refer to Section 18.3.2.7, “eMIOS200 Control Register (EMIOS_CCR[n])”, for a description of how the ODIS and ODISSL bits enable and select the output disable inputs.

The FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on comparators A or B, respectively. Similar to a B1 match, FORCMB sets the internal counter to 0x00_0001. The FLAG bit is not set by the FORCMA or FORCMB bits being asserted.

Figure 18-42 shows the generation of 100% and 0% duty cycle signals. It is assumed EDPOL = 0 and the resultant prescaler value is 1. Initially, A1 = 0x00_0008 and B1 = 0x00_0008. In this case, the B1 match has precedence over the A1 match, thus the output flip-flop is set to the complement of EDPOL bit. This cycle corresponds to a 100% duty cycle signal. The same output signal can be generated for any A1 value greater or equal to B1.

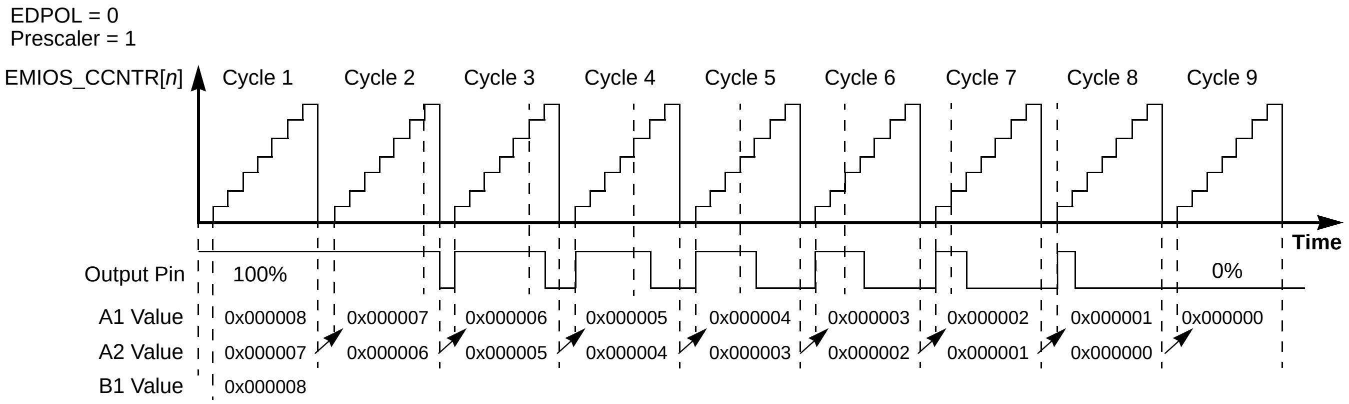


Figure 18-42. OPWFMB Mode from 100% to 0% Duty Cycle

A 0% duty cycle signal is generated if A1 = 0x00_0000 as shown in cycle 9 in Figure 18-42. In this case, the B1 = 0x00_0008 match from cycle 8 occurs at the same time as the A1 = 0x00_0000 match from cycle 9. Refer to Figure 18-39 for a description of the A1 and B1 match generation. In this case, the A1 match has precedence over the B1 match and the output signal transitions to EDPOL.

18.4.1.1.15 Center Aligned Output Pulse Width Modulation with Dead Time (OPWMC) Mode

The OPWMC mode generates a center aligned PWM with dead time insertion in the leading (MODE = 001_11b1) or trailing edge (MODE = 001_11b0).

The selected counter bus must be running an up/down time base, as shown in Figure 18-31. BSL[1:0] bits select the time base. Register A1 contains the ideal duty cycle for the PWM signal and is compared with the selected time base. Register B1 contains the dead time value and is compared with the internal counter. For a leading-edge dead time insertion, the output PWM duty cycle is equal to the difference between register A1 and register B1, and for a trailing edge dead time insertion, the output PWM duty cycle is equal to the sum of register A1 and register B1. Mode[6] bit selects between trailing and leading dead time insertion.

NOTE

> The internal counter may be running in the internal prescaler ratio, while the selected time base may be running in a different prescaler ratio. The output signal may produce an unexpected output if the dead time interval is greater than the duty cycle of the PWM signal.

When OPWMC mode is entered, coming out from GPIO mode, the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR[*n*] register.

When operating with leading edge dead time insertion, the first match between A1 and the selected time base clears the internal counter and switches the selected time base to the internal counter. When a match occurs between register B1 and the selected time base, the output flip-flop is set to the value of the EDPOL bit and the time base is switched to the selected counter bus. In the next match between register A1 and the selected time base, the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously.

When operating with trailing edge dead time insertion, the first match between A1 and the selected time base sets the output flip-flop to the value of the EDPOL bit. In the next match between register A1 and the selected time base, the internal counter is cleared and the selected time base is switched to the internal counter. When a match occurs between register B1 and the selected time base, the output flip-flop is set to the complement of the EDPOL bit and the time base is switched to the selected counter bus. This sequence repeats continuously.

FLAG can be generated in the trailing edge of the output PWM signal when MODE[5] is cleared, or in both edges, when MODE[5] is set.

At any time, the FORCMA or FORCMB bits are equivalent to a successful comparison on comparator A or B with the exception that the FLAG bit is not set.

NOTE

When in freeze state, the FORCMA or FORCMB bits only allow the software to force the output flip-flop to the level corresponding of a match on A or B respectively.

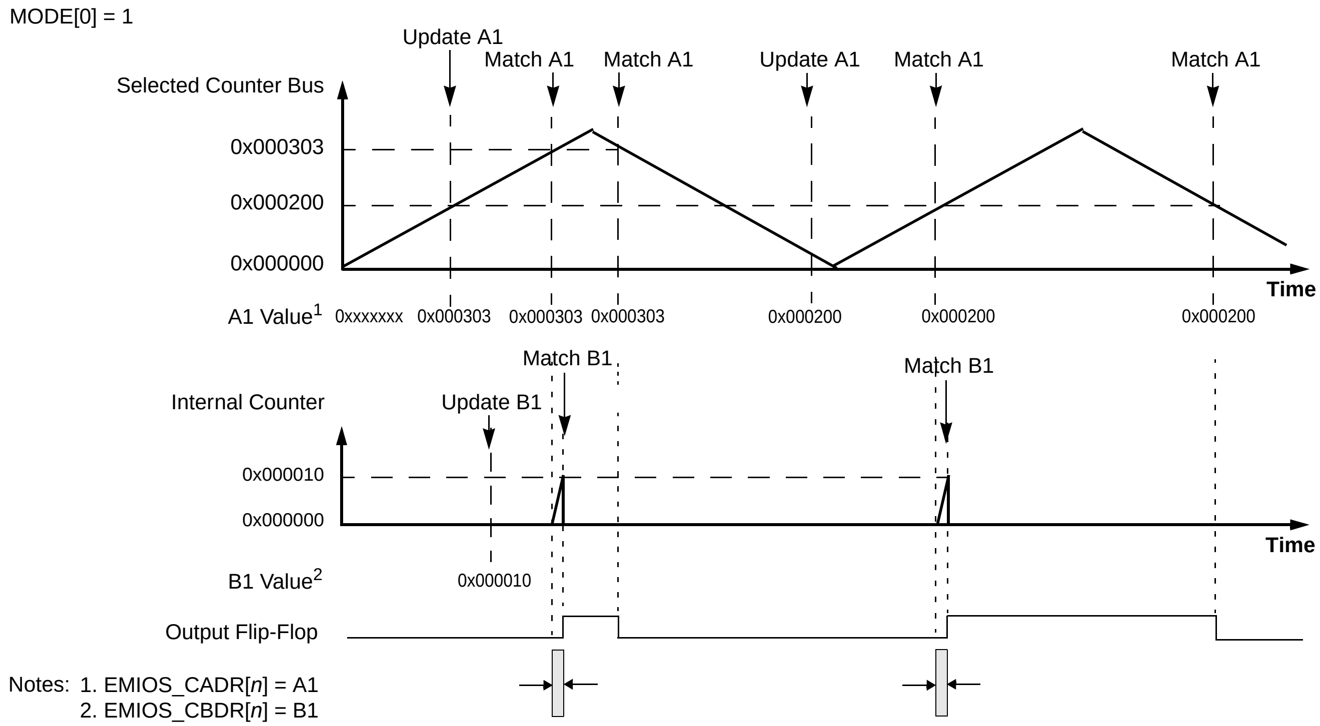
If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

In order to achieve a duty cycle of 100%, both registers A1 and B1 must be set to the same value. When a simultaneous match occurs between the selected time base and registers A1 and B1, the output flip-flop is set to the value of EDPOL bit and the selected time base switches to the selected counter bus, allowing a new cycle to begin at any time, as previously described. 0% duty cycle is possible by writing 0x00_0000 to register A (EMIOS_CADR). When a match occurs, the output flip-flop is set to the complement of EDPOL bit and the selected time base switches to the selected counter bus, allowing a new cycle to begin at any time, as previously described. In both cases, FLAG is generated regardless of the MODE[6] bit.

NOTE

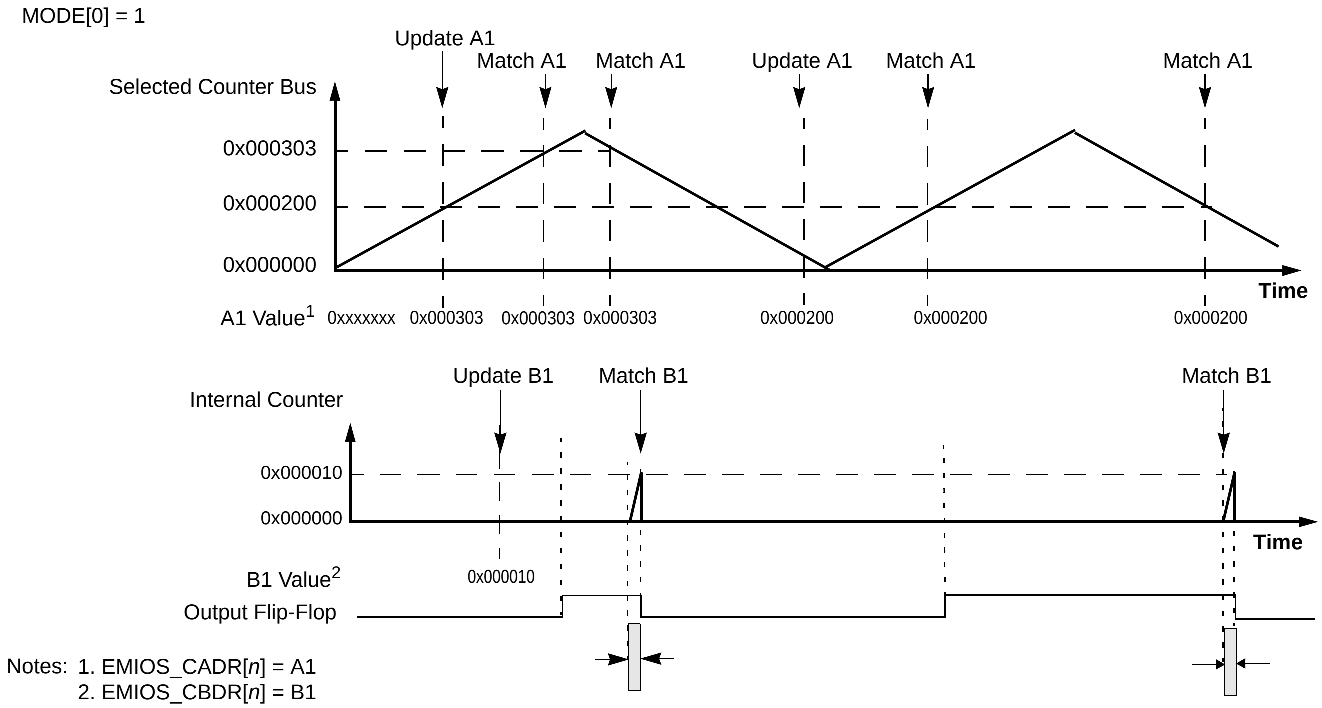
If A1 and B1 are set to 0x00_0000, a 0% duty cycle waveform is produced.

Figure 18-43 and Figure 18-44 show the unified channel running in OPWMC with leading and trailing dead time, respectively.



Notes: 1. EMIOS_CADR[*n*] = A1
2. EMIOS_CBDR[*n*] = B1
A2 = A1 according to OU[*n*] bit
B2 = B1 according to OU[*n*] bit

Figure 18-43. OPWMC with Leading Dead Time Insertion



Notes: 1. EMIOS_CADR[*n*] = A1
2. EMIOS_CBDR[*n*] = B1
A2 = A1 according to OU[*n*] bit
B2 = B1 according to OU[*n*] bit

Figure 18-44. OPWMC with Trailing Dead Time Insertion

18.4.1.1.16 Center-Aligned Output PWM Buffered with Dead Time (OPWMCB) Mode

The OPWMCB mode generates a center-aligned PWM with dead time insertion to the leading or trailing edge. A1 and B1 registers are double-buffered to allow smooth output signal generation when changing A2 or B2 registers values on the fly.

The BSL bits select the time base. The time base selected for a channel configured to OPWMCB mode should be a channel configured to MCB Up/Down mode, as shown in Figure 18-33. It is recommended to start the MCB channel time base after the OPWMCB mode is entered in order to avoid missing A matches at the very first duty cycle.

Register A1 contains the ideal duty cycle for the PWM signal and is compared with the selected time base.

Register B1 contains the dead time value and is compared against the internal counter. For a leading edge dead time insertion, the output PWM duty cycle is equal to the difference between register A1 and register B1. For a trailing edge dead time insertion, the output PWM duty cycle is equal to the sum of register A1 and register B1. Mode[6] selects between trailing and leading dead time insertion.

NOTE

> The internal counter runs in the internal prescaler ratio, while the selected time base may be running in a different prescaler ratio.

When OPWMCB mode is entered (coming out of GPIO mode), the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR[*n*] register.

The following basic steps summarize proper OPWMCB startup, assuming the channels are initially in GPIO mode:

1. *[global]* Disable Global Prescaler.
2. *[MCB channel]* Disable Channel Prescaler.
3. *[MCB channel]* Write $1 at internal counter.
4. *[MCB channel]* Set A register.
5. *[MCB channel]* Set channel to MCB Up mode.
6. *[MCB channel]* Set prescaler ratio.
7. *[MCB channel]* Enable Channel Prescaler.
8. *[OPWMCB channel]* Disable Channel Prescaler.
9. *[OPWMCB channel]* Set A register.
10. *[OPWMCB channel]* Set B register.
11. *[OPWMCB channel]* Select time base input through BSL[1:0] bits.
12. *[OPWMCB channel]* Enter OPWMCB mode.
13. *[OPWMCB channel]* Set prescaler ratio.
14. *[OPWMCB channel]* Enable Channel Prescaler.
15. *[global]* Enable Global Prescaler.

Figure 18-45 shows the load of A1 and B1 registers, which occurs when the selected counter bus transitions from 0x00_0002 to 0x00_0001. This event defines the cycle boundary. Values written to A2 or

B2 within cycle (*n*) are loaded into A1 or B1 registers, respectively, and used to generate matches in cycle (*n* + 1).

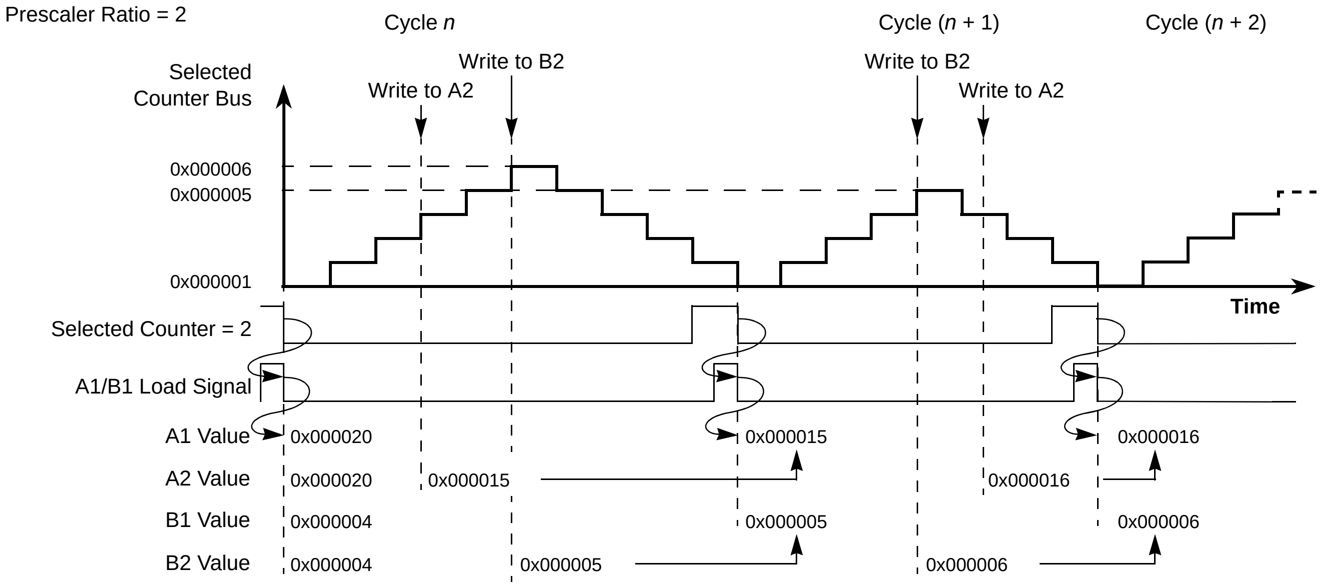


Figure 18-45. OPWMCB A1 and B1 Registers Load

The (OU[*n*] in EMIOS_OUDR) bit can be used to disable the A1 and B1 updates, thus allowing to synchronize the load on these registers with the load of A1 or B1 registers in others channels. Using the update disable bit, A1 and B1 registers can be updated at the same counter cycle, allowing both registers to change at the same time.

In this mode A1 matches always sets the internal counter to 0x00_0001. When operating with leading edge dead time insertion the first A1 match sets the internal counter to 0x00_0001. When a match occurs between register B1 and the internal time base, the output flip-flop is set to the value of the EDPOL bit. In the following match between register A1 and the selected time base, the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously. The internal counter should not reach 0x00_0000 as consequence of a rollover. To avoid this, the user must not write a value greater than twice the difference between external count up limit and EMIOS_CADR[*n*] value to the EMIOS_CBDR[*n*] register.

Figure 18-46 shows two cycles of a center-aligned PWM signal. Both A1 and B1 register values are changing within the same cycle, which allows to vary at the same time the duty cycle and dead time values.

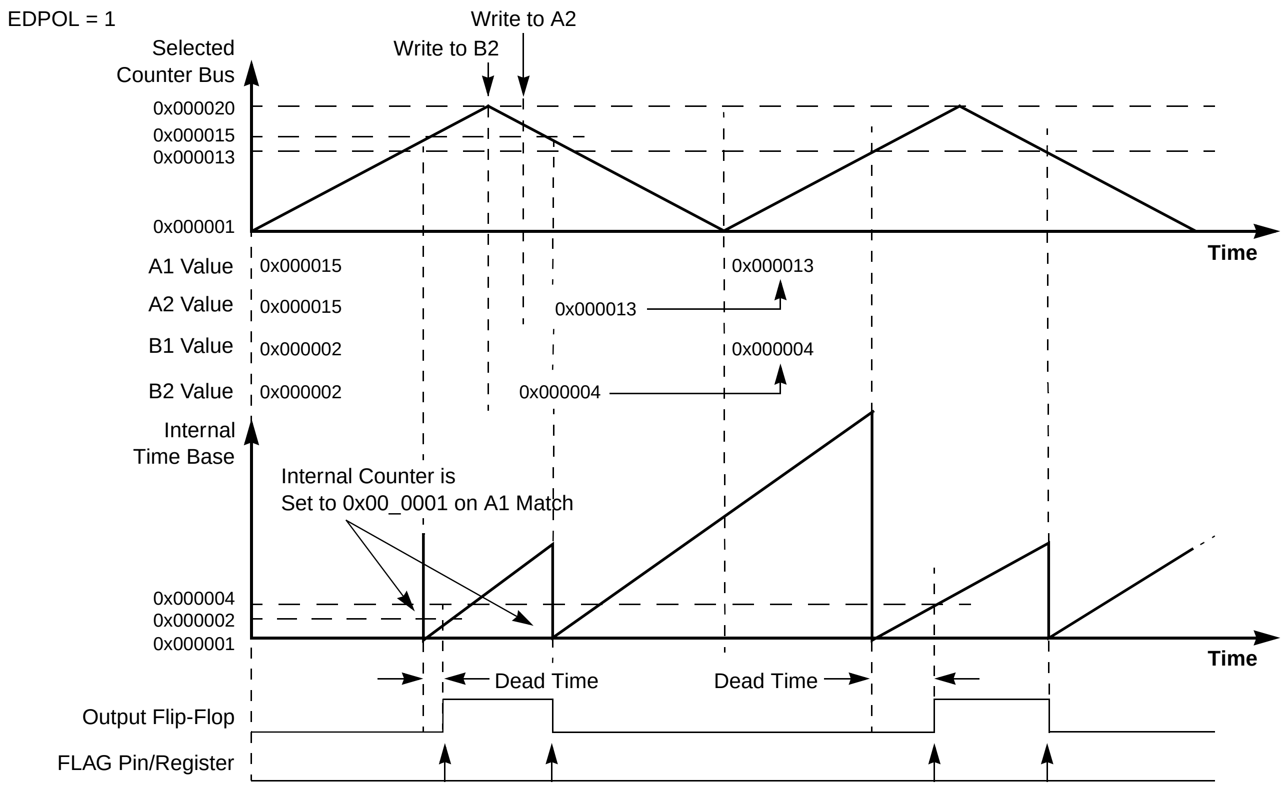


Figure 18-46. OPWMCB with Leading Dead Time Insertion

When operating with trailing edge dead time insertion, the first match between A1 and the selected time base sets the output flip-flop to the value of the EDPOL bit and sets the internal counter to 0x00_0001. In the second match between register A1 and the selected time base, the internal counter is set to 0x00_0001 and B1 matches are enabled. When the match between register B1 and the selected time base occurs, the output flip-flop is set to the complement of the EDPOL bit. This sequence repeats continuously.

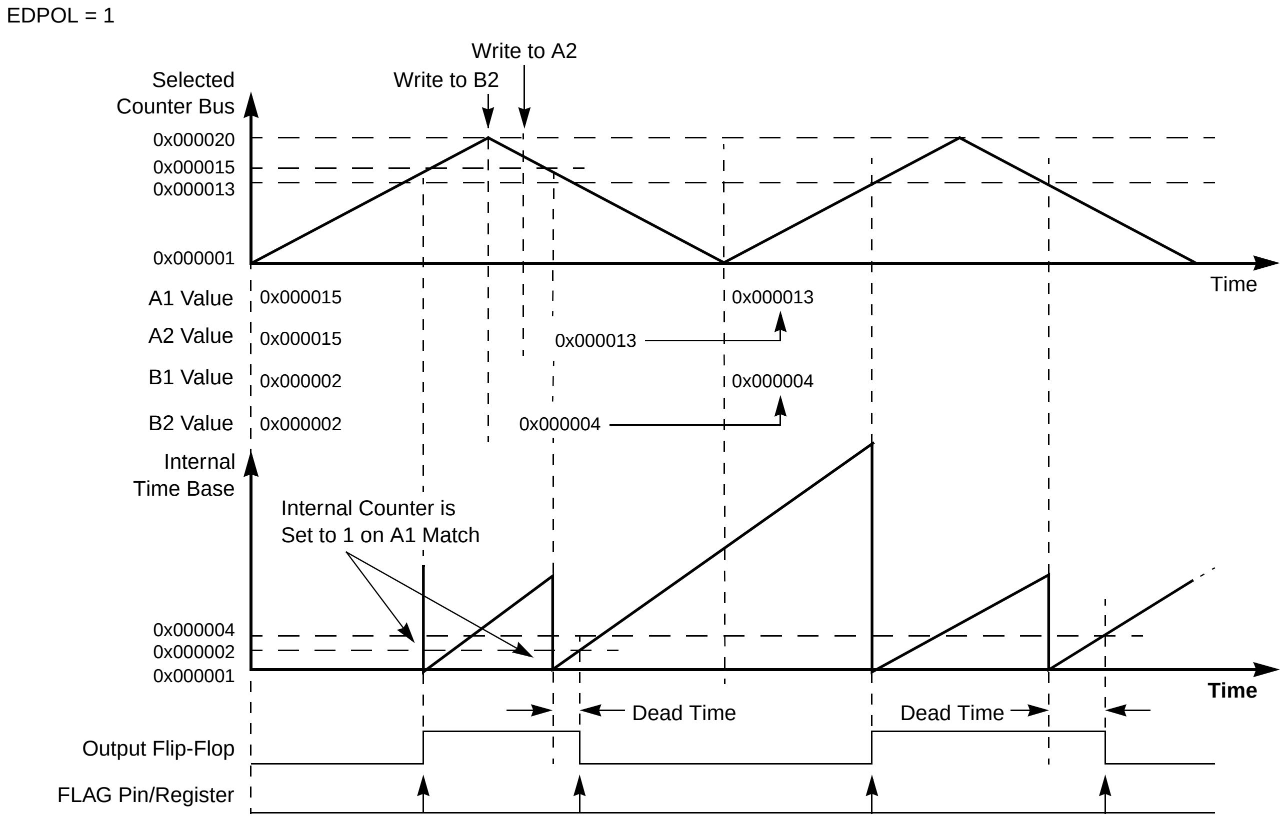


Figure 18-47. OPWMCB with Trailing Dead Time Insertion

FLAG can be generated in the trailing edge of the output PWM signal when MODE[5] is cleared, or in both edges when MODE[5] is set. If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated regardless of the state of the FLAG bit.

NOTE

In OPWMCB mode, FORCMA and FORCMB do not have the same behavior as a regular match. Instead they force the output flip-flop to constant value, which depends on the selected dead time insertion mode, leading or trailing, and the value of the EDPOL bit.

FORCMA has different behaviors depending on the selected dead time insertion mode, leading or trail. In leading dead time insertion, FORCMA forces a transition in the output flip-flop to the opposite of the EDPOL bit. In trailing dead time insertion, the output flip-flop is forced to the value of the EDPOL bit.

If the FORCMB bit is set, the output flip-flop value depends on the selected dead time insertion mode. In leading dead time insertion, FORCMB forces the output flip-flop to transition to the EDPOL bit value. In trailing dead time insertion the output flip-flop is forced to the opposite of the EDPOL bit value.

NOTE

The FORCMA bit set does not set the internal time-base to 0x00_0001 as a regular A1 match.

The FLAG bit is not set either in case of FORCMA or FORCMB, even if both forces are issued at the same time.

NOTE

> FORCMA and FORCMB have the same behavior even in freeze or normal mode regarding the output pin transition.

When FORCMA is issued along with FORCMB, the output flip-flop is set to the opposite of the EDPOL bit value. This is the equivalent of saying that FORCMA has precedence over FORCMB when leading dead time insertion is selected, and FORCMB has precedence over FORCMA when trailing dead time insertion is selected.

Duty cycles from 0% to 100% can be generated by setting appropriate values to the A1 and B1 registers relative to the period of the external time base. Setting A1 = 0x00_0001 generates a 100% duty cycle waveform. If A1 is greater than the maximum value of the selected counter bus period, then a 0% duty cycle is produced. Assuming EDPOL is set to 1 in OPWMCB mode with trailing dead time insertion, 100% duty cycle signals can be generated if B1 occurs at or after the cycle boundary (external counter = 1).

If A1 is greater than the maximum value of the selected counter bus period, then a 0% duty cycle is produced, only if the pin starts the current cycle in the opposite of the EDPOL value. In case of 100% duty cycle, the transition from EDPOL to the opposite of EDPOL may be obtained by forcing the pin using FORCMA and/or FORCMB.

NOTE

> If A1 is set to 0x00_0001 at OPWMCB entry the 100% duty cycle may not be obtained in the very first PWM cycle due to the pin condition at mode entry.

Only values different than 0x0 are allowed to be written to A1 register. If 0x00_0000 is loaded to A1, the results are unpredictable.

NOTE

> A special case occurs when A1 is set to (external counter bus period)/2, which is the maximum value of the external counter. In this case, the output flip-flop is constantly set to the EDPOL bit value.

The internal channel logic prevents matches from one cycle to propagate to the next cycle. In trailing dead time insertion, a B1 match from cycle (*n*) could eventually cross the cycle boundary and occur in cycle (*n* + 1). In this case, the B1 match is masked out and does not cause the output flip-flop to transition. Therefore, matches in cycle (*n* + 1) are not affected by the late B1 matches from cycle (*n*).

Figure 18-48 shows a 100% duty cycle output signal generated by setting A1 = 4 and B1 = 3. In this case the trailing edge is positioned at the boundary of cycle *n* + 1, which is actually considered to belong to cycle *n* + 2 and therefore does not cause the output flip-flip to transition.

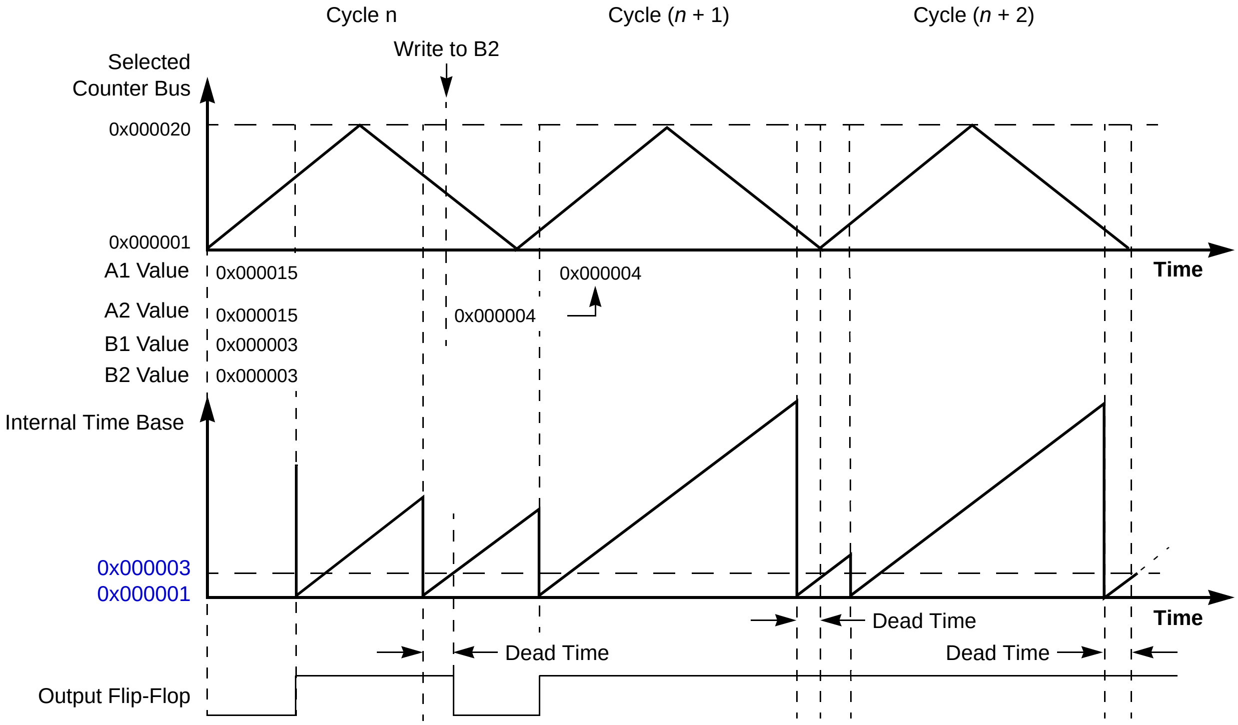


Figure 18-48. OPWMCB with 100% Duty Cycle (A1 = 4 and B1 = 3)

The output disable feature, if enabled, causes the output flip-flop to transition to the EDPOL inverted state. This feature allows an application to force the channel output pin to a “safe” state. The internal channel matches continue to occur even in this case, thus generating flags. As soon as the output disable is deasserted, the channel output pin is again controlled by the A1 and B1 matches. This process is synchronous, meaning that the output channel pin transitions on system clock edges only.

It is important to notice that, as in OPWMB and OPWFMB modes, the match signal used to set or clear the channel output flip-flop is generated on the deassertion of the channel combinational comparator output signal, which compares the selected time base with A1 or B1 register values. Refer to Figure 18-38, which shows the delay from matches to output flip-flop transition in OPWFMB mode. The operation of OPWMCB mode is similar to OPWFMB regarding matches and output pin transition.

18.4.1.1.17 Output Pulse Width Modulation (OPWM) Mode

In OPWM mode, registers A1 and B1 define the leading and trailing edges of the PWM output pulse, respectively. The MODE[6] bit controls the transfer from register B2 to B1, which can be done either immediately (MODE[6] cleared, MODE = 010_00b0), providing the fastest change in the duty cycle, or at every match of register A1 (MODE[6] set, MODE = 010_00b1).

The value loaded in register A1 is compared with the value on the selected time base. When a match on comparator A occurs, the output flip-flop is set to the value of the EDPOL bit. When a match occurs on comparator B, the output flip-flop is set to the complement of the EDPOL bit.

FLAG can be generated at match B, when MODE[5] is cleared, or in both matches, when MODE[5] is set.

At any time, the FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A or B respectively. Note that FLAG bit is not set by the FORCMA and FORCMB operations.

If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

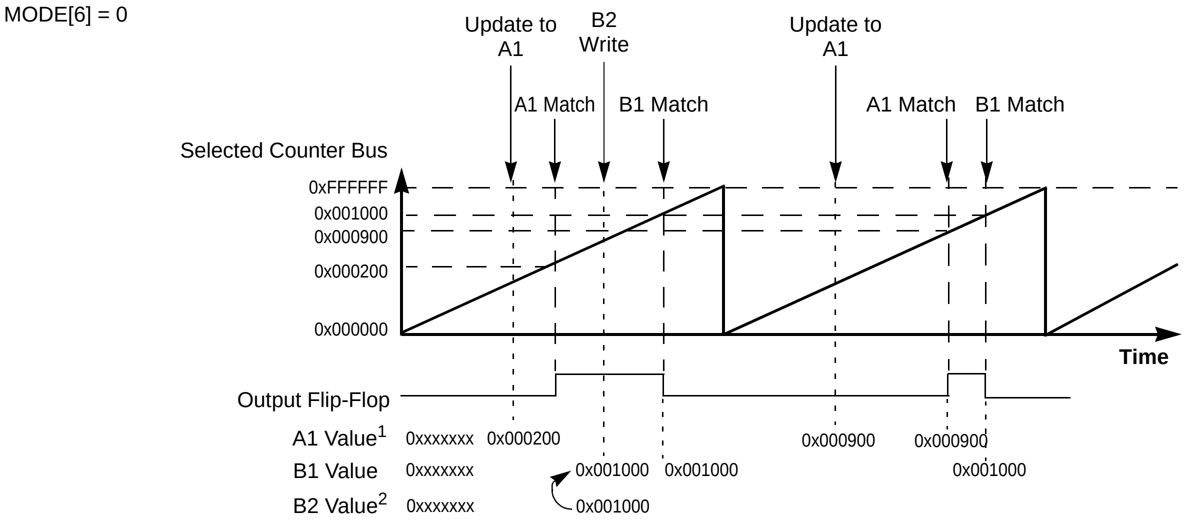
At OPWM mode entry, the output flip-flop is set to the complement of the EDPOL bit in the EMIOS_CCR[*n*] register.

In order to achieve 100% duty cycle, both registers A1 and B1 must be set to the same value. When a simultaneous match on comparators A and B occur, the output flip-flop is set at every period to the value of EDPOL bit. 0% duty cycle is possible by writing 0x0 to register A (EMIOS_CADR). When a match occurs, the output flip-flop is set at every period to the complement of EDPOL bit. The transfer from register B2 to B1 is still controlled by MODE[6] bit.

NOTE

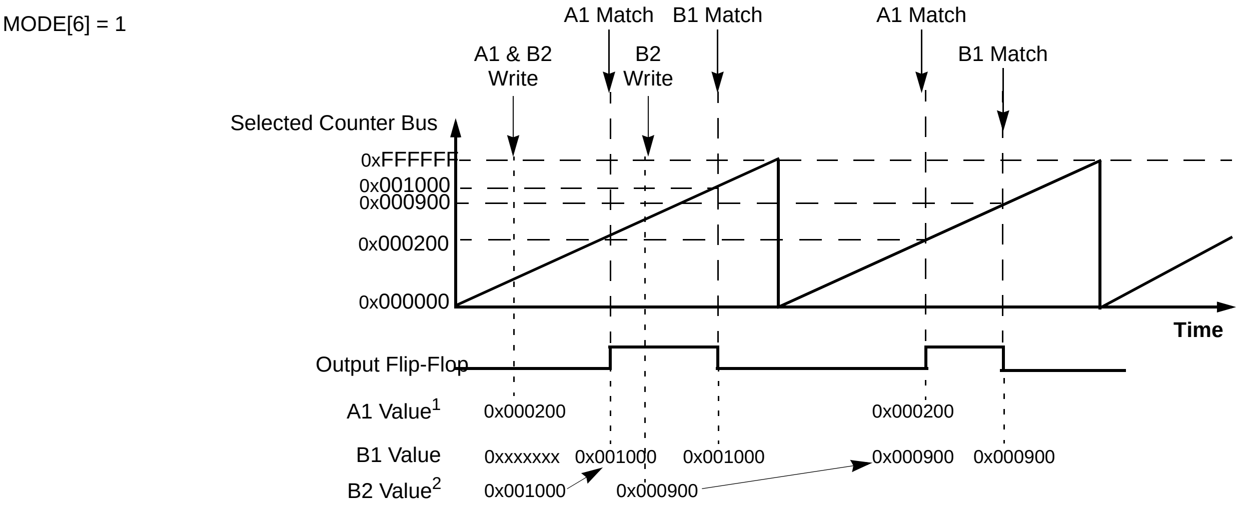
If A1 and B1 are set to 0x00_0000, a 0% duty cycle waveform is produced.

Figure 18-49 and Figure 18-50 show the unified channel running in OPWM with immediate update and next period update, respectively.



Notes: 1. EMIOS_CADR[*n*] = A1
2. EMIOS_CBDR[*n*] = B2
A2 = A1 according to OU[*n*] bit
B2 = B1 according to OU[*n*] bit

Figure 18-49. OPWM with Immediate Update



Notes: 1. EMIOS_CADR[*n*] = A1
2. EMIOS_CBDR[*n*] = B2
A2 = A1 and A2 = A1 according to OU[*n*] bit

Figure 18-50. OPWM with Next Period Update

18.4.1.1.18 Output Pulse-Width Modulation Buffered (OPWMB) Mode

OPWMB mode (MODE = 110_00b0) is used to generate pulses with programmable leading- and trailing-edge placement. An external counter must be selected from one of the counter buses. The A1 register value defines the first edge and B1 defines the second edge. The output signal polarity is defined by the EDPOL bit. If EDPOL is 0, a negative edge occurs when A1 matches the selected counter bus; and a positive edge occurs when B1 matches the selected counter bus.

The A1 and B1 registers are double buffered and updated from A2 and B2, respectively, at the cycle boundary. The load operation is similar to the OPWFMB mode. Please refer to Figure 18-40 for more information about A1 and B1 registers update.

FLAG can be generated at B1 matches, when MODE[5] is cleared, or in both A1 and B1 matches, when MODE[5] is set. If subsequent matches occur on comparators A and B, the PWM pulses continue to be generated, regardless of the state of the FLAG bit.

FORCMA and FORCMB bits allow the software to force the output flip-flop to the level corresponding to a match on A1 or B1 respectively. The FLAG bit is not set by the FORCMA and FORCMB operations.

At OPWMB mode entry the output flip-flop is set to the value of the EDPOL bit in the EMIOS_CCR[*n*] register.

Some rules applicable to the OPWMB mode include:

- B1 matches have precedence over A1 matches if they occur at the same time within the same counter cycle
- A1 = 0 match from cycle(*n*) has precedence over B1 match from cycle(*n* – 1)
- A1 matches are masked out if they occur after B1 match within the same cycle

- Any value written to A2 or B2 on cycle(*n*) is loaded to A1 and B1 registers at the following cycle boundary (assuming (OU[*n*] in EMIOS_OUDR) is not asserted). The new values are used for A1 and B1 matches in cycle(*n* + 1)

Figure 18-51 shows the operation of the OPWMB mode regarding A1 and B1 matches and the transition of the channel output pin. In this example, EDPOL is set to 0.

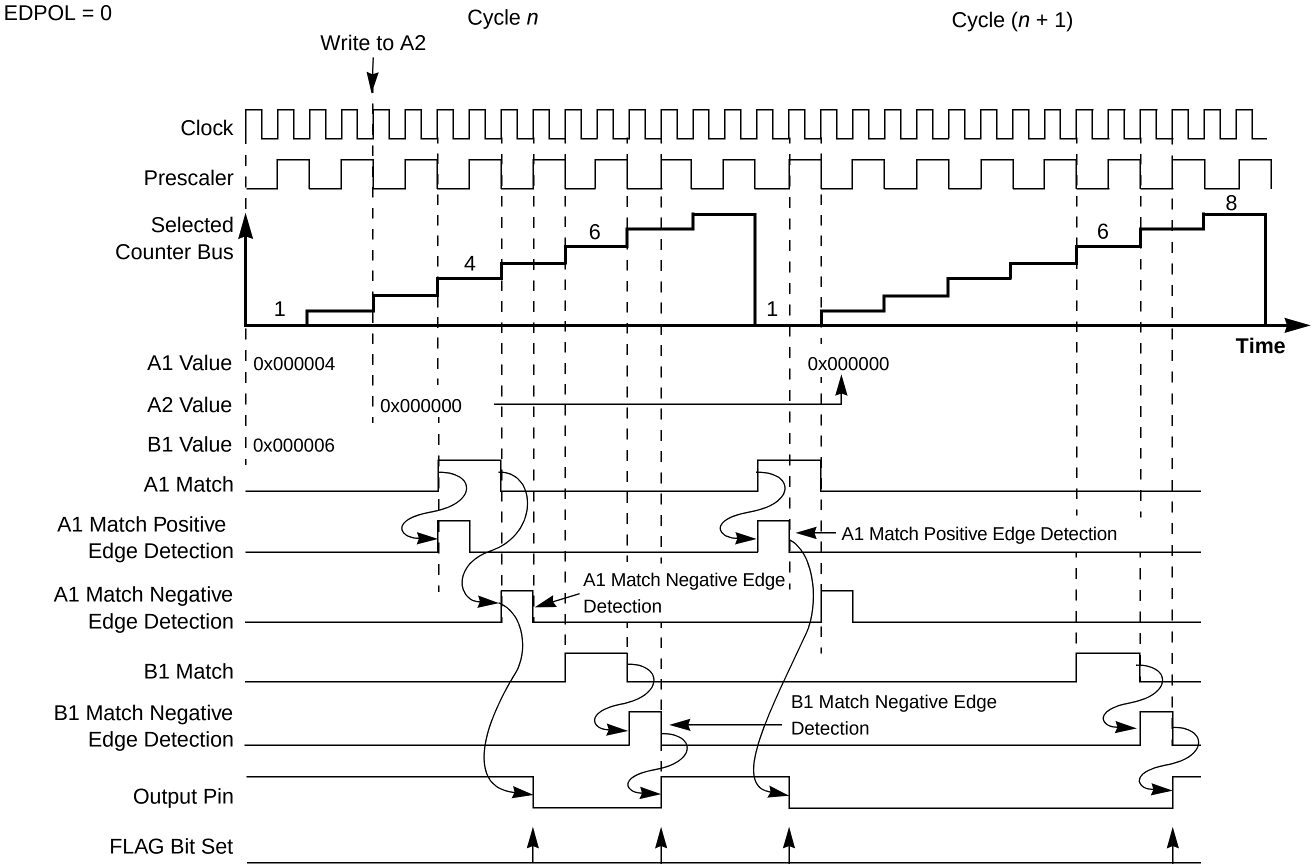


Figure 18-51. OPWMB Mode Matches and Flags

The output pin transitions are based on the negative edges of the A1 and B1 match signals. Figure 18-51 shows in cycle(*n* + 1) the value of the A1 register being set to 0. In this case, the match positive edge is used instead of the negative edge to transition the output flip-flop.

Figure 18-52 shows the channel operation for 0% duty cycle. Note that the A1 match positive edge signal occurs at the same time as the B1 = 0x00_0008 negative edge signal. In this case A1 match has precedence over B1 match, causing the output pin to remain at EDPOL bit value, thus generating a 0% duty cycle signal.

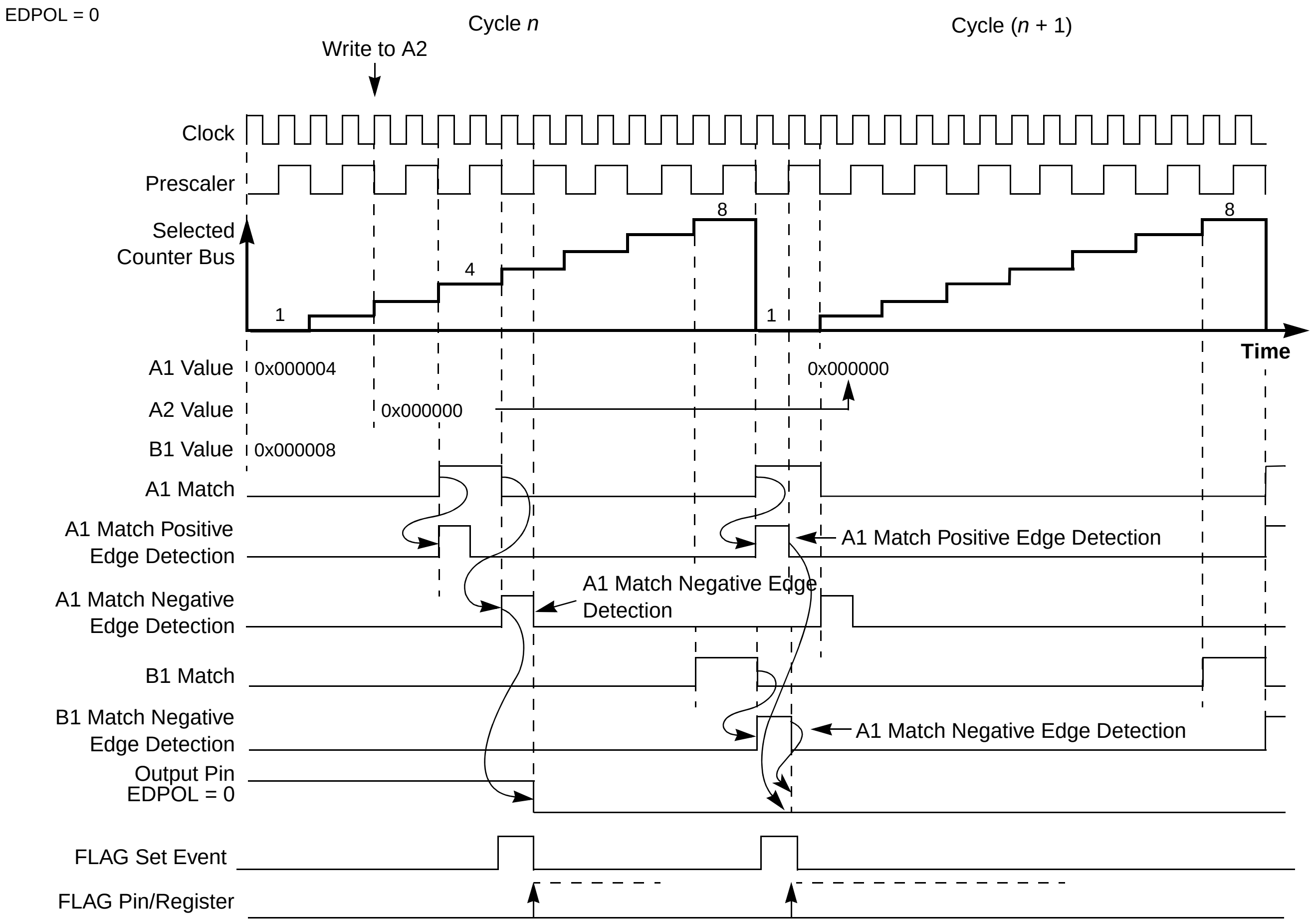


Figure 18-52. OPWMB Mode with 0% Duty Cycle

Figure 18-53 shows the operation of the OPWMB mode with the output disable signal asserted. The output disable forces a transition in the output pin to the EDPOL bit value. After deassertion, the output disable allows the output pin to transition at the following A1 or B1 match. The output disable does not modify the flag bit behavior. There is a delay of one system clock between the assertion of the output disable signal and the transition of the output pin to EDPOL.

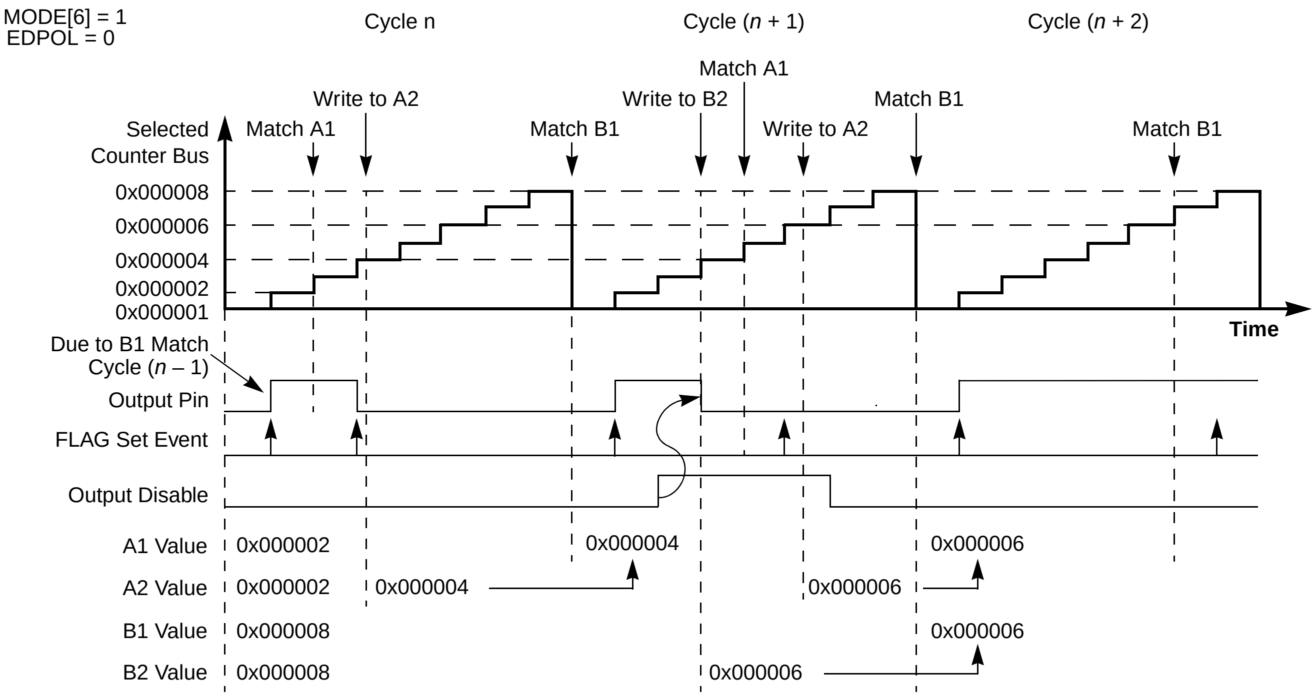


Figure 18-53. OPWMB Mode with Active Output Disable

Figure 18-54 shows a waveform changing from 100% to 0% duty cycle. In this case, EDPOL is 0. In this example, B1 is programmed to the same value as the period of the external selected time base.

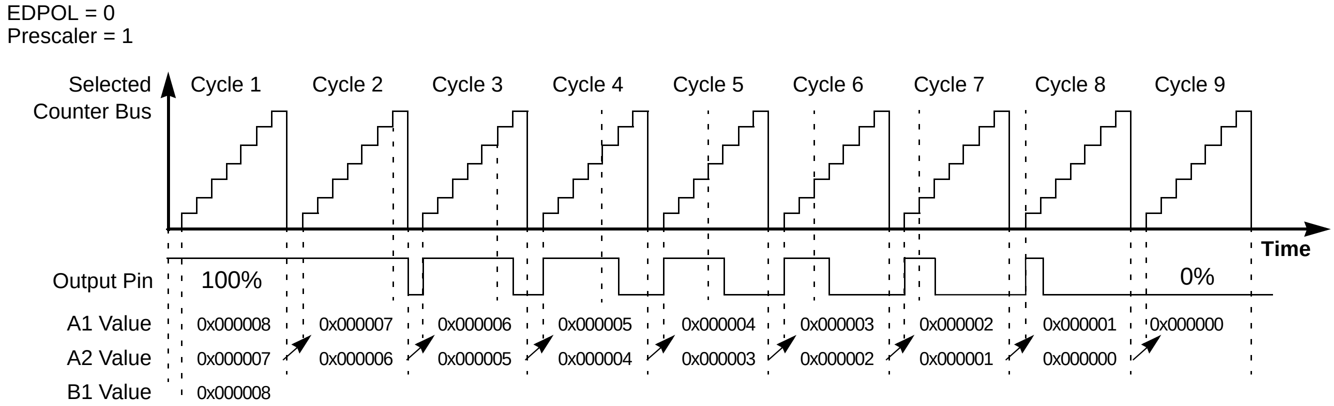


Figure 18-54. OPWMB Mode from 100% to 0% Duty Cycle

In Figure 18-54, if B1 is set to a value lower than 0x00_0008, it is not possible to achieve 0% duty cycle by changing only the A1 register value. Because B1 matches have precedence over A1 matches, the output pin transitions to the opposite of EDPOL bit at B1 match. If B1 is set to 0x00_0009, for instance, B1 match does not occur, thus a 0% duty cycle signal is generated.

18.4.1.2 Input Programmable Filter (IPF)

The IPF is a 5-bit programmable up counter that is incremented by the selected clock source, according to IF bits in EMIOS_CCR[*n*]. The IPF ensures that only valid input pin transitions are received by the unified channel edge detector. Figure 18-55 shows a block diagram of the IPF.

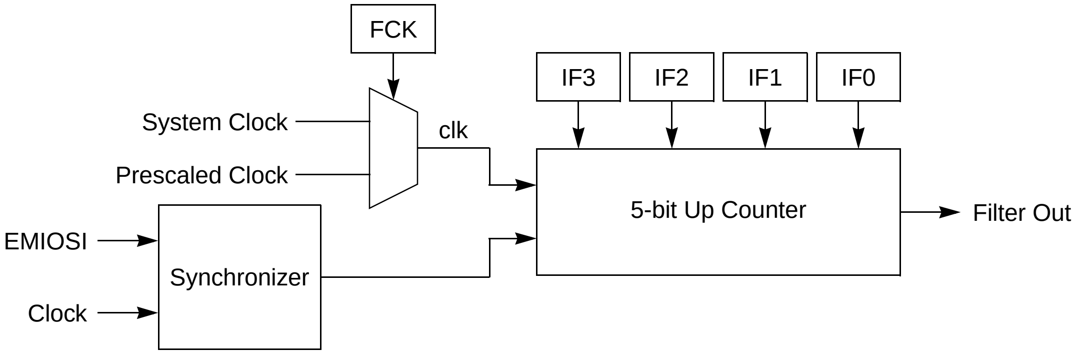


Figure 18-55. Input Programmable Filter Submodule Diagram

The input signal is synchronized by system clock. When a state change occurs in this signal, the 5-bit counter starts counting up. As long as the new state is stable on the pin, the counter remains incrementing. If a counter overflow occurs, the new pin value is validated. In this case, it is transmitted as a pulse edge to the edge detector. If the opposite edge appears on the pin before validation (overflow), the counter is reset. At the next pin transition, the counter starts counting again. Any pulse that is shorter than a full range of the masked counter is regarded as a glitch and it is not passed on to the edge detector. Figure 18-56 shows a timing diagram of the input filter.

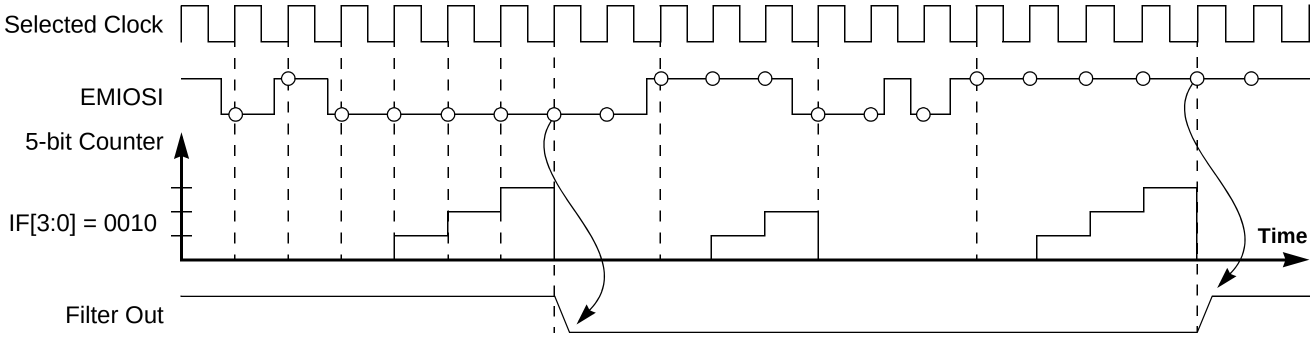


Figure 18-56. Input Programmable Filter Example

The filter is not disabled during either freeze state or negated GTBE input.

18.4.1.3 Clock Prescaler (CP)

The CP divides the GCP output signal to generate a clock enable for the internal counter of the unified channels. It is a programmable 2-bit down counter. The GCP output signal is prescaled by the value defined in the UCPRE bits in the EMIOS_CCR[*n*] register. The output is clocked every time the counter reaches zero. Counting is enabled by setting the UCPREN bit in the EMIOS_CCR[*n*]. The counter can be stopped at any time by clearing this bit, thereby stopping the internal counter in the unified channel.

In order to ensure safe working and avoid glitches the following steps must be performed whenever any update in the prescaling rate is desired:

1. Write 0 at both GPREN bit in EMIOSMCR register and UCPREN bit in EMIOS_CCR[*n*] register, thus disabling prescalers;
2. Write the desired value for prescaling rate at UCPRE[] bits in EMIOS_CCR[*n*] register;
3. Enable channel prescaler by writing 1 at UCPREN bit in EMIOS_CCR[*n*] register;
4. Enable global prescaler by writing 1 at GPREN bit in EMIOSMCR register.

The prescaler is not disabled during either freeze state or negated GTBE input.

18.4.1.4 Effect of Freeze on the Unified Channel

When in debug mode, if the FRZ bit in the EMIOS_MCR register and the FREN bit in the EMIOS_CCR[*n*] are both set, the internal counter and unified channel capture and compare functions are halted. The unified channel is frozen in its current state.

During freeze, all registers are accessible. When the unified channel is operating in an output mode, the force match functions remain available, allowing the software to force the output to the desired level.

During input modes, any input events that may occur while the channel is frozen are ignored.

When exiting debug mode or when the freeze enable bit is cleared (FRZ in the EMIOS_MCR or FREN in the EMIOS_CCR[*n*] register), the channel actions resume but may be inconsistent until the channel enters GPIO mode again.

18.4.2 IP Bus Interface Unit (BIU)

The BIU provides the interface between the internal interface bus (IIB) and the peripheral bus, allowing communication among all submodules and this IP interface.

The BIU allows 8-, 16-, and 32-bit access. They are performed over a 32-bit data bus in a single cycle clock.

18.4.2.1 Effect of Freeze on the BIU

When the FRZ bit in the EMIOS_MCR register is set and the module is in debug mode, the operation of BIU is not affected.

18.4.3 STAC Client Submodule

The shared time and angle count (STAC) bus provides access to one external time base, imported from the STAC bus to the eMIOS unified channels. The eTPU module's time bases and angle count can be exported and/or imported through the STAC client submodule interface. Time bases and/or angle information of either eTPU engine can be exported to the other eTPU engine and to the eMIOS module, which is only a STAC client. There are restrictions on engine export/import targets: one engine cannot export from or import to itself, nor can it import time base and/or angle count if in angle mode.

The device’s STAC server identification assignment is shown in Table 18-11. The time slot assignment is fixed, so only time bases running at system clock divided by four or slower can be integrally exported. The STAC client submodule runs with the system clock, and its time slot timing is synchronized with the eTPU timing on reset. The time slot sequence is 0-1-2-3, such that they alternate between engines one and two.

Table 18-11. STAC Client Submodule Server Slot Assignment

Engine	Time Base	Server ID
1	TCR1	0
1	TCR2	2
2	TCR1	1
2	TCR2	3

Figure 18-57 provides a block diagram for the STAC client submodule.

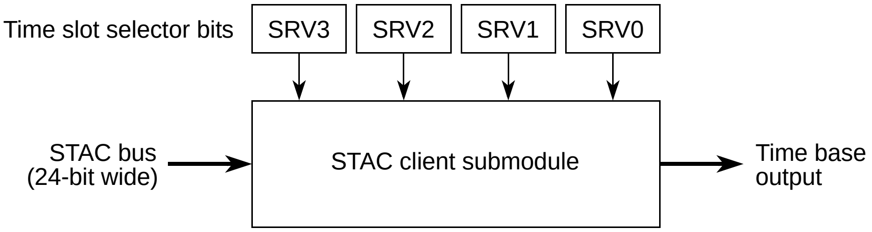


Figure 18-57. STAC Client Submodule Block Diagram

EMIOS_MCR[SRV] bits select the time slot of the STAC output bus. Figure 18-58 shows a timing diagram for the STAC client submodule.

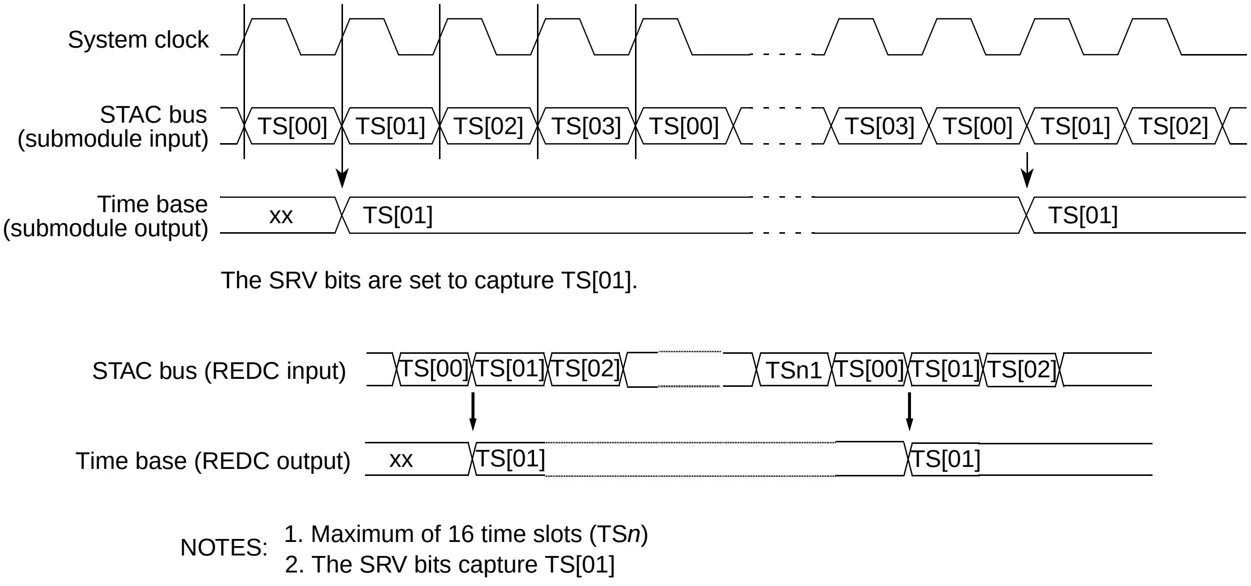


Figure 18-58. Timing Diagram for the STAC Bus and STAC Client Submodule Output

Every time the selected time slot changes, the STAC client submodule output is updated.

18.4.3.1 Effect of Freeze on the STAC Client Submodule

When the FRZ bit in the EMIOS_MCR is set and the module is in debug mode, the operation of the STAC client submodule is not affected; that is, there is no freeze function in this submodule.

18.4.4 Global Clock Prescaler Submodule (GCP)

The GCP divides the system clock to generate a clock for the clock prescalers (CP) of the unified channels. It is a programmable 8-bit up counter. The main clock signal is prescaled by the value defined in the GPRE bits in EMIOS_MCR. The output is clocked every time the counter overflows. Counting is enabled by setting the GPREN bit in the EMIOS_MCR. The counter can be stopped at any time by clearing this bit, thereby stopping the internal counter in all the unified channels.

In order to ensure safe working and avoid glitches the following steps must be performed whenever any update in the prescaling rate is desired:

1. Write 0 at GPREN bit in EMIOSMCR register, thus disabling global prescaler;
2. Write the desired value for prescaling rate at GPRE[0:7] bits in EMIOSMCR register;
3. Enable global prescaler by writing 1 at GPREN bit in EMIOSMCR register.

The prescaler is not disabled during either freeze state or negated GTBE input.

18.4.4.1 Effect of Freeze on the GCP

When the FRZ bit in the EMIOS_MCR register is set and the module is in debug mode, the operation of GCP submodule is not affected, i.e., there is no freeze function in this submodule.

18.5 Reset

The eMIOS200 is reset by the global asynchronous system reset signal.

The MDIS bit in the EMIOS_MCR register is cleared during reset.

On resetting the eMIOS200 all unified channels enter GPIO input mode.

18.6 Interrupts

The eMIOS200 can generate one interrupt per channel. An interrupt request is generated according to the configuration of the channel and input events or matches. See Chapter 27, “Interrupts and Interrupt Controller (INTC)”, for details on the eMIOS200 interrupt vectors.

18.7 Initialization/Application Information

On resetting the eMIOS200 all unified channels enter GPIO input mode.

18.7.1 Considerations

Before changing an operating mode, the unified channel must be programmed to GPIO mode and EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] registers must be updated with the correct values for the next operating mode. Then the EMIOS_CCR[*n*] register can be written with the new operating mode. If a unified channel is changed from one mode to another without performing this procedure, the first operation cycle of the selected time base can be random, i.e., matches can occur in random time if the contents of EMIOS_CADR[*n*] or EMIOS_CBDR[*n*] were not updated with the correct value before the time base matches the previous contents of EMIOS_CADR[*n*] or EMIOS_CBDR[*n*].

When interrupts are enabled, the software must clear the FLAG bits before exiting the interrupt service routine.

18.7.2 Application Information

Correlated output signals can be generated by all output operation modes. The OU[*n*] bits in EMIOS_OUDR can be used to control the update of these output signals.

In order to guarantee the internal counters of correlated channels are incremented in the same clock cycle, the internal prescalers must be set before enabling the global prescaler. If the internal prescalers are set after enabling the global prescaler, the internal counters may increment in the same ratio but at a different clock cycle.

18.7.3 Time Base Generation

For MC, OPWFM, and OPWM with internal clock source operation modes, the internal counter rate can be modified by configuring the clock prescaler ratio. Figure 18-59 shows an example of a time base with prescaler ratio equal to one. When the prescaler is greater than one, the counter is immediately cleared on a match and then incremented in the next prescaled clock edge, except when running in OPWFM mode or MC mode with internal clock source. In these cases, the counter will skip the next prescaled clock edge and continue incremented on subsequent edges, as shown in Figure 18-60.

NOTE

MCB, OPWFMB, and OPWMB modes have a different behavior.

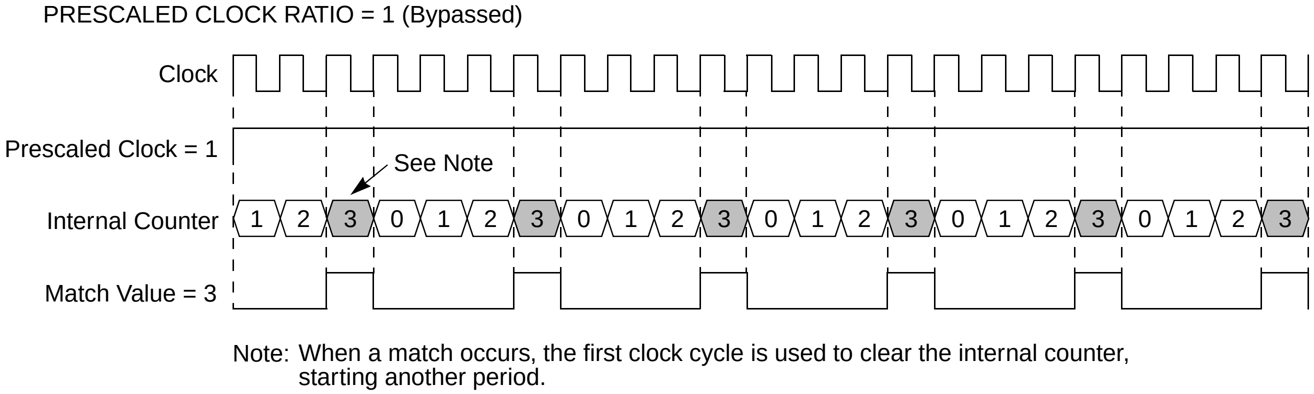


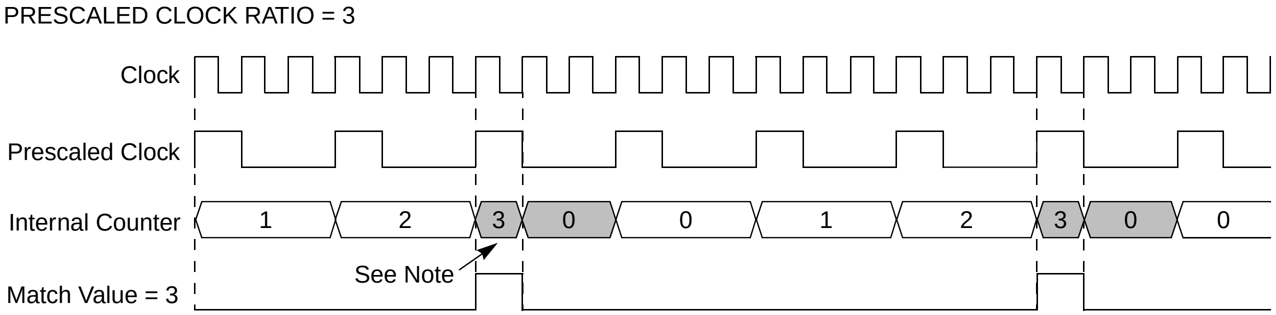
Figure 18-59. Time Base Period when Running in the Fastest Prescaler Ratio

If the prescaler ratio is greater than one or external clock is selected, the counter may behave in three different ways depending on the channel mode:

- If MC mode and Clear on Match Start and External Clock source are selected the internal counter behaves as described in Figure 18-61.
- If MC mode and Clear on Match Start and Internal Clock source are selected the internal counter behaves as described in Figure 18-62.
- If MC mode and Clear on Match End are selected the internal counter behaves as described in Figure 18-63.
- If OPWFM mode is selected the internal counter behaves as described in Figure 18-62. The internal counter clears at the start of the match signal, skips the next prescaled clock edge and then increments in the subsequent prescaled clock edge.

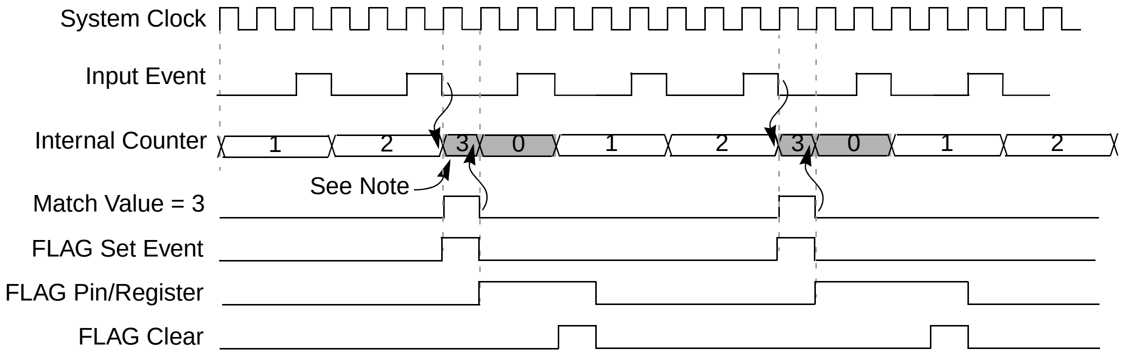
NOTE

MCB and OPWFMB modes have a different behavior.



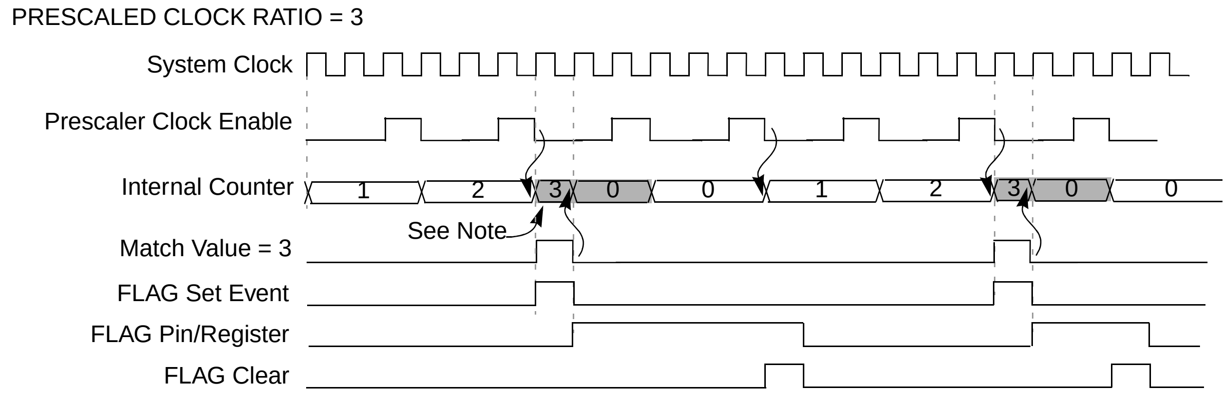
Note: When a match occurs, the first clock cycle is used to clear the internal counter, and only after a second edge of prescaled clock the counter will start counting.

Figure 18-60. Time Base Period when Running with a Prescaler Ratio Greater Than 1



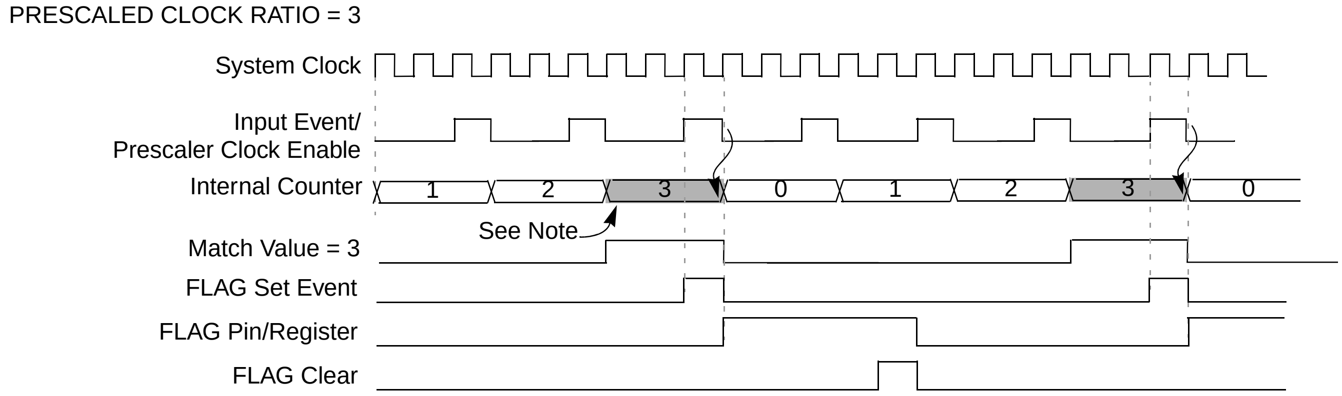
Note: When a match occurs, the first system clock cycle is used to clear the internal counter, and at the next edge of prescaler clock enable the counter will start counting.

Figure 18-61. Time Base Generation with External Clock and Clear on Match Start



Note: When a match occurs, the first clock cycle is used to clear the internal counter, and only after a second edge of prescaled clock the counter will start counting.

Figure 18-62. Time Base Generation with Internal Clock and Clear on Match Start



Note: The match occurs only when the input event/prescaler clock enable is active. Then, the internal counter is immediately cleared.

Figure 18-63. Time Base Generation with Clear on Match End

18.7.4 Coherent Accesses

For IPWM and IPM modes, it is recommended that the software wait for a new FLAG set event before reading EMIOS_CADR[*n*] and EMIOS_CBDR[*n*] registers to get a new measurement. The FLAG indicates that new data has been captured and it is the only way to assure data coherency.

The FLAG set event can be detected by polling the FLAG bit or by enabling the interrupt or DMA request generation.

Reading the EMIOS_CADR[*n*] register again in the same period of the last read of EMIOS_CBDR[*n*] register may lead to incoherent results. This occurs if the last read of EMIOS_CBDR[*n*] register occurred after a disabled B2 to B1 transfer.

Chapter 19
Enhanced Queued Analog-to-Digital Converter (eQADC)

19.1 Overview

The Enhanced Queued Analog-to-Digital Converter (EQADC) module provides fast and accurate conversions for a wide range of applications. There are two EQADC modules on the MPC5676R, EQADC_A and EQADC_B. Each EQADC module provides a parallel interface to two on-chip, independent analog-to-digital converter cores. Each EQADC module has a hardware interface to the twelve decimation filter blocks on the MPC5676R. This allows transferring of conversion and filtered values to and from the EQADC and decimation filters without CPU or DMA interaction. Each EQADC module has 24 dedicated external analog input pins, and 16 pins are shared by each module.

The EQADC transfers commands from multiple Command FIFOs (CFIFOs) to the on-chip ADCs. The multiple Result FIFOs (RFIFOs) can receive data from the on-chip ADCs or from an on-chip DSP module (decimation filters). Data from the on-chip ADCs can be routed to the side interface, processed by the decimation filters and then routed back through the side interface to the RFIFOs. The EQADC supports software and external hardware triggers (via package pins) from other blocks (eTPU and eMIOS) to initiate transfers of commands from the CFIFOs to the on-chip ADCs. It also monitors the fill level of the CFIFOs and RFIFOs, and accordingly generates DMA or interrupt requests to control data movement between the FIFOs and the system memory.

19.1.1 Analog to Digital Conversion Sub-system

Figure 19-1 shows how two eQADC modules fit into the overall analog to digital conversion sub-system.

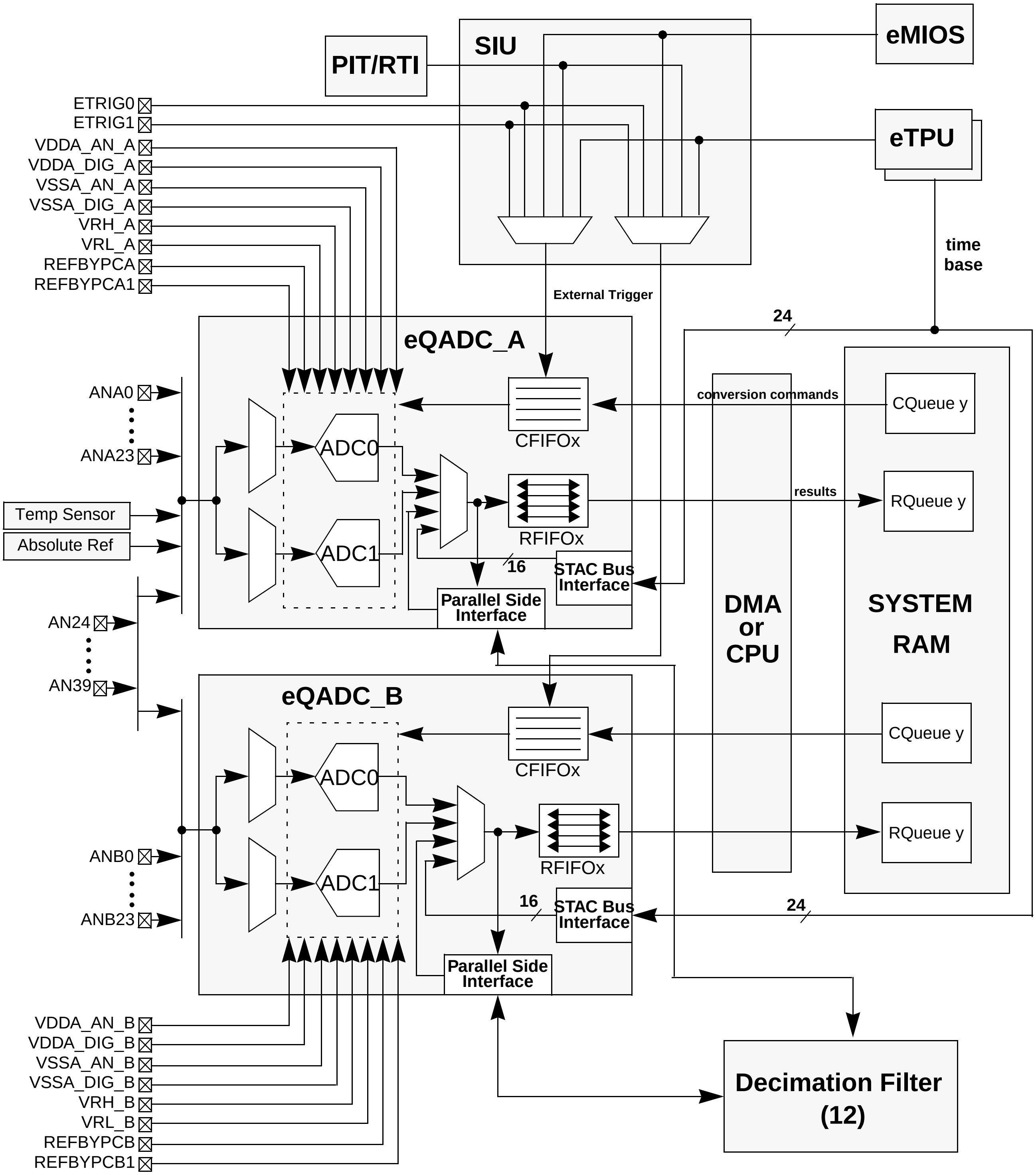


Figure 19-1. Analog to Digital Conversion Sub-system

19.2 Block Diagram

Figure 19-2 is the block diagram for an EQADC block (Table 19-1 describes the main sub-blocks).

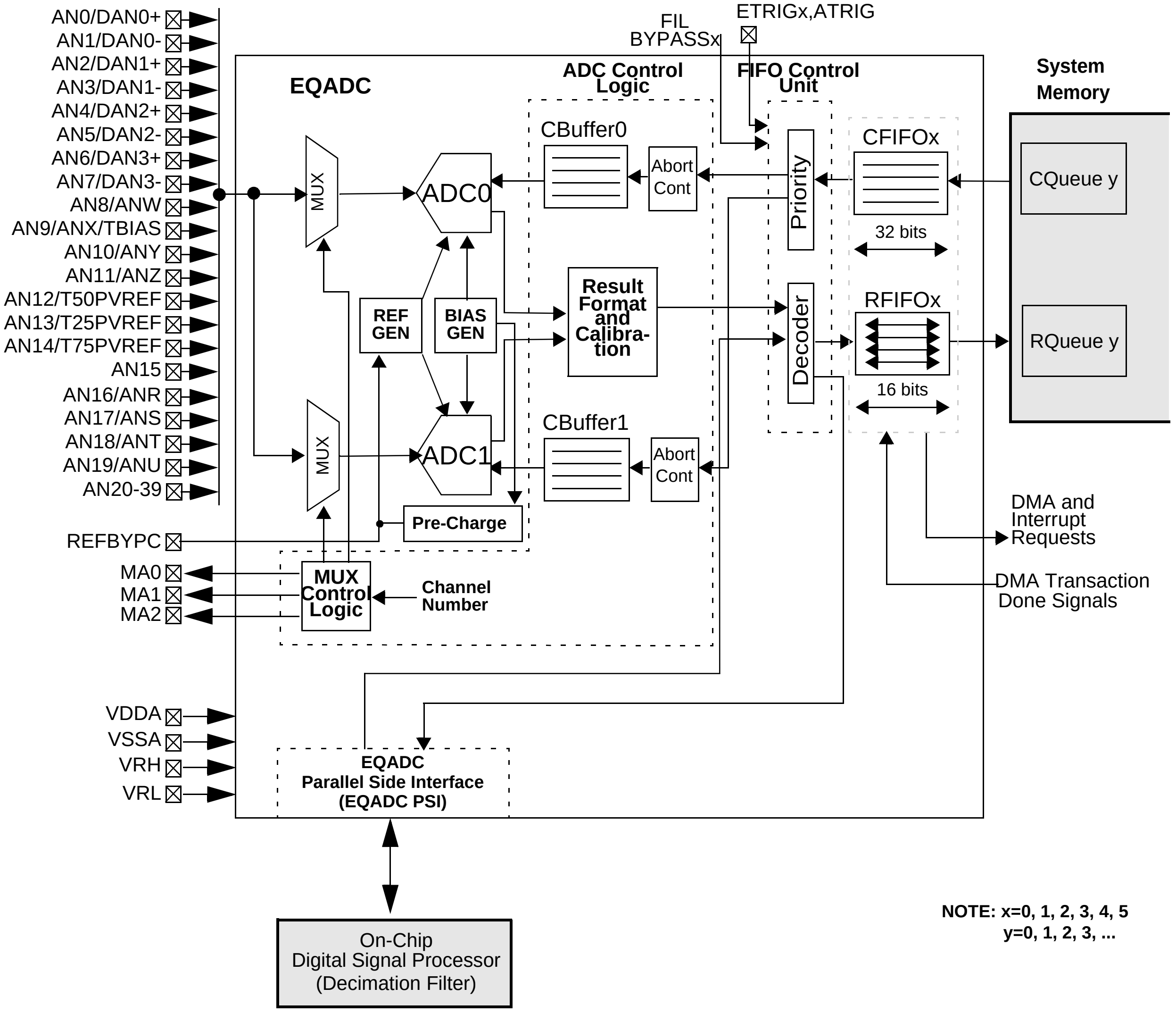


Figure 19-2. EQADC Block Diagram

Table 19-1 shows the primary components inside the EQADC.

Table 19-1. EQADC Primary Component Descriptions

Component	Function
FIFO Control Unit	Controls the CFIFOs and the RFIFOs: • Prioritizes the CFIFOs to determine which CFIFOs will have their commands transferred. • Supports software and hardware triggers to start command transfers from a particular CFIFO. • Decodes command data from the CFIFOs and, accordingly, sends these commands to one of the two on-chip ADCs. • Decodes result data from on-chip ADCs and transfers data to the appropriate RFIFO or to the parallel side interface.
ADC Control Logic	Manages the execution of commands bound for on-chip ADCs: • interfaces with the CFIFOs via two 2-entry command buffers (CBuffers) with abort control and with the RFIFOs via the Result Format and Calibration Sub-Block. • Buffers command data for execution. • Decodes command data and, accordingly, generates control signals for the two on-chip ADCs. • Detects abort requests, stores aborted commands and buffers immediate conversion commands. • Formats and calibrates conversion result data coming from the on-chip ADCs. • Generates the internal multiplexer control signals and the select signals used by the external multiplexers.
EQADC Parallel Side Interface (EQADC PSI)	Allows for a full duplex, synchronous, parallel communication between the EQADC and decimation filters.

Figure 19-2 also depicts data flow through the EQADC. Commands are contained in system memory in a user defined data structure. The most likely data structure to be used is a queue (as shown in Figure 19-2[1]). Command data is moved from the command queue (CQueue) to the CFIFOs by either the host CPU or by the system DMA controller. Once a CFIFO is triggered and becomes the highest priority CFIFO using a certain CBuffer, command data is transferred from the CFIFO to the on-chip ADCs. The ADC executes the command, and the result is moved through the *Result Format and Calibration Sub-Block* to either the side interface or to the RFIFO. Data from the on-chip companion module (decimation filter) bypasses the *Result Format and Calibration Sub-Block* and is moved directly to its specified RFIFO. When data is stored in an RFIFO, data is moved from the RFIFO by the host CPU or by the system DMA controller to a data structure in system memory as a result queue (RQueue).

If you are familiar with the QADC, the EQADC system upgrades the functionality provided by that block. Refer to Section 19.8.7, "EQADC Versus QADC," for a comparison between the EQADC and QADC.

---

[1] Command and result data can be stored in system memory in any user defined data structure. However, in this document it will be assumed that the data structure of choice is a queue, since it is the most likely data structure to be used and because queues are the only type of data structure supported by the DMAC.

19.2.1 Features

Each EQADC block includes these distinctive features (except where noted):

- Two independent on-chip RSD Cyclic ADCs
  - 8, 10, and 12 bits AD Resolution
  - Differential conversions
  - Differential channels include variable gain amplifier for improved dynamic range (x1; x2; x4)
  - Differential channels include programmable pull-up and pull-down resistors for biasing and sensor diagnostics (200k ohms; 100k ohms; 5k ohms)
  - Sample times of 2 (default), 8, 64 or 128 ADC clock cycles
  - Each conversion result can be marked with an imported timestamp from the eTPU, or an independent timestamp
  - Parallel interface to EQADC CFIFOs and RFIFOs
  - Supports both right-justified unsigned and signed formats for conversion results
  - Two REFBYPC pins for each EQADC module: REFBYPC25 and REFBYPC75
  - Temperature sensor (available only to the primary ADC pair (eQADC_A’s ADC0 and ADC1)
  - Ability to directly measure Vdd
- Automatic application of ADC calibration constants
- Parallel Side Interface allows eQADC_B to route conversion results without CPU intervention to the Decimation Filter block for signal processing
- Priority Based CFIFOs
  - Supports six CFIFOs with fixed priority. The lower the CFIFO number, the higher its priority. When commands of distinct CFIFOs are bound for the same CBuffer, the higher priority CFIFO is always served first.
  - Immediate conversion command feature with conversion abort control
  - Streaming mode operation of CFIFO0 to execute defined commands multiple times
  - Supports software and several hardware trigger modes to arm a particular CFIFO
  - Generates interrupt when command coherency is not achieved
- External (to the eQADC) Hardware Triggers
  - Supports rising edge, falling edge, high level and low level triggers
  - Supports configurable digital filter
  - Supports controls to bypass the trigger digital filters (refer to the SIU chapter)
- Two Triggers operation mode for queue0
  - Additional internal trigger (not filtered) called Advance trigger that is used to enable the external trigger of queue0 and to control the loop behavior of CFIFO0 (only available on EQADC_B)
- Supports 4 to 8 external 8-to-1 muxes which can expand the input channel number from 40 to 96

19.3 Modes of Operation

This section describes the operation modes of the EQADC.

19.3.1 Normal Mode

This is the default operational mode when the EQADC is not in streaming mode or background debug or stop mode.

19.3.2 Streaming Mode

This mode is characterized by two main aspects: the abort action by CFIFO0 in any current conversion process started from another queue, and the loop behavior of the CFIFO0.

In some applications, there may be sequences of identical commands each spaced only by a few microseconds. To reduce the DMA data transfer, in this mode a short command queue can be stored in CFIFO0 and repeatedly be executed based on a timed trigger, but advance to the next (repeating) sequence of commands based on another device’s internal trigger.

The CFIFO0 delivers commands to the ADC as before, but those commands are not ‘invalidated’ after they are sent (in fact, they are ‘invalidated’ only because the Transfer Next Data Pointer has moved on). When it encounters these repeated commands the CFIFO0 only fills once, using the DMA as usual, until either it is full or a command with End-of-Queue is encountered. Thereafter the sub-queue repeats/wraps. The number of commands loaded is unaffected by the delivery of commands once the streaming mode is configured, since no commands loaded are invalidated even if sent before all the queue is loaded.

The number of entries in the CFIFO0 is extended to eight (configurable). This is to facilitate the targeted applications. The repeating subqueue must be contained within the eight CFIFO0 entries.

To maintain compatibility, CFIFO0 by default operates as it does before, without streaming and with four entries. Streaming, and additional entries, can be enabled independently.

Streaming mode is selected as another mode for queue 0 using the configuration bits in the EQADC_CFCR register. Streaming mode makes use of an additional bit in the Conversion Command Word (CCW); this bit is called ‘Repeat’. The purpose of this bit is to mark in the command queue, where to start a repeating sequence.

Streaming mode requires 2 trigger inputs. The standard queue 0 trigger, in this mode referred to as ‘Repeat Trigger’ and a second internal trigger input to the eQADC called ‘Advance’ trigger.

19.3.3 Debug Mode

Upon a debug mode entry request, EQADC behavior will vary according to the status of the DBG field in the EQADC_MCR. If DBG is programmed to 0b00, the debug mode entry request is ignored. If DBG is programmed to 0b10 or to 0b11, the EQADC will enter debug mode.

During debug mode, the EQADC will not transfer commands from any CFIFOs, no data will be returned to any RFIFO, no hardware trigger event will be captured, and all EQADC registers can be accessed as in Normal mode. The latter implies that CFIFOs can still be triggered using software triggers, since no scheme is implemented to write-protect registers during debug mode. DMA and interrupt requests continue to be generated as in Normal Mode.

If at the time the debug mode entry request is detected, there are commands in the on-chip CBuffers that were already under execution, these commands will be completed but the generated results, if any, will not be sent to the RFIFOs until debug mode is exited. Commands whose execution has not started will not be executed until debug mode is exited.The clock associated with an on-chip ADC stops, during its low phase, after the ADC ceases executing commands. The time base counter will only stop after all on-chip ADCs cease executing commands.

When exiting debug mode, the EQADC relies on the CFIFO operation modes and on the CFIFO status to determine the next command entry to transfer.

The EQADC's internal behavior after the debug mode entry request is detected differs depending on the status of command transfers.

- No command transfer is in progress.

  The EQADC immediately halts future command transfers from any CFIFO.
- Command transfer is in progress.

  EQADC will complete the transfer and update CFIFO status before halting future command transfers from any CFIFO. Command transfers to the internal CBuffers are considered completed when a command is written to the buffers.

19.3.4 Stop Mode

Upon a stop mode entry request detection, the EQADC progressively halts its operations until it reaches a static, stable state from which it can recover when returning to Normal mode. EQADC then asserts an acknowledge signal, indicating that it is static and that the clock input can be stopped.The latter implies that, as long as the peripheral clock is running, CFIFOs can still be triggered using software triggers, since no scheme is implemented to write-protect registers during stop mode.

If at the time the stop mode entry request is detected, there are commands in the on-chip CBuffers that were already under execution, these commands will be completed but the generated results, if any, will not be sent to the RFIFOs until stop mode is exited. Commands whose execution has not started will not be executed until stop mode is exited.

After these remaining commands are executed, the clock input to the ADCs and the bias generator circuit is stopped. The ADC clock stops during its low phase. The time base counter will only stop after all on-chip ADCs cease executing commands. Only then, the stop acknowledge signal is asserted. When exiting stop mode, the EQADC relies on the CFIFO operation modes and on the CFIFO status to determine the next command entry to transfer.

The EQADC internal behavior after the stop mode entry request is detected differs depending on the status of the command transfer.

- No command transfer is in progress

  The EQADC immediately halts future command transfers from any CFIFO.
- Command transfer is in progress

EQADC will complete the transfer and update CFIFO status before halting future command transfers from any CFIFO. Command transfers to the internal CBuffers are considered completed when a command is written to the buffers.

19.4 External Signal Description

19.4.1 Overview

The following is a list of external pins (applies to both EQADC_A and EQADC_B).

NOTE

At chip integration level, some of the digital and analog signals listed here might share pins or not be available external to the chip. Refer to the Signals chapter for details.

Table 19-2. eQADC Signals

Name	Port	Function	Reset State	Type
AN0/DAN0+	Input	Single-ended analog input / Differential analog input positive terminal	—	Analog
AN1/DAN0-	Input	Single-ended analog input / Differential analog input negative terminal	—	Analog
AN2/DAN1+	Input	Single-ended analog input / Differential analog input positive terminal	—	Analog
AN3/DAN1-	Input	Single-ended analog input / Differential analog input negative terminal	—	Analog
AN4/DAN2+	Input	Single-ended analog input / Differential analog input positive terminal	—	Analog
AN5/DAN2-	Input	Single-ended analog input / Differential analog input negative terminal	—	Analog
AN6/DAN3+	Input	Single-ended analog input / Differential analog input positive terminal	—	Analog
AN7/DAN3-	Input	Single-ended analog input / Differential analog input negative terminal	—	Analog
AN8/ANW	Input	Single-ended analog input / Single-ended analog input from external multiplexers	—	Analog
AN9/ANX	Input	Single-ended analog input / Single-ended analog input from external multiplexers	—	Analog
AN10/ANY	Input	Single-ended analog input/Single-ended analog input from external multiplexers	—	Analog
AN11/ANZ	Input	Single-ended analog input / Single-ended analog input from external multiplexers	—	Analog
AN12	Input / Output	Single-ended analog input	—	Analog
AN13	Input / Output	Single-ended analog input	—	Analog
AN14	Input / Output	Single-ended analog input	—	Analog
AN15	Input	Single-ended analog input	—	Analog
AN16/ANR[1]	Input	Single-ended analog input / Single-ended analog input from external multiplexers	—	Analog

Table 19-2. eQADC Signals (continued)

Name	Port	Function	Reset State	Type
AN17/ANS[1]	Input	Single-ended analog input / Single-ended analog input from external multiplexers	—	Analog
AN18/ANT[1]	Input	Single-ended analog input / Single-ended analog input from external multiplexers	—	Analog
AN19/ANU[1]	Input	Single-ended analog input / Single-ended analog input from external multiplexers	—	Analog
AN20–AN39	Input	Single-ended analog inputs	—	Analog
MA0	Output	External multiplexer control signal	0	Digital
MA1	Output	External multiplexer control signal	0	Digital
MA2	Output	External multiplexer control signal	0	Digital
VDDA	Input	Analog Positive Power Supply	—	Power
VSSA	Input	Analog Negative Power Supply	—	Power
VRH	Input	Voltage Reference High	—	Power
VRL	Input	Voltage Reference Low	—	Power
REFBYPC	Input	External Bypass capacitor Pin	—	Power
ETRIG0	Input	External trigger for CFIFO0	—	Digital
ETRIG1	Input	External trigger for CFIFO1	—	Digital
ETRIG2	Input	External trigger for CFIFO2	—	Digital
ETRIG3	Input	External trigger for CFIFO3	—	Digital
ETRIG4	Input	External trigger for CFIFO4	—	Digital
ETRIG5	Input	External trigger for CFIFO5	—	Digital

[1] Can be disabled or not using configuration parameters.

19.4.2 Detailed Signal Descriptions

Table 19-3 shows the EQADC external signals.

Table 19-3. EQADC External Signals

Signal Name	Description
AN0/DAN0+	Single-ended analog input/Differential analog input positive terminal. AN0 is a single-ended analog input to the two on-chip ADCs. DAN0+ is the positive terminal of the differential analog input DAN0 (DAN0+ - DAN0-).
AN1/DAN0-	Single-ended analog input/Differential analog input negative terminal. AN1 is a single-ended analog input to the two on-chip ADCs. DAN0- is the negative terminal of the differential analog input DAN0 (DAN0+ - DAN0-).
AN2/DAN1+	Single-ended analog input/Differential analog input positive terminal. AN2 is a single-ended analog input to the two on-chip ADCs. DAN1+ is the positive terminal of the differential analog input DAN1 (DAN1+ - DAN1-).
AN3/DAN1-	Single-ended analog input/Differential analog input negative terminal. AN3 is a single-ended analog input to the two on-chip ADCs. DAN1- is the negative terminal of the differential analog input DAN1 (DAN1+ - DAN1-).
AN4/DAN2+	Single-ended analog input/Differential analog input positive terminal. AN4 is a single-ended analog input to the two on-chip ADCs. DAN2+ is the positive terminal of the differential analog input DAN2 (DAN2+ - DAN2-).
AN5/DAN2-	Single-ended analog input/Differential analog input negative terminal. AN5 is a single-ended analog input to the two on-chip ADCs. DAN2- is the negative terminal of the differential analog input DAN2 (DAN2+ - DAN2-).
AN6/DAN3+	Single-ended analog input/Differential analog input positive terminal. AN6 is a single-ended analog input to the two on-chip ADCs. DAN3+ is the positive terminal of the differential analog input DAN3 (DAN3+ - DAN3-).
AN7/DAN3-	Single-ended analog input/Differential analog input negative terminal. AN7 is a single-ended analog input to the two on-chip ADCs. DAN3- is the negative terminal of the differential analog input DAN3 (DAN3+ - DAN3-).
AN8/ANW	Single-ended analog input/ Single-ended analog input from external multiplexers. AN8 is a single-ended analog input to the two on-chip ADCs. ANW is a single-ended analog input to one of the on-chip ADCs in external multiplexed mode.
AN9/ANX	Single-ended analog input/ Single-ended analog input from external multiplexers/Test Bias. AN9 is a single-ended analog input to the two on-chip ADCs. ANX is a single-ended analog input to one of the on-chip ADCs in external multiplexed mode.
AN10/ANY	Single-ended analog input/ Single-ended analog input from external multiplexers. AN10 is a single-ended analog input to the two on-chip ADCs. ANY is a single-ended analog input to one of the on-chip ADCs in external multiplexed mode.
AN11/ANZ	Single-ended analog input/ Single-ended analog input from external multiplexers. AN11 is a single-ended analog input to the two on-chip ADCs. ANZ is a single-ended analog input to one of the on-chip ADCs in external multiplexed mode.
AN12	Single-ended analog input/ Test 50% VREF analog output. AN12 is a single-ended analog input to the two on-chip ADCs.
AN13	Single-ended analog input/ Test 25% VREF analog output. AN13 is a single-ended analog input to the two on-chip ADCs.
AN14	Single-ended analog input/ Test 75% VREF analog output. AN14 is a single-ended analog input to the two on-chip ADCs.

Table 19-3. EQADC External Signals

Signal Name	Description
AN15	Single-ended analog input. AN15 is a single-ended analog inputs to the two on-chip ADCs.
AN16/ANR	Single-ended analog input/ Single-ended analog input from external multiplexers. AN16 is a single-ended analog input to the two on-chip ADCs. ANR is a single-ended analog input to one of the on-chip ADCs in external multiplexed mode. The external multiplexing capability is a device option.
AN17/ANS	Single-ended analog input/ Single-ended analog input from external multiplexers. AN17 is a single-ended analog input to the two on-chip ADCs. ANS is a single-ended analog input to one of the on-chip ADCs in external multiplexed mode. The external multiplexing capability is a device option.
AN18/ANT	Single-ended analog input/ Single-ended analog input from external multiplexers. AN18 is a single-ended analog input to the two on-chip ADCs. ANT is a single-ended analog input to one of the on-chip ADCs in external multiplexed mode. The external multiplexing capability is a device option.
AN19/ANU	Single-ended analog input/ Single-ended analog input from external multiplexers. AN19 is a single-ended analog input to the two on-chip ADCs. ANU is a single-ended analog input to one of the on-chip ADCs in external multiplexed mode. The external multiplexing capability is a device option.
AN20-AN39	Single-ended analog input. AN20 through AN39 are single-ended analog inputs to the two on-chip ADCs.
INA_ADC0_0 - INA_ADC0_9	(not external to the chip) Single-ended analog input. INA_ADC0_0 through INA_ADC0_9 are single-ended analog inputs to the on-chip ADC0.
INA_ADC1_0 - INA_ADC1_9	(not external to the chip) Single-ended analog input. INA_ADC1_0 through INA_ADC1_9 are single-ended analog inputs to the on-chip ADC1.
MA0-MA2	External multiplexer control signals. MA0, MA1, and MA2 combined form a select signal associated with external multiplexers.
VRH, VRL	Voltage reference high and voltage reference low. VRH and VRL are voltage references for the ADCs. VRH is the highest voltage reference, while VRL is the lowest voltage reference.
VDDA, VSSA	5V VDD and VSS for the 5V analog components. VDDA is the positive power supply pin for the ADCs and VSSA is the negative power supply pin for the ADCs. Refer to electrical specifications.
REFBYPC	Reference Bypass Capacitor The REFBYPC pin is used to connect an external bypass capacitor between REFBYPC and VRL. The value of this capacitor should be 100nf. This bypass capacitor is used to provide a stable reference voltage for the ADC.
ETRIG0-ETRIG5	External trigger The external trigger signals are for hardware triggering. The EQADC can detect rising edge, falling edge, high level and low level on each of the external trigger signals. ETRIGx triggers CFIFOx. The EQADC also uses digital filters for these external trigger signals.

19.5 Pin Mapping to Channel Mapping

Table 19-4 shows the pin mapping to channel mapping to show which signals go to which ADC. For internal and external multiplexing channel assignments, see Table 19-45.

Table 19-4. Pin Mapping to Channel Mapping

Analog Input Pin 416 package	Function	ADC Number		eQADC_A Channel Number	eQADC_B Channel Number
		eQADC_A	eQADC_B		
ANA0–ANA23[1]	Single ended conversion	0, 1	—	0–23	—
ANB0–ANB23	Single ended conversion	—	0,1	—	0–23
AN24–AN39	Single ended conversion	0, 1	0, 1	24–39	24–39
—	VRH	0, 1	0, 1	40	40
—	VRL	0, 1	0, 1	41	41
—	50% (VRH–VRL)[2]	0, 1	0, 1	42	42
—	75% (VRH–VRL)	0, 1	0, 1	43	43
—	25% (VRH–VRL)	0, 1	0, 1	44	44
—	Local Band Gap 1220mV	0, 1	—	45	45
ANA8	ANWA[3]	0, 1	0, 1	64–71	—
ANB8	ANWB[3]	—	0, 1	—	64–71
ANA9	ANXA[3]	0, 1	—	72–79	—
ANB9	ANXB[3]	—	0, 1	—	72–79
ANA10	ANYA[3]	0, 1	—	80–87	—
ANB10	ANYB[3]	—	0, 1	—	80–87
ANA11	ANZA[3]	0, 1	—	88–95	—
ANB11	ANZB[3]	—	0, 1	—	88–95
ANA0 and ANA1	DANA0+ and DANA0-	0, 1	—	96	—
ANB0 and ANB1	DANB0+ and DANB0-	—	0, 1	—	96
ANA2 and ANA3	DANA1+ and DANA1-	0, 1	—	97	—
ANB2 and ANB3	DANB1+ and DANB1-	—	0, 1	—	97
ANA4 and ANA5	DANA2+ and DANA2-	0, 1	—	98	—
ANB4 and ANB5	DANB2+ and DANB2-	—	0, 1	—	98
ANA6 and ANA7	DANA3+ and DANA3-	0, 1	—	99	—
ANB6 and ANB7	DANB3+ and DANB3-	—	0, 1	—	99
—	Temperature Sensor	0, 1	—	128	—
—	PMC band gap 0.62 V [4]	0	—	145	—
—	VDD	0	—	146	—
—	Internal 1.2 V regulator	0	—	147	—
—	VDDEH1 50%	0	—	162	—

Table 19-4. Pin Mapping to Channel Mapping (continued)

Analog Input Pin 416 package	Function	ADC Number		eQADC_A Channel Number	eQADC_B Channel Number
		eQADC_A	eQADC_B		
—	VDDEH3 50%	0	—	163	—
—	VDDEH4 50%	0	—	164	—
—	VDDEH5 50%	0	—	165	—
—	VDDEH6 50%	0	—	166	—
—	VDDEH7 50%	0	—	167	—
—	1.2 V LVD VDD	0	—	180	—
—	3.3 V Internal regulator	0	—	181	—
—	3.3 V LVD VDDSYN	0	—	182	—
—	5.0 V LVD VDDREG	0	—	183	—
—	Standby regulator out	1	—	194	—
—	Standby source bias	1	—	195	—
—	4.75 V LVD VDDA (ADC band gap)	1	—	196	—
ANA16	ANRA[3]	0, 1	—	224–231	—
ANB16	ANRB[3]	—	0, 1	—	224–231
ANA17	ANSA[3]	0, 1	—	232–239	—
ANB17	ANSB[3]	—	0, 1	—	232–239
ANA18	ANTA[3]	0, 1	—	240–247	—
ANB18	ANTB[3]	—	0, 1	—	240–247
ANA19	ANUA[3]	0, 1	—	248–255	—
ANB19	ANUB[3]	—	0, 1	—	248–255

[1] AN*x*8–AN*x*11 and AN*x*16–AN*x*19 have alternate functions when used with an external multiplexer.

[2] After calibration, the 50% reference point will read approximately 20 mV below 50%(VRH - VRL). The 50% reference point should not be used to calibrate the EQADC. Use the 25% and 75% reference points to calibrate the EQADC.

[3] This function is when using an external multiplexer.

[4] Refer to the PMC chapter.

19.6 Memory Map/Register Definition

This section provides memory maps and detailed descriptions of all registers. Data written to or read from reserved areas of the memory map is undefined.

19.6.1 EQADC Memory Map

This section provides memory maps for the EQADC block.

Table 19-5. EQADC Memory Map

Address Offset[1]	Register	Access	Section/Page
0x0000	EQADC_MCR—EQADC Module Configuration Register	R/W	19.6.2.1/19-16
0x0004	Reserved	—	
0x0008	Reserved	—	
0x000C	EQADC_ETDFR—EQADC External Trigger Digital Filter Register	R/W	19.6.2.2/19-17
0x0010	EQADC_CFPR0—EQADC CFIFO Push Register 0	W	19.6.2.3/19-19
0x0014	EQADC_CFPR1—EQADC CFIFO Push Register 1	W	
0x0018	EQADC_CFPR2—EQADC CFIFO Push Register 2	W	
0x001C	EQADC_CFPR3—EQADC CFIFO Push Register 3	W	
0x0020	EQADC_CFPR4—EQADC CFIFO Push Register 4	W	
0x0024	EQADC_CFPR5—EQADC CFIFO Push Register 5	W	
0x0028	Reserved	—	
0x002C	Reserved	—	
0x0030	EQADC_RFPR0—EQADC Result FIFO Pop Register 0	R	19.6.2.4/19-19
0x0034	EQADC_RFPR1—EQADC Result FIFO Pop Register 1	R	
0x0038	EQADC_RFPR2—EQADC Result FIFO Pop Register 2	R	
0x003C	EQADC_RFPR3—EQADC Result FIFO Pop Register 3	R	
0x0040	EQADC_RFPR4—EQADC Result FIFO Pop Register 4	R	
0x0044	EQADC_RFPR5—EQADC Result FIFO Pop Register 5	R	
0x0048	Reserved	—	—
0x004C	Reserved	—	—
0x0050	EQADC_CFCR0-1—EQADC CFIFO Control Register 0 and 1	R/W	19.6.2.5/19-20
0x0054	EQADC_CFCR2-3—EQADC CFIFO Control Register 2 and 3	R/W	
0x0058	EQADC_CFCR4-5—EQADC CFIFO Control Register 4 and 5	R/W	
0x005C	Reserved	—	—
0x0060	EQADC_IDCR0—EQADC Interrupt and DMA Control Register 0	R/W	19.6.2.6/19-23
0x0064	EQADC_IDCR1—EQADC Interrupt and DMA Control Register 1	R/W	
0x0068	EQADC_IDCR2—EQADC Interrupt and DMA Control Register 2	R/W	
0x006C	Reserved	—	—
0x0070	EQADC_FISR0—EQADC FIFO and Interrupt Status Register 0	R/W	19.6.2.7/19-26
0x0074	EQADC_FISR1—EQADC FIFO and Interrupt Status Register 1	R/W	
0x0078	EQADC_FISR2—EQADC FIFO and Interrupt Status Register 2	R/W	
0x007C	EQADC_FISR3—EQADC FIFO and Interrupt Status Register 3	R/W	
0x0080	EQADC_FISR4—EQADC FIFO and Interrupt Status Register 4	R/W	
0x0084	EQADC_FISR5—EQADC FIFO and Interrupt Status Register 5	R/W	

Table 19-5. EQADC Memory Map (continued)

Address Offset[1]	Register	Access	Section/Page
0x0088	Reserved	—	—
0x008C	Reserved	—	—
0x0090	EQADC_CFTCR0—EQADC CFIFO Transfer Counter Register 0	R/W	19.6.2.8/19-30
0x0094	EQADC_CFTCR1—EQADC CFIFO Transfer Counter Register 1	R/W	
0x0098	EQADC_CFTCR2—EQADC CFIFO Transfer Counter Register 2	R/W	
0x009C	Reserved	—	—
0x00A0	EQADC_CFSSR0—EQADC CFIFO Status Snapshot Register 0	R	19.6.2.9/19-31
0x00A4	EQADC_CFSSR1—EQADC CFIFO Status Snapshot Register 1	R	
0x00A8	Reserved	—	—
0x00AC	EQADC_CFSR—EQADC CFIFO Status Register	R	19.6.2.10/19-32
0x00B0–0x00CC	Reserved	—	—
0x00D0	EQADC_REDLCCR—EQADC Red Line Client Configuration Register		19.6.2.11/19-33
0x00D4–0x00FC	Reserved	—	—
0x0100–0x010C	EQADC_CF0Rw—EQADC CFIFO0 Registers (w=0, .., 3)	R	19.6.2.12/19-35
0x0110–0x011C	EQADC_CF0ERw—EQADC CFIFO0 Extension Registers (w=0, .., 3)	R	19.6.2.13/19-37
0x0120 –0x013C	Reserved	—	—
0x0140–0x014C	EQADC_CF1Rw—EQADC CFIFO1 Registers (w=0, .., 3)	R	19.6.2.13/19-37
0x0150–0x017C	Reserved	—	—
0x0180–0x018C	EQADC_CF2Rw—EQADC CFIFO2 Registers (w=0, .., 3)	R	19.6.2.13/19-37
0x0190–0x01BC	Reserved	—	—
0x01C0–0x01CC	EQADC_CF3Rw—EQADC CFIFO3 Registers (w=0, .., 3)	R	19.6.2.13/19-37
0x01D0–0x01FC	Reserved	—	—
0x0200–0x020C	EQADC_CF4Rw—EQADC CFIFO4 Registers (w=0, .., 3)	R	19.6.2.13/19-37
0x0210–0x023C	Reserved	—	—
0x0240–0x024C	EQADC_CF5Rw—EQADC CFIFO5 Registers (w=0, .., 3)	R	19.6.2.13/19-37
0x0250–0x02FC	Reserved	—	—
0x0300–0x030C	EQADC_RF0Rw—EQADC RFIFO0 Registers (w=0, .., 3)	R	19.6.2.14/19-37
0x0310–0x033C	Reserved	—	—
0x0340–0x034C	EQADC_RF1Rw—EQADC RFIFO1 Registers (w=0, .., 3)	R	19.6.2.14/19-37
0x0350–0x037C	Reserved	—	—
0x0380–0x038C	EQADC_RF2Rw—EQADC RFIFO2 Registers (w=0, .., 3)	R	19.6.2.14/19-37
0x0390–0x03BC	Reserved	—	—
0x03C0–0x03CC	EQADC_RF3Rw—EQADC RFIFO3 Registers (w=0, .., 3)	R	19.6.2.14/19-37
0x03D0–0x03FC	Reserved	—	—
0x0400–0x040C	EQADC_RF4Rw—EQADC RFIFO4 Registers (w=0, .., 3)	R	19.6.2.14/19-37
0x0410–0x043C	Reserved	—	—
0x0440–0x044C	EQADC_RF5Rw—EQADC RFIFO5 Registers (w=0, .., 3)	R	19.6.2.14/19-37
0x0450–0x07FC	Reserved	—	—

[1] EQADC_A_BASE = 0xFFF8_0000
EQADC_B_BASE = 0xFFF8_4000

19.6.2 EQADC Register Descriptions

19.6.2.1 EQADC Module Configuration Register (EQADC_MCR)

The EQADC Module Configuration Register (EQADC_MCR) contains bits used to control how the EQADC responds to a debug mode entry request.

Address: 0x0000

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	ICEA0	ICEA1	DSM	0		0	DBG	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-3. EQADC Module Configuration Register (EQADC_MCR)

Table 19-6. EQADC_MCR Field Descriptions

Field	Description
ICEA*n*	Immediate Conversion Command Enable ADC*n* (*n*=0,1). ICEA*n* enables the EQADC to abort on-chip ADC*n* current conversion and to start the immediate conversion command from CFIFO0 in the requested ADC*n*. 0 Disable immediate conversion command request. 1 Enable immediate conversion command request.
DSM	Destination Select Mode. DSM selects between two modes of sending the same conversion result to two different destinations. This is applied only when using Alternate Conversion Commands. 0 Legacy mode. Uses a defined set of pairs of companion module destinations as defined by the DEST field encodings of the ADC_ACR1–14 registers specified in Table 19-31. The Extended Alternate Configuration registers (ADC_EACR1-14) are not considered. 1 Selects the Extended Alternate Configuration Registers to send result to a second destination in addition to the destination selected by the Alternate Configuration Registers. These destinations can be RFIFOs and companion modules (Decimation filters) connected to the PSI. See Section 19.6.3.9, "Extended Alternate Configuration 1-14 Control Registers (ADC_EACR1-14)" for details.
Bits 27–28	These bits are reserved and must always be 0b00.
DBG	Debug enable. The DBG field defines the EQADC response to a debug mode entry request. 00 Do not enter debug mode. 01 Reserved 1x Enter debug mode.

19.6.2.2 EQADC External Trigger Digital Filter Register (EQADC_ETDFR)

The EQADC External Trigger Digital Filter Register (EQADC_ETDFR) is used to set the minimum time a signal must be held in a logic state on the CFIFO triggers inputs to be recognized as an edge or level gated trigger. The Digital Filter Length field specifies the minimum number of peripheral clocks that must be counted by the digital filter counter to recognize a logic state change. This filter does not work when the trigger is an internal signal (see SIU chapter), but should be considered when the trigger is an external one.

Address: 0x000C Access: User read/write

R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	DFL			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-4. EQADC External Trigger Digital Filter Register (EQADC_ETDFR)

Table 19-7. EQADC_ETDFR Field Description

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>DFL
[0:3]</td><td>

Digital Filter Length. The DFL field specifies the minimum number of peripheral clocks that must be counted by the digital filter counter to recognize a logic state change. The count specifies the sample period of the digital filter which is calculated according to the following equation:

$$\mathrm{FilterPeriod} = (\mathrm{SystemClockPeriod} \times 2^{\mathrm{DFL}}) + 1(\mathrm{SystemClockPeriod})$$

Minimum clock counts for which an ETRIG signal needs to be stable to be passed through the filter are shown below. Refer to Section 19.7.4.5, “Hardware Trigger Event Detection,” for more information on the digital filter.

<table>
<tr><th>DFL[0:3]</th><th>Minimum Clock Count</th><th>Minimum Time (ns) (peripheral clock = 120 MHz)</th></tr>
<tr><td>0b0000</td><td>2</td><td>16.66</td></tr>
<tr><td>0b0001</td><td>3</td><td>25.00</td></tr>
<tr><td>0b0010</td><td>5</td><td>41.66</td></tr>
<tr><td>0b0011</td><td>9</td><td>75.00</td></tr>
<tr><td>0b0100</td><td>17</td><td>141.66</td></tr>
<tr><td>0b0101</td><td>33</td><td>275.00</td></tr>
<tr><td>0b0110</td><td>65</td><td>541.66</td></tr>
<tr><td>0b0111</td><td>129</td><td>1075.00</td></tr>
<tr><td>0b1000</td><td>257</td><td>2141.66</td></tr>
<tr><td>0b1001</td><td>513</td><td>4275.00</td></tr>
<tr><td>0b1010</td><td>1025</td><td>8541.66</td></tr>
<tr><td>0b1011</td><td>2049</td><td>17075.00</td></tr>
<tr><td>0b1100</td><td>4097</td><td>34141.00</td></tr>
<tr><td>0b1101</td><td>8193</td><td>68275.00</td></tr>
<tr><td>0b1110</td><td>16385</td><td>136541.66</td></tr>
<tr><td>0b1111</td><td>32769</td><td>273075.00</td></tr>
</table>

Note: The DFL field must only be written when the MODEx of all CFIFOs are configured to disabled.When the digital filter is bypassed (i.e. for on-chip triggers), the DFL is not considered and the trigger input signal is not filtered.

</td></tr>
</table>

19.6.2.3 EQADC CFIFO Push Registers (EQADC_CFPR)

The EQADC CFIFO Push Registers (EQADC_CFPR) provide a mechanism to fill the CFIFOs with command messages from the CQueues. Refer to Section 19.7.4, “EQADC Command FIFOs,” for more information on the CFIFOs and to Section 19.7.2.2, “Message Format in EQADC,” for a description on command message formats.

Address: 0x0010, 0x0014, 0x0018, 0x001C, 0x0020, 0x0024 Access: User write-only

R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
W	CF_PUSH*x*																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-5. EQADC CFIFO Push Register *x* (EQADC_CFPR*x*)

Table 19-8. EQADC_CFPR*x* Field Description

Field	Description
CF_PUSH*x* [0:31]	CFIFO Push Data x. When CFIFO*x* is not full, writing to the whole word or any bytes of EQADC_CFPR*x* will push the 32-bit CF_PUSH*x* value into CFIFO*x*. Writing to this field also increments the corresponding EQADC_FISR[CFCTR*x*] value by one. When the CFIFO*x* is full, the EQADC ignores any write to the CF_PUSH*x*. Reading EQADC_CFPRx always returns zero. **Note:** Only whole words must be written to EQADC_CFPR. Writing half-words or bytes to EQADC_CFPR will still push the whole 32-bit CF_PUSH field into the corresponding CFIFO, but undefined data will fill the areas of CF_PUSH that were not specifically designated as target locations for the write.

19.6.2.4 EQADC Result FIFO Pop Registers (EQADC_RFPR)

The EQADC Result FIFO Pop Registers (EQADC_RFPR) provide a mechanism to retrieve data from RFIFOs.

Address: 0x0030, 0x0034, 0x0038, 0x003C, 0x0040, 0x0044 Access: User read-only

R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RF_POP*x*																
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-6. EQADC RFIFO Pop Register *x* (EQADC_RFPR*x*)

Table 19-9. EQADC_RFPR*x* Field Description

Field	Description
RF_POP*x* [0:15]	Result FIFO Pop Data x. When RFIFOx is not empty, the RF_POPx contains the next unread entry value of RFIFOx. Reading a word, a half-word, or any bytes from EQADC_RFPRx will pop one entry from RFIFOx and cause the EQADC_FISR[RFCTRx] field to be decremented by one. When the RFIFOx is empty, any read on EQADC_RFPRx returns undefined data value and does not decrement the RFCTRx value. Writing to EQADC_RFPRx has no effect.

19.6.2.5 EQADC CFIFO Control Registers (EQADC_CFCR)

The EQADC CFIFO Control Registers (EQADC_CFCR) contain bits that affect CFIFOs. These bits specify the CFIFO operation mode and can invalidate all of the CFIFO contents.

Address: 0x0050 (CFIFO0)
0x0052 (CFIFO1)
0x0054 (CFIFO2)
0x0056 (CFIFO3)
0x0058 (CFIFO4)
0x005A (CFIFO5)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	CFE EE0*	STR ME0*	0	0	0	MODEx				AMODE0*			
W						SSEx	CFINVx									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

* only available for CFIFO0

Figure 19-7. EQADC CFIFO Control Register x (EQADC_CFCRx)

Table 19-10. EQADC_CFCR*x* Field Description

Field	Description
CFEEE0	CFIFO0 Entry Number Extension Enable. The CFEEE0 bit is used to enable the extension of the CFIFO0 entries. When in extended mode, the CFIFO0 total entries is the sum of normal entries plus the defined value for extension. For more details, refer to Section 19.7.4.2, “CFIFO0 Streaming Mode Description.” 1 Enable the extension of CFIFO0 entries. 0 CFIFO0 has a normal value of entries.
STRME0	CFIFO0 Streaming Mode Operation Enable. The STRME0 bit is used to enable the streaming mode of operation of CFIFO0. In this case, it is possible to repeat some sequence of commands of this FIFO without having to load new commands from a command queue. For more details, refer to Section 19.7.4.2, “CFIFO0 Streaming Mode Description.” 1 Enable the streaming mode of CFIFO0. 0 Streaming mode of CFIFO0 is disabled.
SSEx	CFIFO Single-Scan Enable Bit x. The SSEx bit is used to set the EQADC_FISR[SSSx] bit. Writing a “1” to SSEx will set the EQADC_FISR[SSSx] field if the CFIFO is in single-scan mode. When EQADC_FISR[SSSx] is already asserted, writing a “1” to SSEx has no effect. If the CFIFO is in continuous-scan mode or is disabled, writing a “1” to SSEx will not set EQADC_FISR[SSSx]. Writing a “0” to SSEx has no effect. SSEx always is read as “0”. 0 No effect. 1 Set the SSSx bit.
CFINVx	CFIFO Invalidate Bit *x*. The CFINV*x* bit causes the EQADC to invalidate all entries of CFIFO*x*. Writing a “1” to CFINVx will reset the value of EQADC_FISR[CFCTR*x*]. Writing a “1” to CFINV*x* also resets the Push Next Data Pointer, Transfer Next Data Pointer to the first entry of CFIFOx in Figure 19-52. CFINVx always is read as “0”. Writing a “0” has no effect. 0 No effect.1Invalidate all of the entries in the corresponding CFIFO. **Note:** Writing CFINV*x* only invalidates commands stored in CFIFO*x*; previously transferred commands that are waiting for execution, that is commands stored in the CBuffers, will still be executed, and results generated by them will be stored in the appropriate RFIFO. **Note:** CFINV*x* must not be written unless the MODEx is configured to disabled, and CFIFO status is IDLE.

Table 19-10. EQADC_CFCR*x* Field Description (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>MODE<i>x</i>
[0:3]</td><td>CFIFO Operation Mode <i>x</i>. The MODE<i>x</i> field selects the CFIFO operation mode for CFIFO<i>x</i>. Refer to Section 19.7.4.6, “CFIFO Scan Trigger Modes,” for more information on CFIFO trigger mode.
Note: If MODE<i>x</i> is not disabled, it must not be changed to any other mode besides disabled. If MODE<i>x</i> is disabled and the CFIFO status is IDLE, MODE<i>x</i> can be changed to any other mode.
<table>
<tr><th>MODE<i>x</i>[0:3]</th><th>CFIFO Operation Mode</th></tr>
<tr><td>0b0000</td><td>Disabled</td></tr>
<tr><td>0b0001</td><td>Software Trigger, Single Scan</td></tr>
<tr><td>0b0010</td><td>Low Level Gated External Trigger, Single Scan</td></tr>
<tr><td>0b0011</td><td>High Level Gated External Trigger, Single Scan</td></tr>
<tr><td>0b0100</td><td>Falling Edge External Trigger, Single Scan</td></tr>
<tr><td>0b0101</td><td>Rising Edge External Trigger, Single Scan</td></tr>
<tr><td>0b0110</td><td>Falling or Rising Edge External Trigger, Single Scan</td></tr>
<tr><td>0b0111– 0b1000</td><td>Reserved</td></tr>
<tr><td>0b1001</td><td>Software Trigger, Continuous Scan</td></tr>
<tr><td>0b1010</td><td>Low Level Gated External Trigger, Continuous Scan</td></tr>
<tr><td>0b1011</td><td>High Level Gated External Trigger, Continuous Scan</td></tr>
<tr><td>0b1100</td><td>Falling Edge External Trigger, Continuous Scan</td></tr>
<tr><td>0b1101</td><td>Rising Edge External Trigger, Continuous Scan</td></tr>
<tr><td>0b1110</td><td>Falling or Rising Edge External Trigger, Continuous Scan</td></tr>
<tr><td>0b1111</td><td>Reserved</td></tr>
</table>
</td></tr>
</table>

Table 19-10. EQADC_CFCR*x* Field Description (continued)

Field	Description
AMODE0 [0:3]	CFIFO0 Advance Trigger Operation Mode 0. The AMODE0 field selects the trigger mode for the ATRIG0 trigger signal in streaming mode. The use of reserved values drives to unknown behavior of the block. **Note:** If AMODE0 is not disabled, it must not be changed to any other mode besides disabled If AMODE0 is disabled and the CFIFO0 status is IDLE, AMODE0 can be changed to any other mode. **Note:** For the streaming mode of operation when the ATRIG0 is used to enable the ETRIG0 or to advance the command queue, the normal mode of operation is external trigger single scan. Other settings are not fully tested.

AMODE*0*[0:3]	CFIFO0 Advance Trigger Operation Mode
0b0000	Disabled
0b0001	Software Trigger, Single Scan
0b0010	Low Level Gated External Trigger, Single Scan
0b0011	High Level Gated External Trigger, Single Scan
0b0100	Falling Edge External Trigger, Single Scan
0b0101	Rising Edge External Trigger, Single Scan
0b0110	Falling or Rising Edge External Trigger, Single Scan
0b0111–0b1111	Reserved

19.6.2.6 EQADC Interrupt and DMA Control Registers (EQADC_IDCR)

The EQADC Interrupt Control Registers (EQADC_IDCR) contain bits to enable the generation of interrupt or DMA requests when the corresponding flag bits are set in Section 19.6.2.7, “EQADC FIFO and Interrupt Status Registers (EQADC_FISR).”

Address: 0x0060 Access: User read/write

R	NCIE0	TORIE0	PIE0	EOQIE0	CFUIE0	0	CFFE0	CFFS0	0	0	0	0	RFOIE0	0	RFDE0	RFDS0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	NCIE1	TORIE1	PIE1	EOQIE1	CFUIE1	0	CFFE1	CFFS1	0	0	0	0	RFOIE1	0	RFDE1	RFDS1
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-8. EQADC Interrupt and DMA Control Register 0 (EQADC_IDCR0)

Address: 0x0064 Access: User read/write

R	NCIE2	TORIE2	PIE2	EOQIE2	CFUIE2	0	CFFE2	CFFS2	0	0	0	0	RFOIE2	0	RFDE2	RFDS2
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	NCIE3	TORIE3	PIE3	EOQIE3	CFUIE3	0	CFFE3	CFFS3	0	0	0	0	RFOIE3	0	RFDE3	RFDS3
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-9. EQADC Interrupt and DMA Control Register 1 (EQADC_IDCR1)

Address: 0x0068 Access: User read/write

R	NCIE4	TORIE4	PIE4	EOQIE4	CFUIE4	0	CFFE4	CFFS4	0	0	0	0	RFOIE4	0	RFDE4	RFDS4
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	NCIE5	TORIE5	PIE5	EOQIE5	CFUIE5	0	CFFE5	CFFS5	0	0	0	0	RFOIE5	0	RFDE5	RFDS5
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-10. EQADC Interrupt and DMA Control Register 2 (EQADC_IDCR2)

Table 19-11. EQADC_RFPR*x* Field Description

Field	Description
NCIEx	Non-Coherency Interrupt Enable x. NCIEx enables the EQADC to generate an interrupt request when the corresponding EQADC_FISR[NCFx] is asserted. 0 Disable non-coherency interrupt request. 1 Enable non-coherency interrupt request.
TORIEx	Trigger Overrun Interrupt Enable x. TORIEx enables the EQADC to generate an interrupt request when the corresponding EQADC_FISR[TORFx] is asserted. Apart from generating an independent interrupt request for a CFIFOx Trigger Overrun event, the EQADC also provides a combined interrupt at which the Result FIFO Overflow Interrupt, the Command FIFO Underflow Interrupt, and the Command FIFO Trigger Overrun Interrupt requests of ALL CFIFOs are ORed. When RFOIEx, CFUIEx, and TORIEx are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFx, CFUFx, and TORFx (assuming that all interrupts are enabled). See Section 19.7.8, "EQADC DMA/Interrupt Request," for details. 0 Disable trigger overrun interrupt request. 1 Enable trigger overrun interrupt request.
PIEx	Pause Interrupt Enable x. PIEx enables the EQADC to generate an interrupt request when the corresponding EQADC_FISR[PFx] is asserted. 1 Enable pause interrupt request. 0 Disable pause interrupt request.
EOQIEx	End of Queue Interrupt Enable x. EOQIEx enables the EQADC to generate an interrupt request when the corresponding EQADC_FISR[EOQFx] is asserted. 0 Disable End of Queue interrupt request. 1 Enable End of Queue interrupt request.
CFUIEx	CFIFO Underflow Interrupt Enable x. CFUIEx enables the EQADC to generate an interrupt request when the corresponding EQADC_FISR[CFUFx] is asserted. Apart from generating an independent interrupt request for a CFIFOx underflow event, the EQADC also provides a combined interrupt at which the Result FIFO Overflow Interrupt, the Command FIFO Underflow Interrupt, and the Command FIFO Trigger Overrun Interrupt requests of all CFIFOs are ORed. When RFOIEx, CFUIEx, and TORIEx are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFx, CFUFx, and TORFx (assuming that all interrupts are enabled). See Section 19.7.8, "EQADC DMA/Interrupt Request," for details. 0 Disable Underflow Interrupt request. 1 Enable Underflow Interrupt request.
CFFEx	CFIFO Fill Enable x. CFFEx enables the EQADC to generate an interrupt request (CFFSx is asserted) or DMA request (CFFSx is negated) when EQADC_FISR[CFFFx] is asserted. 0 Disable CFIFO Fill DMA or Interrupt request. 1 Enable CFIFO Fill DMA or Interrupt request. **Note:** CFFEx must not be negated while a DMA transaction is in progress.

Table 19-11. EQADC_RFPR*x* Field Description (continued)

Field	Description
CFFSx	CFIFO Fill Select x. CFFSx selects if a DMA or interrupt request is generated when EQADC_FISR[CFFFx] is asserted. If CFFEx is asserted, the EQADC generates an interrupt request when CFFSx is negated, or it generates a DMA request if CFFSx is asserted. 0 Generate interrupt request to move data from the system memory to CFIFOx. 1 Generate DMA request to move data from the system memory to CFIFOx. **Note:** CFFSx must not be negated while a DMA transaction is in progress.
RFOIEx	RFIFO Overflow Interrupt Enable x. RFOIEx enables the EQADC to generate an interrupt request when the corresponding EQADC_FISR[RFOFx] is asserted. Apart from generating an independent interrupt request for an RFIFOx overflow event, the EQADC also provides a combined interrupt at which the Result FIFO Overflow Interrupt, the Command FIFO Underflow Interrupt, and the Command FIFO Trigger Overrun Interrupt requests of ALL CFIFOs are ORed. When RFOIEx, CFUIEx, and TORIEx are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFx, CFUFx, and TORFx (assuming that all interrupts are enabled). See Section 19.7.8, “EQADC DMA/Interrupt Request,” for details. 0 Disable Overflow Interrupt request. 1 Enable Overflow Interrupt request.
RFDEx	RFIFO Drain Enable x. RFDEx enables the EQADC to generate an interrupt request (RFDSx is asserted) or DMA request (RFDSx is negated) when EQADC_FISR[RFDFx] is asserted. 0 Disable RFIFO Drain DMA or Interrupt request. 1 Enable RFIFO Drain DMA or Interrupt request. **Note:** RFDEx must not be negated while a DMA transaction is in progress.
RFDSx	RFIFO Drain Select x. RFDSx selects if a DMA or interrupt request is generated when EQADC_FISR[RFDFx] is asserted. If RFDEx is asserted, the EQADC generates an interrupt request when RFDSx is negated, or it generates a DMA request when RFDSx is asserted. 0 Generate interrupt request to move data from RFIFOx to the system memory. 1 Generate DMA request to move data from RFIFOx to the system memory. **Note:** RFDSx must not be negated while a DMA transaction is in progress.

19.6.2.7 EQADC FIFO and Interrupt Status Registers (EQADC_FISR)

The EQADC FIFO and Interrupt Status Registers (EQADC_FISR) contain flag and status bits for each CFIFO and RFIFO pair. Write “1” to a flag bit to clear it. Writing “0” has no effect. Status bits are read only. These bits indicate the status of the FIFO itself.

Address: 0x0070, 0x0074, 0x0078, 0x007C, 0x0080, 0x0084 Access: User read/write

R	NCFx	TORFx	PFx	EOQFx	CFUFx	SSSx	CFFFx	0	0	0	0	0	RFOFx	0	RFDFx	0
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
R	CFCTRx				TNXTPTRx				RFCTRx				POPNXTPTRx			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-11. EQADC FIFO and Interrupt Status Register x (EQADC_FISRx)

Table 19-12. EQADC_FISRx Field Description

Field	Description
NCFx	Non-Coherency Flag. NCFx is set whenever a command sequence being transferred through CFIFOx becomes non coherent. If EQADC_IDCR[NCIEx] and NCF*x* are asserted, an interrupt request will be generated. Write “1” to clear NCFx. Writing a “0” has no effect. For more information refer to Section 19.7.4.7.5, “Command Sequence Non-Coherency Detection.” 0 Command sequence being transferred by CFIFOx is coherent. 1 Command sequence being transferred by CFIFOx became non-coherent.
TORFx	Trigger Overrun Flag for CFIFOx. TORFx is set when trigger overrun occurs for the specified CFIFO in edge or level trigger mode. Trigger overrun occurs when an already triggered CFIFO receives an additional trigger. When EQADC_IDCR[TORIE*x*] and TORF*x* are asserted, an interrupt request will be generated. Apart from generating an independent interrupt request for a CFIFOx Trigger Overrun event, the EQADC also provides a combined interrupt at which the Result FIFO Overflow Interrupt, the Command FIFO Underflow Interrupt, and the Command FIFO Trigger Overrun Interrupt requests of ALL CFIFOs are ORed. When RFOIEx, CFUIEx, and TORIEx are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFx, CFUFx, and TORFx (assuming that all interrupts are enabled). See Section 19.7.8, “EQADC DMA/Interrupt Request,” for details. Write “1” to clear the TORF*x* bit. Writing a “0” has no effect. 0 No trigger overrun occurred. 1 Trigger overrun occurred. **Note:** The trigger overrun flag will not set for CFIFOs configured for software trigger mode.

Table 19-12. EQADC_FISRx Field Description (continued)

Field	Description
PFx	Pause Flag x. PF behavior changes according to the CFIFO trigger mode. In edge trigger mode, PFx is set when the EQADC completes the transfer of an entry with an asserted Pause bit from CFIFOx. In level trigger mode, when CFIFOx is in TRIGGERED status, PFx is set when CFIFO status changes from TRIGGERED due to the detection of a closed gate. An interrupt routine, generated due to the asserted PF, can be used to verify if a complete scan of the CQueue was performed. If a closed gate is detected while no command transfers are taking place, it will have immediate effect on the CFIFO status. If a closed gate is detected while a command transfer to an on-chip CBuffer is taking place, it will only affect the CFIFO status when the transfer completes. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the appropriate CBuffer. If EQADC_IDCR[PIE*x*] and PFx are asserted, an interrupt will be generated. Write “1” to clear the PF*x*. Writing a “0” has no effect. Refer to Section 19.7.4.7.3, “Pause Status,” for more information on the Pause Flag. 0 Entry with asserted PAUSE bit was not transferred from CFIFOx (CFIFO in edge trigger mode), or CFIFO status did not change from TRIGGERED due to detection of a closed gate (CFIFO in level trigger mode). 1 Entry with asserted PAUSE bit was transferred from CFIFOx (CFIFO in edge trigger mode), or CFIFO status changes from TRIGGERED due to detection of a closed gate (CFIFO in level trigger mode). **Note:** In edge trigger mode, an asserted PFx only implies that the EQADC has finished transferring a command with an asserted PAUSE bit from CFIFOx. It does not imply that result data for the current command and for all previously transferred commands has been returned to the appropriate RFIFO. **Note:** In software or level trigger mode, when the EQADC completes the transfer of an entry from CFIFOx with an asserted PAUSE bit, PFx will not be set and transfer of commands will continue without pausing.
EOQFx	End of Queue Flag x. EOQFx indicates that an entry with an asserted EOQ bit was transferred from CFIFOx to the on-chip ADCs - see Section 19.7.2.2, “Message Format in EQADC,” for details about command message formats. When the EQADC completes the transfer of an entry with an asserted EOQ bit from CFIFOx, EOQFx will be set. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the appropriate CBuffer. If the EQADC_IDCR[EOQIEx] and EOQFx are asserted, an interrupt will be generated. Write “1” to clear the EOQFx bit. Writing a “0” has no effect. Refer to Section 19.7.4.7.2, “CQueue Completion Status,” for more information on the End of Queue Flag. 1 Entry with asserted EOQ bit was transferred from CFIFOx. 0 Entry with asserted EOQ bit was not transferred from CFIFOx. **Note:** An asserted EOQFx only implies that the EQADC has finished transferring a command with an asserted EOQ bit from CFIFOx. It does not imply that result data for the current command and for all previously transferred commands has been returned to the appropriate RFIFO.

Table 19-12. EQADC_FISRx Field Description (continued)

Field	Description
CFUFx	CFIFO Underflow Flag x. CFUFx indicates an underflow event on CFIFOx. CFUFx is set when CFIFOx is in TRIGGERED state and it becomes empty. No commands will be transferred from an underflowing CFIFO, nor will command transfers from lower priority CFIFOs be blocked. When EQADC_IDCR[CFUIEx] and CFUFx are both asserted, the EQADC generates an interrupt request. Apart from generating an independent interrupt request for a CFIFOx underflow event, the EQADC also provides a combined interrupt at which the Result FIFO Overflow Interrupt, the Command FIFO Underflow Interrupt, and the Command FIFO Trigger Overrun Interrupt requests of ALL CFIFOs are ORed. When RFOIEx, CFUIEx, and TORIEx are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFx, CFUFx, and TORFx (assuming that all interrupts are enabled). See Section 19.7.8, “EQADC DMA/Interrupt Request,” for details. Write “1” to clear CFUFx. Writing a “0” has no effect. 0 No CFIFO underflow event occurred. 1 A CFIFO underflow event occurred.
SSSx	CFIFO Single-Scan Status Bit x. An asserted SSSx bit enables the detection of trigger events for CFIFOs programmed into single-scan level- or edge-trigger mode, and works as trigger for CFIFOs programmed into single-scan software-trigger mode. Refer to Section 19.7.4.6.2, “Single-Scan Mode,” for further details. The SSSx bit is set by writing a “1” to EQADC_CFCR[SSEx]. The EQADC clears the SSSx bit when a command with an asserted EOQ bit is transferred from a CFIFO in single-scan mode, when a CFIFO is in single-scan level-trigger mode and its status changes from TRIGGERED due to the detection of a closed gate, or when the value of the CFIFO operation mode (EQADC_CFCR[MODEx]) is changed to disabled. Writing to SSSx has no effect. SSSx has no effect in continuous-scan or in disabled mode. 0 CFIFO in single-scan level- or edge-trigger mode will ignore trigger events, or CFIFO in single-scan software-trigger mode is not triggered. 1 CFIFO in single-scan level- or edge-trigger mode will detect a trigger event, or CFIFO in single-scan software-trigger mode is triggered.
CFFFx	CFIFO Fill Flag x. CFFFx is set when the CFIFOx is not full. When EQADC_IDCR[CFFEx] and CFFFx are both asserted, an interrupt or a DMA request will be generated depending on the status of the CFFSx bit. When CFFSx is negated (interrupt requests selected), software clears CFFFx by writing a “1” to it. Writing a “0” has no effect. When CFFSx is asserted (DMA requests selected), CFFFx is automatically cleared by the EQADC when the CFIFO becomes full. 0 CFIFOx is full. 1 CFIFOx is not full. **Note:** Writing “1” to CFFFx when CFFSx is asserted (DMA requests selected) is not allowed. **Note:** When generation of interrupt requests is selected (CFFSx=0), CFFFx must only be cleared in the ISR after the CFIFOx push register is accessed.

Table 19-12. EQADC_FISRx Field Description (continued)

Field	Description
RFOFx	RFIFO Overflow Flag x. RFOFx indicates an overflow event on RFIFOx. RFOF*x* is set when RFIFOx is already full, and a new data is received from the on-chip ADCs. The RFIFOx will not overwrite older data in the RFIFO, and the new data will be ignored. When EQADC_IDCR[RFOIEx] and RFOFx are both asserted, the EQADC generates an interrupt request. Apart from generating an independent interrupt request for an RFIFOx overflow event, the EQADC also provides a combined interrupt at which the Result FIFO Overflow Interrupt, the Command FIFO Underflow Interrupt, and the Command FIFO Trigger Overrun Interrupt requests of ALL CFIFOs are ORed. When RFOIEx, CFUIEx, and TORIEx are all asserted, this combined interrupt request is asserted whenever one of the following 18 flags becomes asserted: RFOFx, CFUFx, and TORFx (assuming that all interrupts are enabled). See Section 19.7.8, “EQADC DMA/Interrupt Request,” for details. Write “1” to clear RFOFx. Writing a “0” has no effect. 0 No RFIFO overflow event occurred. 1 An RFIFO overflow event occurred.
RFDF*x*	RFIFO Drain Flag x. RFDFx indicates if RFIFOx has valid entries or not. RFDFx is set when the RFIFOx has at least one valid entry in it. When EQADC_IDCR[RFDEx] and RFDFx are both asserted, an interrupt or a DMA request will be generated depending on the status of the RFDSx bit. When RFDSx is negated (interrupt requests selected), software clears RFDFx by writing a “1” to it. Writing a “0” has no effect. When RFDSx is asserted (DMA requests selected), RFDFx is automatically cleared by the EQADC when the RFIFO becomes empty. 0 RFIFOx is empty. 1 RFIFOx has at least one valid entry. **Note:** Writing “1” to RFDFx when RFDSx is asserted (DMA requests selected) is not allowed. **Note:** When the generation of interrupt requests is selected (RFDSx=0), RFDFx must only be cleared in the ISR after the RFIFOx pop register is accessed.
CFCTRx [0:3]	CFIFOx Entry Counter. CFCTRx indicates the number of commands stored in the CFIFOx. When the EQADC completes transferring a piece of new data from the CFIFOx, it decrements CFCTRx by one. Writing a word or any bytes to the corresponding EQADC_CFPR increments CFCTRx by one. Writing any value to CFCTRx has no effect.
TNXTPTRx [0:3]	CFIFOx Transfer Next Pointer. TNXTPTRx indicates the index of the next entry to be removed from CFIFOx when it completes a transfer. When TNXTPTRx is zero, it points to the entry with the smallest memory-mapped address inside CFIFOx. TNXTPTRx is only updated when a command transfer is completed. If the maximum index number (CFIFO depth minus one) is reached, TNXTPTRx is wrapped to zero, else, it is incremented by one. For details refer to Section 19.7.4.1, “CFIFO Basic Functionality.” Writing any value to TNXTPTRx has no effect.
RFCTRx [0:3]	RFIFOx Entry Counter. RFCTRx indicates the number of data items stored in the RFIFOx. When the EQADC stores a piece of new data into RFIFOx, it increments RFCTRx by one. Reading the whole word, half-word or any bytes of the corresponding EQADC_RFPR decrements RFCTRx by one. Writing any value to RFCTRx itself has no effect.
POPNXT PTRx [0:3]	RFIFOx Pop Next Pointer. POPNXTPTRx indicates the index of the entry that will be returned when EQADC_RFPRx is read. When POPNXTPTRx is zero, it points to the entry with the smallest memory-mapped address inside RFIFOx. POPNXTPTRx is updated when EQADC_RFPRx is read. If the maximum index number (RFIFO depth minus one) is reached, POPNXTPTRx is wrapped to zero, else, it is incremented by one. For details refer to Section 19.7.5.1, “RFIFO Basic Functionality.” Writing any value to POPNXTPTRx has no effect.

19.6.2.8 EQADC CFIFO Transfer Counter Registers (EQADC_CFTCR)

The EQADC CFIFO Transfer Counter Registers (EQADC_CFTCR) record the number of commands transferred from a CFIFO. The EQADC_CFTCR supports the monitoring of command transfers from a CFIFO.

Address: 0x0090 Access: User read/write

R	0	0	0	0	0	TC_CF0										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	TC_CF1										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-12. EQADC CFIFO Transfer Counter Register 0 (EQADC_CFTCR0)

Address: 0x0094 Access: User read/write

R	0	0	0	0	0	TC_CF2										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	TC_CF3										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-13. EQADC CFIFO Transfer Counter Register 1 (EQADC_CFTCR1)

Address: 0x0098 Access: User read/write

R	0	0	0	0	0	TC_CF4										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	TC_CF5										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-14. EQADC CFIFO Transfer Counter Register 2 (EQADC_CFTCR2)

Table 19-13. EQADC_CFTCRx Field Descriptions

Field	Description
TC_CFx [0:10]	Transfer Counter for CFIFOx. TC_CFx counts the number of commands which have been completely transferred from CFIFOx. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the appropriate CBuffer. The EQADC increments the TC_CFx value by one after a command is transferred. TC_CFx resets to zero after EQADC completes transferring a command with an asserted EOQ bit. Writing any value to TC_CFx sets the counter to that written value. **Note:** If CFIFOx is in TRIGGERED state when its MODEx field is programmed to disabled, the exact number of entries transferred from the CFIFO until that point - TC_CFx - is only known after the CFIFO status changes to IDLE, as indicated by CFSx. For details refer to Section 19.7.4.6.1, “Disabled Mode.”

19.6.2.9 EQADC CFIFO Status Snapshot Registers (EQADC_CFSSR)

The EQADC CFIFO Status Snapshot Registers (EQADC_CFSSR) contain status fields to track the operation status of each CFIFO and the transfer counter of the last CFIFO to initiate a command transfer to the internal CBuffers. EQADC_CFSSR0-1 are related to the on-chip CBuffers (CBuffer0-1). All fields of a particular EQADC_CFSSR register are captured at the beginning of a command transfer to the CBuffer associated with that register. Note that captured status register values are associated with previous command transfer. This means that the CFSSR registers capture the status registers before the status registers change because of the transfer of the current command that is about to be popped from the CFIFO. The EQADC_CFSSR registers are read only. Writing to the EQADC_CFSSR registers has no effect.

Address: 0x00A0 Access: User read/write

R	CFS0_TCB0		CFS1_TCB0		CFS2_TCB0		CFS3_TCB0		CFS4_TCB0		CFS5_TCB0		0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	0	LCFTCB0				TC_LCFTCB0										
W																
Reset	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

Figure 19-15. EQADC CFIFO Status Snapshot Register 0 (EQADC_CFSSR0)

Address: 0x00A4 Access: User read/write

R	CFS0_TCB1		CFS1_TCB1		CFS2_TCB1		CFS3_TCB1		CFS4_TCB1		CFS5_TCB1		0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	0	LCFTCB1				TC_LCFTCB1										
W																
Reset	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0

Figure 19-16. EQADC CFIFO Status Snapshot Register 1 (EQADC_CFSSR1)

Table 19-14. EQADC_CFSSRx Field Descriptions

Field	Description
CFSx_TCB*n*[0:1]	CFIFO Status at Transfer to CBuffer*n (n=0,1).* CFSx_TCB*n* indicates the CFIFOx status of previously completed command transfer. CFSx_TCB*n* is a copy of the corresponding CFSx in the Section 19.6.2.10, “EQADC CFIFO Status Register (EQADC_CFSR),” captured at the time a current command transfer to CBuffer*n* is initiated.
LCFTCB*n*[0:3]	Last CFIFO to Transfer to CBuffer*n (n=0,1).* LCFTCB*n* holds the CFIFO number to have completed a previous command transfer to CBuffer*n*. 0000 Last command was transferred from CFIFO0 0001 Last command was transferred from CFIFO1 0010 Last command was transferred from CFIFO2 0011 Last command was transferred from CFIFO3 0100 Last command was transferred from CFIFO4 0101 Last command was transferred from CFIFO5 0110–1110 Reserved 1111 No command was transferred to CBuffer*n*
TC_LCFTCB*n*[0:10]	Transfer Counter for Last CFIFO to transfer commands to CBuffer*n.* TC_LCFTCB*n* indicates the number of commands which have been completely transferred from CFIFOx when a current command transfer from CFIFOx to CBuffer*n* is initiated. TC_LCFTCB*n* is a copy of the corresponding TC_CFx in the Section 19.6.2.8, “EQADC CFIFO Transfer Counter Registers (EQADC_CFTCR),” captured at the time a current command transfer from CFIFOx to CBuffer*n* is initiated. This field has no meaning when LCFTCB*n* is 0b1111.

19.6.2.10 EQADC CFIFO Status Register (EQADC_CFSR)

The EQADC CFIFO Status Register (EQADC_CFSR) contains the current CFIFO status. The EQADC_CFSR registers is read only. Writing to the EQADC_CFSR register has no effect.

Address: 0x00AC Access: User read only

R	CFS0		CFS1		CFS2		CFS3		CFS4		CFS5		0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-17. EQADC CFIFO Status Register (EQADC_CFSR)

Table 19-15. EQADC_CFSR Field Descriptions

Field	Description
CFSx[0:1]	CFIFO Status. CFSx indicates the current status of CFIFOx. 00 Idle: CFIFO is disabled. CFIFO is in single-scan edge or level trigger mode and does not have SSS asserted. EQADC completed the transfer of the last entry of the CQueue in single-scan mode. 01 Reserved 10 Waiting for trigger: CFIFO Mode is modified to continuous-scan edge or level trigger mode. CFIFO Mode is modified to single-scan edge or level trigger mode and SSS is asserted. CFIFO Mode is modified to single-scan software trigger mode and SSS is negated. CFIFO is paused. EQADC transferred the last entry of the queue in continuous-scan edge trigger mode. 11 Triggered: CFIFO is triggered.

19.6.2.11 EQADC Red Line Client Configuration Register (EQADC_REDLCCR)

The EQADC Red Line Client Configuration Register (EQADC_REDLCCR) is used to configure an external timestamp from any of the ETPU modules on the device to be applied to an ADC sample.

The EQADC is connected to the ETPU modules via the STAC bus (Red Line Client), which gives the EQADC access to 24-bit timestamp information from the ETPU modules. Each TCR domain of each ETPU module has an independent STAC bus server data slot, which is specified in the SRVn field of the EQADC_REDLCCR.

The EQADC only supports a 16-bit timestamp, so the REDBSn field in the EQADC_REDLCCR is provided to allow nine different selections of the 16 bits to use in the 24-bit ETPU timestamp.

The EQADC_REDLCCR register provides two sets of SRV/REDBS fields to allow two independent timestamp configurations to be specified at once. The EQADC_REDLCCR statically defines two possible timestamp configurations to choose from. Whether a timestamp is used for each ADC sample is specified in the TSR bit in the conversion command word, Section 19.7.2.2.1, "Conversion Command Format for the Standard Configuration". Which of the two EQADC_REDLCCR configurations used is selected in the ADC0/1 control registers, Section 19.6.3.1, "ADC0/1 Control Registers (ADC0_CR and ADC1_CR)".

Address: EQADC_BASE+0x0D0 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	REDBS2				SRV2				REDBS1				SRV1			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-18. EQADC Red Line Client Configuration Register (EQADC_REDLCCR)

Table 19-16. EQADC_REDLCCR Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>REDBS<i>m</i>[0:3]</td><td>Red Line Timebase Bits Selection <i>m</i> (<i>m</i>=1,2)—Selects 16 bits from the total of 24 bits that are received from the Red Line interface as described in below. Consider TBASEm[0:23] the selected time base from slot SRVm:
<table>
<tr><th>REDBS<i>m</i>[0:3]</th><th>Selected Bits</th></tr>
<tr><td>0b0000</td><td>TBASE<i>m</i>[0:15]</td></tr>
<tr><td>0b0001</td><td>TBASE<i>m</i>[1:16]</td></tr>
<tr><td>0b0010</td><td>TBASE<i>m</i>[2:17]</td></tr>
<tr><td>0b0011</td><td>TBASE<i>m</i>[3:18]</td></tr>
<tr><td>0b0100</td><td>TBASE<i>m</i>[4:19]</td></tr>
<tr><td>0b0101</td><td>TBASE<i>m</i>[5:20]</td></tr>
<tr><td>0b0110</td><td>TBASE<i>m</i>[6:21]</td></tr>
<tr><td>0b0111</td><td>TBASE<i>m</i>[7:22]</td></tr>
<tr><td>0b1000</td><td>TBASE<i>m</i>[8:23]</td></tr>
<tr><td>Others</td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td>SRV<i>m</i>[0:3]</td><td>Red Line Server Data Slot Selector <i>m</i> (<i>m</i>=1,2)—Indicates the slot number that contains the desired time base value sent by the Red Line server. This value selects which eTPU timebase on the STAC bus is used to timestamp an ADC sample.
<table>
<tr><th>SRV<i>m</i>[0:3]</th><th>eTPU Timebase</th></tr>
<tr><td>0b0000</td><td>eTPUA TCR1</td></tr>
<tr><td>0b0001</td><td>eTPUB TCR1</td></tr>
<tr><td>0b0010</td><td>eTPUA TCR2</td></tr>
<tr><td>0b0011</td><td>eTPUB TCR2</td></tr>
<tr><td>0b0100</td><td>eTPUC TCR1</td></tr>
<tr><td>0b0101</td><td>eTPUC TCR2</td></tr>
<tr><td>0b0110 - 0b1111</td><td>Reserved</td></tr>
</table>
</td></tr>
</table>

19.6.2.12 EQADC CFIFO Registers (EQADC_CFxRw) (x=0, ..,5; w=0, .., 3)

The EQADC CFIFO Registers (EQADC_CFxRw) (x=0, .., 5; w=0, .., 3) provide visibility of the contents of a CFIFO for debugging purposes. Each CFIFO has four registers which are uniquely mapped to its four 32-bit entries. Refer to Section 19.7.4, “EQADC Command FIFOs,” for more information on CFIFOs. These registers are read only. Data written to these registers is ignored.

Address: 0x0100, 0x0104, 0x0108, 01010C Access: User read only

R	CFIFO0_DATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	CFIFO0_DATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-19. EQADC CFIFO0 Registers (EQADC_CF0Rw) (w=0, .., 3)

Address: 0x0140, 0x0144, 0x0148, 01014C Access: User read only

R	CFIFO0_DATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	CFIFO0_DATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-20. EQADC CFIFO1 Registers (EQADC_CF1Rw) (w=0, .., 3)

Address: 0x0180, 0x0184, 0x0188, 01018C Access: User read only

R	CFIFO2_DATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	CFIFO2_DATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-21. EQADC CFIFO2 Registers (EQADC_CF2Rw) (w=0, .., 3)

Address: 0x01C0, 0x01C4, 0x01C8, 0101CC

Access: User read only

R	CFIFO3_DATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

R	CFIFO3_DATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-22. EQADC CFIFO3 Registers (EQADC_CF3Rw) (w=0, .., 3)

Address: 0x0200, 0x0204, 0x0208, 01020C

Access: User read only

R	CFIFO4_DATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

R	CFIFO4_DATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-23. EQADC CFIFO4 Registers (EQADC_CF4Rw) (w=0, .., 3)

Address: 0x0240, 0x0244, 0x0248, 01024C

Access: User read only

R	CFIFO5_DATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

R	CFIFO5_DATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-24. EQADC CFIFO5 Registers (EQADC_CF5Rw) (w=0, .., 3)

Table 19-17. EQADC_CFxRw Field Description

Field	Description
CFIFOx_DATAw[0:31]	CFIFOx Data w (w = 0, .., 3). Reading CFIFOx_DATAw returns the value stored on the wth entry of CFIFOx. Each CFIFO is composed of four 32-bit entries, with register 0 being mapped to the one with the smallest memory mapped address.

19.6.2.13 EQADC CFIFO0 Extension Registers (EQADC_CF0ERw) (w=0, .., 3)

The EQADC CFIFO0 Extension Registers (EQADC_CF0ERw) (w=0, .., 3) provide visibility of the contents of the extended portion of CFIFO0 for debugging purposes. There are four registers which are uniquely mapped to its four 32-bit entries. Refer to Section 19.7.4, "EQADC Command FIFOs," for more information on CFIFOs. These registers are read only. Data written to these registers is ignored.

Address: 0x0110, 0x0114, 0x0118, 01011C Access: User read only

R	CFIFO0_EDATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	CFIFO0_EDATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-25. EQADC CFIFO0 Extension Registers (EQADC_CF0ERw) (w=0, .., 3)

Table 19-18. EQADC_CF0ERw Field Description

Field	Description
CFIFO0_EDATAw[0:31]	CFIFOx Extension Data w (w = 0, .., 3). Reading CFIFO0_EDATAw returns the value stored on the wth entry of CFIFO0's extended portion.

19.6.2.14 EQADC RFIFO Registers (EQADC_RFxRw) (x=0, .., 5; w=0, .., 3)

The EQADC RFIFO Registers (EQADC_RFxRw) (x=0, .., 5; w=0, .., 3) provide visibility of the contents of a RFIFO for debugging purposes. Each RFIFO has four registers which are uniquely mapped to its four 16-bit entries. Refer to Section 19.7.5, "EQADC Result FIFOs," for more information on RFIFOs. These registers are read only. Data written to these registers is ignored.

Address: 0x0300, 0x0304, 0x0308, 01030C Access: User read only

R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	RFIFO0_DATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-26. EQADC RFIFO0 Registers (EQADC_RF0Rw) (w=0, .., 3)

Address: 0x0340, 0x0344, 0x0348, 01034C Access: User read only

R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

R	RFIFO1_DATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-27. EQADC RFIFO1 Registers (EQADC_RF0Rw) (w=0, .., 3)

Address: 0x0380, 0x0384, 0x0388, 01038C Access: User read only

R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

R	RFIFO2_DATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-28. EQADC RFIFO2 Registers (EQADC_RF0Rw) (w=0, .., 3)

Address: 0x0380, 0x0384, 0x0388, 01038C Access: User read only

R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

R	RFIFO2_DATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-29. EQADC RFIFO2 Registers (EQADC_RF0Rw) (w=0, .., 3)

Address: 0x03C0, 0x03C4, 0x03C8, 0103CC Access: User read only

R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

R	RFIFO3_DATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-30. EQADC RFIFO3 Registers (EQADC_RF0Rw) (w=0, .., 3)

Address: 0x0400, 0x0404, 0x0408, 01040C

Access: User read only

R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	RFIFO4_DATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-31. EQADC RFIFO4 Registers (EQADC_RF0Rw) (w=0, .., 3)

Address: 0x0440, 0x0444, 0x0448, 01044C

Access: User read only

R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	RFIFO5_DATAw															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-32. EQADC RFIFO5 Registers (EQADC_RF0Rw) (w=0, .., 3)

Table 19-19. EQADC_CFxRw Field Descriptions

Field	Description
RFIFOx_DATAw [0:15]	RFIFOx Data w (w = 0, .., 3). Reading RFIFOx_DATAw returns the value stored on the wth entry of RFIFOx. Each RFIFO is composed of four 16-bit entries, with register 0 being mapped to the one with the smallest memory mapped address.

19.6.3 On-Chip ADC Registers

This section describes a list of registers that control on-chip ADC operation. The ADC registers are not part of the CPU accessible memory map. These registers can only be accessed indirectly through configuration commands. There are a set of non-memory mapped registers per ADC, plus a set of registers shared by both ADCs. The address, usage, and access privilege of each register is shown in Table 19-20. Data written to or read from reserved areas of the memory map is undefined.

Their assigned addresses are the values used to set the ADC_REG_ADDRESS field of the read/write configurations commands bound for the on-chip ADCs. These are half-word addresses. Further, the following restrictions apply when accessing these registers:

- Registers ADC0_CR, ADC0_GCCR, ADC0_OCCR, ADC0_AGR1/2 and ADC0_AOR1/2 can only be accessed by configuration commands sent to CBuffer0.
- Registers ADC1_CR, ADC1_GCCR, ADC1_OCCR, ADC1_AGR1/2 and ADC1_AOR1/2 can only be accessed by configuration commands sent to CBuffer1.
- Registers ADC_TSCR, ADC_TBCR, ADC_ACR1-14, ADC_EACR1-14, ADC_PUDCR0-7, and ADC_REDTBR1/2 can be accessed by configuration commands sent to CBuffer0 or to CBuffer1. A data write to any of these registers through a configuration command sent to CBuffer0 will write the same memory location as when writing to it through a configuration command sent to CBuffer1.

NOTE

Simultaneous write accesses from CBuffer0 and CBuffer1 to any of the shared registers are not allowed.

Table 19-20. On-Chip ADC Memory Map

ADC Address	Use	Access
0x00	ADC0/ADC1[1] Conversion Command for Standard Configuration (See page 63)	Write
0x01	ADC0/ADC1 Configuration Control Register (ADC0_CR, ADC1_CR)	Write/Read
0x02	Time Stamp Control Register (ADC_TSCR)	Write/Read
0x03	Time Base Counter Register (ADC_TBCR)	Write/Read
0x04	ADC0/ADC1 Gain Calibration Constant Register (ADC0_GCCR, ADC1_GCCR)	Write/Read
0x05	ADC0/ADC1 Offset Calibration Constant Register (ADC0_OCCR, ADC1_OCCR)	Write/Read
0x06– 0x07	Reserved	—
0x08	ADC0/ADC1 Conversion Command for Alternate Configuration 1 (See page 65)	Write
0x09	ADC0/ADC1 Conversion Command for Alternate Configuration 2 (See page 65)	Write
0x0A	ADC0/ADC1 Conversion Command for Alternate Configuration 3 (See page 65)	Write
0x0B	ADC0/ADC1 Conversion Command for Alternate Configuration 4 (See page 65)	Write
0x0C	ADC0/ADC1 Conversion Command for Alternate Configuration 5 (See page 65)	Write
0x0D	ADC0/ADC1 Conversion Command for Alternate Configuration 6 (See page 65)	Write
0x0E	ADC0/ADC1 Conversion Command for Alternate Configuration 7 (See page 65)	Write

Table 19-20. On-Chip ADC Memory Map (continued)

0x0F	ADC0/ADC1 Conversion Command for Alternate Configuration 8 (See page 65)	Write
0x10	ADC0/ADC1 Conversion Command for Alternate Configuration 9 (See page 65)	Write
0x11	ADC0/ADC1 Conversion Command for Alternate Configuration 10 (See page 65)	Write
0x12	ADC0/ADC1 Conversion Command for Alternate Configuration 11 (See page 65)	Write
0x13	ADC0/ADC1 Conversion Command for Alternate Configuration 12 (See page 65)	Write
0x14	ADC0/ADC1 Conversion Command for Alternate Configuration 13 (See page 65)	Write
0x15	ADC0/ADC1 Conversion Command for Alternate Configuration 14 (See page 65)	Write
0x16–0x2F	Reserved	—
0x30	Alternate Configuration 1 Control Register (ADC_ACR1)	Write/Read
0x31	ADC0/ADC1 Alternate Gain 1 Register (ADC0_AGR1, ADC1_AGR1)	Write/Read
0x32	ADC0/ADC1 Alternate Offset 1 Register (ADC0_AOR1, ADC1_AOR1)	Write/Read
0x33	Extended Alternate Configuration 1 Control Register (ADC_EACR1)	Write/Read
0x34	Alternate Configuration 2 Control Register (ADC_ACR2)	Write/Read
0x35	ADC0/ADC1 Alternate Gain 2 Register (ADC0_AGR2, ADC1_AGR2)	Write/Read
0x36	ADC0/ADC1 Alternate Offset 2 Register (ADC0_AOR2, ADC1_AOR2)	Write/Read
0x37	Extended Alternate Configuration 2 Control Register (ADC_EACR2)	Write/Read
0x38	Alternate Configuration 3 Control Register (ADC_ACR3)	Write/Read
0x39	Reserved	—
0x3A	Reserved	—
0x3B	Extended Alternate Configuration 3 Control Register (ADC_EACR3)	Write/Read
0x3C	Alternate Configuration 4 Control Register (ADC_ACR4)	Write/Read
0x3D	Reserved	—
0x3E	Reserved	—
0x3F	Extended Alternate Configuration 4 Control Register (ADC_EACR4)	Write/Read
0x40	Alternate Configuration 5 Control Register (ADC_ACR5)	Write/Read
0x41	Reserved	—
0x42	Reserved	—
0x43	Extended Alternate Configuration 5 Control Register (ADC_EACR5)	Write/Read
0x44	Alternate Configuration 6 Control Register (ADC_ACR6)	Write/Read
0x45	Reserved	—
0x46	Reserved	—
0x47	Extended Alternate Configuration 6 Control Register (ADC_EACR6)	Write/Read
0x48	Alternate Configuration 7 Control Register (ADC_ACR7)	Write/Read
0x49	Reserved	—
0x4A	Reserved	—
0x4B	Extended Alternate Configuration 7 Control Register (ADC_EACR7)	Write/Read

Table 19-20. On-Chip ADC Memory Map (continued)

0x4C	Alternate Configuration 8 Control Register (ADC_ACR8)	Write/Read
0x4D	Reserved	—
0x4E	Reserved	—
0x4F	Extended Alternate Configuration 8 Control Register (ADC_EACR8)	Write/Read
0x50	Alternate Configuration 9 Control Register (ADC_ACR9)	Write/Read
0x51	Reserved	—
0x52	Reserved	—
0x53	Extended Alternate Configuration 9 Control Register (ADC_EACR9)	Write/Read
0x54	Alternate Configuration 10 Control Register (ADC_ACR10)	Write/Read
0x55	Reserved	—
0x56	Reserved	—
0x57	Extended Alternate Configuration 10 Control Register (ADC_EACR10)	Write/Read
0x58	Alternate Configuration 11 Control Register (ADC_ACR11)	Write/Read
0x59	Reserved	—
0x5A	Reserved	—
0x5B	Extended Alternate Configuration 11 Control Register (ADC_EACR11)	Write/Read
0x5C	Alternate Configuration 12 Control Register (ADC_ACR12)	Write/Read
0x5D	Reserved	—
0x5E	Reserved	—
0x5F	Extended Alternate Configuration 12 Control Register (ADC_EACR12)	Write/Read
0x60	Alternate Configuration 13 Control Register (ADC_ACR13)	Write/Read
0x61	Reserved	—
0x62	Reserved	—
0x63	Extended Alternate Configuration 13 Control Register (ADC_EACR13)	Write/Read
0x64	Alternate Configuration 14 Control Register (ADC_ACR14)	Write/Read
0x65	Reserved	—
0x66	Reserved	—
0x67	Extended Alternate Configuration 14 Control Register (ADC_EACR14)	Write/Read
0x68-0x6F	Reserved	—
0x70	Pull Up/Down Control Register0 (ADC_PUDCR0)	Write/Read
0x71	Pull Up/Down Control Register1 (ADC_PUDCR1)	Write/Read
0x72	Pull Up/Down Control Register2 (ADC_PUDCR2)	Write/Read
0x73	Pull Up/Down Control Register3 (ADC_PUDCR3)	Write/Read
0x74	Pull Up/Down Control Register4 (ADC_PUDCR4)	Write/Read
0x75	Pull Up/Down Control Register5 (ADC_PUDCR5)	Write/Read
0x76	Pull Up/Down Control Register6 (ADC_PUDCR6)	Write/Read

Table 19-20. On-Chip ADC Memory Map (continued)

0x77	Pull Up/Down Control Register7 (ADC_PUDCR7)	Write/Read
0x78-0x9F	Reserved	—
0xA0	Red Line Time Base 1 Register (ADC_REDTBR1)	Read
0xA1	Red Line Time Base 2 Register (ADC_REDTBR2)	Read
0xA2-0xFF	Reserved	—

[1] Throughout the table, ADC0/ADC1 indicates that if the command is stored in CBuffer0 it will be applied to ADC0 and if in CBuffer1 it applies to ADC1. If this indication is omitted the register applies for both ADC0 and ADC1, independent of the CBuffer used.

19.6.3.1 ADC0/1 Control Registers (ADC0_CR and ADC1_CR)

The ADC0/1 Control Registers (ADC0/1_CR) is used to define the standard configuration of the ADC. In the standard configuration, the parameters contained in the Alternate Configuration Control Registers (ADC_ACR1-14) are fixed at their reset value. A conversion uses the standard configuration when the conversion command (with the standard format) is written to address 0x00 of the on-chip ADC memory map. Refer to Section 19.7.2.2.1, "Conversion Command Format for the Standard Configuration.

ADC0 Register Address: 0x01

R	ADC0_EN	0	0	0	ADC0_EMUX	0	ADC0_TBSEL		ADC0_ODD_PS	ADC0_CLK_DTY	ADC0_CLK_SEL	ADC0_CLK_PS				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

ADC1 Register Address: 0x01

R	ADC1_EN	0	0	0	ADC1_EMUX	0	ADC1_TBSEL		ADC1_ODD_PS	ADC1_CLK_DTY	ADC1_CLK_SEL	ADC1_CLK_PS				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

Figure 19-33. ADC0/1 Control Registers (ADC0/1_CR)

Table 19-21. EQADC_CFxRw Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>ADC0/1_EN</td><td>Enable bit for ADC0/1. ADC0/1_EN enables ADC0/1 to perform A/D conversions. Refer to Section 19.7.6.1, “Enabling and Disabling the On-chip ADCs,” for details.
0 ADC is disabled. Clock supply to ADC0/1 is stopped.
1 ADC is enabled and ready to perform A/D conversions.

Note: The bias generator circuit inside the ADC hard macro ceases functioning when both ADC0_EN and ADC1_EN bits are negated.
Note: Conversion commands sent to the CBuffer of a disabled ADC are ignored by the ADC control hardware.
Note: When the ADC0/1_EN status is changed from asserted to negated, the ADC Clock will not stop until it reaches its low phase.
Note: Both ADC0 and ADC1 of an eQADC module pair must be enabled before calibrating or using either ADC0 or ADC1 of the pair. Failure to enable both ADC0 and ADC1 of the pair can result in inaccurate conversions.</td></tr>
<tr><td>ADC0/1_EMUX</td><td>External Multiplexer enable for ADC0/1. When ADC0/1_EMUX is asserted, the MA pins will output digital values according to the number of the external channel being converted for selecting external multiplexer inputs. Refer to Section 19.7.7, “Internal/External Multiplexing,” for a detailed description about how ADC0/1_EMUX affects channel number decoding.

0 External multiplexer disabled; no external multiplexer channels can be selected.
1 External multiplexer enabled; external multiplexer channels can be selected.

Note: Both ADC0/1_EMUX bits must not be asserted at the same time.
Note: The ADC0/1_EMUX bit must only be written when the ADC0/1_EN bit is negated. ADC0/1_EMUX can be set during the same write cycle used to set ADC0/1_EN.</td></tr>
<tr><td>ADC0/1_TBSEL</td><td>Timebase Selection for ADC0/1. The ADC0/1_TBSEL[0:11:0] field selects the time information to be used as timestamp.
<table>
<tr><th>ADC0/1_TBSEL[0:1]</th><th>Definition</th></tr>
<tr><td>00</td><td>Selects internally generated time base as time stamp.</td></tr>
<tr><td>01</td><td>Selects imported time base 1 indicated by SRV1 bit field of EQADC_REDLCCR register.</td></tr>
<tr><td>10</td><td>Selects imported time base 2 indicated by SRV2 bit field of EQADC_REDLCCR register.</td></tr>
<tr><td>11</td><td>Reserved</td></tr>
</table>
Note: This selection is overridden by the corresponding field ATBSEL in the ADC_ACR1-14 registers when the alternate conversion command is used.</td></tr>
<tr><td>ADC0/1_ODD_PS</td><td>Clock Prescaler Odd Rates Selector for ADC0/1. The ADC0/1_ODD_PS field is used together with the ADC0/1_CLK_PS field to define even/odd divide factors in the generation of the ADC0/1 clocks. Refer to Table 19-22 for available divide factors.

0 Even divide factor is selected. The final divide factor is dependent of ADC0/1_CLK_PS field.
1 Odd divide factor is selected. The final divide factor is dependent of ADC0/1_CLK_PS field.</td></tr>
</table>

Table 19-21. EQADC_CFxRw Field Descriptions (continued)

Field	Description
ADC0/1_CLK_DTY	Clock Duty Rate Selector for ADC0/1 (for odd divide factors). The ADC0/1_CLK_DTY field controls the duty rate of the ADC0/1 clock when the ADC0/1_CLK_PS field is asserted. The generated clock has an odd number of peripheral clock cycles, therefore this field is used to select a clock duty higher or lower than 50%. 0 clock low interval is longer 1 clock cycle than high pulse. 1 clock high pulse is longer 1 clock cycle than low portion.
ADC0/1_CLK_SEL	Clock Selector for ADC0/1. The ADC0/1_CLK_SEL is used to select between the peripheral clock signal or the prescaler output signal. The prescaler provides the peripheral clock signal divided by a even factor from 2 to 64. This is required to permit the ADC to run as fast as possible when the device is in Low Power Active mode and peripheral clock is around 1 MHz. 0 Prescaler output clock is selected. 1 Peripheral clock is selected - maximum frequency. **Note:** The ADC0/1_CLK_SEL bits must only be written when the ADC0/1_EN bit is negated. ADC0/1_CLK_SEL can be set during the same write cycle used to set ADC0/1_EN.
ADC0/1_CLK_PS [0:4]	Clock Prescaler Field for ADC0/1. The ADC0/1_CLK_PS field controls the peripheral clock divide factor for the ADC0/1 clock as in Table 19-22. See Section 19.7.6.2, “ADC Clock and Conversion Speed,” for details about how to set ADC0/1_CLK_PS. **Note:** The ADC0/1_CLK_PS field must only be written when the ADC0/1_EN bit is negated. This field can be configured during the same write cycle used to set ADC0/1_EN.

Table 19-22. Peripheral clock Divide Factor for ADC Clock

ADC0/1_CLK_PS[0:4]	Peripheral clock Divide Factor	
	ADC0/1_ODD_PS = 0	ADC0/1_ODD_PS = 1
0b00000	2	3
0b00001	4	5
0b00010	6	7
0b00011	8	9
0b00100	10	11
0b00101	12	13
0b00110	14	15
0b00111	16	17
0b01000	18	19
0b01001	20	21
0b01010	22	23
0b01011	24	25
0b01100	26	27
0b01101	28	29

Table 19-22. Peripheral clock Divide Factor for ADC Clock (continued)

ADC0/1_CLK_PS[0:4]	Peripheral clock Divide Factor	
	ADC0/1_ODD_PS = 0	ADC0/1_ODD_PS = 1
0b01110	30	31
0b01111	32	33
0b10000	34	35
0b10001	36	37
0b10010	38	39
0b10011	40	41
0b10100	42	43
0b10101	44	45
0b10110	46	47
0b10111	48	49
0b11000	50	51
0b11001	52	53
0b11010	54	55
0b11011	56	57
0b11100	58	59
0b11101	60	61
0b11110	62	63
0b11111	64	65

19.6.3.2 ADC Time Stamp Control Register (ADC_TSCR)

The ADC Time Stamp Control Register (ADC_TSCR) contains a peripheral clock divide factor used in the making of the time base counter clock. It determines at what frequency the time base counter will run. ADC_TSCR can be accessed by configuration commands sent to CBuffer0 or to CBuffer1. A data write to ADC_TSCR through a configuration command sent to CBuffer0 will write the same memory location as when writing to it through a configuration command sent to CBuffer1.

NOTE

Simultaneous write accesses from CBuffer0 and CBuffer1 to ADC_TSCR are not allowed.

ADC0/1 Register Address: 0x02

R	0	0	0	0	0	0	0	0	0	0	0	0	TBC_CLK_PS			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-34. ADC Time Stamp Control Register (ADC_TSCR)

Table 19-23. ADC_TSCR Field Descriptions

Field	Description
TBC_CLK_PS[0:3]	Time Base Counter Clock Prescaler. The TBC_CLK_PS field contains the peripheral clock divide factor for the time base counter. It controls the accuracy of the time stamp. The prescaler is disabled when TBC_CLK_PS is set to 0b0000.

Table 19-24. Clock Divide Factor for Time Stamp

TBC_CLK_PS[0:3]	Peripheral Clock Divide Factor	Clock to Time Stamp Counter for a 120 MHz Peripheral Clock (MHz)
0b0000	Disabled	Disabled
0b0001	1	120
0b0010	2	60
0b0011	4	30
0b0100	6	20
0b0101	8	15
0b0110	10	12
0b0111	12	10
0b1000	16	7.5
0b1001	32	3.75
0b1010	64	1.88
0b1011	128	0.94
0b1100	256	0.47
0b1101	512	0.23
0b1110 - 0b1111	Reserved	-

NOTE

If TBC_CLK_PS is not set to disabled, it must not be changed to any other value besides disabled. If TBC_CLK_PS is set to disabled it can be changed to any other value.

19.6.3.3 ADC Time Base Counter Registers (ADC_TBCR)

The ADC Time Base Counter Register (ADC_TBCR) contains the current value of the time base counter. ADC_TBCR can be accessed by configuration commands sent to CBuffer0 or to CBuffer1. A data write to ADC_TBCR through a configuration command sent to CBuffer0 will write the same memory location as when writing to it through a configuration command sent to CBuffer1.

NOTE

Simultaneous write accesses from CBuffer0 and CBuffer1 to ADC_TBCR are not allowed.

ADC0/1 Register Address: 0x03

R W	TBC_VALUE															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-35. ADC Time Base Counter Register (ADC_TBCR)

Table 19-25. ADC_TSCR Field Descriptions

Field	Description
TBC_VALUE[0:15]	Time Base Counter value. The TBC_VALUE field contains the current value of the time base counter. Reading TBC_VALUE returns the current value of time base counter. Writes to TBC_VALUE register load the written data to the counter. The time base counter counts from 0x0000 to 0xFFFF and wraps when reaching 0xFFFF.

19.6.3.4 ADC0/1 Gain Calibration Constant Registers (ADC0_GCCR and ADC1_GCCR)

The ADC0/1 Gain Calibration Constant Register (ADC0/1_GCCR) contains the gain calibration constant used to fine-tune the ADC0/1 conversion results. Refer to Section 19.7.6.6, “ADC Calibration Feature,” for details about the calibration scheme used in the EQADC.

ADC0 Register Address: 0x04

R	0	GCC0														
W																
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ADC1 Register Address: 0x04

R	0	GCC1														
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-36. ADC0/1 Gain Calibration Constant Registers (ADC0/1_GCCR)

Table 19-26. ADC0/1_GCCR Field Descriptions

Field	Description
GCC0/1[0:14]	Gain calibration constant for ADC0/1. GCC0/1 contains the gain calibration constant used to fine-tune ADC0/1 conversion results. It is a unsigned 15-bit fixed pointed value. The gain calibration constant is an unsigned fixed point number expressed in the GCC_INT.GCC_FRAC binary format. The integer part of the gain constant (GCC_INT) contains a single binary digit while its fractional part (GCC_FRAC) contains fourteen digits. For details about the GCC data format refer to Section 19.7.6.6.2, "MAC Unit and Operand Data Format."

19.6.3.5 ADC0/1 Offset Calibration Constant Registers (ADC0_OCCR and ADC1_OCCR)

The ADC0/1 Offset Calibration Constant Register (ADC0/1_OCCR) contains the offset calibration constant used to fine-tune of ADC0/1 conversion results. The offset constant is a signed 14-bit integer value. Refer to Section 19.7.6.6, "ADC Calibration Feature," for details about the calibration scheme used in the EQADC.

ADC0 Register Address: 0x05

R	0	0	OCC0													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ADC1 Register Address: 0x05

R	0	0	OCC1													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-37. ADC0/1 Offset Calibration Constant Registers (ADC0/1_OCCR)

Table 19-27. ADC0/1_OCCR Field Descriptions

Field	Description
OCC0/1[0:13]	Offset Calibration Constant of ADC0/1. OCC0/1 contains the offset calibration constant used to fine-tune ADC0/1 conversion results. Negative values should be expressed using the two's complement representation.

19.6.3.6 Alternate Configuration 1-14 Control Registers (ADC_ACR1-14)

The Alternate Configuration Control Registers (ADC_ACR1-14) are used to configure the alternate configurations of the ADC. There are 14 possible alternate configurations, each one associated with one of the ADC_ACR1-14 registers. All alternate configurations share the same standard configuration parameters from the ADC0/1_CR registers, plus additional configuration parameters contained in the ADC_ACR1-14. A conversion uses one of the alternate configurations when the conversion command (with the alternate configuration format) is written to an address in the range 0x08-0x15 of the on-chip ADC memory map. It also can use the Extended Alternated Configurations as described in Section 1.5.3.9, “Extended Alternate Configuration 1-14 Control Registers (ADC_EACR1-14)” if bit EQADC_MCR[DSM] = 1. Refer to Section 19.7.2.2.2, “Conversion Command Format for Alternate Configurations.”

ADC0/1 Register Address: 0x30, 0x34, 0x38, 0x3C,... 0x5C, 0x60, 0x64 Access: User read only

R	RET_ INH	0	DEST				FMTA	RMSEL	RESSEL		0	0	ATBSEL		PRE_GAIN	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-38. Alternate Configuration 1-14 Control Registers (ADC_ACR1-14)

Table 19-28. ADC_ACR1-8 Field Descriptions

Field	Description
RET_INH	Result Transfer Inhibit / Decimation Filter Pre-Fill. This bit is used to inhibit the transfer of the result data from the peripheral module to the result queue. When the module is a Decimation Filter, this bit sets the filter in a special mode (PRE-FILL) in which it does not generate decimated samples out from the conversion results received from the EQADC block, but the conversion samples are used by the filter algorithm. This feature allows a proper initialization of the Decimation Filter without generating any decimated result. Or this bit is useful for sending the result of the ADC to the STAC bus master but not putting the result in the result queue. 0 Result transfer to result queue / Decimation Filter in filtering mode 1 No result transfer to result queue / Decimation Filter PRE-FILL mode
DEST[0:3]	Conversion Result Destination Selection. The DEST[0:3] field selects the Decimation Filter destinations of the conversion result generated by the Alternate Conversion Command. This field also affects the behavior of the FMTA bit and the FFMT bit of the conversion command for alternate configurations (see Section 19.7.2.2.2, "Conversion Command Format for Alternate Configurations).” **Note:** When EQADC_MCR[DSM]=0, decimation filter H through L are only accessible from the CPU, and cannot be accessed by an eQADC. When EQADC_MCR[DSM]=1, the functionality of the DEST field is defined by Table 19-32.

DEST[0:3]	Description
0000	The conversion result is sent to the RFIFOs. The data format is specified by the FFMT bit in the conversion command.
0001	The conversion result is sent to Decimation Filter block A.
0010	The conversion result is sent to Decimation Filter block B.
0011	The conversion result is sent to Decimation Filter block C.
0100	The conversion result is sent to Decimation Filter block D.
0101	The conversion result is sent to Decimation Filter block E.
0110	The conversion result is sent to Decimation Filter block F.
0111	The conversion result is sent to Decimation Filter block G.
1000	The conversion result is sent to Decimation Filter blocks A and E.
1001	The conversion result is sent to Decimation Filter blocks B and E.
1010	The conversion result is sent to Decimation Filter blocks C and E.
1011	The conversion result is sent to Decimation Filter blocks D and E.
1100	The conversion result is sent to Decimation Filter blocks A and G.
1101	The conversion result is sent to Decimation Filter blocks B and G.
1110	The conversion result is sent to Decimation Filter blocks C and G.
1111	The conversion result is sent to Decimation Filter blocks D and G.

Table 19-28. ADC_ACR1-8 Field Descriptions (continued)

Field	Description
FMTA	Conversion Data Format for Alternate Configuration. If the DEST field is not 0b000, the FMTA bit specifies how the 12-bit conversion data returned by the ADCs is formatted into the 16-bit data which is sent to the parallel side interface. 0 Right justified unsigned 1 Right justified signed
RMSEL	Return Module Selection. The RMSEL bit specifies which eQADC module will receive data returning from a decimation filter. **Note:** A decimation filter may take samples from two eQADCs, allowing double sample rate. If the RMSEL bits for each are different, the results are sent to just one of the eQADCs. 0 Decimation filter return data is sent to the eQADC that supplied the filter input data. 1 Decimation filter return data is sent to the eQADC that did *not* supply the filter input data.
RESSEL[0:1]	ADC Resolution Selection. 00 ADC set to 12-bits resolution 01 ADC set to 10-bits resolution 10 ADC set to 8-bits resolution 11 Reserved
ATBSEL	Alternate Command Timebase Selector. The ATBSEL field selects the time information to be used as timestamp. **Note:** This selection overrides the corresponding fields ADC0/1_TBSEL in the ADC0/1_CR registers when the alternate conversion command is used.
PRE_GAIN[0:1]	ADC Pre-gain control. The PRE_GAIN[0:1] controls the gain of the ADC input stage by changing the internal ADC iterations in the gain stage. 00 X1 gain 01 X2 gain 10 X4 gain 11 Reserved

ATBSEL[0:11]	Description
00	Selects internally generated time base as time stamp.
01	Selects imported time base 1 indicated by SRV1 bit field of EQADC_REDLCCR register.
10	Selects imported time base 2 indicated by SRV2 bit field of EQADC_REDLCCR register.
11	Reserved

19.6.3.7 ADC0/1 Alternate Gain Registers (ADC0_AGR1-2 and ADC1_AGR1-2)

The Alternate Gain Registers (ADC0_AGRx and ADC1_AGRx, x=1-2) contain the gain calibration constants used to fine-tune the ADCs conversion results for alternate configurations 1 or 2. A conversion from an ADC uses the corresponding ADC0_AGRx or ADC1_AGRx register when the conversion command (with the alternate configuration format) is written to an address in the range 0x08-0x09 of the on-chip ADC memory map. Refer to Section 19.7.6.6, “ADC Calibration Feature,” for details about the calibration scheme used in the EQADC.

ADC0 Register Address: 0x31, 0x35

R	0	ALTGCC0x														
W																
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ADC1 Register Address: 0x31, 0x35

R	0	ALTGCC1x														
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-39. ADC0/1 Alternate x Gain Register (ADC0/1_AGRx, x=1-2)

Table 19-29. ADC0/1_AGRx Field Descriptions

Field	Description
ALTGCC0/1x[0:14]	Alternate Gain Calibration Constant. ALTGCC0/1x[0:14] contain the gain calibration constants used to fine-tune ADC0/1 conversion results for alternate configurations 1 and 2. The gain calibration constants are 15-bit unsigned fixed point numbers expressed in the GCC_INT.GCC_FRAC binary format. The integer part of the gain constants (GCC_INT) contain a single binary digit while their fractional part (GCC_FRAC) contain fourteen digits. For details about the GCC data format refer to Section 19.7.6.6.2, “MAC Unit and Operand Data Format.”

19.6.3.8 ADC0/1 Alternate Offset Register (ADC0_AOR1-2 and ADC1_AOR1-2)

The Alternate Offset Registers (ADC0_AORx and ADC1_AORx, x=1-2) contain the offset calibration constants used to fine-tune ADCs conversion results for alternate configurations 1 and 2. The offset constants are signed 14-bit integer values. Refer to Section 19.7.6.6, “ADC Calibration Feature,” for details about the calibration scheme used in the EQADC.

ADC0 Register Address: 0x32, 0x36

R	0	0	ALTOCC0x													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

ADC1 Register Address: 0x31, 0x35

R	0	0	ALTOCC1x													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-40. ADC0/1 Alternate x Offset Registers (ADC0/1_AORx, x=1-2)

Table 19-30. ADC0/1_AORx Field Descriptions

Field	Description
ALTOCC0/1x[0:13]	Alternate Offset Calibration Constant.ALTOCC0/1x[0:13] contain the offset calibration constants used to fine-tune ADCs conversion results for alternate configurations 1 or 2. Negative values should be expressed using the two’s complement representation.”

19.6.3.9 Extended Alternate Configuration 1-14 Control Registers (ADC_EACR1-14)

The Extended Alternate Configuration Control Registers (ADC_EACR1-14) are used together with the Alternate Configuration Control Registers (ADC_ACR1-14) of the same number to complement the alternate configurations of the ADC. A conversion uses one pair of the alternate configuration regs (normal and extended) when the conversion command (with the alternate conversion format) is written to an address in the range 0x08-0x15 of the on-chip ADC memory map. Refer to "Conversion Command Format for Alternate Configurations".

ADC0/1 Register Address: 0x33, 0x37, 0x3B, 0x3F,... 0x5F, 0x63, 0x67 Access: User read/write

R	RET_ INH2	0	DEST2				FMTA2	RMSEL 2	FLEN 2	0	0	TEN2	MESSAGE_TAG2			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-41. Extended Alternate Configuration 1-14 Control Registers (ADC_EACR1-14)

Table 19-31. ADC_EACR*x* Field Descriptions

Field	Description
RET_INH2	Second Result Transfer Inhibit / Decimation Filter Pre-Fill. Same definition as RET_INH field in EQADC_ACRx but applying to the second destination selected by DEST2. 0 Result transfer to result queue / Decimation Filter in filtering mode 1 No result transfer to result queue / Decimation Filter PRE-FILL mode
DEST2[0:3]	Second Conversion Result Destination Selection. The DEST2[0:3] field selects the second Decimation Filter destination of the conversion result generated by the Alternate Conversion Command. In addition, the eQADC operation due to this field is also affected by the ADC_ACRx DEST field, by the ADC_EACRx TEN2 field, by MESSAGE_TAG field, by the ADC_EACRx MESSAGE_TAG2 field as defined in Table 19-32.
FMTA2	Second Conversion Data Format for Alternate Configuration. The FMTA2 bit specifies how the 12-bit conversion data returned by the ADCs is formatted into the 16-bit data which is sent to the RFIFOs or the parallel side interface defined by DEST2. Refer to Table 19-32 for more details. 0 Right justified unsigned 1 Right justified signed
RMSEL2	Second Return Module Selection. The RMSEL bit specifies which eQADC module will receive data returning from the decimation filter specified by DEST2. **Note:** A decimation filter may take samples from two eQADCs, allowing double sample rate. If the RMSEL bits for each are different, the results are sent to just one of the eQADCs. 0 Decimation filter return data is sent to the eQADC that supplied the filter input data. 1 Decimation filter return data is sent to the eQADC that did *not* supply the filter input data.
FLEN2	Flush Enable for Second Destination. The FLEN2 bit enables the transfer of the flush command given by the FFMT field from the alternate conversion command when DEST>0. 0 No flush command is sent to second destination. 1 Flush command is sent to second destination if specified by FFMT when DEST>0.
TEN2	Enable MESSAGE_TAG2 for Second Destination. This bit specifies if the MESSAGE_TAG2 value should be used for the second destination result transfer. Refer to Table 19-32 for more details. 0 MESSAGE_TAG2 is disabled. 1 MESSAGE_TAG2 is enabled.
MESSAGE_TAG2 [0:3]	Same as the MESSAGE_TAG field in the Conversion Command as described in "Conversion Command Format for the Standard Configuration".

Table 19-32. Conversion Destination Selection (EQADC_MCR[DSM] = 1)

DEST[b]	DEST2[c]	TEN2[c]	MESSAGE_TAG2[c] = MESSAGE_TAG[a]?	Same formats (two RFIFOs)[d]?	Same formats/controls (two filters)[e]?	Primary destination			Secondary destination			Notes
						Destination	Format specifier	Flush control	Destination	Format specifier	Flush control	
0000b (RFIFO)	0000b (RFIFO)	0b	X	X	X	RFIFO (MESSAGE_TAG[a] specifies the RFIFO.)	FFMT[a]	N/A	None (DEST2[c] is 0000b and TEN2[c] is 0b, so the second destination is ignored.)			—
		1b	True	True					None (Two identical result sets would be sent to the same RFIFO, so the secondary destination is ignored.)			
				False					RFIFO (MESSAGE_TAG2[c] specifies the RFIFO.)	FMTA2[c]	N/A	
			False	X								
0000b (RFIFO)	0001–1111b (Filter module)	0b	X	X	X	RFIFO (MESSAGE_TAG[a] specifies the RFIFO.)	FFMT[a]	N/A	Filter module (1–15) (DEST2[c] specifies the module; MESSAGE_TAG[a] specifies the return RFIFO. RET_INH2[c] and RMSEL2[c] control the module.)	FMTA2[c]	None (FFMT[a] is used as the primary-destination format specifier)	—
		1b	True						Filter module (1–15) (DEST2[c] specifies the module; MESSAGE_TAG2[c] specifies the return RFIFO. RET_INH2[c] and RMSEL2[c] control the module.)			
			False									

[a] In the alternate conversion command.

[b] In the associated ADC_ACRx register.

[c] In the associated ADC_EACRx register.

[d] Considered only when both destinations are RFIFOs. The sample formats are the same: FMTA2[c] = FFMT[a].

[e] Considered only when both destinations are Decimation filter modules. The sample formats and controls are the same: (FMTA2[c] = FMTA[b]) && (RET_INH2[c] = RET_INH[b]) && (RMSEL2[c] = RMSEL[b]) && (((FLEN2[c] = 0b) && (FFMT[a] = 0b)) || (FLEN2[c] = 1b)).

[f] The primary-destination result set is sent first, and the secondary-destination result set is sent later.

[g] The primary- and secondary-destination result sets are sent to the Decimation filter modules simultaneously.

[h] The order in which processed result sets (returned from the Decimation filter modules) are stored in the RFIFO is not guaranteed. One filter module may take longer than the other to process its result set, and when two or more modules attempt to return result sets at the same time, the EQADC responds to the highest-priority module first (module 1 has the highest priority; module 15 has the lowest priority).

DEST[b]	DEST2[c]	TEN2[c]	MESSAGE_TAG2[c] = MESSAGE_TAG[a]?	Same formats (two RFIFOs)[d]?	Same formats/controls (two filters)[e]?	Primary destination			Secondary destination			Notes
						Destination	Format specifier	Flush control	Destination	Format specifier	Flush control	
0001–1111b (Filter module)	0000b (RFIFO)	0b	X	X	X	Filter module (1–15) (DEST[b] specifies the module; MESSAGE_TAG[a] specifies the return RFIFO. RET_INH[b] and RMSEL[b] control the module.)	FMTA[b]	FFMT[a]	None (DEST2[c] is 0000b and TEN2[c] is 0b, so the second destination is ignored.)			—
		1b	True			Filter module (1–15) (DEST[b] specifies the module; MESSAGE_TAG[a] specifies the return RFIFO. RET_INH[b] and RMSEL[b] control the module.)			RFIFO (MESSAGE_TAG2[c] specifies the RFIFO.)	FMTA2[c]	N/A	
			False			Filter module (1–15) (DEST[b] specifies the module; MESSAGE_TAG[a] specifies the return RFIFO. RET_INH[b] and RMSEL[b] control the module.)						
0001–1111b (Filter module)	0001–1111b (Filter module) (equal to DEST)	0b	X	X	False	Filter module (1–15) (DEST[b] specifies the module; MESSAGE_TAG[a] specifies the return RFIFO. RET_INH[b] and RMSEL[b] control the module.)	FMTA[b]	FFMT[a]	Filter module (1–15) (DEST2[c] specifies the module; MESSAGE_TAG[a] specifies the return RFIFO. RET_INH2[c] and RMSEL2[c] control the module.)	FMTA2[c]	FFMT[a] when FLEN2[c] is 1b	—
					True				None (Two identical result sets would be sent to the same RFIFO, so the secondary destination is ignored.)			
		1b	True		True							
					False				Filter module (1–15) (DEST2[c] specifies the module; MESSAGE_TAG2[c] specifies the return RFIFO. RET_INH2[c] and RMSEL2[c] control the module.)	FMTA2[c]	FFMT[a] when FLEN2[c] is 1b	
			False		X							

[a] In the alternate conversion command.

[b] In the associated ADC_ACRx register.

[c] In the associated ADC_EACRx register.

[d] Considered only when both destinations are RFIFOs. The sample formats are the same: FMTA2[c] = FFMT[a].

[e] Considered only when both destinations are Decimation filter modules. The sample formats and controls are the same: (FMTA2[c] = FMTA[b]) && (RET_INH2[c] = RET_INH[b]) && (RMSEL2[c] = RMSEL[b]) && (((FLEN2[c] = 0b) && (FFMT[a] = 0b)) \|\| (FLEN2[c] = 1b)).

[f] The primary-destination result set is sent first, and the secondary-destination result set is sent later.

[g] The primary- and secondary-destination result sets are sent to the Decimation filter modules simultaneously.

[h] The order in which processed result sets (returned from the Decimation filter modules) are stored in the RFIFO is not guaranteed. One filter module may take longer than the other to process its result set, and when two or more modules attempt to return result sets at the same time, the EQADC responds to the highest-priority module first (module 1 has the highest priority; module 15 has the lowest priority).

DEST^b	DEST2^c	TEN2^c	MESSAGE_TAG2^c = MESSAGE_TAG^a?	Same formats (two RFIFOs)^d?	Same formats/controls (two filters)^e?	Primary destination			Secondary destination			Notes
						Destination	Format specifier	Flush control	Destination	Format specifier	Flush control	
0001–1111b (Filter module)	0001–1111b (Filter module) (not equal to DEST)	0b	X	X	False	Filter module (1–15) (DEST^b specifies the module; MESSAGE_TAG^a specifies the return RFIFO. RET_INH^b and RMSEL^b control the module.)	FMTA^b	FFMT^a	Filter module (1–15) (DEST2^c specifies the module; MESSAGE_TAG^a specifies the return RFIFO. RET_INH2^c and RMSEL2^c control the module.)	FMTA2^c	FFMT^a when FLEN2^c is 1b	f, h
					True							g, h
		1b	True		True				Filter module (1–15) (DEST2^c specifies the module; MESSAGE_TAG2^c specifies the return RFIFO. RET_INH2^c and RMSEL2^c control the module.)			
					False							f, h
			False		X							—

[a] In the alternate conversion command.

[b] In the associated ADC_ACRx register.

[c] In the associated ADC_EACRx register.

[d] Considered only when both destinations are RFIFOs. The sample formats are the same: FMTA2^c = FFMT^a.

[e] Considered only when both destinations are Decimation filter modules. The sample formats and controls are the same: (FMTA2^c = FMTA^b) && (RET_INH2^c = RET_INH^b) && (RMSEL2^c = RMSEL^b) && (((FLEN2^c = 0b) && (FFMT^a = 0b)) || (FLEN2^c = 1b)).

[f] The primary-destination result set is sent first, and the secondary-destination result set is sent later.

[g] The primary- and secondary-destination result sets are sent to the Decimation filter modules simultaneously.

[h] The order in which processed result sets (returned from the Decimation filter modules) are stored in the RFIFO is not guaranteed. One filter module may take longer than the other to process its result set, and when two or more modules attempt to return result sets at the same time, the EQADC responds to the highest-priority module first (module 1 has the highest priority; module 15 has the lowest priority).

19.6.3.10 ADC Pull Up/Down Control Register *x* (ADC_PUDCRx, *x*=0-7)

The ADC Pull Up/Down Control Register *x* (ADC_PUDCR*x*) contains configuration bits for pull up and pull down resistors present at ADC input channels *x*, *x*=0 to 7.

ADC0/1 Register Address: 0x70–0x77 Access: User read/write

R	0	0	CH_PULL*x*		0	0	PULL_STR*x*		0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-42. ADC Pull Up/Down Control Register *x* (ADC_PUDCR*x*, *x*=0-7)

Table 19-33. ADC_PUDCR*x* Field Descriptions

Field	Description
CH_PULLx[0:1]	Channel x Pull Up/Down Control bits. The CH_PULLx[0:1] field controls the pull up/down configuration of the channel x. 00 No Pull resistors connected to the channel 01 Pull Up resistor connected to the channel 10 Pull Down resistor connected to the channel 11 Pull Up and Pull Down resistors connected to the channel
PULL_STRx[0:1]	Pull Up/Down Strength Control bits of channel x. The PULL_STRx[0:1] bit field defines the strength of the channel x pull up or down resistors. 00 Reserved 01 200 Kohm pull resistor 10 100 Kohm pull resistor 11 5 Kohm (approx.) pull resistor (not available for CH_PULL_*x* = 0b11)

19.6.3.11 Red Line Time Base 1/2 Registers (ADC_REDTBR1/2)

The Red Line Time Base Register1/2 (ADC_REDTBR1/2) contain the current value of the imported time base counter as selected by bit fields on register EQADC_REDLCCR. See "Conversion Command Format for the Standard Configuration" for more details.

ADC0/1 Register Address: 0xA0, 0xA1 Access: User read

R	TB_VALUE															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-43. ADC Time Base Counter Register (ADC_REDTBR1/2)

Table 19-34. ADC_REDTBR1/2 Field Descriptions

Field	Description
TB_VALUE[0:15]	Time Base VALUE Field. Contains the current value of the imported time base.

19.7 Functional Description

19.7.1 Overview

The EQADC provides a parallel interface to two on-chip ADCs and a parallel side interface to an on-chip companion module, like a decimation filter. The two on-chip ADCs are architectured to allow access to all the analog channels.

Initially, command data is contained in system memory in a user defined data structure which is likely to be a queue as depicted in Figure 19-2[1]. Command data is moved between the CQueues and CFIFOs by the

---

[1] Command and result data can be stored in the system memory in any user defined data structure. However, in this document it will be assumed that the data structure of choice is a queue, since it is the most likely data structure to be used and because queues are the only type of data structure supported by the DMAC.

host CPU or by the DMAC which respond to interrupt and DMA requests generated by the EQADC. The EQADC supports software and hardware triggers from other blocks or external pins to initiate transfers of commands from the multiple CFIFOs to the on-chip ADCs.

CFIFOs can be configured in single-scan or continuous-scan mode. When a CFIFO is configured in single-scan mode, the EQADC scans the CQueue one time. The EQADC stops transferring commands from the triggered CFIFO after detecting the EOQ bit set in the last transfer. After an EOQ bit is detected, software involvement is required to rearm the CFIFO so that it can detect new trigger events.

When a CFIFO is configured for continuous-scan mode, the whole CQueue is scanned multiple times. After the detection of an asserted EOQ bit in the last command transfer, command transfers can continue or not depending on the mode of operation of the CFIFO.

CFIFO0 has a special configuration option to allow a repetitive sequence of conversion commands (streaming mode) with high priority characteristics (abort operation) or not. This feature is useful with the immediate conversion command feature that allows the immediate execution of a conversion command or a sequence of commands with critical timing even with the possibility of abortion of some current ADC conversion in progress. The aborted command is stored and executed again as soon as the critical timing commands have been finished.

The multiple Result FIFOs (RFIFOs) can receive data from the on-chip ADCs or from an on-chip companion module (decimation filter). Data from the on-chip ADCs can be routed to the side interface, processed by the on-chip companion module (decimation filter) and then routed back through the side interface to the RFIFOs.

19.7.2 Data Flow in EQADC

19.7.2.1 Overview and Basic Terminology

Figure 19-44 shows how command data flows inside the EQADC system. A Command Message is the predefined format at which command data is stored in the CQueues. A Command message has 32 bits and is composed of two parts: a CFIFO header and an ADC Command. Command messages are moved from the CQueues to the CFIFOs by the host CPU or by the DMAC as they respond to interrupt and DMA requests generated by the EQADC. The EQADC generates these requests whenever a CFIFO is not full. The *FIFO Control Unit* will only transfer to a CBuffer the ADC command part of the Command Message. Information in the CFIFO header together with the upper bit of the ADC command is used by the *FIFO Control Unit* to arbitrate which triggered CFIFO will be transferring the next command.

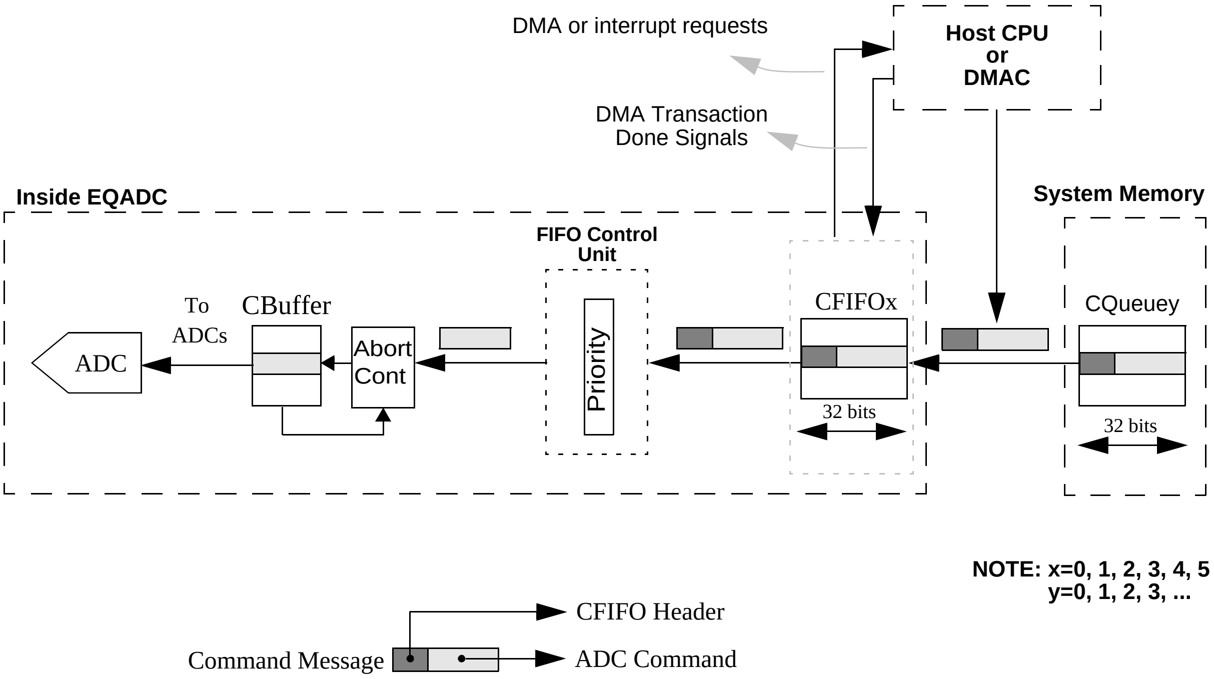


Figure 19-44. Command Flow during EQADC operation

ADC commands sent to the on-chip CBuffers are executed in a first-in-first-out basis with exception when the immediate conversion command function is enabled. Three types of results can be expected: data read from an ADC register, a conversion result, or a time stamp.

NOTE

While the EQADC pops commands out from a CFIFO, it also is checking the number of entries in the CFIFO and generating requests to fill it. The process of pushing and popping commands to and from a CFIFO can occur simultaneously. However, this is not true for CFIFO0 when configured to operate in streaming mode for popping.

The *FIFO Control Unit* expects all incoming results to be shaped in a predefined Result Message format. Figure 19-45 shows how result data flows inside the EQADC system. Results generated on the on-chip ADCs are adjusted considering the selected resolution of the ADC and are formatted into result messages inside the *Result Format and Calibration Sub-Block*. This result message can be routed directly to one of the RFIFOs or to an on-chip companion module (decimation filter) via the parallel side interface. After the data is processed by the companion module, it can be routed back to one of the RFIFOs via the side interface with the correct format. A result message is composed of an RFIFO header and an ADC Result. The *FIFO Control Unit* decodes the information contained in the RFIFO header to determine the RFIFO to which the ADC result should be sent. Once in an RFIFO, the ADC result is moved to the corresponding RQueue by the host CPU or by the DMAC as they respond to interrupt and DMA requests generated by the EQADC. The EQADC generates these requests whenever an RFIFO has at least one entry.

NOTE

> While conversion results are returned, the EQADC is checking the number of entries in the RFIFO and generating requests to empty it. The process of pushing and popping ADC results to and from an RFIFO can occur simultaneously.

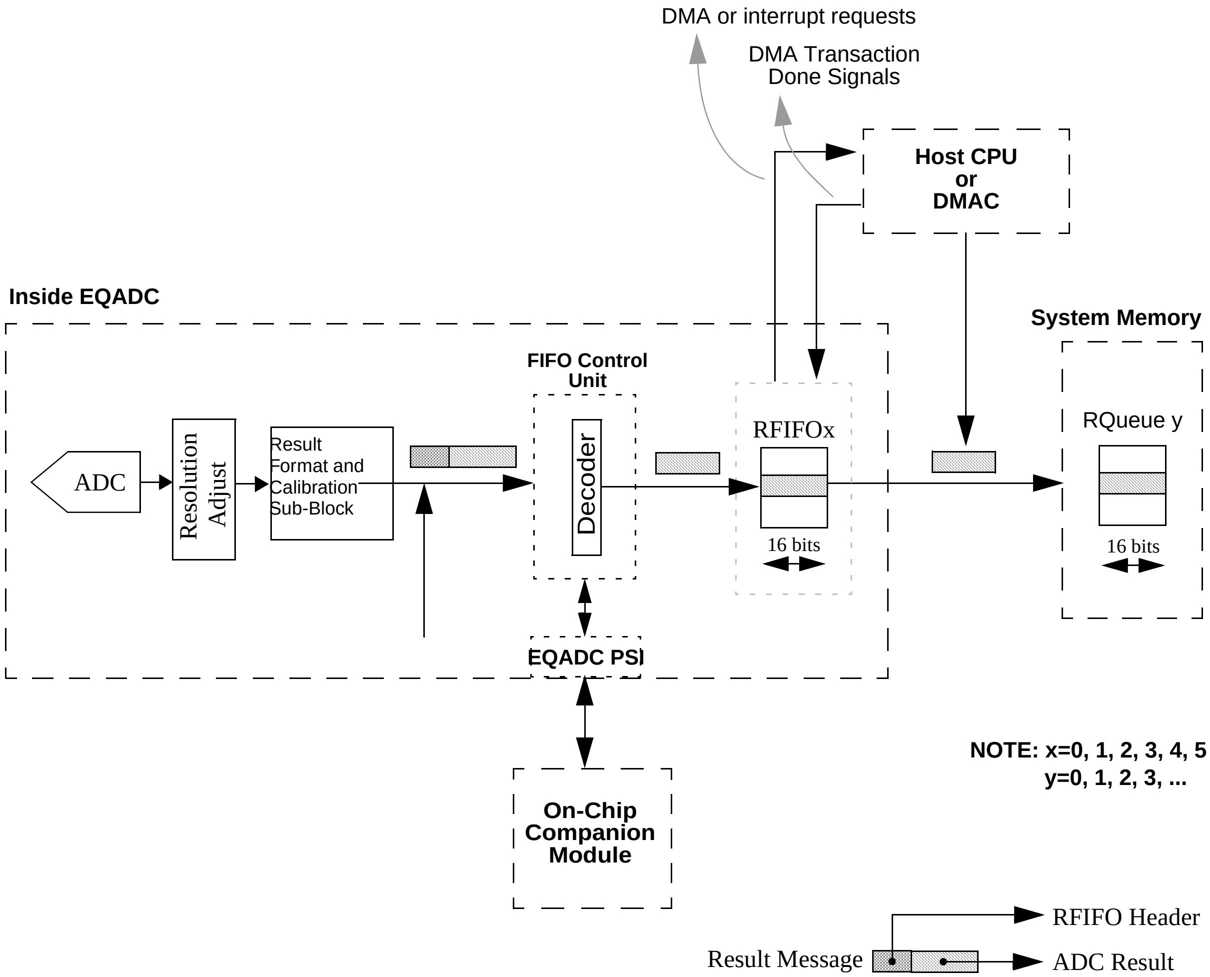


Figure 19-45. Result Flow during EQADC operation

19.7.2.2 Message Format in EQADC

This section explains the command and result message formats used for on-chip ADC operation.

A Command Message is the predefined format at which command data is stored in the cqueues. A Command message has 32 bits and is composed of two parts: a CFIFO header and an ADC Command. The size of the CFIFO header is fixed to 6 bits, and it works as inputs to the *FIFO Control Unit*. It controls when a CQueue ends, when it pauses, when a loop starts (only for CFIFO0 in streaming mode), if commands are sent to internal or external buffers, and if it can abort a serial data transmission. Information contained in the CFIFO header, together with the upper bit of the ADC Command is used by the *FIFO*

Control Unit to arbitrate which triggered CFIFO will transfer the next command. ADC commands are encoded inside the least significant 26 bits of the command message.

A Result message is composed of an RFIFO header and an ADC Result. The *FIFO Control Unit* decodes the information contained in the RFIFO header to determine the RFIFO to which the ADC result should be sent. An ADC result is always 16 bits long.

19.7.2.2.1 Conversion Command Format for the Standard Configuration

Figure 19-46 describes the format for conversion commands when interfacing with the on-chip ADCs in the standard configuration. The standard configuration is selected when the least significant byte (bits 24-31) of the conversion command is set to zero. In the standard configuration, the conversion result is always routed to one of the RFIFOs. A time stamp information can be optionally requested.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
EOQ	PAUSE	REP	RESERVED		EB (0b0)	BN	CAL	MESSAGE_TAG				LST		TSR	FMT

CFIFO Header

ADC Command

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
CHANNEL_NUMBER								0	0	0	0	0	0	0	0

ADC Command

Figure 19-46. Conversion Command Format for the Standard Configuration

The register bit descriptions are given in Table 19-35.

Table 19-35. Field Descriptions

Field	Description
EOQ	End Of Queue Bit. The EOQ bit is asserted in the last command of a CQueue to indicate to the EQADC that a scan of the CQueue is completed. EOQ instructs the EQADC to reset its current CFIFO transfer counter value (TC_CF) to zero. Depending on the CFIFO mode of operation, the CFIFO status will also change upon the detection of an asserted EOQ bit on the last transferred command - see Section 19.7.4.6, “CFIFO Scan Trigger Modes,” for details. 0 Not the last entry of the CQueue. 1 Last entry of the CQueue. **Note:** If both the PAUSE and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.
PAUSE	Pause Bit. The Pause bit allows software to create sub-queues within a CQueue. When the EQADC completes the transfer of a command with an asserted Pause bit, the CFIFO enters the WAITING FOR TRIGGER state. Refer to Section 19.7.4.7.1, “CFIFO Operation Status,” for a description of the state transitions. The Pause bit is only valid when CFIFO operation mode is configured to single or continuous-scan edge trigger mode. 0 Do not enter WAITING FOR TRIGGER state after transfer of the current Command Message. 1 Enter WAITING FOR TRIGGER state after transfer of the current Command Message. **Note:** If both the PAUSE and EOQ bits are asserted in the same command message the respective flags are set, but the CFIFO status changes as if only the EOQ bit were asserted.

Table 19-35. Field Descriptions (continued)

Field	Description
REP	Repeat/loop Start Point Indication Bit. The REP bit is asserted in the command to indicate where is the start point of the sub-queue to be repeated when the streaming mode is enabled. The PAUSE bit indicates the end point of the sub-queue. Therefore, both can occur in the same command or in separated ones. If two or more REP bits are read before a PAUSE bit, this is an error case and the intermediary REP bits are ignored. 0 It is not the start point of a loop. 1 Indicates the start point of the sub-queue to be repeated.
EB	External Buffer Bit. A negated EB bit indicates that the command is sent to an internal CBuffer. 0 Command is sent to an internal buffer. 1 Reserved.
BN	Buffer Number Bit. BN indicates which buffer the message will be stored in. 1 Message stored in buffer 1. 0 Message stored in buffer 0.
CAL	Calibration Bit. CAL indicates if the returning conversion result must be calibrated. 1 Calibrate conversion result. 0 Do not calibrate conversion result.
MESSAGE_TAG [0:3]	MESSAGE_TAG Field. The MESSAGE_TAG allows the EQADC to separate returning results into different RFIFOs. When the EQADC transfers a command, the MESSAGE_TAG is included as part of the command. Eventually the on-chip ADC returns the result with the same MESSAGE_TAG. The EQADC separates incoming messages into different RFIFOs by decoding the MESSAGE_TAG of the incoming data. 0000 Result is sent to RFIFO 0 0001 Result is sent to RFIFO 1 0010 Result is sent to RFIFO 2 0011 Result is sent to RFIFO 3 0100 Result is sent to RFIFO 4 0101 Result is sent to RFIFO 5 0110–0111 Reserved 1000 Null Message Received 1001 Reserved for customer use (see note) 1010 Reserved for customer use (see note) 1011–1111 Reserved **Note:** These messages are treated as null messages.
LST[0:1]	Long Sampling Time. These two bits determine the duration of the sampling time in ADC clock cycles. **LST[0:1]** — **Sampling cycles (ADC Clock Cycles)** 0b00 — 2 0b01 — 8 0b10 — 64 0b11 — 128
TSR	Time Stamp Request. TSR indicates the request for a time stamp. When TSR is asserted, the on-chip *ADC Control Logic* returns a time stamp for the current conversion command after the conversion result is sent to the RFIFOs. See Section 19.7.6.3, “Time Stamp Feature,” for details. 0 Return conversion result only. 1 Return conversion time stamp after the conversion result.

Table 19-35. Field Descriptions (continued)

Field	Description
FMT	Conversion Data Format. FMT specifies to the EQADC how to format the 12-bit conversion data returned by the ADCs into the 16-bit format which is sent to the RFIFOs. See Section 19.7.2.2.5, “ADC Result Format for On-Chip ADC Operation,” for details. 0 Right justified unsigned. 1 Right justified signed.
CHANNEL_ NUMBER[0:7]	Channel Number Field. The CHANNEL_NUMBER field selects the analog input channel. The software programs this field with the channel number corresponding to the analog input pin to be sampled and converted. See Section 19.7.7.1, “Channel Assignment,” for details.

19.7.2.2.2 Conversion Command Format for Alternate Configurations

Figure 19-47 describes the format for conversion commands when interfacing with the on-chip ADCs in one of the 14 alternate configurations. An alternate configuration is selected when the least significant byte (bits 24-31) of the conversion command is set to a value in the range 0x08-0x15. Each value in this range selects one of the 14 alternate configuration (0x08 selects Alternate Configuration 1, 0x15 selects Alternate Configuration 14). In the alternate configurations, the conversion result can be routed to one of the RFIFOs or to an on-chip companion module (decimation filter). Bit fields in the corresponding Alternate Configuration Control Register and Extended Alternate Configuration Control Register select the Internal RFIFO or decimation filter as the destination for the conversion result. Time stamp information can be optionally requested.

NOTE

All fields, except FFMT and ALT_CONFIG_SEL, are identical to the ones in the standard configuration format. Only the fields which are different from the standard format will be described here.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
EOQ	PAUSE	REP	RESERVED		EB (0b0)	BN	CAL	MESSAGE_TAG				LST		TSR	FFMT

CFIFO Header

ADC Command

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
CHANNEL_NUMBER								ALT_CONFIG_SEL							

ADC Command

Figure 19-47. Conversion Command Format for Alternate Configurations

Table 19-36. Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>FFMT</td><td>Flush or Format. The function of this bit depends on the DEST field of the Alternate Configuration Control Register. If DEST is equal to 0b000, then FFMT defines the format in which the 12-bit conversion result are stored in the RFIFOs. If DEST is not equal to 0b000, then the FFMT bit is used to send a flush (soft-reset) signal through the parallel side interface to the companion module addressed by the DEST field.
In case DEST is not equal to 0b000, the FMTA bit in the Alternate Configuration Control register is used to define the conversion result format.
0 Conversion Result Format set to right justified unsigned if DEST is equal to 0b000. No flush signal is sent through the side interface if DEST is not equal to 0b000.
1 Conversion Result Format set to right justified signed if DEST is equal to 0b000. A flush signal is sent through the side interface if DEST is not equal to 0b000.
Note: The flush signal can be asserted along with a valid conversion result. In this case the companion module should execute the software-reset first and then consider the conversion result as a valid data for the filtering algorithm.</td></tr>
<tr><td>ALT_CONFIG_SEL</td><td>Alternate Configuration Selection. This field selects one of the alternate configurations.
<table>
<tr><th>ALT_CONFIG_SEL[0:7]</th><th>Alternate Configuration</th></tr>
<tr><td>0x08</td><td>1</td></tr>
<tr><td>0x09</td><td>2</td></tr>
<tr><td>0x0A</td><td>3</td></tr>
<tr><td>0x0B</td><td>4</td></tr>
<tr><td>0x0C</td><td>5</td></tr>
<tr><td>0x0D</td><td>6</td></tr>
<tr><td>0x0E</td><td>7</td></tr>
<tr><td>0x0F</td><td>8</td></tr>
<tr><td>0x10</td><td>9</td></tr>
<tr><td>0x11</td><td>10</td></tr>
<tr><td>0x12</td><td>11</td></tr>
<tr><td>0x13</td><td>12</td></tr>
<tr><td>0x14</td><td>13</td></tr>
<tr><td>0x15</td><td>14</td></tr>
</table>
</td></tr>
</table>

19.7.2.2.3 Write Configuration Command Format for On-Chip ADC Operation

Figure 19-48 describes the command message format for a write configuration command when interfacing with the on-chip ADCs. A write configuration command is used to set the control registers of the on-chip ADCs. No conversion data will be returned for a write configuration command. Write configuration commands are differentiated from read configuration commands by a negated R/W bit.

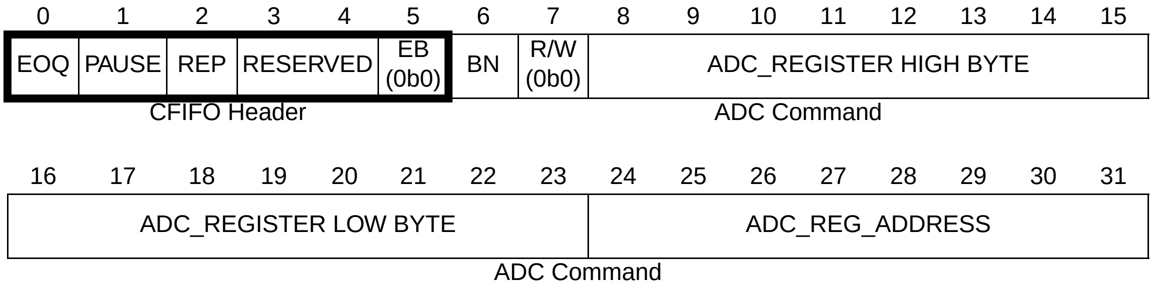


Figure 19-48. Write Configuration Command Format for On-Chip ADC Operation

The register bit descriptions are given in Table 19-37.

Table 19-37. Field Descriptions

Field	Description
EOQ	End Of Queue Bit
PAUSE	Pause Bit
REP	Repeat/loop Start Point Indication Bit
EB	Must be 0b0
BN	Buffer Number Bit. Refer to Section 19.7.2.2.1, "Conversion Command Format for the Standard Configuration."
R/W	Read/Write bit. A negated R/W indicates a write configuration command. 0 Write1Read
ADC_REGISTER_ HIGH_BYTE[0:7]	ADC Register High Byte Field. REGISTER_HIGH_BYTE is the value to be written into the most significant 8 bits of control/configuration register when the R/W bit is negated.
ADC_REGISTER_ LOW_BYTE[0:7]	ADC Register Low Byte Field. REGISTER_LOW_BYTE is the value to be written into the least significant 8 bits of a control/configuration register when the R/W bit is negated.
ADC_REG_ ADDRESS[0:7]	ADC Register Address. The ADC_REG_ADDRESS field selects a register on the ADC register set to be written or read. Only half-word addresses can be used.

19.7.2.2.4 Read Configuration Command Format for On-Chip ADC Operation

Figure 19-49 describes the command message format for a read configuration command when interfacing with the on-chip ADCs. A read configuration command is used to read the contents of the on-chip ADC registers which are only accessible via command messages. Read configuration commands are differentiated from write configuration commands by an asserted R/W bit.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
EOQ	PAUSE	REP	RESERVED		EB (0b0)	BN	R/W (0b1)	MESSAGE_TAG				RESERVED			
CFIFO Header						ADC Command									

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
RESERVED								ADC_REG_ADDRESS							
ADC Command															

Figure 19-49. Read Configuration Command Format for On-Chip ADC Operation

The register bit descriptions are given in Table 19-38.

Table 19-38. Field Descriptions

Field	Description
EOQ	End Of Queue Bit
PAUSE	Pause Bit
REP	Repeat/loop Start Point Indication Bit
EB	Must be 0b0
BN	Buffer Number Bit. Refer to Section 19.7.2.2.1, "Conversion Command Format for the Standard Configuration."
R/W	Read/Write bit. An asserted R/W bit indicates a read configuration command. 0 Write1Read
MESSAGE_ TAG[0:3]	MESSAGE_TAG Field. Refer to Section 19.7.2.2.1, "Conversion Command Format for the Standard Configuration."
ADC_REG_ ADDRESS[0:7]	ADC Register Address. The ADC_REG_ADDRESS field selects a register on the ADC register set to be written or read. Only half-word addresses can be used.

19.7.2.2.5 ADC Result Format for On-Chip ADC Operation

When the *FIFO Control Unit* receives a return data message, it decodes the MESSAGE_TAG field and stores the 16-bit data into the appropriate RFIFO. This section describes the *ADC result* portion of the *result message* returned by the on-chip ADCs. The 16-bit data stored in the RFIFOs can be:

- Data read from an ADC register with a read configuration command. In this case, the stored 16-bit data corresponds to the contents of the ADC register that was read.
- A time stamp. In this case, the stored 16-bit data is the value of the time base counter latched when the EQADC detects the end of the analog input voltage sampling. For details see Section 19.7.6.3, "Time Stamp Feature."

- A conversion result, coming directly from the ADCs. In this case, the stored 16-bit data contains a right justified 14-bit result data. The conversion result can be calibrated or not depending on the status of CAL bit in the command that requested the conversion[1]. When the CAL bit is negated, this 14-bit data is obtained by executing a 2-bit left-shift on the 12-bit data resultant from the resolution adjustment on the 8 or 10 or 12-bit data received from the ADC. The resolution adjustment consists of changing the conversion result input from 8, 10 or 12 bits right aligned to a 12-bit word left aligned - refer to Section 19.7.6.5, "ADC Resolution Selection Feature," for details. When the CAL bit is asserted, this 14-bit data is the result of the calculations performed in the EQADC MAC unit using the 12-bit data result of the resolution adjustment and the calibration constants GCC and OCC, or ALTGCC and ALTOCC - refer to Section 19.7.6.6, "ADC Calibration Feature," for details. Then, this 14-bit data is further formatted into a 16-bit format according to the status of the FMT bit in conversion command of the standard configuration or FFMT bit in the conversion command of the alternate configurations[2]. When FMT/FFMT is asserted, the 14-bit result data is reformatted to look as if it was measured against an imaginary ground at VREF/2 (the MSB bit of the 14-bit result is inverted), and is sign-extended to a 16-bit format as in Figure 19-50. When FMT/FFMT is negated, the EQADC zero-extends the 14-bit result data to a 16-bit format as in Figure 19-51. Correspondence between the analog voltage in a channel and the calculated digital values is shown in Table 19-40.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SIGN_EXT		RESOLUTION ADJUSTED CONVERSION_RESULT (With inverted MSB bit)													

ADC Result

Figure 19-50. ADC Result Format when FMT=1 (Right Justified Signed)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	RESOLUTION ADJUSTED CONVERSION_RESULT													

ADC Result

Figure 19-51. ADC Result Format when FMT=0 (Right Justified Unsigned)

---

[1] In case the conversion result is routed through an on-chip DSP via side interface, the calibration is applied before the data is sent to the DSP.

[2] For simplicity, the following text will refer to FMT only, but when using alternate configurations, refer to Section 19.7.2.2.2, "Conversion Command Format for Alternate Configurations."

Table 19-39. Field Descriptions

Field	Description
SIGN_EXT[0:1]	Sign Extension field. SIGN_EXT only has meaning when FMT is asserted. SIGN_EXT is 0b00 when CONVERSION_RESULT is positive, and 0b11 when CONVERSION_RESULT is negative.
CONVERSION_ RESULT[0:13]	Conversion Result field. CONVERSION_RESULT is a digital value corresponding to the analog input voltage in a channel when the conversion command was initiated. The two's complement representation is used to express negative values.

Table 19-40. Correspondence between analog voltages and digital values[1, 2]

	Voltage Level on Channel (V)	Corresponding 8-bit Conversion Result Returned by the ADC [3]	Corresponding 10-bit Conversion Result Returned by the ADC [4]	Corresponding 12-bit Conversion Result Returned by the ADC [5]	16-bit Result Sent to RFIFOs (FMT=0) [6]	16-bit Result Sent to RFIFOs (FMT=1) [6]
Single-Ended Conversions	5.12	—	—	0xFFF	0x3FFC	0x1FFC
		—	0x3FF	—	0x3FF0	0x1FF0
		0xFF	—	—	0x3FC0	0x1FC0
	5.12 – LSB	—	—	0xFFF	0x3FFC	0x1FFC
		—	0x3FF	—	0x3FF0	0x1FF0
		0xFF	—	—	0x3FC0	0x1FC0
	...	...	...	...	...	...
	2.56	—	—	0x800	0x2000	0x0000
		—	0x200	—	0x2000	0x0000
		0x80	—	—	0x2000	0x0000
	...	...	...	...	...	...
	1 LSB	—	—	0x001	0x0004	0xE004
		—	0x001	—	0x0010	0xE010
		0x01	—	—	0x0040	0xE040
	0	0x00	0x000	0x000	0x0000	0xE000
Differential Conversions	2.56	—	—	0xFFF	0x3FFC	0x1FFC
		—	0x3FF	—	0x3FF0	0x1FF0
		0xFF	—	—	0x3FC0	0x1FC0
	2.56 - LSB	—	—	0xFFF	0x3FFC	0x1FFC
		—	0x3FF	—	0x3FF0	0x1FF0
		0xFF	—	—	0x3FC0	0x1FC0
	...	...	...	...	...	...
	0	—	—	0x800	0x2000	0x0000
		—	0x200	—	0x2000	0x0000
		0x80	—	—	0x2000	0x0000
	...	...	...	...	...	...
	2.56 – LSB	—	—	0x001	0x0004	0xE004
		—	0x001	—	0x0010	0xE010
		0x01	—	—	0x0040	0xE040
	–2.56	0x00	0x000	0x000	0x0000	0xE000

[1] VREF=VRH-VRL=5.12V. Resulting in one 12-bit count (LSB) =1.25mV.

[2] The two's complement representation is used to express negative values.

[3] For non-corner voltages (voltages different from V_{RL} and V_{RH}), the maximum quantization error is ± 0.5 LSB. First code transition (from 0x000 to 0x001 for unsigned values) happens at +0.5 LSB (V) while the last one (from 0xFFE to 0xFFF for unsigned values) happens at V_{REF} - 1.5 LSB (V).

[4] For non-corner voltages (voltages different from V_{RL} and V_{RH}), the maximum quantization error is ± 0.5 LSB. First code transition (from 0x000 to 0x001 for unsigned values) happens at +0.5 LSB (V) while the last one (from 0xFFE to 0xFFF for unsigned values) happens at V_{REF} - 1.5 LSB (V).

[5] For non-corner voltages (voltages different from V_{RL} and V_{RH}), the maximum quantization error is ± 0.5 LSB. First code transition (from 0x000 to 0x001 for unsigned values) happens at +0.5 LSB (V) while the last one (from 0xFFE to 0xFFF for unsigned values) happens at V_{REF} - 1.5 LSB (V).

[6] Assuming uncalibrated conversion results.

19.7.3 Command/Result Queues

Each CQueue entry is a 32-bit Command Message.The last entry of a CQueue has the EOQ bit asserted to indicate that it is the last entry of the CQueue. RQueue entry is a 16-bit data.

See Section 19.7.2.1, “Overview and Basic Terminology,” for a description of the message formats and their flow in EQADC.

Refer to Section 19.8.5, “CQueue and RQueues Usage,” for examples of how CQueues and RQueues can be used.

19.7.4 EQADC Command FIFOs

19.7.4.1 CFIFO Basic Functionality

There are six prioritized CFIFOs located in the EQADC. Each CFIFO is four entries deep, except CFIFO0 that can be configured to eight entries deep in extended mode, and each CFIFO entry is 32 bits long. A CFIFO serves as a temporary storage location for the command messages stored in the CQueues in the system memory. When a CFIFO is not full, the EQADC sets the corresponding CFFF bit in EQADC_FISR. If CFFE is asserted in EQADC_IDCR, the EQADC generates requests for more commands from a CQueue. An interrupt request, served by the host CPU, is generated when CFFS is negated, and a DMA request, served by the DMAC, is generated when CFFS is asserted. The host CPU or the DMAC respond to these requests by writing to the EQADC_CFPR to fill the CFIFO.

NOTE

The DMAC should be configured to write a single command (32-bit data) to the CFIFO push registers for every asserted DMA request it acknowledges. Refer to Section 19.8.2, “EQADC/DMAC Interface,” for DMAC configuration guidelines.

NOTE

CFIFO0 can be configured to work in an alternative way called Streaming Mode. This mode is very different from the mode described here because it maintains some stored commands to execute them several times in sequence and in loop.

NOTE

> Only whole words must be written to EQADC_CFPR. Writing half-words or bytes to EQADC_CFPR will still push the whole 32-bit CF_PUSH field into the corresponding CFIFO, but undefined data will fill the areas of CF_PUSH that were not specifically designated as target locations for writing.

Figure 19-52 describes the important components in the CFIFO. Each CFIFO is implemented as a circular set of registers to avoid the need to move all entries at each push/pop operation. The Push Next Data Pointer points to the next available CFIFO location for storing data written into the EQADC Command FIFO Push Register. The Transfer Next Data Pointer points to the next entry to be removed from CFIFOx when it completes a transfer. The *CFIFO Transfer Counter Control Logic* counts the number of entries in the CFIFO and generates DMA or interrupt requests to fill the CFIFO. TNXTPTR in EQADC_FISR, indicates the index of the entry that is currently being addressed by the Transfer Next Data Pointer, and CFCTR, in the same register, provides the number of entries stored in the CFIFO. Using TNXTPTR and CFCTR, the absolute addresses for the entries indicated by the Transfer Next Data Pointer and by the Push Next Data Pointer can be calculated using the following formulas:

Transfer Next Data Pointer Address = CFIFOx_BASE_ADDRESS + TNXTPTRx*4

Push Next Data Pointer Address = CFIFOx_BASE_ADDRESS +
[(TNXTPTRx+CFCTRx) mod CFIFO_DEPTH] * 4

where

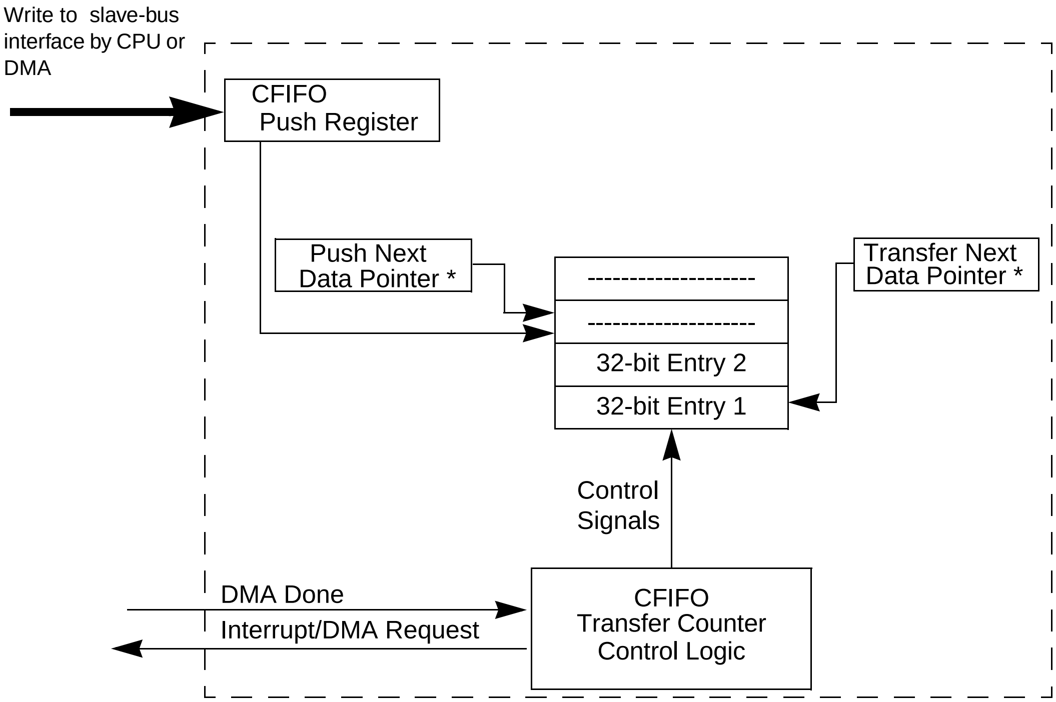
- *a mod b* returns the remainder of the division of *a* by *b*.
- CFIFOx_BASE_ADDRESS is the smallest memory mapped address allocated to a CFIFOx entry.
- CFIFO_DEPTH is the number of entries contained in a CFIFO - four in this implementation.

When CFS*x* in EQADC_CFSR is TRIGGERED, the EQADC generates the proper control signals for the transfer of the entry pointed by Transfer Next Data Pointer. CFUFx in EQADC_FISR is set when a CFIFOx underflow event occurs. A CFIFO underflow occurs when the CFIFO is in TRIGGERED state and it becomes empty. No commands will be transferred from an underflowing CFIFO, nor will command transfers from lower priority CFIFOs be blocked. CFIFOx is empty when the Transfer Next Data Pointer x equals the Push Next Data Pointer x and CFCTRx is zero. CFIFOx is full when the Transfer Next Data Pointer x equals the Push Next Data Pointer x and CFCTRx is not zero.

When the EQADC completes the transfer of an entry from CFIFOx: the transferred entry is popped from CFIFOx, the CFIFO counter CFCTR in the EQADC_FISR is decremented by one, and Transfer Next Data Pointer x is incremented by one (or wrapped around) to point to the next entry in the CFIFO. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the appropriate CBuffer.

When the EQADC_CFPRx is written and CFIFOx is not full, the CFIFO counter CFCTRx is incremented by one, and the Push Next Data Pointer x then is incremented by one (or wrapped around) to point to the next entry in the CFIFO.

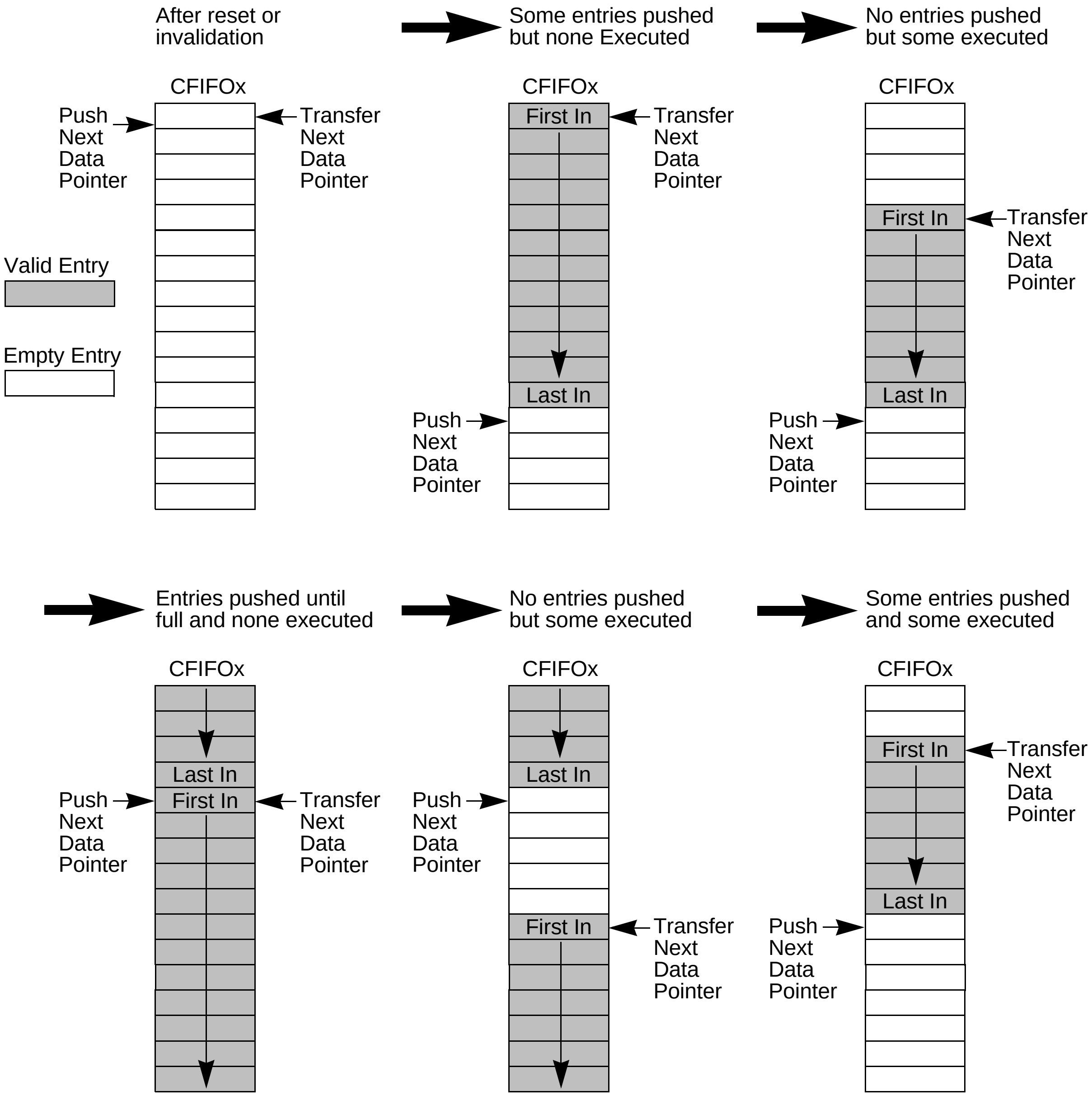
When the EQADC_CFPRx is written but CFIFOx is full, the EQADC will not increment the counter value and will not overwrite any entry in CFIFOx.



* All CFIFO entries are memory mapped and the entries addressed by these pointers can have their absolute addresses calculated using TNXTPTR and CFCTR.

Figure 19-52. CFIFO Diagram

The detailed behavior of the Push Next Data Pointer and Transfer Next Data Pointer is described in the example shown in Figure 19-53 where a CFIFO with 16 entries is shown for clarity of explanation, the actual hardware implementation has only four entries. In this example, CFIFOx with 16 entries is shown in sequence after pushing and transferring entries.



NOTE: x=0, 1, 2, 3, 4, 5

Figure 19-53. CFIFO Entry Pointer Example

19.7.4.2 CFIFO0 Streaming Mode Description

CFIFO0 can be configured to operate in streaming mode to allow repetition of a group of commands several times without the need of refilling the registers as in the normal mode of operation of CFIFOs. This mode makes use of the additional bit in the Conversion Command Word (CCW) called ‘Repeat’ (REP bit). The purpose of this bit is to mark in the command queue, where to start a repeating sequence. This location is stored in an additional pointer ‘Repeat Pointer’.

Streaming mode requires 2 trigger inputs. The standard queue 0 trigger, in this mode referred to as Repeat Trigger and a new internal trigger input to the eQADC called Advance Trigger (no filter available).

CFIFO0 is configured to operate in streaming mode by setting the bit STRME0 as described in Section 19.6.2.5, “EQADC CFIFO Control Registers (EQADC_CFCR).” CFIFO0 is eight entries deep in extended mode by setting the bit CFEEE0 in the same EQADC_CFCR register, and each entry is 32 bits long. This CFIFO0 serves as a local storage of a few commands that need to be executed sequentially as in a FIFO but can contain sub-queues that need to be executed several times. The CFFF0 bit in Section 19.6.2.7, “EQADC FIFO and Interrupt Status Registers (EQADC_FISR),” is used to assure the CFIFO0 is not full and command messages are stored from address 0x0 to 0x7.

19.7.4.2.1 CFIFO0 Operation in Streaming Mode

In Streaming mode, the CFIFO0 is filled with CCWs using the DMA exactly the same as existing modes. The CFIFO executes commands as per the existing modes until it executes a Conversion Command Word with the Repeat bit set. When this CCW is executed, the Repeat Pointer is set to point to this FIFO location and from this CCW onwards, CFIFO0 entries is not invalidated, that is, the Repeat Pointer prevents this and subsequent entries from being overwritten.

The queue continues to execute until a CCW with an asserted Pause bit is completed; then the queue stops and enters the Pause state, waiting for a trigger. This is the same as normal behavior.

The Pause state is exited in one of two ways: Repeat Trigger or Repeat Trigger with Advance Trigger. The Repeat trigger with no Advance trigger causes the Transfer Next Data Pointer to be loaded with the Repeat Pointer location and CCWs are then executed from the Repeat Pointer back to the Pause bit. This means that a section of the CFIFO0 is repeatedly executed every time a Repeat Trigger occurs.

The Repeat trigger with the Advance trigger pending causes all CCWs from the Repeat pointer to the Pause bit to be invalidated and the CCW after the pause bit to be executed. This is achieved by invalidating the Repeat Pointer. The effect is that the queue advances beyond the repeating section of the CFIFO0 to execute new CCWs.

Note that the Advance trigger can occur at any time between Repeat triggers, but is only actioned when the next Repeat trigger occurs. Prior to that it is pending.

In a typical application, the queue is made of some configuration commands to the ADC (to flush the decimator or turn on pad pull-up/down) followed by a repeating section of ADC conversions on one or more ADC channels from one or more sensors; followed by a few more configuration commands; then more repeating ADC conversions, until the entire engine cycle is complete; when the queue is restarted. The mechanism described permits any number of repeating sub-queues to be loaded and executed, interspersed by configuration commands.

19.7.4.2.2 Triggering Description in Streaming Mode

The additional advance-trigger signal ATRIG0 is detected by a separate circuit that is configured by the bit field AMODE0 as described in Section 19.6.2.5, “EQADC CFIFO Control Registers (EQADC_CFCR).” This trigger signal is used as an advance control of pop pointer of CFIFO0. In addition, it is used as the enable trigger for the Repeat trigger. This means it is necessary to have an Advance trigger first to enable the detection of the Repeat trigger. When the Repeat trigger is enabled, the

Advance trigger is used to advance the pop pointer beyond some loop sub-queue. And it is to disable the Repeat trigger by executing a Pause without a previous REP bit.

In Streaming mode, the CFIFO0 is filled with CCWs using the DMA as usual. The two triggers are configured to positive edge and single scan mode.

The SSS bit is asserted and the trigger detector of the Repeat trigger is disabled in the start of the queue. It is necessary to receive the first Advance trigger to enable the detector of the other trigger. This enable is useful when the Repeat trigger is received all the time and the trigger signal can be disabled when it is not desired.

The Advance trigger is received and detected and the Repeat trigger detector is enabled. No commands are executed until now.

The Repeat trigger is detected and the commands start to be executed in sequence. If a REP bit is decoded with the PAUSE bit, the loop is configured and the CFIFO0 commands stop to be executed. The next Repeat trigger is waited to start the execution of the loop again, or the Advance trigger can be detected to break the loop and advance the queue in CFIFO0. The Repeat trigger detector remains enabled.

If the Advance trigger is received and the next command in the CFIFO0 does not present the REP bit set, this means the CFIFO0 is not starting a new loop. In this case (outside a loop) if a PAUSE bit is decoded, this means to disable the Repeat trigger detector. This can be useful if the Repeat trigger is not required for some interval of time. The Repeat trigger detector is enabled again when the next Advance trigger event is detected.

19.7.4.2.3 CFIFO0 Diagram Description in Streaming Mode

Figure 19-54 represents the main components of CFIFO0 in streaming mode. However, some signals behave in a different way from the common operation. The Push Next Data Pointer points to the next available CFIFO0 location for storing data written into the EQADC Command FIFO Push Register. The Transfer Next Data Pointer points to the next entry to be transferred to Cbuffer. The Repeat Pointer points to the first entry of the repeating sub-queue. TNXTPTR in Section 19.6.2.7, “EQADC FIFO and Interrupt Status Registers (EQADC_FISR),” indicates the index of the entry that is currently being addressed by the Transfer Next Data Pointer, and CFCTR, in the same register, provides the number of entries stored in the CFIFO.

When CFS0 in Section 19.6.2.10, “EQADC CFIFO Status Register (EQADC_CFSR),” is TRIGGERED, the EQADC generates the proper control signals for the transfer of the entry pointed by Transfer Next Data Pointer. CFUF0 in Section 19.6.2.7, “EQADC FIFO and Interrupt Status Registers (EQADC_FISR),” is set when CFIFO0 underflow event occurs. A CFIFO underflow occurs when the CFIFO is in TRIGGERED state and it is empty. No commands will be transferred from an underflowing CFIFO, nor will command transfers from lower priority CFIFOs be blocked. CFIFO0 is empty when CFCTR0 is zero. CFIFO0 is full when (CFCTR0 mod CFIFO_DEPTH) is zero but CFCTR0 is not zero.

When the EQADC completes the transfer of an entry from CFIFO0 in loop condition: the transferred entry is not popped from CFIFO0, the CFIFO counter CFCTR in the Section 19.6.2.7, “EQADC FIFO and Interrupt Status Registers (EQADC_FISR),” is not decremented by one, and Transfer Next Data Pointer 0 is incremented by one (or wrapped around) to point to the next entry in the CFIFO0.

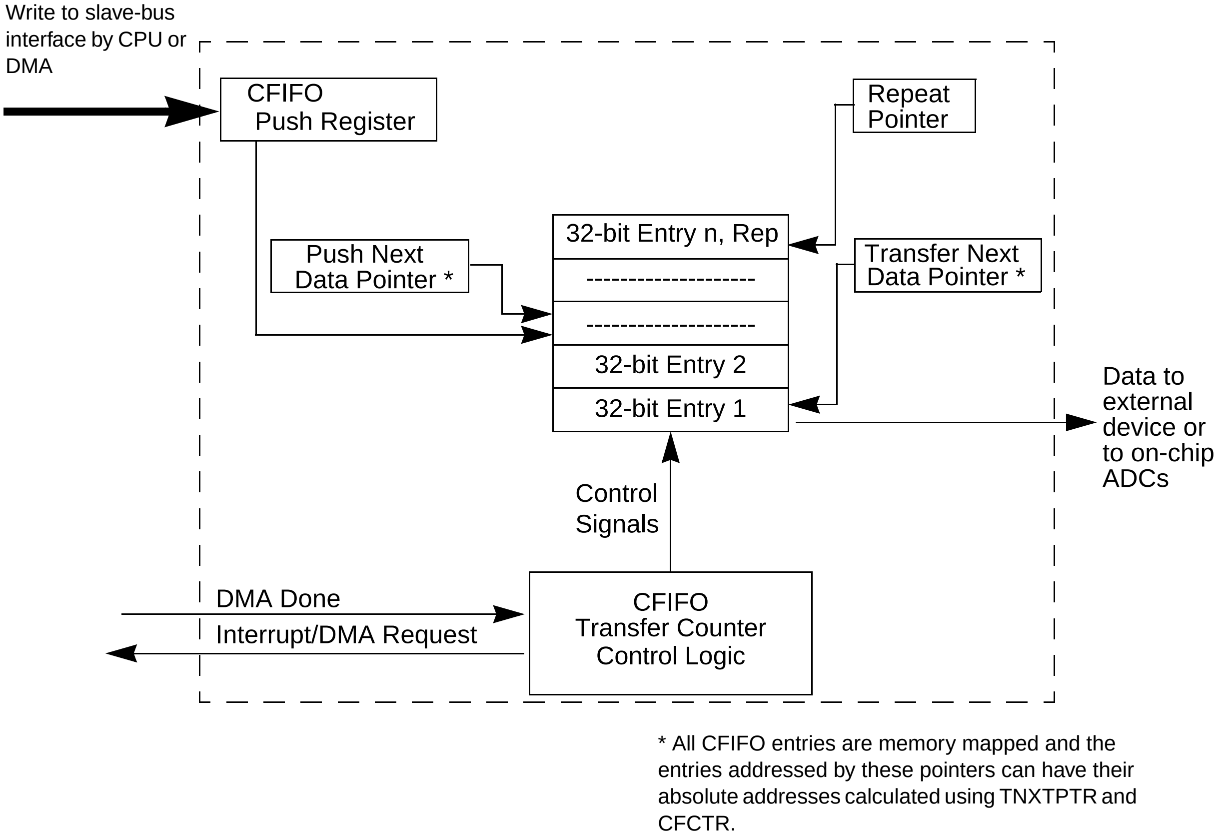


Figure 19-54. CFIFO0 in Streaming Mode Diagram

The detailed behavior of the Push Next Data Pointer and Transfer Next Data Pointer is described in the example shown in Figure 19-55 where a CFIFO with 16 entries is shown for clarity of explanation, the actual hardware implementation has only four/eight entries. In this example, CFIFO0 with 16 entries is shown in sequence after pushing and transferring entries.

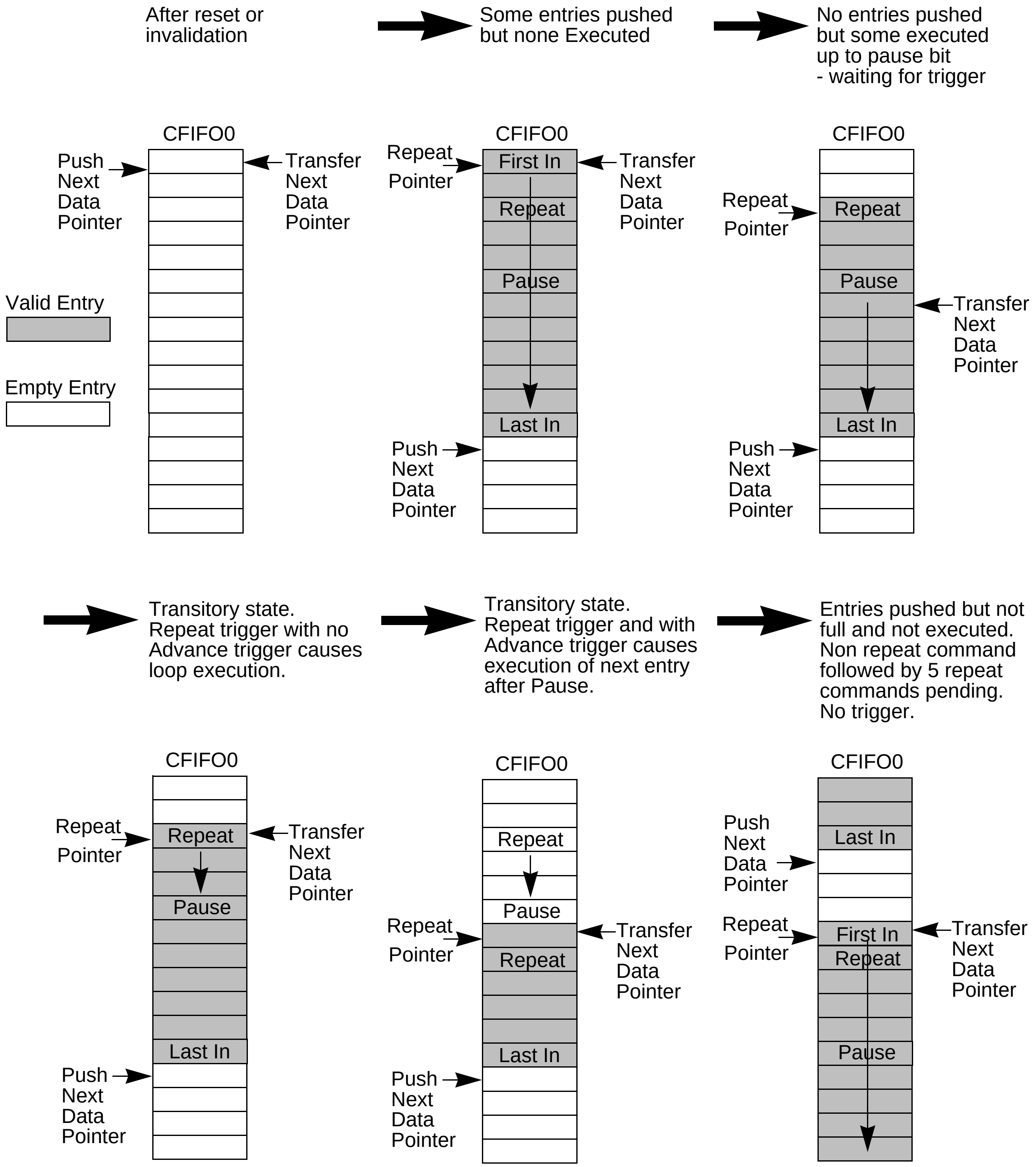


Figure 19-55. CFIFO0 in Streaming Mode Entry Pointer Example

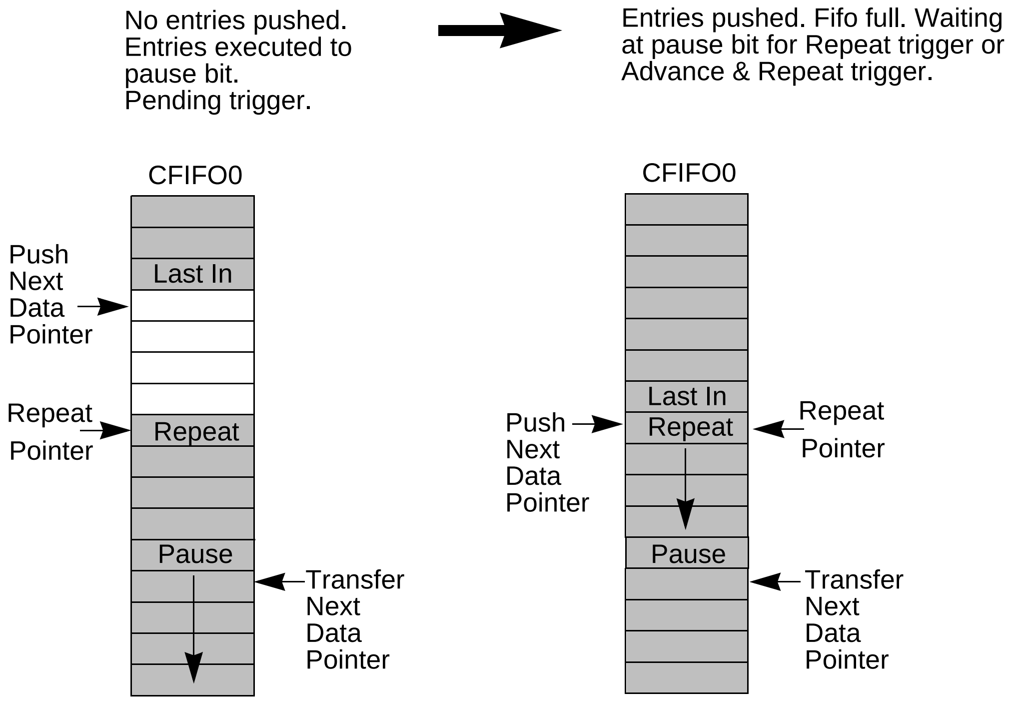


Figure 19-56. CFIFO0 in Streaming Mode Entry Pointer Example (Cont.)

19.7.4.2.4 Streaming Mode Error Conditions

In the repeat state, the existing error conditions still apply, but now there are new ways to trigger them. Now, the CCWs are not being invalidated so the DMA is not able to load more CCWs into those locations. So a queue overflow becomes more likely, and occurs if the repeat loop is longer than 8 entries. If all CCWs in the CFIFO0 are executed and no Pause bit or EOQ bit is detected, the eQADC will signal an underflow error. In practice this may limit a repeating queue to 7 entries since otherwise an underflow will occur at the point a Repeat with Advance trigger occurs, and there is no command in the CFIFO0 to execute. The exception is a final command with both a Pause and an EOQ bit set. The End of Queue bit EOQ continues to operate as in normal mode, unless the Repeat mode is enabled. In this case the Pause bit takes precedence and a Repeat trigger causes the jump back described. A Repeat trigger with Advance trigger causes the queue to end.

Another error condition occur when the repeat trigger is in the TRIGGERED state and a new repeat trigger is received. In this case, a trigger overflow occurs but the CFIFO0 is defined to not restart the loop. The trigger in this case is not used in the CFIFO0, but the overflow is indicated.

19.7.4.3 CFIFO Common Prioritization and Command Transfer

The CFIFO priority is fixed according to the CFIFO number. A CFIFO with a smaller number has a higher priority. When commands of distinct CFIFOs are bound for the same destination (CBuffer), the higher priority CFIFO is always served first. A TRIGGERED, not-underflowing CFIFO will start the transfer of its commands when:

- its commands are bound for an internal CBuffer that is not full, and it is the highest priority triggered CFIFO sending commands to that CBuffer.

A triggered CFIFO with commands bound for a certain CBuffer consecutively transfers its commands to it until:

- an asserted End Of Queue bit is reached, or;
- an asserted Pause bit is encountered and the CFIFO is configured for edge trigger mode, or;
- CFIFO is configured for level trigger mode and a closed gate is detected, or;
- in case its commands are bound for an internal CBuffer, a higher priority CFIFO that uses the same internal CBuffer is triggered, or;

The prioritization logic of the EQADC, depicted in Figure 19-57, is composed of two independent sub-blocks: one prioritizing CFIFOs with commands bound for CBuffer0 and another prioritizing CFIFOs with commands for CBuffer1. As these sub-blocks are independent, simultaneous writes to CBuffer0 and CBuffer1. The hardware identifies the destination of a command by decoding the BN bit in the command message - see Section 19.7.2.2, “Message Format in EQADC,” for details.

NOTE

Triggered but empty CFIFOs, underflowing CFIFOs, are not considered for prioritization. No data from these CFIFOs will be sent to the CBuffers and nor will they stop lower priority CFIFOs from transferring commands.

Whenever CBuffer0 is able to receive new entries, the prioritization sub-block selects the highest-priority triggered CFIFO with a command bound for CBuffer0, and writes its command into the buffer. In case CBuffer0 is able to receive new entries but there are no triggered CFIFOs with commands bound for it, nothing is written to the buffer. The sub-block prioritizing CBuffer1 usage behaves in the same way.

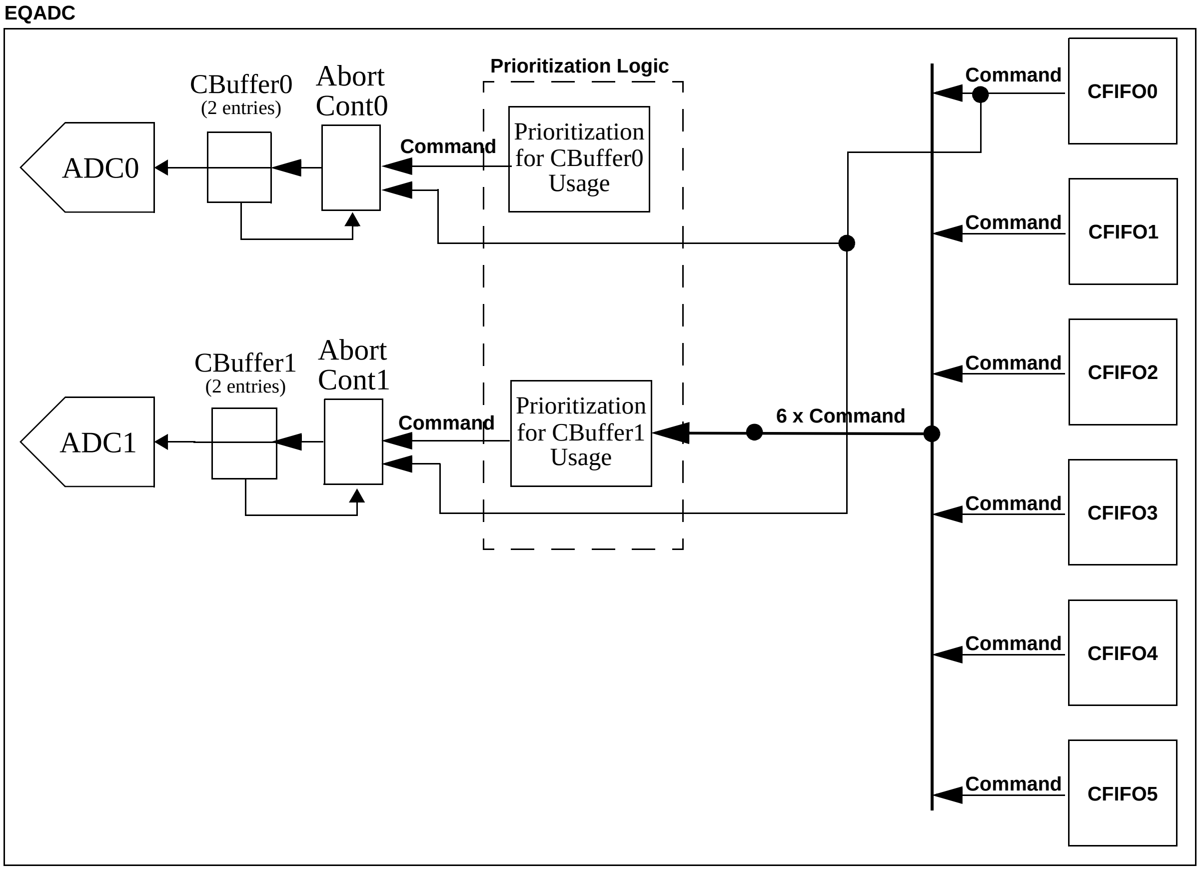


Figure 19-57. CFIFO Prioritization Logic

19.7.4.4 CFIFO Prioritization in Abort Mode

The CFIFO priority does not change when the EQADC is configured to allow abortion of conversion execution in on-chip ADC analog blocks. However, CFIFO0 is the only one that can be enabled to abort conversions.

This feature is necessary when the timing of some conversion is very important. In normal priority scheme, when CFIFO0 is triggered, its conversion command can be put behind 2 pending conversion commands in the Cbuffer due to the queue structure. Considering that these 2 pending commands are from lower priority CFIFOs and that the delay between the trigger and the sampling of the command from Cqueue0 can be unacceptable, EQADC can be configured to permit immediate conversion commands from CFIFO0 with abort function.

When CFIFO0 is triggered and abort is enabled, up to 2 commands in Cbuffer0 or Cbuffer1 are stored in a side register. The abort request signal is generated to ADC0 or ADC1 and the confirmation of ADC reset/ready is waited to send the command from CFIFO0 to the decoded Cbuffer.

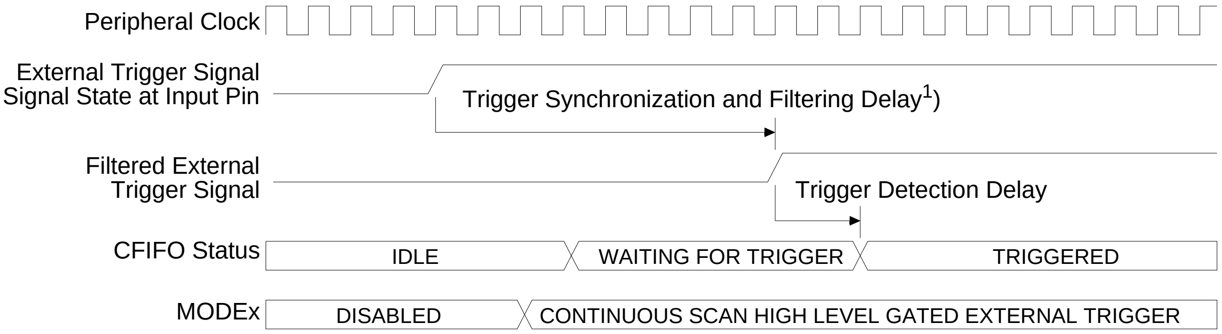
After the transfer of all commands from CFIFO0, the recovery phase restores the up to 2 commands that were in Cbuffer when the abort occurred. After this recovery phase, it is established the normal process of prioritization of commands from CFIFOs.

19.7.4.5 Hardware Trigger Event Detection

On this device, the on-chip triggers bypass the filter and the off-chip ETRIG triggers are always filtered and subject to the minimum filter length of 2 clocks. When the filter is bypassed, the ETRIG input signal is not filtered and the logic after the filter receives a copy of this input trigger signal.

The Digital Filter Length field in Section 19.6.2.2, "EQADC External Trigger Digital Filter Register (EQADC_ETDFR)," specifies the minimum number of peripheral clocks that the ETRIG0-5 signals must be held at a logic level to be recognized as valid. All ETRIG signals are filtered. A counter for each queue trigger is implemented to detect a transition between logic levels. The counter counts at the peripheral clock rate. The corresponding counter is cleared and restarted each time the signal transitions between logic levels. When the corresponding counter matches the value specified by the Digital Filter Length field in Section 19.6.2.2, "EQADC External Trigger Digital Filter Register (EQADC_ETDFR)," the EQADC considers the ETRIG logic level to be valid and passes that new logic level to the rest of the EQADC.

The filter is only for filtering the ETRIG signal. Logic after the filter checks for transitions between filtered values, such as for detecting the transition from a filtered logic level zero to a filter logic level one in rising edge external trigger mode. The EQADC can detect rising edge, falling edge, or level gated external triggers. The digital filter will always be active independently of the status of the MODE*x* field in Section 19.6.2.5, "EQADC CFIFO Control Registers (EQADC_CFCR)," but the edge, level detection logic is only active when MODEx is set to a value different from disabled, and in case MODEx is set to single scan mode, when the SSS bit is asserted. Note that the time necessary for a external trigger event to result into a CFIFO status change is not solely determined by the DFL field in the Section 19.6.2.2, "EQADC External Trigger Digital Filter Register (EQADC_ETDFR)." After being synchronized to the peripheral clock and filtered, a trigger event is checked against the CFIFO trigger mode. Only then, after a valid trigger event is detected, the EQADC accordingly changes the CFIFO status. Refer to Figure 19-58 for an example.



Notes:

1. This delay is about 2 clocks when the filter bypass control is asserted.

Figure 19-58. ETRIG Event Propagation Example

19.7.4.6 CFIFO Scan Trigger Modes

The EQADC supports two different scan modes, single-scan and continuous-scan. Refer to Table 19-41 for a summary of these two scan modes. When a CFIFO is triggered, the EQADC scan mode determines whether the EQADC will stop command transfers from a CFIFO, and wait for software intervention to rearm the CFIFO to detect new trigger events, upon detection of an asserted EOQ bit in the last transfer. Refer to Section 19.7.2.2, “Message Format in EQADC,” for details about command formats.

CFIFOs can be configured in single-scan or continuous-scan mode. When a CFIFO is configured in single-scan mode, the EQADC scans the CQueue one time. The EQADC stops future command transfers from the triggered CFIFO after detecting the EOQ bit set in the last transfer. After a EOQ bit is detected, software involvement is required to rearm the CFIFO so that it can detect new trigger events.

When a CFIFO is configured for continuous-scan mode, no software involvement is necessary to rearm the CFIFO to detect new trigger events after an asserted EOQ is detected. In continuous-scan mode the whole CQueue is scanned multiple times.

The EQADC also supports different triggering mechanisms for each scan mode. The EQADC will not transfer commands from a CFIFO until the CFIFO is triggered. The combination of scan modes and triggering mechanisms allows the support of different requirements for scanning input channels. The scan mode and trigger mechanism are configured by programming the MODEx field in Section 19.6.2.5, “EQADC CFIFO Control Registers (EQADC_CFCR).”

Enabled CFIFOs can be triggered by software or external trigger events. The elapsed time from detecting a trigger to transferring a command is a function of clock frequency, trigger synchronization, trigger filtering or not, programmable trigger events, command transfer, CFIFO prioritization, CBuffer availability, etc. Fast and predictable transfers can be achieved by ensuring that the CFIFO is not underflowing and that the target CBuffer is not full when the CFIFO is triggered.

19.7.4.6.1 Disabled Mode

The MODE*x* field in Section 19.6.2.5, “EQADC CFIFO Control Registers (EQADC_CFCR),” for all of the CFIFOs can be changed from any other mode to disabled at any time. No trigger event can initiate command transfers from an CFIFO which has its MODE field programmed to disabled.

NOTE

> If MODEx is not disabled, it must not be changed to any other mode besides disabled. If MODEx is disabled and the CFIFO status is IDLE, MODEx can be changed to any other mode.

If MODE*x* is changed to disabled:

- The CFIFO execution status will change to IDLE. The timing of this change depends on whether a command is being transferred or not:
  - When no command transfer is in progress, the EQADC switches the CFIFO to IDLE status immediately.
  - When a command transfer to an on-chip CBuffer is in progress, the EQADC will complete the transfer, update TC_CF, and switch CFIFO status to IDLE. Command transfers to the internal CBuffers are considered completed when a command is written to the buffers.
- The CFIFOs are not invalidated automatically. The CFIFO still can be invalidated by writing a “1” to the CFINVx bit in Section 19.6.2.5, “EQADC CFIFO Control Registers (EQADC_CFCR).” Certify that CFS has changed to IDLE before setting CFINVx.
- The TC_CFx value also is not reset automatically, but it can be reset by writing “0” to it.

- The SSS bit in Section 19.6.2.7, “EQADC FIFO and Interrupt Status Registers (EQADC_FISR),” is negated. The SSS bit can be set even if a “1” is written to the SSE bit in Section 19.6.2.5, “EQADC CFIFO Control Registers (EQADC_CFCR),” in the same write that the MODEx field is changed to a value other than disabled.
- The trigger detection hardware is reset. If MODEx is changed from disabled to an edge trigger mode, a new edge, matching that edge trigger mode, is needed to trigger the command transfers from the CFIFO.

NOTE

CFIFO fill requests, which generated when CFFF is asserted, are not automatically halted when MODEx is changed to disabled. CFIFO fill requests will still be generated until CFFE is cleared in Section 19.6.2.6, “EQADC Interrupt and DMA Control Registers (EQADC_IDCR).”

19.7.4.6.2 Single-Scan Mode

In single-scan mode, a single pass through a sequence of command messages in a CQueue is performed.

In single-scan software trigger mode, the CFIFO is triggered by an asserted Single-Scan Status bit (SSS) in Section 19.6.2.7, “EQADC FIFO and Interrupt Status Registers (EQADC_FISR).” The SSS bit is set by writing “1” to the Single-Scan Enable bit (SSE) in Section 19.6.2.5, “EQADC CFIFO Control Registers (EQADC_CFCR).”

In single-scan edge- or level-trigger mode, the respective triggers are only detected when the SSS bit is asserted. When the SSS bit is negated, all trigger events for that CFIFO are ignored. Writing a “1” to the SSE bit can be done during the same write cycle that the CFIFO operation mode is configured.

Only the EQADC can clear the SSS bit. Once SSS is asserted, it remains asserted until the EQADC completes the CQueue scan, or the CFIFO operation mode (MODEx) in Section 19.6.2.5, “EQADC CFIFO Control Registers (EQADC_CFCR),” is changed to disabled. The SSSx bit will be negated while MODE*x* is disabled.

Single-Scan Software Trigger

When single-scan software trigger mode is selected, the CFIFO is triggered by an asserted SSS bit. The SSS bit is asserted by writing “1” to the SSE bit. Writing to SSE while SSS is already asserted will not have any effect on the state of the SSS bit, nor will it cause a trigger overrun event.

The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using a not-full on-chip CBuffer. When an asserted EOQ bit is encountered, the EQADC will clear the SSS bit. Setting the SSS bit is required for the EQADC to start the next scan of the queue.

The Pause bit has no effect in single-scan software trigger mode.

Single-Scan Edge Trigger

When SSS is asserted and an edge triggered mode is selected for a CFIFO, an appropriate edge on the associated trigger signal causes the CFIFO to become TRIGGERED. For example, if rising-edge trigger mode is selected, the CFIFO becomes TRIGGERED when a rising edge is sensed on the trigger signal.

The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using a not-full on-chip CBuffer or an not-full external CBuffer.

When an asserted EOQ bit is encountered, the EQADC clears SSS and stops command transfers from the CFIFO. An asserted SSS bit and a subsequent edge trigger event are required to start the next scan for the CFIFO. When an asserted Pause bit is encountered, the EQADC stops command transfers from the CFIFO, but SSS remains set. Another edge trigger event is required for command transfers to continue. A trigger overrun happens when the CFIFO is in TRIGGERED state and an edge trigger event is detected.

Single-Scan Level Trigger

When SSS is asserted and a level gated trigger mode is selected, the input level on the associated trigger signal puts the CFIFO in TRIGGERED state. When the CFIFO is asserted to high-level gated trigger, a high level signal opens the gate, and a low level closes the gate. When the CFIFO is set to low-level gated trigger mode, a low level signal opens the gate, and a high level closes the gate. If the corresponding level is already present, setting the SSS bit triggers the CFIFO. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using a not-full on-chip CBuffer or a not -full external CBuffer.

The EQADC clears the SSS bit and stops transferring commands from a TRIGGERED CFIFO when an asserted EOQ bit is encountered or when CFIFO status changes from TRIGGERED due to the detection of a closed gate. Command transfers will restart from the point they have stopped.

The Pause bit has no effect in single-scan level-trigger mode.

19.7.4.6.3 Continuous-Scan Mode

In continuous-scan mode, multiple passes looping through a sequence of command messages in a CQueue are executed. When a CFIFO is programmed for a continuous-scan mode, the SSE bit in the Section 19.6.2.5, “EQADC CFIFO Control Registers (EQADC_CFCR),” does not have any effect.

Continuous-Scan Software Trigger

When a CFIFO is programmed to continuous-scan software trigger mode, the CFIFO is triggered immediately. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using a not-full on-chip CBuffer. When a CFIFO is programmed to run in continuous-scan software trigger mode, the EQADC will not halt transfers from the CFIFO until the CFIFO operation mode is modified to disabled or a higher priority CFIFO preempts it. Although command transfers will not stop upon detection of an asserted EOQ bit, the EOQF is set and, if enabled, an EOQ interrupt request is generated.

The Pause bit has no effect in continuous-scan software trigger mode.

Continuous-Scan Edge Trigger

When rising, falling, or either edge trigger mode is selected for a CFIFO, a corresponding edge on the associated ETRIG signal places the CFIFO in TRIGGERED state. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using a not-full on-chip CBuffer.

When an EOQ or a Pause is encountered, the EQADC halts command transfers from the CFIFO and, if enabled, the appropriate interrupt requests are generated. Another edge trigger event is required to resume command transfers but no software involvement is required to rearm the CFIFO in order to detect such event.

A trigger overrun happens when the CFIFO is already in TRIGGERED state and a new edge trigger event is detected.

Continuous-Scan Level Trigger

When high or low level gated trigger mode is selected, the input level on the associated trigger signal places the CFIFO in TRIGGERED state. When high-level gated trigger is selected, a high-level signal opens the gate, and a low level closes the gate. The CFIFO commands start to be transferred when the CFIFO becomes the highest priority CFIFO using a not-full on-chip CBuffer. Although command transfers will not stop upon detection of an asserted EOQ bit at the end of a command transfer, the EOQF is asserted and, if enabled, an EOQ interrupt request is generated.

The EQADC stops transferring commands from a TRIGGERED CFIFO when CFIFO status changes from TRIGGERED due to the detection of a closed gate. If a closed gate is detected while no command transfers are taking place and the CFIFO status is TRIGGERED, the CFIFO status is immediately changed to WAITING FOR TRIGGER and the PF flag is asserted.Command transfers will restart as the gate opens.

The Pause bit has no effect in continuous-scan level-trigger mode.

19.7.4.6.4 CFIFO Scan Trigger Mode Start/Stop Summary

Table 19-41 summarizes the start and stop conditions of command transfers from CFIFOs for all of the single-scan and continuous-scan trigger modes.

Table 19-41. CFIFO Scan Trigger Mode - Command Transfer Start/Stop Summary

Trigger Mode	Requires Asserted SSS to Recognize Trigger Events?	Command Transfer Start/Restart Condition	Stop on asserted EOQ bit[1]?	Stop on asserted Pause bit[2]?	Other Command Transfer Stop Condition[3] [4]
Single Scan Software	Don’t Care	Asserted SSS bit.	Yes	No	None.
Single Scan Edge	Yes	A corresponding edge occurs.	Yes	Yes	None.
Single Scan Level	Yes	Gate is opened.	Yes	No	EQADC also stops transfers from the CFIFO when CFIFO status changes from TRIGGERED due to the detection of a closed gate.[5]
Continuous Scan Software	No	CFIFO starts automatically after being configured into this mode.	No	No	None.
Continuous Scan Edge	No	A corresponding edge occurs.	Yes	Yes	None.

Table 19-41. CFIFO Scan Trigger Mode - Command Transfer Start/Stop Summary (continued)

Trigger Mode	Requires Asserted SSS to Recognize Trigger Events?	Command Transfer Start/Restart Condition	Stop on asserted EOQ bit[1]?	Stop on asserted Pause bit[2]?	Other Command Transfer Stop Condition[3] [4]
Continuous Scan Level	No	Gate is opened.	No	No	EQADC also stops transfers from the CFIFO when CFIFO status changes from TRIGGERED due to the detection of a closed gate.[5]

---

[1] Refer to Section 19.7.4.7.2, “CQueue Completion Status,” for more information on EOQ.

[2] Refer to Section 19.7.4.7.3, “Pause Status,” for more information on Pause.

[3] EQADC always stops command transfers from a CFIFO when the CFIFO operation mode is disabled.

[4] EQADC always stops command transfers from a CFIFO when a higher priority CFIFO is triggered. Refer to Section 19.7.4.3, “CFIFO Common Prioritization and Command Transfer,” for information on CFIFO priority.

[5] If a closed gate is detected while no command transfers are taking place, it will have immediate effect on the CFIFO status. If a closed gate is detected during the serial transmission of a command to the external device, it will have no effect on the CFIFO status until the transmission completes.

19.7.4.7 CFIFO and Trigger Status

19.7.4.7.1 CFIFO Operation Status

Each CFIFO has its own CFIFO status field. CFIFO status (CFS) can be read from Section 19.6.2.10, “EQADC CFIFO Status Register (EQADC_CFSR).” Figure 19-59 and Table 19-42 indicate the CFIFO status switching condition. Refer to Figure 19-17 for the meaning of each CFIFO operation status. The last CFIFO to transfer a command to an on-chip CBuffer can be read from the LCFTCB*n* (n=0,1) fields in the Section 19.6.2.9, “EQADC CFIFO Status Snapshot Registers (EQADC_CFSSR).” The last CFIFO to transfer a command to a specific external CBuffer can be identified by reading the LCFTSSI and ECBNI fields in the Section 19.6.2.9, “EQADC CFIFO Status Snapshot Registers (EQADC_CFSSR).”

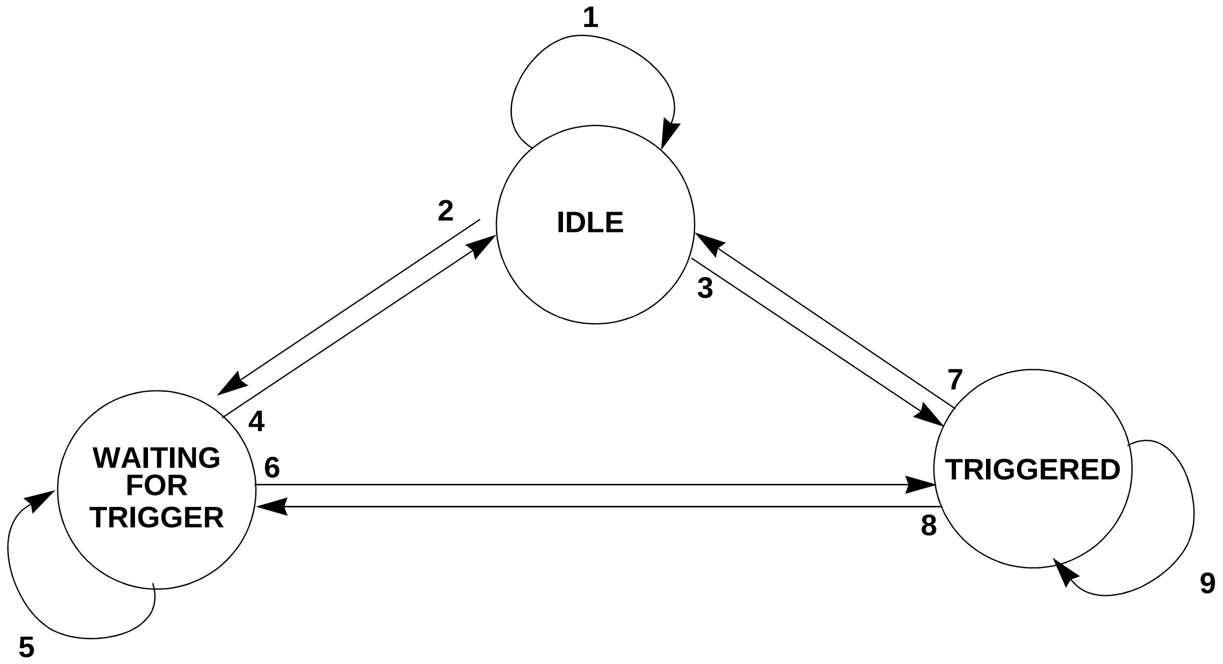


Figure 19-59. State Machine of CFIFO Status

Table 19-42. Command FIFO Status Switching Condition

No.	From Current CFIFO Status (CFS)	To New CFIFO Status (CFS)	Status Switching Condition
1	IDLE (00)	IDLE (0b00)	— CFIFO Mode is programmed to disabled, OR — CFIFO Mode is programmed to single-scan edge or level trigger mode and SSS is negated.
2		WAITING FOR TRIGGER (0b10)	— CFIFO Mode is programmed to continuous-scan edge or level trigger mode, OR — CFIFO Mode is programmed to single-scan edge or level trigger mode and SSS is asserted, OR — CFIFO Mode is programmed to single-scan software trigger mode.
3		TRIGGERED (0b11)	— CFIFO Mode is programmed to continuous-scan software trigger mode
4	WAITING FOR TRIGGER (10)	IDLE (0b00)	— CFIFO Mode is modified to disabled mode.
5		WAITING FOR TRIGGER (0b10)	— No trigger occurred.
6		TRIGGERED (0b11)	— Appropriate edge or level trigger occurred, OR — CFIFO Mode is programmed to single-scan software trigger mode and SSS bit is asserted.

Table 19-42. Command FIFO Status Switching Condition (continued)

No.	From Current CFIFO Status (CFS)	To New CFIFO Status (CFS)	Status Switching Condition
7	TRIGGERED (11)	IDLE (0b00)	— CFIFO in single-scan mode, EQADC detects the EOQ bit asserted at end of command transfer, and CFIFO Mode is not modified to disabled.OR — CFIFO, in single-scan level trigger mode, and the gate closes while no commands are being transferred from the CFIFO, and CFIFO Mode is not modified to disabled. OR — CFIFO, in single-scan level trigger mode, and EQADC detects a closed gated at end of command transfer, and CFIFO Mode is not modified to disabled. OR — CFIFO Mode is modified to disabled mode and CFIFO was not transferring commands. —CFIFO Mode is modified to disabled mode while CFIFO was transferring commands, and CFIFO completes or aborts the transfer.
8		WAITING FOR TRIGGER (0b10)	— CFIFO in single or continuous-scan edge trigger mode, EQADC detects the Pause bit asserted at the end of command transfer, the EOQ bit in the same command is negated, and CFIFO Mode is not modified to disabled, OR — CFIFO in continuous-scan edge trigger mode, EQADC detects the EOQ bit asserted at the end of command transfer, and CFIFO Mode is not modified to disabled, OR — CFIFO, in continuous-scan level trigger mode, and the gate closes while no commands are being transferred from the CFIFO, and CFIFO Mode is not modified to disabled, OR — CFIFO, in continuous-scan level trigger mode, and EQADC detects a closed gated at end of command transfer, and CFIFO Mode is not modified to disabled.
9		TRIGGERED (0b11)	— No event to switch to IDLE or WAITING FOR TRIGGER status has happened.

19.7.4.7.2 CQueue Completion Status

The End of Queue Flag (EOQF) in Section 19.6.2.7, “EQADC FIFO and Interrupt Status Registers (EQADC_FISR),” is asserted when the EQADC completes the transfer of a CFIFO entry with an asserted EOQ bit. Software sets the EOQ bit in the last Command Message of a CQueue to indicate that this entry is the end of the CQueue - see Section 19.7.2.2, “Message Format in EQADC,” for information on command message formats. The transfer of entries bound for the on-chip ADCs is considered completed when they are stored in the appropriate CBuffer.

The command with a EOQ bit asserted is valid and will be transferred. When EOQIE in Section 19.6.2.5, “EQADC CFIFO Control Registers (EQADC_CFCR),” and EOQF are asserted, the EQADC will generate an End of Queue interrupt request.

In single-scan modes, command transfers from the corresponding CFIFO will cease when EQADC completes the transfer of a entry with an asserted EOQ. Software involvement is required to rearm the CFIFO so that it can detect new trigger events.

NOTE

An asserted EOQFx only implies that EQADC has finished transferring a command with an asserted EOQ bit from CFIFOx. It does not imply that result data for the current command and for all previously transferred commands has been returned to the appropriate RFIFO.

19.7.4.7.3 Pause Status

In edge trigger mode, when the EQADC completes the transfer of a CFIFO entry with an asserted Pause bit, the EQADC will stop future command transfers from the CFIFO and set the corresponding Pause Flag (PF) in Section 19.6.2.7, "EQADC FIFO and Interrupt Status Registers (EQADC_FISR)." Refer to Section 19.7.2.2, "Message Format in EQADC," for information on command message formats. The EQADC ignores the Pause bit in command messages in any software and external level trigger mode. The EQADC sets the PF flag upon detection of an asserted Pause bit only in single or continuous-scan edge trigger mode. When the PF flag is set for a CFIFO in single-scan edge trigger mode, the SSS bit will not be cleared in Section 19.6.2.7, "EQADC FIFO and Interrupt Status Registers (EQADC_FISR)."

In level trigger mode, the definition of the PF flag has been redefined. In level trigger mode, when CFIFOx is in TRIGGERED status, PFx is set when CFIFO status changes from TRIGGERED due to detection of a closed gate. The pause flag interrupt routine can be used to verify if the a complete scan of the CQueue was performed. If a closed gate is detected while no command transfers are taking place, it will have immediate effect on the CFIFO status.

When PIE in Section 19.6.2.5, "EQADC CFIFO Control Registers (EQADC_CFCR)," and PF are asserted, the EQADC will generate a Pause interrupt request.

NOTE

In edge trigger mode, an asserted PFx only implies that the EQADC finished transferring a command with an asserted PAUSE bit from CFIFOx. It does not imply that result data for the current command and for all previously transferred commands has been returned to the appropriate RFIFO.

NOTE

In software or level trigger mode, when the EQADC completes the transfer of an entry from CFIFOx with an asserted Pause bit, PFx will not be set and command transfers will continues without pausing.

19.7.4.7.4 Trigger Overrun Status

NOTEWhen a CFIFO is configured for edge- or level-trigger mode and is in TRIGGERED state, an additional trigger occurring for the same CFIFO results in a trigger overrun. The trigger overrun bit for the corresponding CFIFO will be set (TORF*x* = 1) in Section 19.6.2.7, "EQADC FIFO and Interrupt Status Registers (EQADC_FISR)." When TORIE in Section 19.6.2.5, "EQADC CFIFO Control Registers (EQADC_CFCR)," and TORF are asserted, the EQADC generates a trigger overrun interrupt request.

For CFIFOs configured for level-trigger mode, a trigger overrun does not occur.

NOTE

The trigger overrun flag will not set for CFIFOs configured for software trigger mode.

19.7.4.7.5 Command Sequence Non-Coherency Detection

The EQADC provides a mechanism to indicate if a command sequence has been completely executed without interruptions. A command sequence is defined as a group of consecutive commands bound for the same CBuffer and it is expected to be executed without interruptions. A command sequence is coherent if its commands are executed in order without interruptions. Since commands are stored in the CBuffers before being executed in the EQADC, a command sequence is coherent if, while it is transferring commands to a CBuffer, the buffer is only fed with commands from that sequence without ever becoming empty.

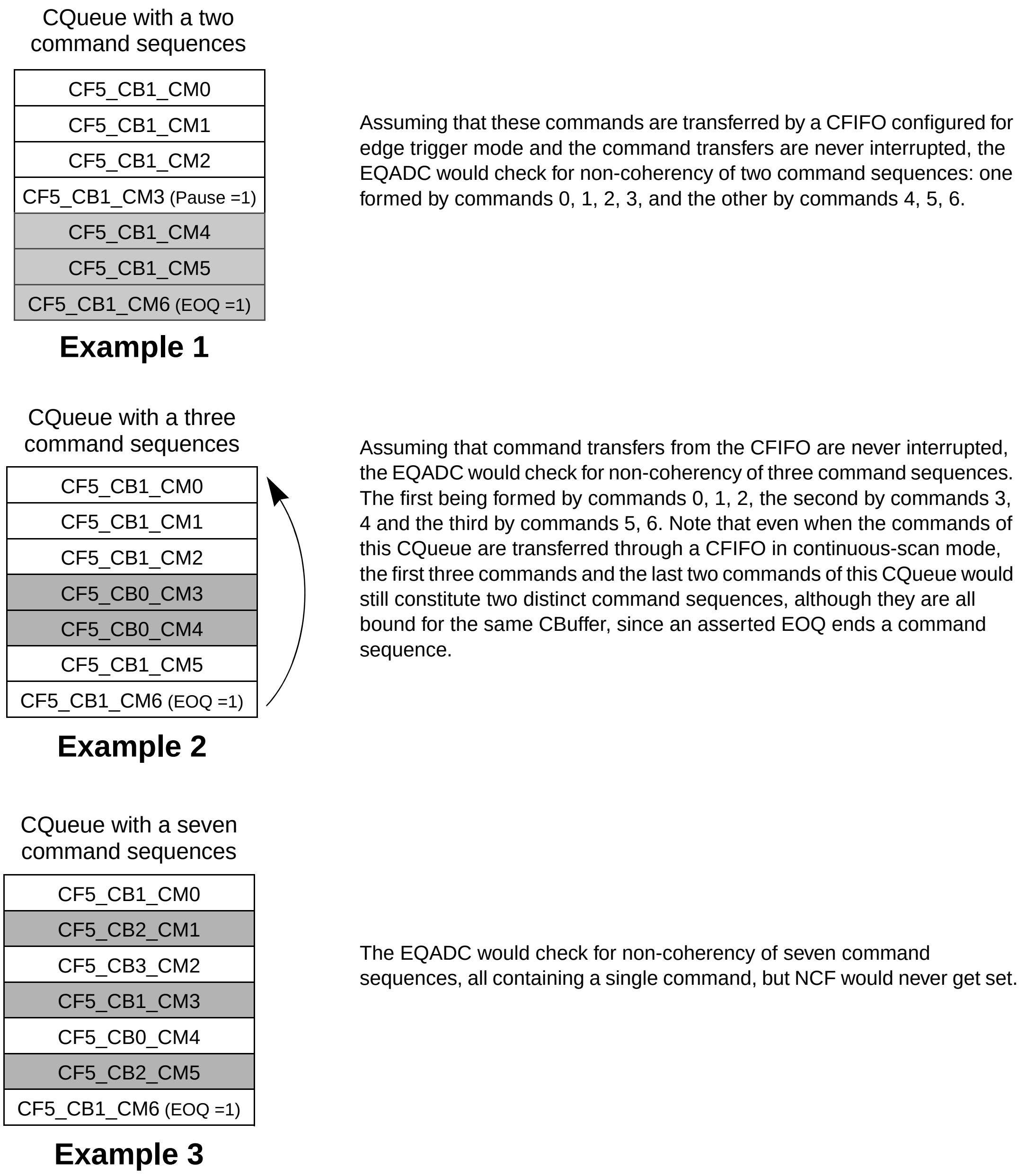
A command sequence starts when:

- a CFIFO in TRIGGERED state transfers its first command to CBuffer.
- the CFIFO is constantly transferring commands and the previous command sequence ended.
- the CFIFO resumes command transfers after being interrupted.

And a command sequence ends when:

- an asserted EOQ bit is detected on the last transferred command.
- CFIFO is in edge-trigger mode and asserted PAUSE bit is detected on the last transferred command.
- the CBuffer to which the next command is bound is different from the one to which the last command was transferred.

Figure 19-60 shows examples of how the EQADC would detect command sequences when transferring commands from a CFIFO to a CBuffer. The smallest possible command sequence can have a single command as shown in example 3 of Figure 19-60.



Assuming that these commands are transferred by a CFIFO configured for edge trigger mode and the command transfers are never interrupted, the EQADC would check for non-coherency of two command sequences: one formed by commands 0, 1, 2, 3, and the other by commands 4, 5, 6.

Assuming that command transfers from the CFIFO are never interrupted, the EQADC would check for non-coherency of three command sequences. The first being formed by commands 0, 1, 2, the second by commands 3, 4 and the third by commands 5, 6. Note that even when the commands of this CQueue are transferred through a CFIFO in continuous-scan mode, the first three commands and the last two commands of this CQueue would still constitute two distinct command sequences, although they are all bound for the same CBuffer, since an asserted EOQ ends a command sequence.

The EQADC would check for non-coherency of seven command sequences, all containing a single command, but NCF would never get set.

CF*x*_CB*a*_CM*n* - Command *n* in CFIFO*x* bound for CBuffer*a*

Figure 19-60. Command Sequence Examples

The NCF flag is used to indicate command sequence non-coherency. When the NCFx flag is asserted, it indicates that the command sequence being transferred through CFIFOx became non-coherent. The NCF flag only becomes asserted for CFIFOs in TRIGGERED state.

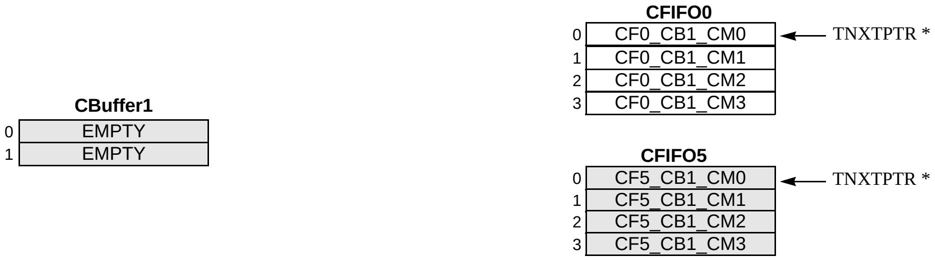
A command sequence is non-coherent when, after transferring the first command of a sequence from a CFIFO to a CBuffer, it cannot successively send all the other commands of the sequence before any of the following conditions are true:

- The CFIFO through which commands are being transferred is preempted by a higher priority CFIFO which sends commands to the same CBuffer. The NCF flag becomes asserted immediately after the first command transfer from the preempting CFIFO, that is the higher priority CFIFO, to the CBuffer in use is completed. See Figure 19-61.

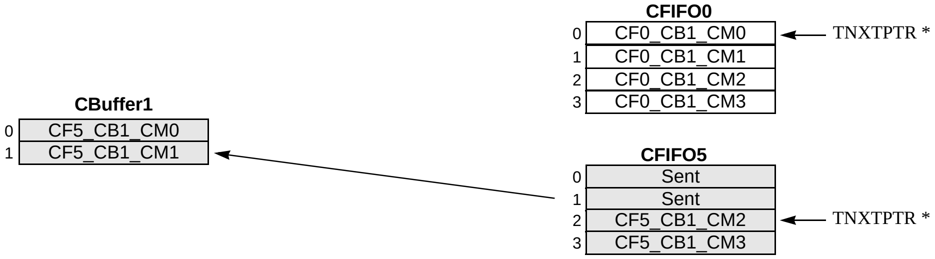
Once command transfers restart/continue, the non-coherency hardware will behave as if the command sequence started from that point. Figure 19-62 depicts how the non-coherency hardware will behave when a non-coherency event is detected.

NOTE

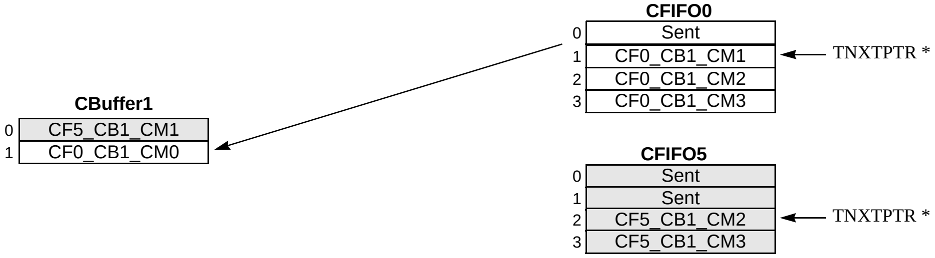
If MODEx is changed to disabled while a CFIFO is transferring commands, the NCF flag for that CFIFO will not become asserted.



(a) CFIFO0 and CFIFO5 both have commands to be sent to CBuffer1, and both are not triggered



(b) CFIFO5 becomes triggered and transfers two commands to CBuffer1



(c) CFIFO0 becomes triggered and transfers a command to CBuffer1. The sequence sent through CFIFO5 becomes non-coherent.

* TNXTPTR - Transfer Next Data Pointer
CF*x*_CB*a*_CM*n* - Command *n* in CFIFO*x* bound for CBuffer*a*

Figure 19-61. Non-Coherency Event when Different CFIFOs use the same CBuffer

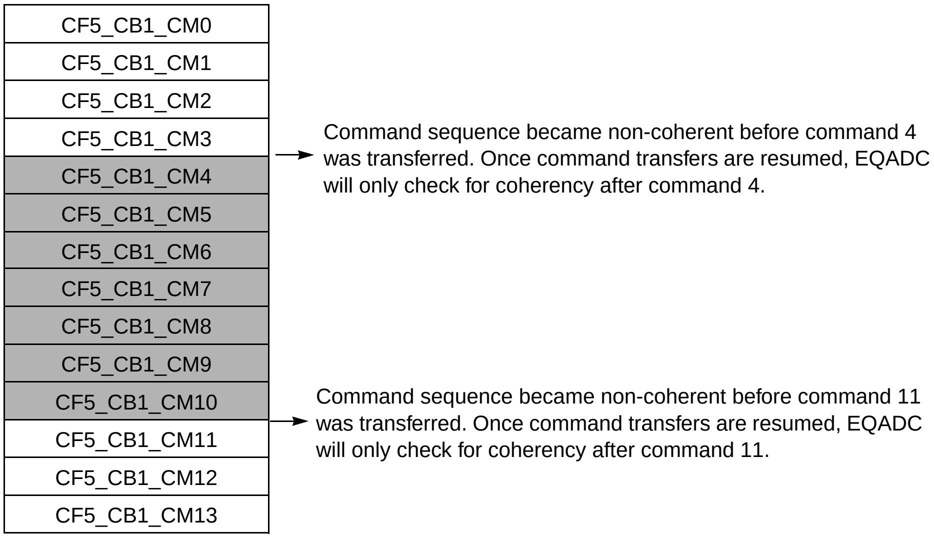


Figure 19-62. Non-coherency Detection when Transfers from a Command Sequence are Interrupted

19.7.5 EQADC Result FIFOs

19.7.5.1 RFIFO Basic Functionality

There are six RFIFOs located in the EQADC. Each RFIFO is four entries deep, and each RFIFO entry is 16 bits long. Each RFIFO serves as a temporary storage location for the one of the RQueues allocated in system memory. Result data is saved in the RFIFOs before being moved into the system RQueues. When an RFIFO is not empty, the EQADC sets the corresponding RFDF bit in Section 19.6.2.7, “EQADC FIFO and Interrupt Status Registers (EQADC_FISR).” If RFDE is asserted in Section 19.6.2.6, “EQADC Interrupt and DMA Control Registers (EQADC_IDCR),” the EQADC generates a request so that an RFIFO entry is moved to the RQueue. An interrupt request, served by the host CPU, is generated when RFDS is negated, and a DMA request, served by the DMAC, is generated when RFDS is asserted. The host CPU or the DMAC responds to these requests by reading Section 19.6.2.4, “EQADC Result FIFO Pop Registers (EQADC_RFPR),” to retrieve data from the RFIFO.

NOTE

The DMAC should be configured to read a single result (16-bit data) from the RFIFO pop registers for every asserted DMA request it acknowledges. Refer to Section 19.8.2, “EQADC/DMAC Interface,” for DMAC configuration guidelines.

NOTE

Reading a word, a half-word, or any bytes from EQADC_RFPRx will pop an entry from RFIFOx, and the RFCTRx field will be decremented by one.

Figure 19-63 describes the important components in the RFIFO. Each RFIFO is implemented as a circular set of registers to avoid the need to move all entries at each push/pop operation. The Pop Next Data Pointer always points to the next RFIFO message to be retrieved from the RFIFO when reading EQADC_RFPR.

The Receive Next Data Pointer points to the next available RFIFO location for storing the next incoming message from the on-chip ADCs. The *RFIFO Counter Logic* counts the number of entries in RFIFO and generates interrupt or DMA requests to drain the RFIFO.

POPNXTPTR in Section 19.6.2.7, "EQADC FIFO and Interrupt Status Registers (EQADC_FISR)," indicates which entry is currently being addressed by the Pop Next Data Pointer, and RFCTR, in the same register, provides the number of entries stored in the RFIFO. Using POPNXTPTR and RFCTR, the absolute addresses for Pop Next Data Pointer and Receive Next Data Pointer can be calculated using the following formulas:

Pop Next Data Pointer Address= RFIFOx_BASE_ADDRESS + POPNXTPTRx*4

Receive Next Data Pointer Address = RFIFOx_BASE_ADDRESS +
[(POPNXTPTRx+RFCTRx) mod RFIFO_DEPTH] * 4

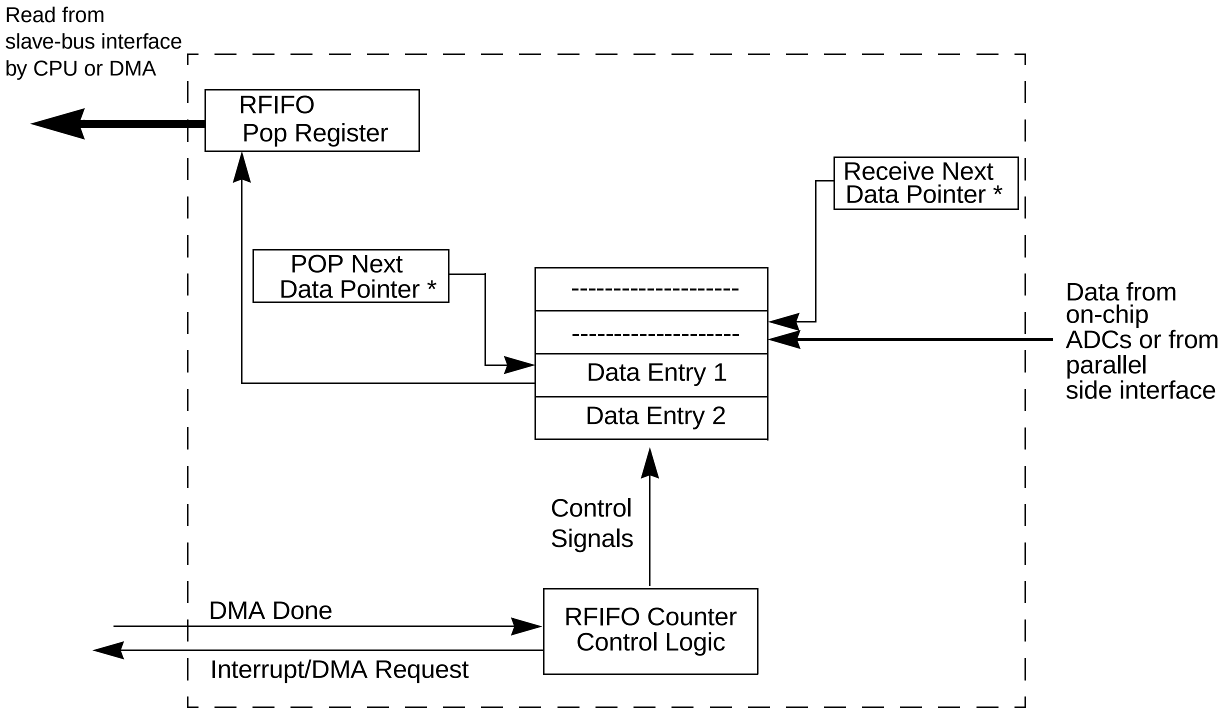
where

- *a mod b* returns the remainder of the division of *a* by *b*.
- RFIFOx_BASE_ADDRESS is the smallest memory mapped address allocated to an RFIFOx entry.
- RFIFO_DEPTH is the number of entries contained in a RFIFO - four in this implementation.

When a new message arrives and RFIFOx is not full, the EQADC copies its contents into the entry pointed by the Receive Next Data Pointer. The RFIFO counter RFCTRx in Section 19.6.2.7, "EQADC FIFO and Interrupt Status Registers (EQADC_FISR)," is incremented by one, and the Receive Next Data Pointer x is also incremented by one (or wrapped around) to point to the next empty entry in RFIFOx. However, if the RFIFOx is full, the EQADC sets the RFOF in Section 19.6.2.7, "EQADC FIFO and Interrupt Status Registers (EQADC_FISR)." The RFIFOx will not overwrite the older data in the RFIFO, the new data will be ignored, and the Receive Next Data Pointer x is not incremented or wrapped around. RFIFOx is full when the Receive Next Data Pointer x equals the Pop Next Data Pointer x and RFCTRx is not zero. RFIFOx is empty when the Receive Next Data Pointer x equals the Pop Next Data Pointer x and RFCTRx is zero.

When the EQADC RFIFO Pop Register x is read and the RFIFOx is not empty, the RFIFO counter RFCTRx is decremented by one, and the POP Next Data Pointer is incremented by one (or wrapped around) to point to the next RFIFO entry.

When the EQADC RFIFO Pop Register x is read and RFIFOx is empty, EQADC will not decrement the counter value and the POP Next Data Pointer x will not be updated. The read value will be undefined.



* All RFIFO entries are memory mapped and the entries addressed by these pointers can have their absolute addresses calculated using POPNXTPTR and RFCTR.

Figure 19-63. RFIFO Diagram

The detailed behavior of the Pop Next Data Pointer and Receive Next Data Pointer is described in the example shown in Figure 19-64 where an RFIFO with 16 entries is shown for clarity of explanation, the actual hardware implementation has only four entries. In this example, RFIFOx with 16 entries is shown in sequence after popping or receiving entries.

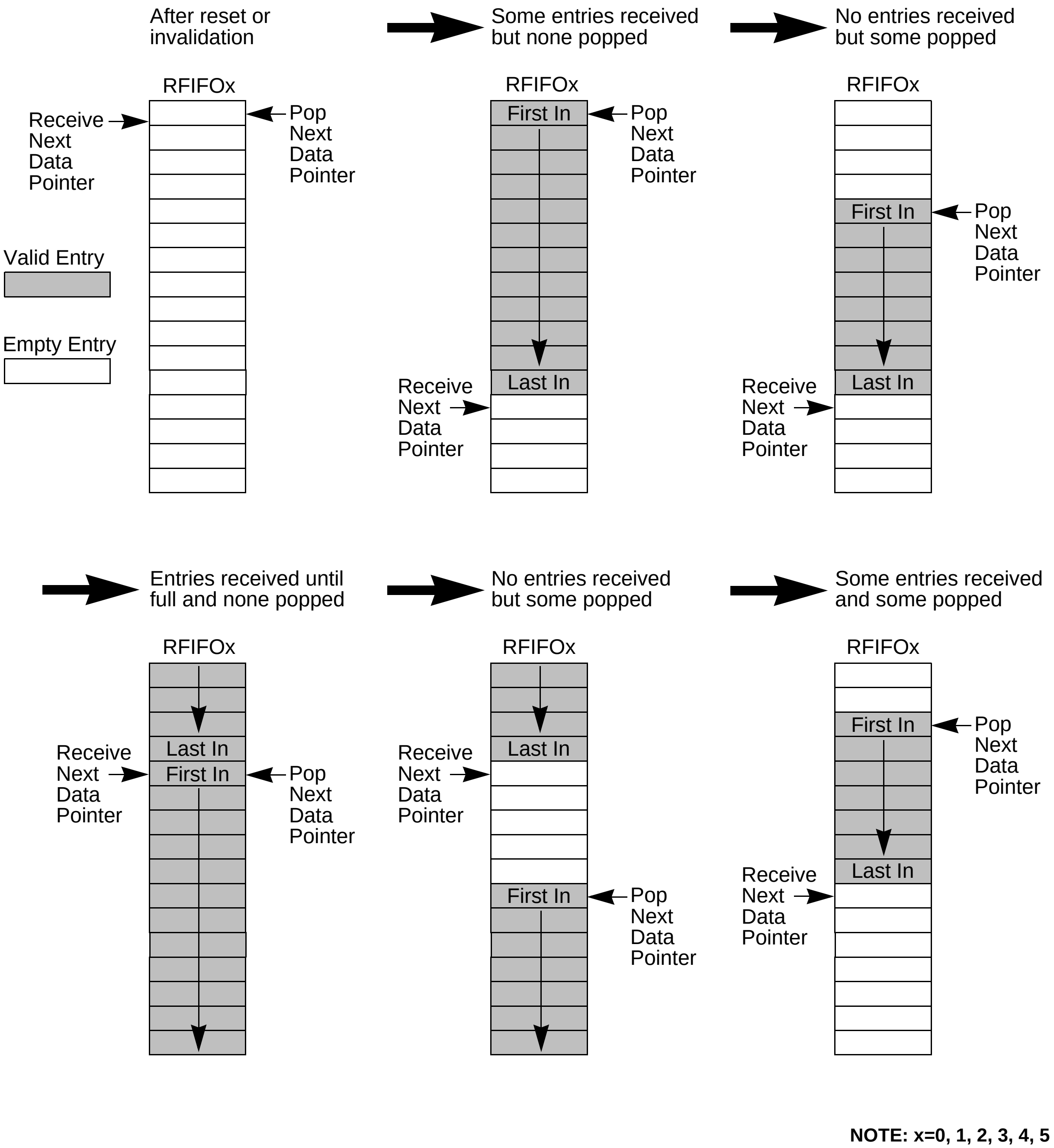


Figure 19-64. RFIFO Entry Pointer Example

19.7.5.2 Distributing Result Data into RFIFOs

Data to be moved into the RFIFOs can come from these sources: ADC0, ADC1, or the on-chip companion modules (Decimation filters). All result data comes with a MESSAGE_TAG field and a DEST field defining what should be done with the received data. The EQADC hardware decodes the MESSAGE_TAG / MESSAGE_TAG2 and DEST / DEST2 fields and:

- stores the 16-bit data into the appropriate RFIFO if the MESSAGE_TAG / MESSAGE_TAG2 indicates a valid RFIFO number, or;
- sends the 16-bit data, the MESSAGE_TAG / MESSAGE_TAG2 and the DEST / DEST2 data to an on-chip companion module (decimation filter), or;
- ignores the data in case of a null or “reserved for customer use” MESSAGE_TAG / MESSAGE_TAG2.

In general received data is moved into RFIFOs as they become available, while an exception happens when multiple results from different sources become available at the same time. In that case, result data from ADC0 is processed first, result data from ADC1 is only process after all ADC0 data is processed, and finally returned data from the companion module is processed (after all data from ADC0/1 is processed).

When time-stamped results return from the on-chip ADCs, the conversion result and the time stamp are always moved to the RFIFOs in consecutive clock cycles in order to guarantee they are always stored in consecutive RFIFO entries.

19.7.6 On-Chip ADC Configuration and Control

19.7.6.1 Enabling and Disabling the On-chip ADCs

The on-chip ADCs have an enable bit (ADC0/1_EN) in the Section 19.6.3.1, “ADC0/1 Control Registers (ADC0_CR and ADC1_CR),” which allows the enabling of the ADCs only when necessary. When the enable bit for an ADC is negated, the clock input to that ADC is stopped. The ADCs are disabled out of reset - ADC0/1_EN bits are negated - to allow for their safe configuration. The ADC must only be configured when its enable bit is negated. Once the enable bit of an ADC is asserted, clock input to is started.

NOTE

Conversion commands sent to the CBuffer of a disabled ADC are ignored by the ADC control hardware.

NOTE

A 8ms wait time from VDDA power up to enabling ADC is required to pre-charge the external 100nf capacitor on REFBYPC pin. This time must be guaranteed by crystal startup time plus reset duration or user. The ADC internal bias generator circuit will start up after 10us upon VRH/VRL and VDDA/VSSA power up and produces a stable/required bias current to the pre-charge circuit, but the current to other analog circuits are disabled until ADCs are enabled. As soon as the ADCs are enabled, the bias currents to all of analog circuits will be enabled.

NOTE

Due to legacy reasons, the EQADC will always wait 120 ADC clocks before issuing the first conversion command following the enabling of one of on-chip ADCs, or the exiting of stop mode. There are two independent counters checking for this delay: one clocked by ADC0_CLK and another by ADC1_CLK. Conversion commands can start to be executed whenever one of these counters completes counting 120 ADC clocks.

19.7.6.2 ADC Clock and Conversion Speed

The clock input to the ADCs is defined by setting the ADC0/1_CLK_SEL and the ADC0/1_CLK_PS fields in the ADC0_CR and ADC1_CR. When the ADC0/1_CLK_SEL is set, the ADC clock frequency is the same as the peripheral clock, but it has the inverted phase. When it is clear, the ADC0/1_CLK_PS field selects the clock divide factor by which the peripheral clock will be divided as showed in Table 19-22. The ADC clock frequency is calculated as below and should be lower than the maximum value specified to the ADC analog block. This is also the maximum frequency of peripheral clock when the ADC0/1_CLK_SEL is asserted.

$$\mathrm{ADCClockFrequency} = \frac{\mathrm{SystemClockFrequency(MHz)}}{\mathrm{SystemClockDivideFactor}};(\mathrm{ADCClockFrequency} \le 15\mathrm{MHz})$$

Figure 19-65 depicts how the ADC clocks for ADC0 and ADC1 are generated.

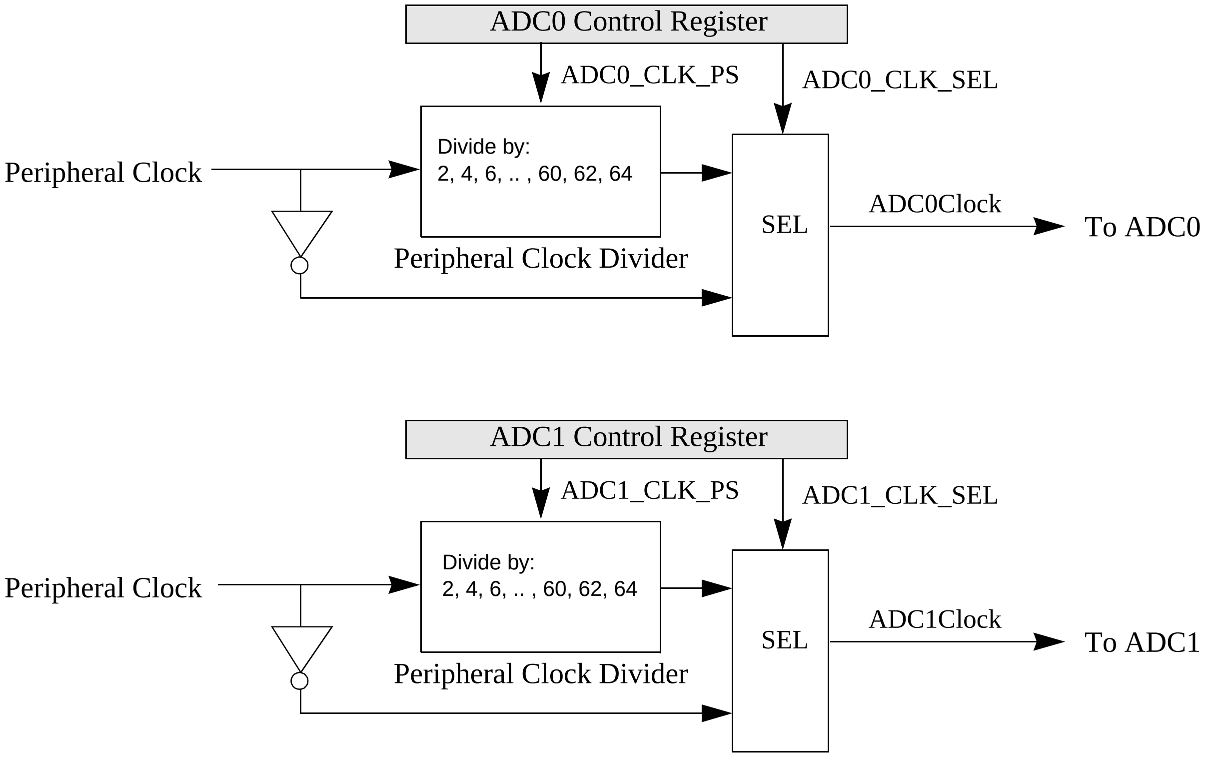


Figure 19-65. ADC0/1 Clock Generation

The ADC conversion speed (in K samples per second - Ksps) is calculated by the following formula. T*he number of sampling cycles* is determined by the LST bits in the command message - see Section 19.7.2.2.1, “Conversion Command Format for the Standard Configuration” - and it can take one of the following values: 2, 8, 64, or 128 ADC clock cycles. The *number of AD conversion cycles* is 13 for differential conversions and 14 for single-ended conversions (12 bits resolution and unitary input gain). The maximum conversion speed is achieved when the ADC Clock frequency is set to its maximum, the number of sampling cycles set to its minimum (2 cycles), and the resolution is also set to the minimum (8 bits) with input unitary gain.

$$\text{ADCConversionSpeed} = \frac{\text{ADCClockFrequency(MHz)}}{(\text{NumberOfSamplingCycles} + \text{NumberOfADConversionCycles})}$$

Table 19-43 shows an example of how the ADC0/1_CLK_PS can be set when using a 120 MHz peripheral clock and the corresponding conversion speeds for all possible ADC clock frequencies. The table also shows that according to the peripheral clock frequency, certain clock divide factors are invalid (2, 4, 6, 8 clock divide factors in the example) since their use would result in a ADC clock frequency higher than the maximum one supported by the ADC. In this example, the maximum ADC clock frequency is 15 MHz (12 bits resolution conversions with unitary input gain).

Table 19-43. ADC Clock Configuration Example (Peripheral Clock Frequency=120 MHz)

ADC0/1_CLK_PS[0:4]	Peripheral Clock Divide Factor	ADC Clock (Peripheral Clock = 120 MHz)	Differential Conversion Speed with Default Sampling Time (2 cycles)	Single-Ended Conversion Speed with Default Sampling Time (2 cycles)
0b00000	2	N/A	N/A	N/A
0b00001	4	N/A	N/A	N/A
0b00010	6	N/A	N/A	N/A
0b00011	8	15.0 MHz	1.0 Msps	938 Ksps
0b00100	10	12.0 MHz	800 Ksps	750 Ksps
0b00101	12	10.0 MHz	667 Ksps	625 Ksps
0b00110	14	8.57 MHz	571 Ksps	536 Ksps
0b00111	16	7.5 MHz	500 Ksps	469 Ksps
0b01000	18	6.67 MHz	444 Ksps	417 Ksps
0b01001	20	6.0 MHz	400 Ksps	375 Ksps
0b01010	22	5.45 MHz	364 Ksps	341 Ksps
0b01011	24	5.0 MHz	333 Ksps	313 Ksps
0b01100	26	4.62 MHz	308 Ksps	288 Ksps
0b01101	28	4.29 MHz	286 Ksps	268 Ksps
0b01110	30	4.0 MHz	267 Ksps	250 Ksps
0b01111	32	3.75 MHz	250 Ksps	234 Ksps
0b10000	34	3.53 MHz	235 Ksps	221 Ksps
0b10001	36	3.33 MHz	222 Ksps	208 Ksps
0b10010	38	3.16 MHz	211 Ksps	198 Ksps
0b10011	40	3.0 MHz	200 Ksps	188 Ksps
0b10100	42	2.86 MHz	190 Ksps	179 Ksps

Table 19-43. ADC Clock Configuration Example (Peripheral Clock Frequency=120 MHz)

ADC0/1_CLK_PS[0:4]	Peripheral Clock Divide Factor	ADC Clock (Peripheral Clock = 120 MHz)	Differential Conversion Speed with Default Sampling Time (2 cycles)	Single-Ended Conversion Speed with Default Sampling Time (2 cycles)
0b10101	44	2.73 MHz	182 Ksps	170 Ksps
0b10110	46	2.61 MHz	174 Ksps	163 Ksps
0b10111	48	2.5 MHz	167 Ksps	156 Ksps
0b11000	50	2.4 MHz	160 Ksps	150 Ksps
0b11001	52	2.31 MHz	154 Ksps	144 Ksps
0b11010	54	2.22 MHz	148 Ksps	139 Ksps
0b11011	56	2.14 MHz	143 Ksps	134 Ksps
0b11100	58	2.07 MHz	138 Ksps	129 Ksps
0b11101	60	2.0 MHz	133 Ksps	125 Ksps
0b11110	62	1.94 MHz	129 Ksps	121 Ksps
0b11111	64	1.88 MHz	125 Ksps	117 Ksps

19.7.6.3 Time Stamp Feature

The on-chip ADCs can provide a time stamp for the conversions they execute. A time stamp is the value of the time base counter latched when the EQADC detects the end of the analog input voltage sampling. A time stamp for a conversion command is requested by setting the TSR bit in the corresponding command. When TSR is negated, that is a time stamp is not requested, the ADC returns a single result message containing the conversion result. When TSR is asserted, that is a time stamp is requested, the ADC returns two result messages; one containing the conversion result, and afterwards another containing the time stamp for that conversion. The result messages are sent in this order to the RFIFOs and both messages are sent to the same RFIFO was specified in the MESSAGE_TAG field of the executed conversion command.

The time base counter is a 16-bit up counter that wraps after reaching 0xFFFF. It is disabled after reset and it is enabled according to the setting of TBC_CLK_PS field in Section 19.6.3.2, “ADC Time Stamp Control Register (ADC_TSCR).” TBC_CLK_PS defines if the counter is enabled or disabled, and, if enabled, at what frequency it is incremented. The time stamps are returned regardless of whether the time base counter is enabled or disabled. The time base counter can be reset by writing 0x0000 to the Section 19.6.3.3, “ADC Time Base Counter Registers (ADC_TBCR),” with a write configuration command.

19.7.6.3.1 Red Line Client Submodule (REDLC)

The Red Line Client sub-block REDLC extracts the external time base from the Red Line bus to the EQADC. Figure 19-66 provides a block diagram for the REDLC sub-module that contains actually two slot selectors.

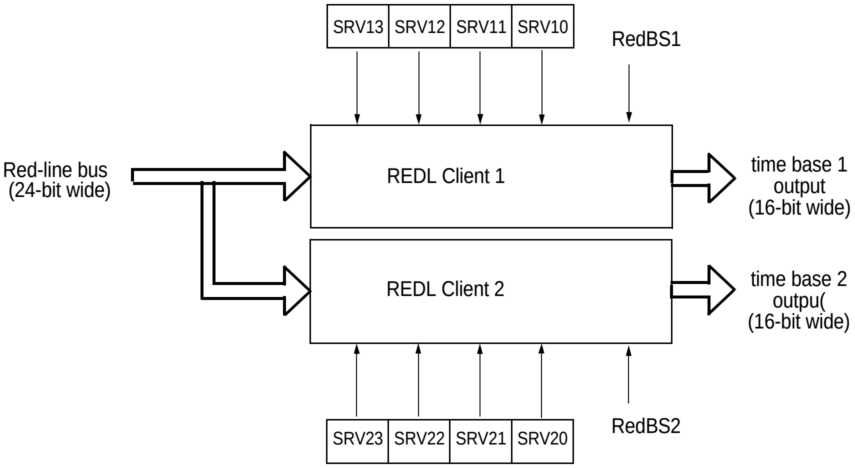
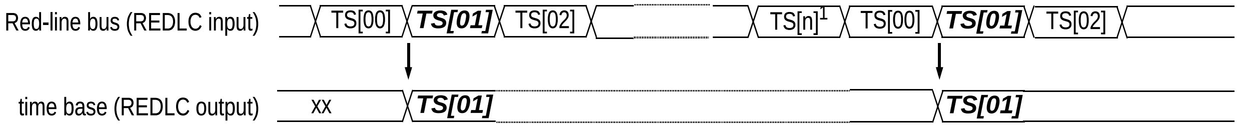


Figure 19-66. REDLC Block Diagram

The Red Line data stream is composed by several identified time slots. Each time slot contains a time value from a specified source. The bits SRV1/2[0:3] in register EQADC_REDLCCR are used to select the desired time slots of the Red-line bus to be used internally by the EQADC. Figure 19-67 shows a timing diagram for the REDLC.



NOTES: 1. Maximum of 16 Time Slots (TS[n])
2. In this case, SRV bits were set to capture TS[01]

Figure 19-67. Timing diagram for the Red Line bus and REDLC output

Every time the selected time slot change, the REDLC outputs are updated. As this time base is an external data to the EQADC, this output is not affected by the stop or by the debug internal state.

After the slot selection is done and the timebase data is extracted, the REDBS1/2 bits select 16 bits from the original 24-bit timebase data. These selected bits are the timebase to be used internal to the EQADC.

19.7.6.4 ADC Pre-gain Feature

Each ADC can be configured to have a selectable input gain as defined in Section 19.6.3.6, "Alternate Configuration 1-14 Control Registers (ADC_ACR1-14)." This means the input signal is sampled and the result is amplified by factor 2, or 4 before the conversion phase. In present implementation of this feature, the conversion is 1 or 2 ADC clock cycles longer for gain 2 or gain 4, respectively.

19.7.6.5 ADC Resolution Selection Feature

The ADCs conversion resolutions can be 8 bits, 10 bits or 12 bits as described in Section 19.6.3.6, "Alternate Configuration 1-14 Control Registers (ADC_ACR1-14)." For conversions at a resolution less than 12, the ADC is executing less operations and the conversion time is smaller. In this ADC, it is verified that there is 1 ADC clock cycle for each bit of resolution. Therefore, for the same ADC clock frequency, the ADC sample frequency is higher for lower resolutions.

19.7.6.6 ADC Calibration Feature

19.7.6.6.1 Overview

There are three sets of calibration coefficients for each ADC. Each set is composed by a gain factor and an offset factor: GCCn/OCCn, ALTGCCn1/ALTGCCn1, and ALTGCCn2/ALTGCCn2, where n is the ADC number 0 or 1. The pair GCCn/OCCn is selected when it is used the normal configuration or the alternate configurations 3 to 14. The pair ALTGCCn1/ALTGCCn1 is used only when the alternate configuration 1 is selected. And the pair ALTGCCn2/ALTGCCn2 is for the alternate configuration 2. The description below is for a generic pair of gain/offset GCC/OCC.

The EQADC provides a calibration scheme to remove the effects of gain and offset errors from the results generated by the on-chip ADCs. Only results generated by the on-chip ADCs are calibrated. The results generated by ADCs on the external device are directly sent to RFIFOs unchanged. The main component of calibration hardware is a Multiply-and-Accumulate (MAC) unit, one per on-chip ADC, that is used to calculate the following transfer function which relates a calibrated result to a raw, uncalibrated one.

CAL_RES = GCC * RAW_RES + OCC+2;

where:

- CAL_RES is the calibrated result corresponding the input voltage V_i.
- GCC is the gain calibration constant.
- RAW_RES is the raw, uncalibrated result with resolution adjustment corresponding to an specific input voltage V_i.
- OCC is the offset calibration constant.
- The addition of two reduces the maximum quantization error of the ADC. See Section 19.8.6.3, "Quantization Error Reduction During Calibration."

Calibration constants GCC and OCC are determined by taking two samples of known reference voltages and using these samples to calculate the values for the constants. For details and an example about how to calculate the calibration constants and use them in result calibration refer to Section 19.8.6, "ADC Result Calibration." Once calculated, GCC is stored in the Section 19.6.3.4, "ADC0/1 Gain Calibration Constant

Registers (ADC0_GCCR and ADC1_GCCR),” and OCC in Section 19.6.3.5, “ADC0/1 Offset Calibration Constant Registers (ADC0_OCCR and ADC1_OCCR),” from where their values are fed to the MAC unit. The alternate gain values are stored in Section 19.6.3.7, “ADC0/1 Alternate Gain Registers (ADC0_AGR1-2 and ADC1_AGR1-2),” and the alternate offset values in Section 19.6.3.8, “ADC0/1 Alternate Offset Register (ADC0_AOR1-2 and ADC1_AOR1-2).” Since the analog characteristics of each on-chip ADCs differs, each ADC has an independent pair of calibration constants.

A conversion result is calibrated according to the status of CAL bit in the command that initiated the conversion. If the CAL bit is asserted, the EQADC will automatically calculate the calibrated result before sending the result to the appropriate RFIFO or companion module. If the CAL bit is negated, the result is not calibrated, it bypasses the calibration hardware, and is directly sent to the appropriate RFIFO or companion module.

19.7.6.6.2 MAC Unit and Operand Data Format

The MAC unit diagram is shown in Figure 19-68. Each on-chip ADC has a separate MAC unit to fine-tune its conversion results. The description below considers the general calibration constant registers but it is the same for the alternate calibration constants.

The OCC0/1 operand is a 14-bit signed value stored in the Section 19.6.3.5, “ADC0/1 Offset Calibration Constant Registers (ADC0_OCCR and ADC1_OCCR).” The RAW_RES operand is the raw uncalibrated result, and it is the direct output from the on-chip ADCs but passing through the resolution adjustment block. The GCC0/1 operand is a 15-bit fixed point unsigned value stored in the Section 19.6.3.4, “ADC0/1 Gain Calibration Constant Registers (ADC0_GCCR and ADC1_GCCR).” The GCC is expressed in the *GCC_INT.GCC_FRAC* binary format. The integer part of the GCC (GCC_INT=GCC[1]) contains a single binary digit while its fractional part (GCC_FRAC=GCC[2:15]) contains 14 bits - see Figure 19-69. The gain constant equivalent decimal value ranges from 0 to 1.999938..., as shown in Table 19-44. Two is always added to the MAC output - see Section 19.8.6.3, “Quantization Error Reduction During Calibration.” CAL_RES output is the calibrated result, and it is a 14-bit unsigned value. CAL_RES is truncated to 0x3FFF, in case of a overflow, and to 0x0000, in case of an underflow.

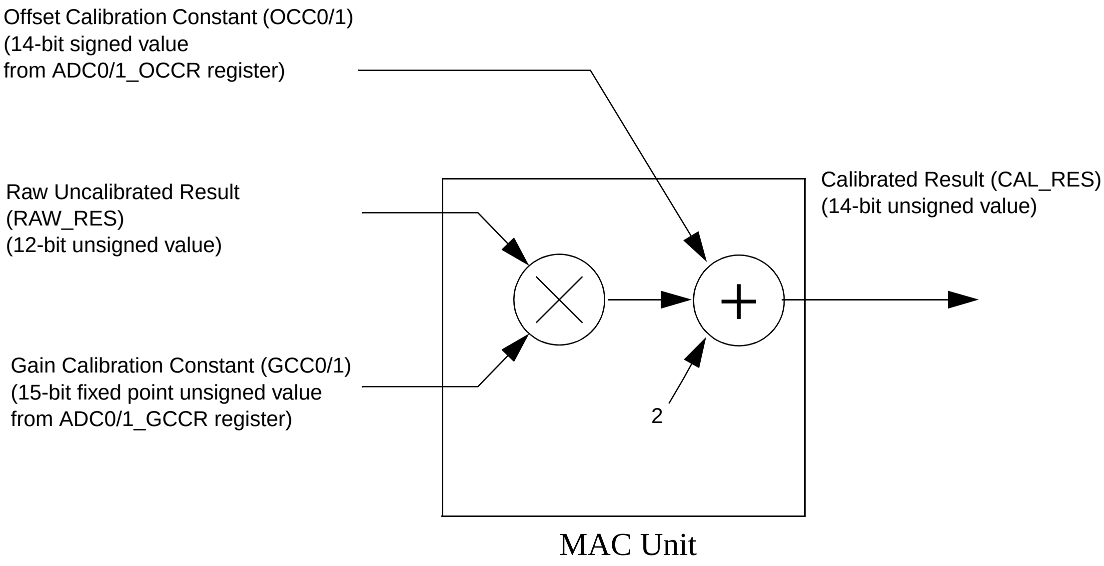


Figure 19-68. MAC Unit Diagram

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
GCC_INT	GCC_FRAC													
GCC[1]	GCC[2:15]													

Figure 19-69. Gain Calibration Constant Format

GCC_INT - Integer part of the gain calibration constant for ADC0/1

GCC_INT is the integer part of the gain calibration constant for ADC0/1.

GCC_FRAC[1:14] - Fractional part of the gain calibration constant for ADC0/1

GCC_FRAC is the fractional part of the gain calibration constant for ADC0/1. GCC_FRAC can expresses decimal values ranging from 0 to 0.999938...

Table 19-44. Binary and Decimal Representations of the Gain Constant

Gain Constant (GCC_INT.GCC_FRAC binary format)	Corresponding Decimal Value
0.0000_0000_0000_00	0
...	...
0.1000_0000_0000_00	0.5
...	...
0.1111_1111_1111_11	0.999938...
1.0000_0000_0000_00	1
...	...
1.1100_0000_0000_00	1.75
...	...
1.1111_1111_1111_11	1.999938...

19.7.6.7 *ADC Control Logic* overview and command execution

Figure 19-70 shows the basic logic blocks involved in the ADC Control and how they interact. CFIFOs/RFIFOs interact with CBuffers/*Abort Cont*/*Result Message Return Logic* through the *FIFO Control Unit*. The EB and BN bits in the Command Message uniquely identify the CBuffer to which a command should be sent. The *FIFO Control Unit* decodes these bits and sends the ADC command to the proper CBuffer. Other blocks of logic are the *Resolution Adjustment*, *Result Format and Calibration Sub-Block*, the *Time Stamp Logic*, and the *MUX Control Logic*.

The *Resolution Adjustment Sub-Block* receives the 12-bit data bus directly from the ADC and changes the received conversion results from right aligned format of ADC to the left aligned format depending on the selected resolution of the conversion. This operation helps the calibration processing to use the calibration coefficients always with the same format.

The *Result Format and Calibration Sub-Block* formats the returning data into Result Messages and sends them to the RFIFOs[1]. The returning data can be data read from an ADC register, a conversion result, or a time stamp. The formatting and calibration of conversion results also take place inside this sub-block.

The *Time Stamp Logic* latches the value of the time base counter when detecting the end of the analog input voltage sampling, and sends it to the *Result Format and Calibration Sub-Block* as time stamp information.

The *MUX Control Logic* generates the proper MUX control signals and, when the ADC0/1_EMUX bits are asserted, the MA signals based on the channel numbers extracted from the ADC Command.

When the on-chip ADC abort feature is not enabled, ADC Commands are stored in the CBuffers as they come and they are executed in the first-in-first-out basis. After the execution of a command in ENTRY1 finishes all commands are shifted one entry. After the shift, ENTRY0 is always empty and ready to receive a new command. Execution of configuration commands only start when they reach ENTRY1. Consecutive conversion commands are pipelined and their execution can start while in ENTRY0. This is explained below.

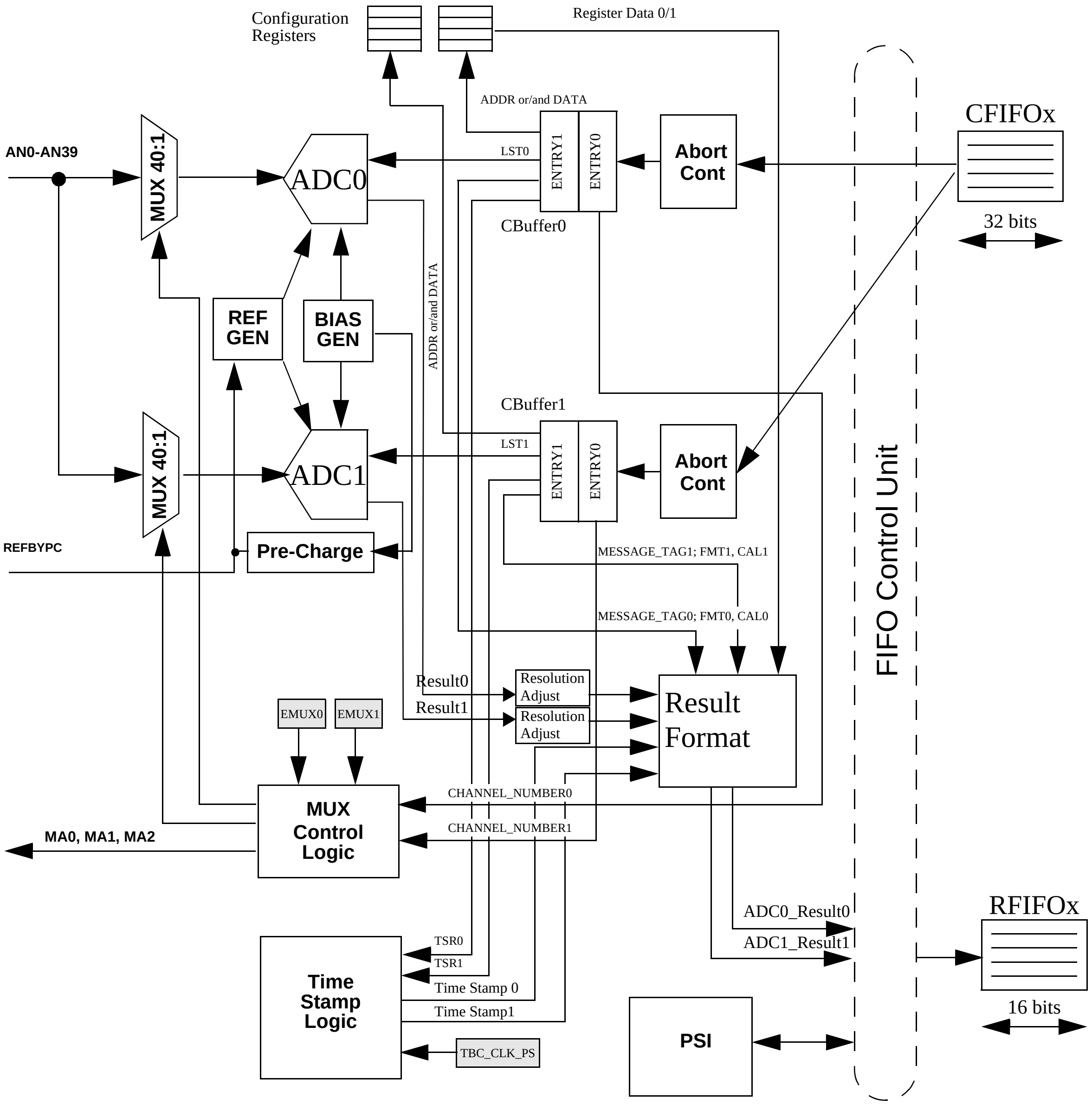
AD conversion accuracy can be affected by the settling time of the input channel multiplexers. Some time is required for the channel multiplexers internal capacitances to settle after the channel number is changed. If the time prior to sampling is not long enough to absorb this settling, then the settling time will take from ADC sampling time which may result in inaccurate sampling and ultimately compromise conversion result accuracy - see Figure 19-71 (a). The EQADC attempts to compensate for this settling time by switching the multiplexers in preparation for the next conversion command's sampling while performing the previous conversion (Figure 19-71 (b)). In EQADC, this is done in the following way; when a conversion command is in buffer ENTRY1 and another conversion command is identified in ENTRY0, then the channel number of ENTRY0 is sent to the *MUX Control Logic* some cycles before the sampling phase of the command in ENTRY0 starts. In this way, sampling for the next command can promptly start after the current conversion finishes because the internal capacitance of the multiplexers will be settled by that time, allowing for more accurate sampling. This is specially important for applications that require high conversion speeds, that is with the ADC running at maximum clock frequency and with the analog input voltage sampling time set to a minimum (2 ADC clock cycles), when the short sampling time does not allow the multiplexers to completely settle. The second advantage of pipelining conversion commands is

---

[1] The result messages may also be routed to an on-chip companion module via the side interface, and then fed back to the RFIFOs.

to provide precise conversion intervals, which means the time intervals between two consecutive conversions are the same. This is important for any digital signal process application.

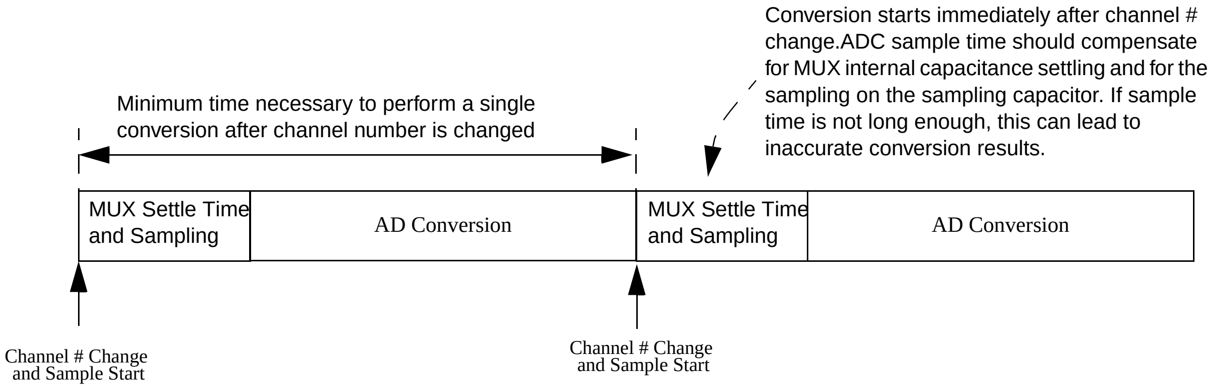
When the on-chip ADC abort feature is enabled, ADC Commands from CFIFO0 should be considered immediately, even stopping the execution of some command that is already in ENTRY1. When the abort request is sent to the ADC, the already stored commands in the CBuffers are copied in a temporary set of registers. The first ADC command from CFIFO0 is sent after the abort acknowledge indication from ADC. The process is the same as usual until the transfer of the last command from CFIFO0. Then the temporarily stored commands that were postponed by the abortion are recovered and they are pipelined for execution. After the last command from this temporary memory is transferred, the next commands are pipelined from the CFIFOs.



REGISTER FIELD Words in shaded boxes represent configuration register fields

NOTE: x=0, 1, 2, 3, 4, 5

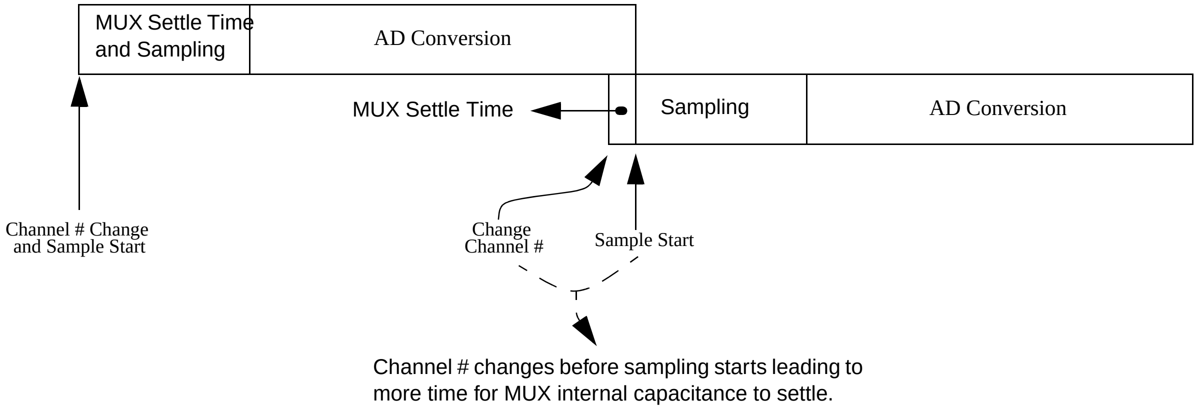
Figure 19-70. On-Chip ADC Control Scheme



Conversion starts immediately after channel # change.ADC sample time should compensate for MUX internal capacitance settling and for the sampling on the sampling capacitor. If sample time is not long enough, this can lead to inaccurate conversion results.

Minimum time necessary to perform a single conversion after channel number is changed

(a) Command Execution Sequence for Two Non-Overlapped Commands



Channel # changes before sampling starts leading to more time for MUX internal capacitance to settle.

(b) Command Execution Sequence for Two Overlapped Commands

Figure 19-71. Overlapping Consecutive Conversion Commands

19.7.7 Internal/External Multiplexing

19.7.7.1 Channel Assignment

The internal analog multiplexers select one of the 40 analog input pins for conversion, based on the CHANNEL_NUMBER field of a Command Message. The analog input pin channel number assignments and the pin definitions vary depending on how the ADC0/1_EMUX are configured. Allowed combinations of ADC0/1_EMUX bits are shown in Table 19-45 together with references to tables indicating how CHANNEL_NUMBER field of each conversion command must be set to avoid channel selection conflicts.In addition to this master multiplexer, each ADC is equipped with a secondary multiplexer that are used to monitor signals internal to the chip. These analog input pins are also selected by the CHANNEL_NUMBER field.

During differential conversions the analog multiplexer passes differential signals to both the positive and negative terminals of the ADC. The differential conversions can only be initiated on four channels: DAN0, DAN1, DAN2, and DAN3. For each value of CHANNEL_NUMBER, this table lists the ADCs you can use to perform conversions for that channel (which you specify using BN), whether external multiplexing (specified by ADCx_CR[EMUX]) must be disabled or enabled to access the channel, and the assigned analog input signal and conversion type.

- More than one ADC can access the same analog input, but not at the same time.
- When one ADC is performing a differential conversion on a pair of pins, another ADC must not access either of those two pins as single-ended channels.
-

Table 19-45. Multiplexed and non-multiplexed channel assignments

CHANNEL_NUMBER		ADCs (BN)	ADCx_CR[EMUX]	Analog input signal	Conversion type
(b)	(d)				
0000_0000	0	0, 1	X	AN0 (accessible when DAN0+ and DAN0– are not being accessed)	Single-ended
0000_0001	1	0, 1	X	AN1 (accessible when DAN0+ and DAN0– are not being accessed)	Single-ended
0000_0010	2	0, 1	X	AN2 (accessible when DAN1+ and DAN1– are not being accessed)	Single-ended
0000_0011	3	0, 1	X	AN3 (accessible when DAN1+ and DAN1– are not being accessed)	Single-ended
0000_0100	4	0, 1	X	AN4 (accessible when DAN2+ and DAN2– are not being accessed)	Single-ended
0000_0101	5	0, 1	X	AN5 (accessible when DAN2+ and DAN2– are not being accessed)	Single-ended
0000_0110	6	0, 1	X	AN6 (accessible when DAN3+ and DAN3– are not being accessed)	Single-ended
0000_0111	7	0, 1	X	AN7 (accessible when DAN3+ and DAN3– are not being accessed)	Single-ended
0000_1000	8	0, 1	X	AN8 (see also ANW)	Single-ended
0000_1001	9	0, 1	X	AN9 (see also ANX)	Single-ended
0000_1010	10	0, 1	X	AN10 (see also ANY)	Single-ended
0000_1011	11	0, 1	X	AN11 (see also ANZ)	Single-ended
0000_1100 to 0000_1111	12 to 15	0, 1	X	AN12 to AN15	Single-ended
0001_0000	16	0, 1	X	AN16 (see also ANR)	Single-ended
0001_0001	17	0, 1	X	AN17 (see also ANS)	Single-ended
0001_0010	18	0, 1	X	AN18 (see also ANT)	Single-ended
0001_0011	19	0, 1	X	AN19 (see also ANU)	Single-ended

Table 19-45. Multiplexed and non-multiplexed channel assignments

CHANNEL_NUMBER		ADCs (BN)	ADCx_CR[EMUX]	Analog input signal	Conversion type
(b)	(d)				
1000_0100 to 0010_0111	20 to 39	0, 1	X	AN20 to AN39	Single-ended
0010_1000	40	0, 1	X	VRH	Single-ended
0010_1001	41	0, 1	X	VRL	Single-ended
0010_1010	42	0, 1	X	50% × (VRH – VRL) (do not use for calibration)	Single-ended
0010_1011	43	0, 1	X	75% × (VRH – VRL) (used for calibration)	Single-ended
0010_1100	44	0, 1	X	25% × (VRH – VRL) (used for calibration)	Single-ended
0010_1101	45	0	X	INA_ADC0_0 (see Table 19-4)	Single-ended
0010_1101	45	1	X	INA_ADC1_0 (see Table 19-4)	Single-ended
0010_1110 to 0011_1111	46 to 63	(Reserved)			
0100_0xxx	64 to 71	0, 1	1	ANW (the MA2–MA0 pins act as the selector inputs for an external 8-to-1 mux; see also AN8)	Single-ended
0100_1xxx	72 to 79	0, 1	1	ANX (the MA2–MA0 pins act as the selector inputs for an external 8-to-1 mux; see also AN9)	Single-ended
0101_0xxx	80 to 87	0, 1	1	ANY (the MA2–MA0 pins act as the selector inputs for an external 8-to-1 mux; see also AN10)	Single-ended
0101_1xxx	88 to 95	0, 1	1	ANZ (the MA2–MA0 pins act as the selector inputs for an external 8-to-1 mux; see also AN11)	Single-ended
0110_0000	96	0, 1	X	DAN0+ and DAN0– (accessible when A0 and A1 are not being accessed)	Differential
0110_0001	97	0, 1	X	DAN1+ and DAN1– (accessible when A2 and A3 are not being accessed)	Differential
0110_0010	98	0, 1	X	DAN2+ and DAN2– (accessible when A4 and A5 are not being accessed)	Differential
0110_0011	99	0, 1	X	DAN3+ and DAN3– (accessible when A6 and A7 are not being accessed)	Differential
0110_0100 to 0111_1111	100 to 127	(Reserved)			
1000_0000	128	0	X	INA_ADC0_1 (see Table 19-4)	Single-ended
1000_0000	128	1	X	INA_ADC1_1 (see Table 19-4)	Single-ended
1000_0001	129	0	X	INA_ADC0_2 (see Table 19-4)	Single-ended
1000_0001	129	1	X	INA_ADC1_2 (see Table 19-4)	Single-ended

Table 19-45. Multiplexed and non-multiplexed channel assignments

CHANNEL_NUMBER		ADCs (BN)	ADCx_CR[EMUX]	Analog input signal	Conversion type
(b)	(d)				
1000_0010 to 1000_1111	130 to 143	(Reserved)			
1001_0000	144	0	X	INA_ADC0_9 (see Table 19-4)	Single-ended
1001_0000	144	1	X	INA_ADC1_9 (see Table 19-4)	Single-ended
1001_0001 to 1010_0001	145 to 161	(Reserved)			
1010_0010	162	X	0	INA_ADC0_3 (see Table 19-4)	Single-ended
1010_0011	163	X	0	INA_ADC0_4 (see Table 19-4)	Single-ended
1010_0100	164	X	0	INA_ADC0_5 (see Table 19-4)	Single-ended
1010_0101	165	X	0	INA_ADC0_6 (see Table 19-4)	Single-ended
1010_0110	166	X	0	INA_ADC0_7 (see Table 19-4)	Single-ended
1010_0111	167	X	0	INA_ADC0_8 (see Table 19-4)	Single-ended
1010_1000 to 1100_0001	168 to 193	(Reserved)			
1100_0010	194	X	0	INA_ADC1_3 (see Table 19-4)	Single-ended
1100_0011	195	X	0	INA_ADC1_4 (see Table 19-4)	Single-ended
1100_0100	196	X	0	INA_ADC1_5 (see Table 19-4)	Single-ended
1100_0101	197	X	0	INA_ADC1_6 (see Table 19-4)	Single-ended
1100_0110	198	X	0	INA_ADC1_7 (see Table 19-4)	Single-ended
1100_0111	199	X	0	INA_ADC1_8 (see Table 19-4)	Single-ended
1100_1000 to 1101_1111	200 to 223	(Reserved)			
1110_0xxx	224 to 231	0, 1	1	ANR (the MA2–MA0 pins act as the selector inputs for an external 8-to-1 mux; see also AN16)	Single-ended
1110_1xxx	232 to 239	0, 1	1	ANS (the MA2–MA0 pins act as the selector inputs for an external 8-to-1 mux; see also AN17)	Single-ended
1111_0xxx	240 to 247	0, 1	1	ANT (the MA2–MA0 pins act as the selector inputs for an external 8-to-1 mux; see also AN18)	Single-ended
1111_1xxx	248 to 255	0, 1	1	ANU (the MA2–MA0 pins act as the selector inputs for an external 8-to-1 mux; see also AN19)	Single-ended

19.7.7.2 External Multiplexing

The EQADC can use from one to eight external multiplexer chips to expand the number of analog signals that may be converted. Up to 64 analog channels can be converted through external multiplexer selection. The externally multiplexed channels are automatically selected by the CHANNEL_NUMBER field of a Command Message, in the same way done with internally multiplexed channels. The software selects the external multiplexed mode by setting the ADC0/1_EMUX bit in either ADC0_CR or ADC1_CR depending on which ADC will perform the conversion. Table 19-46 shows the channel number assignments for the multiplexed mode. There are 4 differential pairs, 39 single-ended, and, at most, 64 externally multiplexed channels which can be selected. Only one ADC can have its EMUX bit asserted at a time.

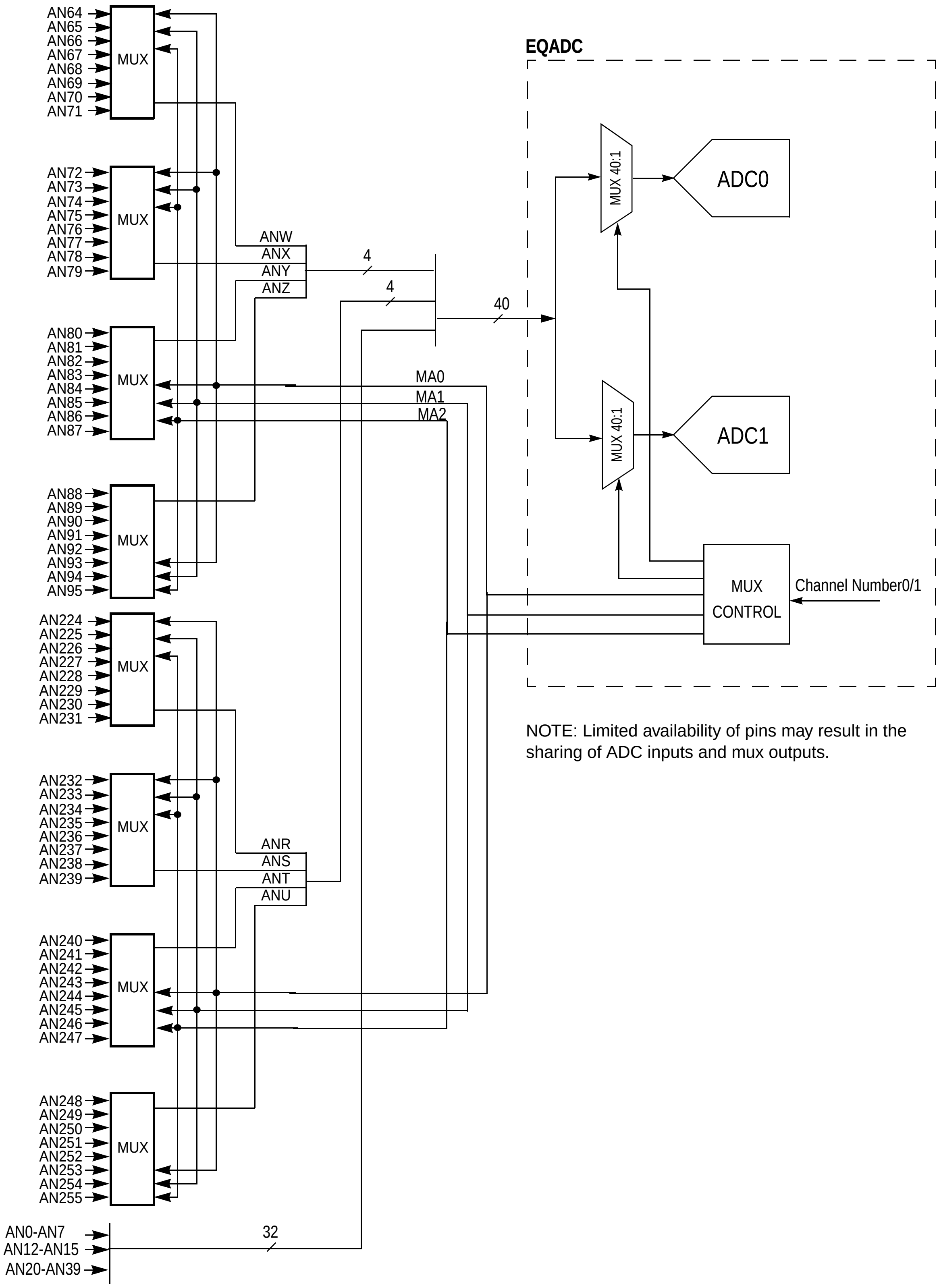
Figure 19-72 shows the maximum configuration of eight external multiplexer chips connected to the EQADC. The external multiplexer chip selects one of eight analog inputs and connects it to a single analog output, which is fed to a specific input of the EQADC. The EQADC provides three multiplexed address signals, MA0, MA1, and MA2, to select one of eight inputs. These three multiplexed address signals are connected to all eight external multiplexer chips. The analog output of the eight multiplex chips are each connected to eight separate EQADC inputs, ANR, ANS, ANT, ANU, ANW, ANX, ANY, and ANZ. The MA pins correspond to the three least significant bits of the channel number that selects ANR, ANS, ANT, ANU, ANW, ANX, ANY, and ANZ with MA0 being the most significant bit - See Table 19-46.

Table 19-46. Encoding of MA Pins[1]

Channel Number selecting ANR, ANS, ANT, ANU, ANW, ANX, ANY, ANZ (decimal)								MA0	MA1	MA2
ANR	ANS	ANT	ANU	ANW	ANX	ANY	ANZ			
224	232	240	248	64	72	80	88	0	0	0
225	233	241	249	65	73	81	89	0	0	1
226	234	242	250	66	74	82	90	0	1	0
227	235	243	251	67	75	83	91	0	1	1
228	236	244	252	68	76	84	92	1	0	0
229	237	245	253	69	77	85	93	1	0	1
230	238	246	254	70	78	86	94	1	1	0
231	239	247	255	71	79	87	95	1	1	1

[1] ‘0’ means pin is driven LOW and ‘1’ that pin is driven HIGH.

When the external multiplexed mode is selected for either ADC, the EQADC automatically creates the MA output signals from CHANNEL_NUMBER field of a Command Message. The EQADC also converts the proper input channel (ANR, ANS, ANT, ANU, ANW, ANX, ANY, and ANZ) by interpreting the CHANNEL_NUMBER field. As a result, up to 64 externally multiplexed channels appear to the conversion queues as directly connected signals.



NOTE: Limited availability of pins may result in the sharing of ADC inputs and mux outputs.

Figure 19-72. Example of External Multiplexing

19.7.8 EQADC DMA/Interrupt Request

Table 19-47 lists methods to generate interrupt requests in the EQADC queuing control and triggering control. The DMA/interrupt request select bits and the DMA/interrupt enable bits are described in Section 19.6.2.6, “EQADC Interrupt and DMA Control Registers (EQADC_IDCR),” and the interrupt flag bits are described in Section 19.6.2.7, “EQADC FIFO and Interrupt Status Registers (EQADC_FISR).” Table 19-73 depicts all interrupts and DMA requests generated by the EQADC.The Result FIFO Overflow Interrupt, the Command FIFO Underflow Interrupt, and the Command FIFO Trigger Overrun Interrupt requests of ALL CFIFOs are ORed to generate single interrupt request from the eQADC. This combined interrupt request is asserted whenever one of the following flags becomes asserted: RFOFx, CFUFx, and TORFx (assuming that all interrupts are enabled).

Table 19-47. EQADC FIFO Interrupt Summary[1]

Interrupt	Condition	Clearing Mechanism
Non Coherency Interrupt	NCIEx = 1 NCFx = 1	Clear NCFx bit by writing a “1” to the bit.
Result FIFO Overflow Interrupt[2]	RFOIEx = 1 RFOFx = 1	Clear RFOFx bit by writing a “1” to the bit.
Command FIFO Underflow Interrupt[2]	CFUIEx = 1 CFUFx = 1	Clear CFUFx bit by writing a “1” to the bit.
Result FIFO Drain Interrupt	RFDEx = 1 RFDSx = 0 RFDFx = 1	Clear RFDFx bit by writing a “1” to the bit.
Command FIFO Fill Interrupt	CFFEx = 1 CFFSx = 0 CFFFx = 1	Clear CFFFx bit by writing a “1” to the bit.
End of Queue Interrupt	EOQIEx = 1 EOQFx = 1	Clear EOQFx bit by writing a “1” to the bit.
Pause Interrupt	PIEx = 1 PFx =1	Clear PFx bit by writing a “1” to the bit.
Trigger Overrun Interrupt[2]	TORIEx = 1 TORFx =1	Clear TORFx bit by writing a “1” to the bit.

[1] For details refer to Section 19.6.2.7, “EQADC FIFO and Interrupt Status Registers (EQADC_FISR),” and Section 19.6.2.6, “EQADC Interrupt and DMA Control Registers (EQADC_IDCR).”

[2] Apart from generating an independent interrupt request for when a RFIFO Overflow Interrupt, a CFIFO Underflow Interrupt, and a CFIFO Trigger Overrun Interrupt occurs, the EQADC also provides a combined interrupt request at which these requests from ALL CFIFOs are ORed. Refer to Figure 19-73 for details.

Table 19-48 describes a list of methods to generate DMA requests in the EQADC.

Table 19-48. EQADC FIFO DMA Summary[1]

DMA Request	Condition	Clearing Mechanism
Result FIFO Drain DMA Request	RFDEx = 1 RFDSx = 1 RFDFx = 1	The EQADC automatically clears the RFDFx when RFIFOx becomes empty. Writing “1” to the RFDFx bit is not allowed.
Command FIFO Fill DMA Request	CFFEx = 1 CFFSx = 1 CFFFx = 1	The EQADC automatically clears the CFFFx when CFIFOx becomes full. Writing “1” to the CFFFx bit is not allowed.

[1] For details refer to Section 19.6.2.7, "EQADC FIFO and Interrupt Status Registers (EQADC_FISR)," and Section 19.6.2.6, "EQADC Interrupt and DMA Control Registers (EQADC_IDCR)."

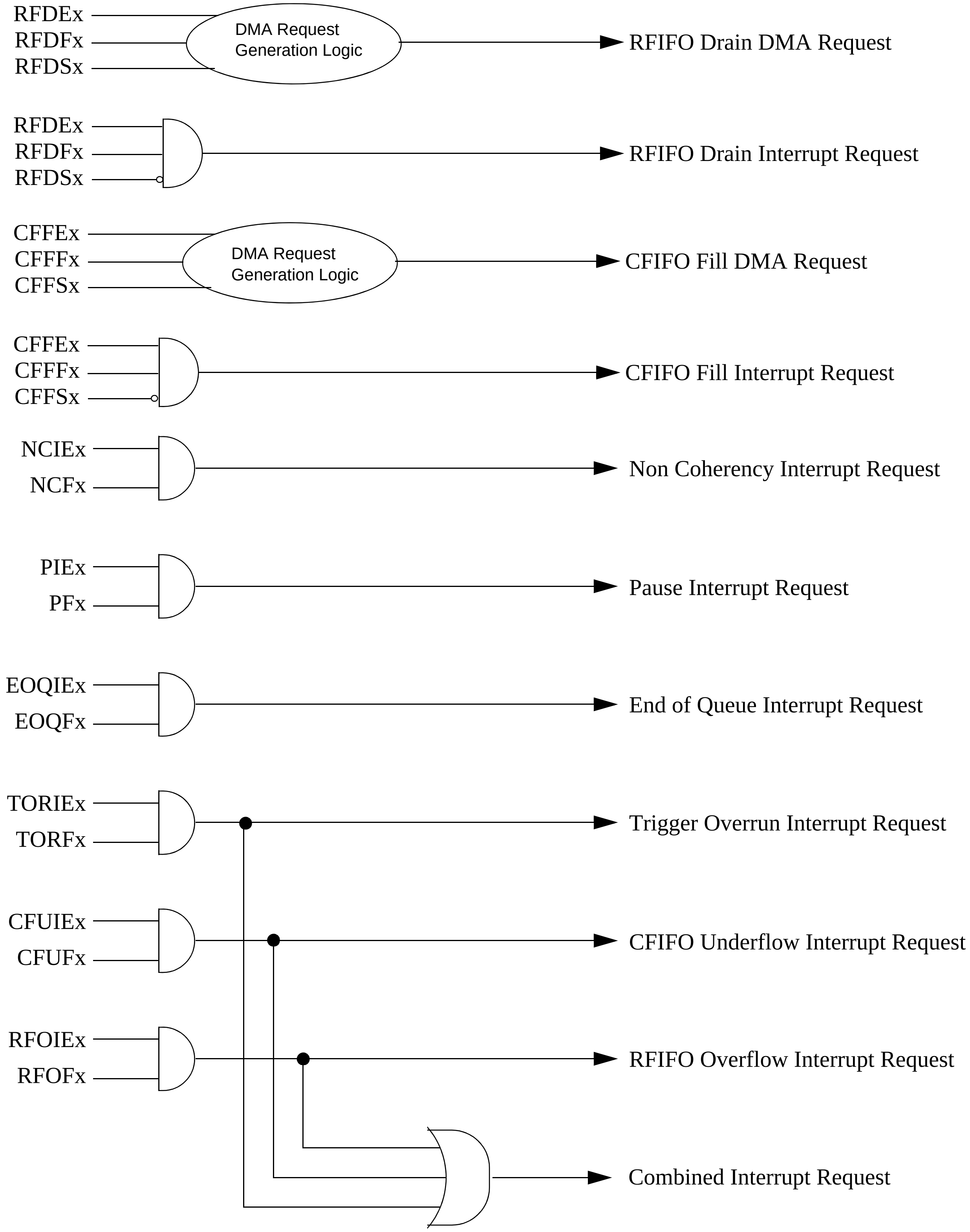


Figure 19-73. EQADC DMA and Interrupt Requests

19.7.9 Analog Sub-Block

19.7.9.1 Analog to Digital Converter (ADC)

19.7.9.1.1 ADC Architecture

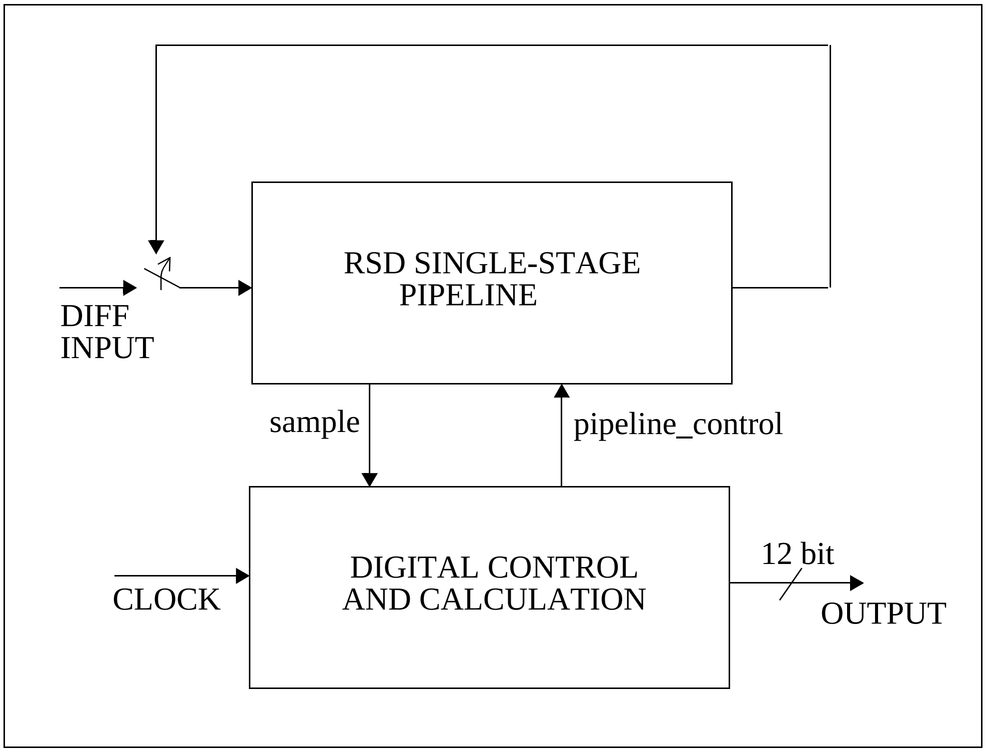


Figure 19-74. RSD ADC Block Diagram

The RSD Cyclic ADC consists of two main portions, the analog RSD Stage, and the digital control and calculation block, as shown in Figure 19-74. To begin an analog to digital conversion, a differential input is passed into the analog RSD stage. The signal is passed through the RSD stage, and then from the RSD stage output, back to its input to be passed again. To complete a 12-bit conversion, the signal must pass through the RSD stage 12 times. For 10-bit and 8-bit resolution, the signal must pass 10 or 8 times through the RSD. Each time an input signal is read into the RSD stage, a digital sample is taken by the digital control/calculation block. The digital control/calculation block uses this sample to tell the analog block how to condition the signal. The digital block also saves each successive sample and adds them according to the RSD algorithm at the end of the entire conversion cycle.

19.7.9.1.2 RSD Overview

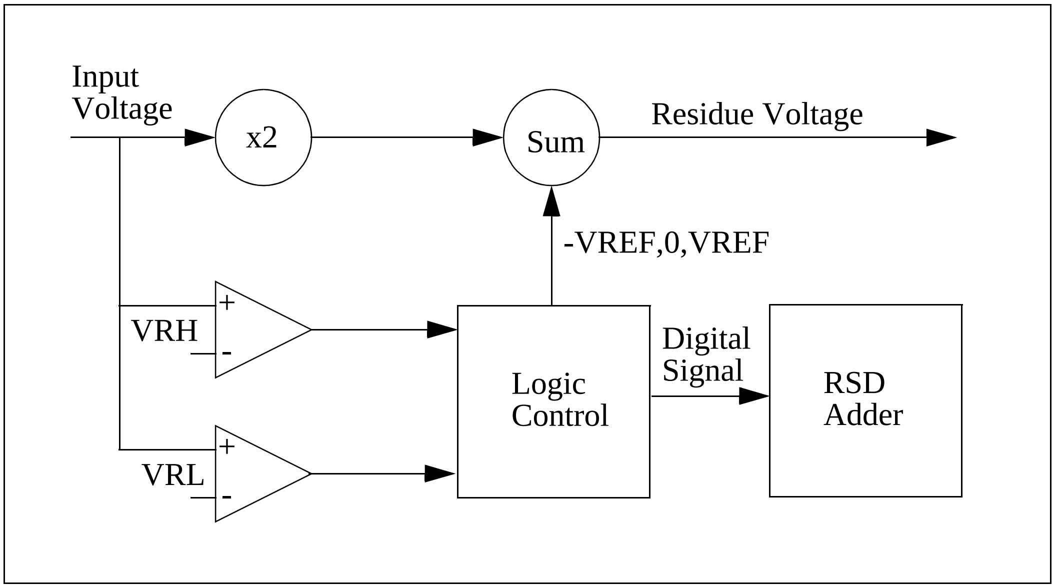


Figure 19-75. RSD Stage Block Diagram

On each pass through the RSD stage, the input signal will be multiplied by exactly two, and summed with either -VREF, 0, or VREF, depending on the Logic Control. The Logic Control will determine -VREF, 0, or VREF depending on the two comparator inputs. As the Logic Control sets the summing operation, it also sends a digital value to the RSD adder. Each time an analog signal passes through the RSD single-stage, a digital value is collected by the RSD adder. At the end of an entire AD conversion cycle, the RSD adder uses these collected values to calculate the 12-bit/10-bit/8-bit digital output.

Figure 19-76 shows the transfer function for the RSD stage. Note how the digital value (a, b) is dependent on the two comparator inputs.

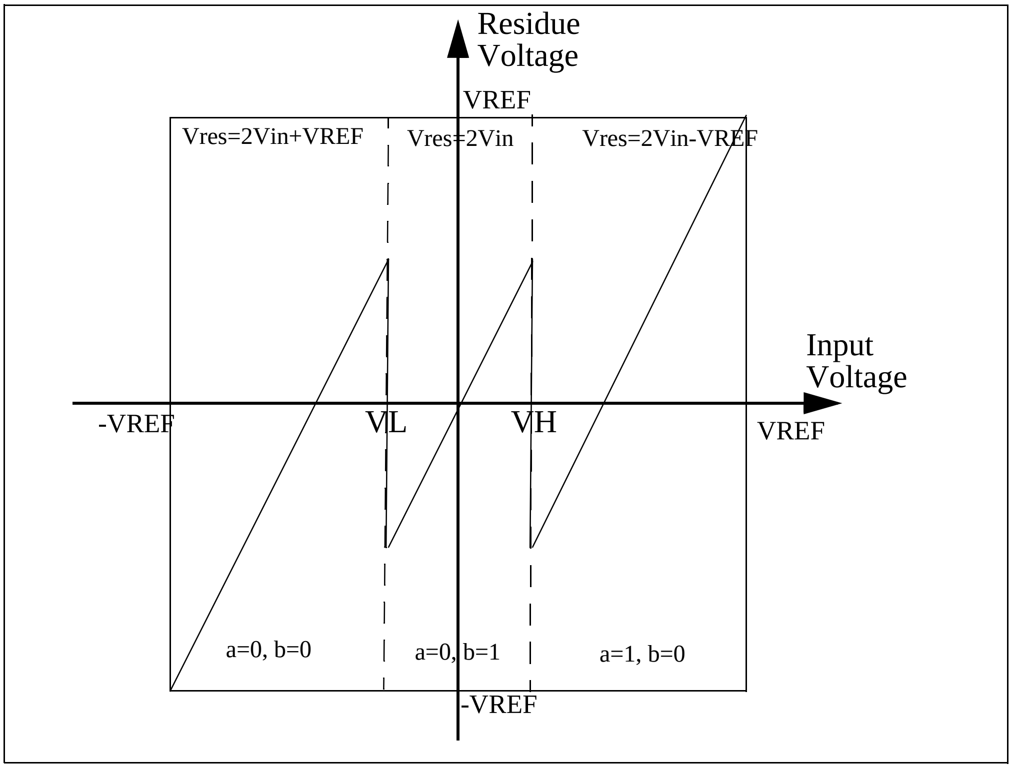


Figure 19-76. RSD Stage Transfer Function

In each pass through the RSD stage, the residue will be sent back to be the new input, and the digital signals, a and b, will be stored. For the 12-bit ADC, input signal is sampled during the input phase, and after each of the 12 passes through the RSD stage. Thus, 13 total a and b values are collected. Upon collecting all these values, they will be added according to the RSD algorithm to create the 12-bit digital representation of the original analog input. The bits are added in the following manner:

19.7.9.1.3 RSD Adder

The array, s1 to s12,will be the digital output of the RSD ADC with s1 being the MSB and s12 being the LSB.

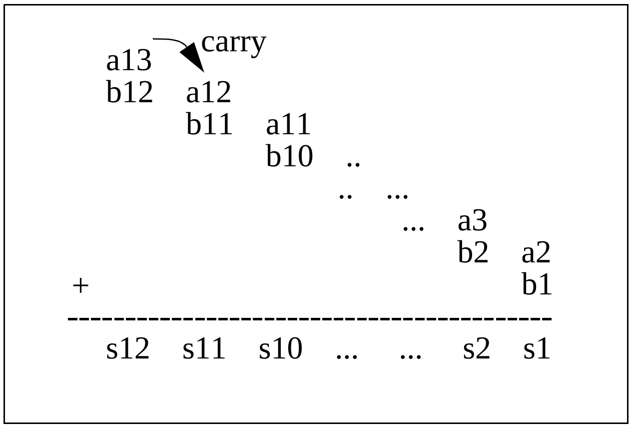


Figure 19-77. RSD Adder

19.7.9.1.4 Variable Gain Amplification (VGA) for Pre-gain

The VGA starts after sampling completes. It is enabled by a 2-bit signal PRE_GAIN described in Section 19.6.3.6, “Alternate Configuration 1-14 Control Registers (ADC_ACR1-14).”

The ADC takes 2, 8, 64 or 128 clock cycles to do sampling which is selected by the LST[0:1] field in the conversion command message. After the sampling, if 2x VGA is enabled, there is a 2x gain stage without comparison before the regular conversion cycles. When 4x VGA is enabled, there are the 2x gain stage without comparison by 2 times before the normal conversion processing.

NOTE

The VGA feature is only supported for 12-bit differential conversions. It is not disabled for 8 or 10 bit differential conversions, but the results will only be correct for 12-bit. The VGA is not available for single-ended conversions.

19.8 Initialization/Application Information

19.8.1 Multiple Queues Control Setup Example

This section provides an example of how to configure multiple CQueues. Table 19-49 describes how each CQueue can be used for a different application. Also documented in this section are general guidelines on how to initialize the on-chip ADCs and the external device, and how to configure the CQueues and the EQADC.

Table 19-49. Application of Each CQueue

CQueue Number	CQueue Type	Running Speed	Number of Contiguous Conversions	Example
0	Very fast burst time-based CQueue	every 2 µs for 200 µs; pause for 300 µs and then repeat	2	Injector current profiling
1	Fast hardware-triggered CQueue	every 900 µs	3	Current sensing of PWM controlled actuators
2	Fast repetitive time-based CQueue	every 2 ms	8	Throttle position
3	Software-triggered CQueue	every 3.9 ms	3	Command triggered by software strategy
4	Repetitive angle-based CQueue	every 625 µs	7	Airflow read every 30 degrees at 8000 RPM
5	Slow repetitive time-based CQueue	every 100 ms	10	Temperature sensors

19.8.1.1 EQADC Initialization

The following steps provide an example about how to configure the EQADC controls and how to initialize the on-chip ADCs. In this example, all conversion commands will be transferred through CFIFO0.

1. Load all required configuration commands in the RAM in such way that they form a queue; this data structure will be referred below as CQueue0. Figure 19-78 shows an example of a CQueue able to configure the on-chip ADCs at the same time. Although, this example uses the DMAC to store commands in CFIFO0, configuration commands could have also been directly written to the CFIFO0 push register.
2. Select source driving EQADC hardware trigger ports (ETRIG). Before proceeding to next step, allow some time (minimum of two peripheral clocks - filter period is set to minimum after reset) so that the logic level at the source is filtered and reaches the EQADC control logic.

NOTE

ETRIG ports could be driven by an external pin or by the output port of other blocks in the device, such as timers. In order to avoid unexpected triggering of CFIFOs in hardware trigger modes, the source driving the ETRIG port must be selected and set to a known logic level before putting the CFIFOs into the WAITING FOR TRIGGER state.

3. Configure Section 19.6.2.2, “EQADC External Trigger Digital Filter Register (EQADC_ETDFR).”
4. Configure the DMAC to transfer data from CQueue0 to CFIFO0 in the EQADC.
5. Configure Section 19.6.2.6, “EQADC Interrupt and DMA Control Registers (EQADC_IDCR).”
    a. Set CFFS0 to configure the EQADC to generate a DMA request to load commands from CQueue 0 to the CFIFO0.

   b. Set CFFE0 to enable the EQADC to generate a DMA request to transfer commands from CQueue0 to CFIFO0; Command transfers from the RAM to the CFIFO0 will start immediately.
   c. Set EOQIE0 to enable the EQADC to generate an interrupt after transferring all of the commands of CQueue0 through CFIFO0.
6. Configure Section 19.6.2.5, “EQADC CFIFO Control Registers (EQADC_CFCR).”
   a. Write 0b0001 to the MODE0 field in EQADC_CFCR0 to program CFIFO0 for software single-scan mode.
   b. Write “1” to SSE0 to assert SSS0 and trigger CFIFO0.
7. Since CFIFO0 is in single-scan software mode and it is also the highest priority CFIFO, the EQADC starts to transfer configuration commands to the on-chip ADCs and to the external device.
8. When all of the configuration commands have been transferred, CF0 in Section 19.6.2.7, “EQADC FIFO and Interrupt Status Registers (EQADC_FISR),” will be set. The EQADC generates a End of Queue interrupt. The initialization procedure is complete.

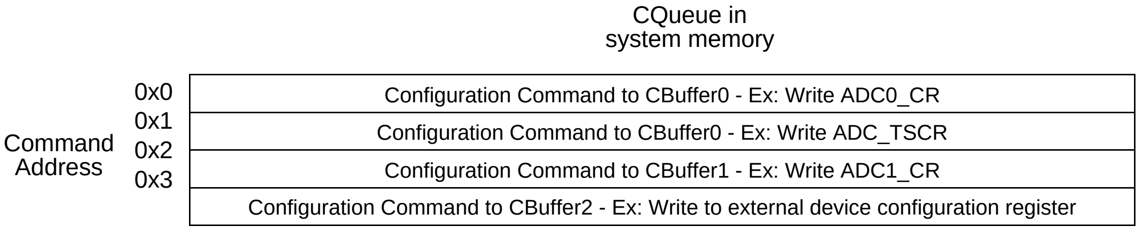


Figure 19-78. Example of a CQueue Configuring the On-Chip ADCs/External Device

The initialization procedure described above does not generate ADC clocks that are in phase because the timing at which the ADC0/1_EN bits, in the Section 19.6.3.1, “ADC0/1 Control Registers (ADC0_CR and ADC1_CR),” are set is different. Below follows an example on how to simultaneously set these bits so that in-phase ADC clocks are generated. In this example, ADC0/1_CLK are configured to the same frequency.

1. Push an ADC0_CR write configuration command in CFIFO0 that enables ADC0 (ADC0_EN=1) and that sets the ADC0_CLK_PS to an appropriate value. For example, 0x80800801.
2. Push an ADC1_CR write configuration command in CFIFO1 that enables ADC1 (ADC1_EN=1) and that sets the ADC1_CLK_PS to an appropriate value. For example, 0x82800801.
3. Configure CFIFO0 and CFIFO1 to single scan software trigger mode and simultaneously trigger them by writing 0x04100410 to the EQADC_CFCR0 register - see Section 19.6.2.5, “EQADC CFIFO Control Registers (EQADC_CFCR).”

19.8.1.2 Configuring EQADC for Applications

This section provides an example based on the applications in Table 19-49. The example describes how to configure multiple CQueues to be used for those applications and provides a step-by-step procedure to configure the EQADC and the associated CQueue structures. In the example, the “Fast hardware-triggered CQueue”, described on the second row of Table 19-49, will have its commands transferred to CBuffer1;

the conversion commands will be executed by ADC1. The generated results will be returned to RFIFO3 before being transferred to the RQueues in the RAM by the DMAC.

NOTE

> There is no fixed relationship between CFIFOs and RFIFOs with the same number. The results of commands being transferred through CFIFO1 can be returned to any RFIFO, regardless of its number. The destination of a result is determined by the MESSAGE_TAG field of the command that requested the result. See Section 19.7.2.2, “Message Format in EQADC,” for details.

Step One: Setup the CQueues and RQueues.

1. Load the RAM with configuration and conversion commands. Table 19-50 is an example of how CQueue1 commands should be set.
    a. Each trigger event will cause four commands to be executed. When the EQADC detects the Pause bit asserted, it will wait for another trigger to restart transferring commands from the CFIFO.
    b. At the end of the CQueue, the “EOQ” bit is asserted as shown in Table 19-50.
    c. Results will be returned to RFIFO3 as specified in the MESSAGE_TAG field of commands.
2. Reserve memory space for storing results.

Table 19-50. Example of CQueue Commands[1]

Bit #	0	1	2	3 4	5	6	7	8 9 10 11	12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Bit Name	EOQ	PAUSE	REP	RESERVED	EB	BN	RESERVED	MESSAGE TAG	ADC COMMAND
CMD 1	0	0	0	0	0	1	0	0b0011	Conversion Command
CMD 2	0	0	0	0	0	1	0	0b0011	Conversion Command
CMD 3	0	0	0	0	0	1	0	0b0011	Conversion Command
CMD 4	0	1	0	0	0	1	0	0b0011 [2]	Configure peripheral device for next conversion sequence
CMD 5	0	0	0	0	0	1	0	0b0011	Conversion Command
CMD 6	0	0	0	0	0	1	0	0b0011	Conversion Command
CMD 7	0	0	0	0	0	1	0	0b0011	Conversion Command
CMD 8	0	1	0	0	0	1	0	0b0011 [2]	Configure peripheral device for next conversion sequence
							etc............		
CMD EOQ	1	0	0	0	0	1	0	0b0011	EOQ Message

CFIFO Header

ADC Command

[1] Fields LST, TSR, FMT, and CHANNEL_NUMBER are not showed for clarity. See "Conversion Command Format for the Standard Configuration"," for details.

[2] MESSAGE_TAG field is only defined for read configuration commands.

Step Two: Configure the DMAC to handle data transfers between the CQueues/RQueues in RAM and the CFIFOs/RFIFOs in the EQADC.

1. For transferring, set the source address of the DMAC to point to the start address of CQueue1. Set the destination address of the DMAC to point to EQADC_CFPR1. Refer to Section 19.6.2.3, "EQADC CFIFO Push Registers (EQADC_CFPR)."
2. For receiving, set the source address of the DMAC to point to EQADC_RFPR3. Refer to Section 19.6.2.4, "EQADC Result FIFO Pop Registers (EQADC_RFPR)." Set the destination address of the DMAC to point to the starting address of RQueue1.

Step Three: Configure the EQADC Control Registers.

3. Configure Section 19.6.2.6, "EQADC Interrupt and DMA Control Registers (EQADC_IDCR)."
    a. Set EOQIE1 to enable the End of Queue Interrupt request.
    b. Set CFFS1 and RFDS3 to configure the EQADC to generate DMA requests to push commands into CFIFO1 and to pop result data from RFIF03.
    c. Set CFINV1 to invalidate the contents of CFIFO1.
    d. Set RFDE3 and CFFE1 to enable the EQADC to generate DMA requests. Command transfers from the RAM to the CFIFO1 will start immediately.
    e. Set RFOIE3 to indicate if RFIFO3 overflows.
    f. Set CFUIE1 to indicate if CFIFO1 underflows.
4. Configure MODE1 to continuous-scan rising edge external trigger mode in Section 19.6.2.5, "EQADC CFIFO Control Registers (EQADC_CFCR)."

Step Four: Command transfer to ADCs and Result data reception.

When an external rising edge event occurs for CFIFO1, the EQADC automatically will begin transferring commands from CFIFO1 when it becomes the highest priority CFIFO trying to send commands to CBuffer1. The received results will be placed in RFIFO3 and then moved to RQueue1 by the DMAC.

19.8.2 EQADC/DMAC Interface

This section provides an overview about the EQADC/DMAC interface and general guidelines about how the DMAC should be configured in order for it to correctly transfer data between the queues in system memory and the EQADC FIFOs.

NOTE

Advanced DMACs provide more functionality then the ones discussed in this section.

19.8.2.1 CQueue/CFIFO Transfers

In transfers involving CQueues and CFIFOs, the DMAC moves data from a queued source to a single destination as showed in Figure 19-79. The location of the data to be moved is indicated by the source address, and the final destination for that data, by the destination address. The DMAC contains a data structure containing these addresses and other parameters used in the control of data transfers. For every DMA request issued by the EQADC, the DMAC has to be configured to transfer a single command (32-bit data) from the CQueue, pointed to by the source address, to the CFIFO push register, pointed to by the destination address. After the service of a DMA request is completed, the source address has to be updated to point to the next valid command. The destination address remains unchanged. When the last command of a queue is transferred one of the following actions is recommended. Refer to the DMAC block guide for details about how this functionality is supported.

- The corresponding DMA channel should be disabled. This might be desirable for CFIFOs in single scan mode.
- The source address should be updated to point to a valid command which can be the first command in the queue that has just been transferred (cyclic queue), or the first command of any other CQueue. This is desirable for CFIFOs in continuous scan mode, and at some cases, for CFIFOs in single scan mode.

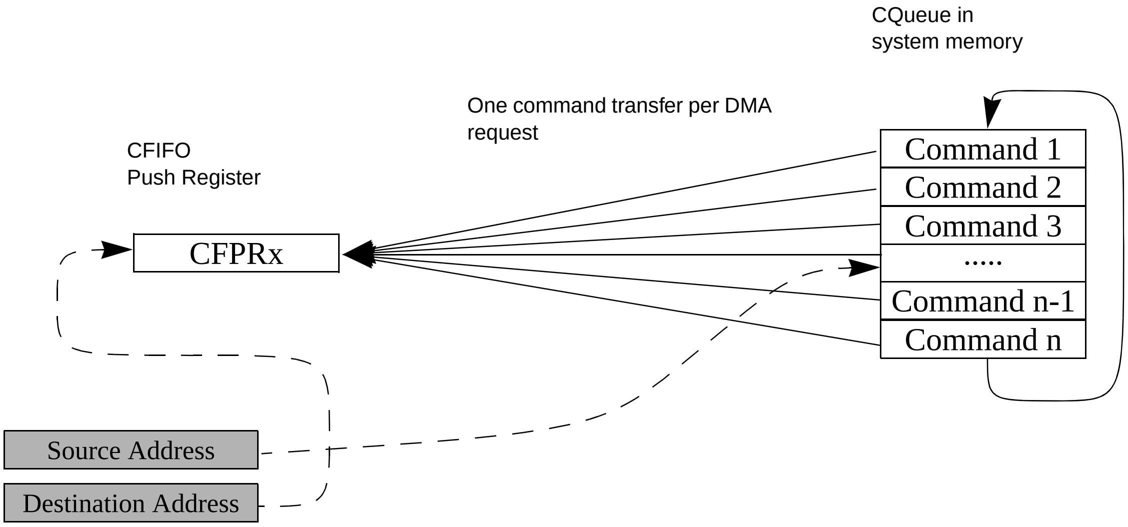


Figure 19-79. CQueue/CFIFO Interface

19.8.2.2 RQueue/RFIFO Transfers

In transfers involving RQueues and RFIFOs, the DMAC moves data from a single source to a queue destination as showed in Figure 19-80. The location of the data to be moved is indicated by the source address, and the final destination for that data, by the destination address. For every DMA request issued by the EQADC, the DMAC has to be configured to transfer a single result (16-bit data), pointed to by the source address, from the RFIFO pop register to the RQueue, pointed to by the destination address. After the service of a DMA request is completed, the destination address has to be updated to point to the location where the next 16-bit result will be stored. The source address remains unchanged. When the last expected result is written to the RQueue, one of the following actions is recommended. Refer to the DMAC block guide for details about how this functionality is supported.

- The corresponding DMA channel should be disabled.
- The destination address should be updated to point to the next location where new coming results are stored, which can be the first entry of the current RQueue (cyclic queue), or the beginning of a new RQueue.

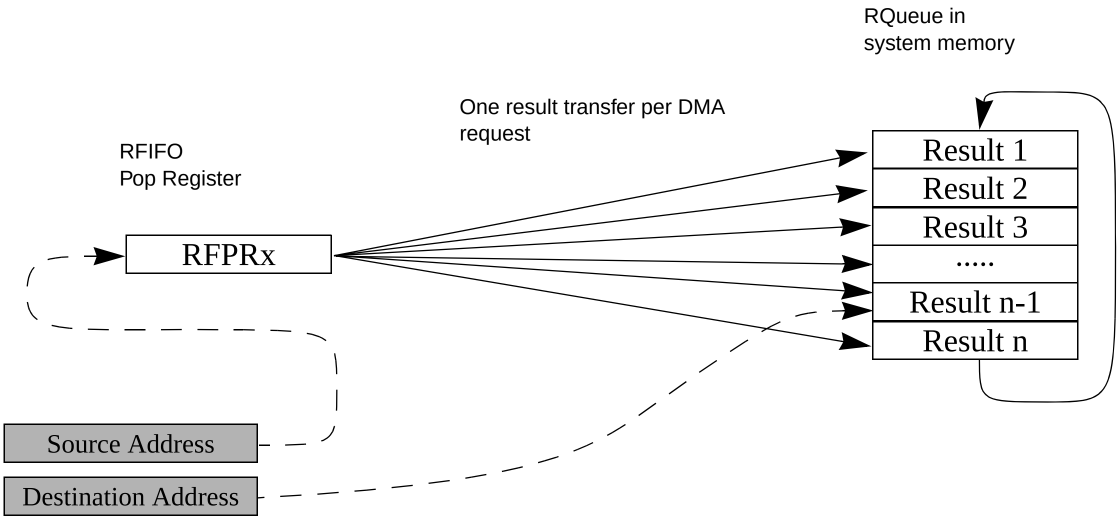


Figure 19-80. RQueue/RFIFO Interface

19.8.3 Sending Immediate Command Setup Example

In the EQADC, there is no immediate command register for sending a command immediately after writing to that register. However, a CFIFO can be configured to perform the same function as an immediate command register. The following steps illustrate how to configure CFIFO5 as an immediate command CFIFO. The results will be returned to RFIFO5.

1. Configure the Section 19.6.2.6, “EQADC Interrupt and DMA Control Registers (EQADC_IDCR).”
   a. Clear CFIFO Fill Enable5 (CFFE5 = 0) in EQADC_IDCR2.
   b. Clear CFIFO Underflow Interrupt Enable5 (CFUIE5 = 0) in EQADC_IDCR2.
   c. Clear RFDS5 to configure the EQADC to generate interrupt requests to pop result data from RFIF05.
   d. Set RFIFO Drain Enable5 (RFDE5 = 1) in EQADC_IDCR2.
2. Configure the Section 19.6.2.5, “EQADC CFIFO Control Registers (EQADC_CFCR).”
   a. Write “1” to CFINV5 in EQADC_FCR2. This will invalidate the contents of CFIFO5.
   b. Set MODE5 to Continuous-Scan Software Trigger mode in EQADC_CFCR2.
3. To transfer a command, write it to EQADC CFIFO Push Register 5 (EQADC_CFPR5) with

Message Tag = 0b0101. Refer to Section 19.6.2.3, “EQADC CFIFO Push Registers (EQADC_CFPR).”

4. Up to four commands can be queued in CFIFO5. Check the CFCTR5 status in EQADC_FISR5 before pushing another command to avoid overflowing the CFIFO. Refer to Section 19.6.2.7, “EQADC FIFO and Interrupt Status Registers (EQADC_FISR).”
5. When the EQADC receives a conversion result for RFIFO5, it generates an interrupt request. RFIFO Pop Register 5 (EQADC_RFPR5) can be popped to read the result. Refer to Section 19.6.2.4, “EQADC Result FIFO Pop Registers (EQADC_RFPR).”

19.8.4 Modifying Queues

More CQueues may be needed than the six supported by the EQADC. These additional CQueues can be supported by interrupting command transfers from a configured CFIFO, even if it is TRIGGERED and transferring, modifying the corresponding CQueue in the RAM or associating another CQueue to it, and restarting the CFIFO. More details on disabling a CFIFO are described in Section 19.7.4.6.1, “Disabled Mode.”

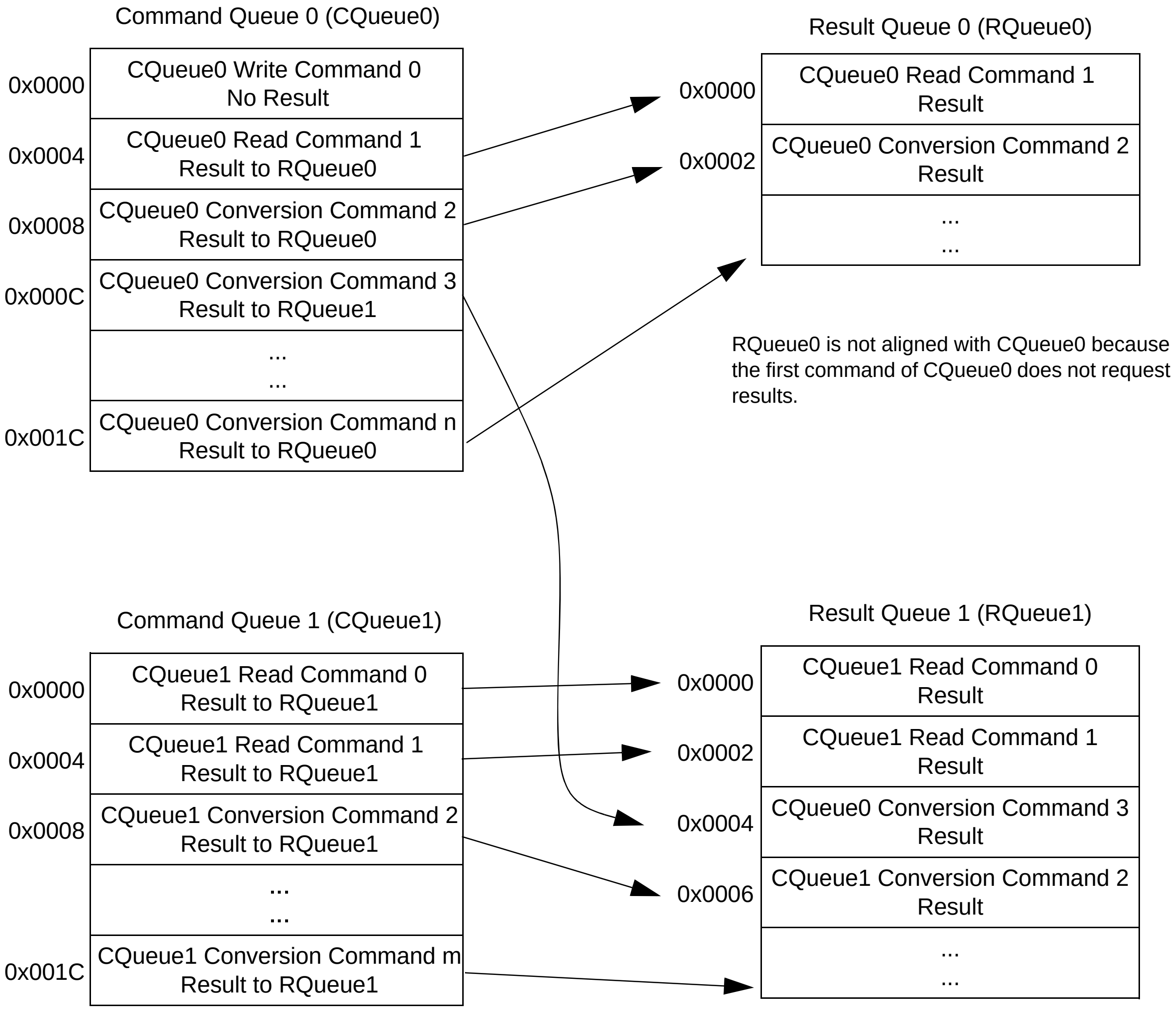
1. Determine the resumption conditions when later resuming the scan of the CQueue at the point before it was modified.
    a. Change MODEx in Section 19.6.2.5, “EQADC CFIFO Control Registers (EQADC_CFCR),” to Disabled. Refer to Section 19.7.4.6.1, “Disabled Mode,” for a description of what happens when MODEx is changed to Disabled.
    b. Poll CFSx until it becomes IDLE in Section 19.6.2.10, “EQADC CFIFO Status Register (EQADC_CFSR).”
    c. Read and save TC_CFx in Section 19.6.2.8, “EQADC CFIFO Transfer Counter Registers (EQADC_CFTCR),” for later resuming the scan of the queue. The TC_CFx provides the point of resumption.
    d. Since all result data may not have being stored in the appropriate RFIFO at the time MODEx is changed to disable, wait for all expected results to be stored in the RFIFO/RQueue before reconfiguring the DMAC to work with the modified RQueue. The number of results that must return can be estimated from the TC_CFx value obtained above.
2. Disable the DMAC from responding to the DMA request generated by CFFFx and RFDFx in Section 19.6.2.7, “EQADC FIFO and Interrupt Status Registers (EQADC_FISR).”
3. Write “0x0000” to the TC_CFx field.
4. Load the new configuration and conversion commands into RAM. Configure the DMAC to support the new CQueue/RQueue, but do not configure it yet to respond to DMA requests from CFIFOx/RFIFOx.
5. If necessary, modify Section 19.6.2.6, “EQADC Interrupt and DMA Control Registers (EQADC_IDCR),” to suit the modified CQueue.
6. Write “1” to CFINVx in Section 19.6.2.5, “EQADC CFIFO Control Registers (EQADC_CFCR),” to invalidate the entries of CFIFOx. Perform any other modifications to EQADC_CFCR except

changing MODEx from Disabled.

7. Configure the DMAC to respond to DMA requests generated by CFFFx and RFDFx.
8. Change MODEx to the modified CFIFO operation mode. Write “1” to SSEx to trigger CFIFOx if MODEx is software trigger.

19.8.5 CQueue and RQueues Usage

Figure 19-81 is an example of CQueue and RQueue usage. It shows the CQueue0 commands requesting results that will be stored in RQueue0 and RQueue1, and CQueue1 commands requesting results that will be stored only in RQueue1. Some Command Messages request data to be returned from the on-chip ADC, but some only configure them and do not request returning data. When a CQueue contains both write and read commands like CQueue0, the CQueue and RQueue entries will not be aligned; as shown in Figure 19-81, the result for the second command of CQueue0 is the first entry of RQueue0. The figure also shows that CQueue and RQueue entries can also become unaligned even if all commands in a CQueue request data as CQueue1. CQueue1 entries became unaligned to RQueue1 entries because a result requested by the forth CQueue0 command was sent to RQueue1. This happens because the system can be configured so that several CQueues can have its results sent to a single RQueue.



RQueue0 is not aligned with CQueue0 because the first command of CQueue0 does not request results.

RQueue1 is not aligned with CQueue1 because it contains results for CQueue0 and CQueue1 commands. The timing at which the CQueue0 command result is stored in RQueue1 depends on the relative speed at which commands from both CQueues are executed. This is influenced by factors like resource sharing, ADC clock frequency, sampling time, and triggering time.

Figure 19-81. EQADC Command and Result Queues

19.8.6 ADC Result Calibration

The ADC result calibration process consists of two steps: determining the gain and offset calibration constants, and calibrating the raw results generated by the on-chip ADCs by solving the following equation discussed in Section 19.7.6.6, “ADC Calibration Feature.”

$$\mathrm{CAL\_RES = GCC * RAW\_RES + OCC+2;} \qquad \textit{Eqn. 19-1}$$

The calibration constants GCC and OCC can be calculated from equation Equation 19-1 provided that two pairs of expected (CAL_RES) and measured (RAW_RES) result values are available for two different input voltages. Most likely calibration points to be used are 25% VREF[1] and 75% VREF since they are far apart but not too close to the end points of the full input voltage range. This allows for calculations of more representative calibration constants. The EQADC provides these voltages via channel numbers 43 and 44. The raw, uncalibrated results for these input voltages are obtained by converting these channels with conversion commands that have the CAL bit negated.

The transfer equations for when sampling these reference voltages are:

$\mathrm{CAL\_RES_{75\%VREF} = GCC * RAW\_RES_{75\%VREF} + OCC+2;}$

$\mathrm{CAL\_RES_{25\%VREF} = GCC * RAW\_RES_{25\%VREF} + OCC+2;}$

Thus;

$$\mathrm{GCC = (CAL\_RES_{75\%VREF} - CAL\_RES_{25\%VREF}) / (RAW\_RES_{75\%VREF} - RAW\_RES_{25\%VREF});} \qquad \textit{Eqn. 19-2}$$

$$\mathrm{OCC = CAL\_RES_{75\%VREF} - GCC*RAW\_RES_{75\%VREF} - 2;} \qquad \textit{Eqn. 19-3}$$

or

$$\mathrm{OCC = CAL\_RES_{25\%VREF} - GCC*RAW\_RES_{25\%VREF} - 2;} \qquad \textit{Eqn. 19-4}$$

After being calculated, the GCC and OCC values must be written to ADC registers: Section 19.6.3.4, “ADC0/1 Gain Calibration Constant Registers (ADC0_GCCR and ADC1_GCCR),” and Section 19.6.3.5, “ADC0/1 Offset Calibration Constant Registers (ADC0_OCCR and ADC1_OCCR),” using write configuration commands.

The EQADC will automatically calibrate the results, according to equation Equation 19-1, of every conversion command that has its CAL bit asserted using the GCC and OCC values stored in the ADC calibration registers.

19.8.6.1 MAC Configuration Procedure

The following steps illustrate how to configure the calibration hardware, namely, determining the values of the gain and offset calibration constants, and the writing of these constants to the calibration registers. The procedure below should be performed for ADC0 and for ADC1.

1. Convert channel 44 with a command that has its CAL bit negated and obtain the raw, uncalibrated result for 25%VREF ($\mathrm{RAW\_RES_{25\%VREF}}$).

---

[1] VREF=VRH-VRL

2. Convert channel 43 with a command that has its CAL bit negated and obtain the raw, uncalibrated result for 75%VREF ($RAW\_RES_{75\%VREF}$).
3. Since the expected values for the conversion of these voltages are known ($CAL\_RES_{25\%VREF}$ and $CAL\_RES_{75\%VREF}$), GCC and OCC values can be calculated from equations Equation 19-2 and Equation 19-3 using these values, and the ones determined in steps 1 and 2.
4. Reformat GCC and OCC to the proper data formats as specified in Section 19.7.6.6.2, “MAC Unit and Operand Data Format.” GCC is an unsigned 15-bit fixed point value and OCC is a signed 14-bit value.
5. Write GCC value to Section 19.6.3.4, “ADC0/1 Gain Calibration Constant Registers (ADC0_GCCR and ADC1_GCCR),” and OCC value to Section 19.6.3.5, “ADC0/1 Offset Calibration Constant Registers (ADC0_OCCR and ADC1_OCCR),” using write configuration commands.

19.8.6.2 Example

The raw results obtained when sampling reference voltages 25%VREF and 75%VREF were, respectively, 3798 and 11592. The results that should have been obtained from the conversion of these reference voltages are, respectively, 4096 and 12288. Therefore, using equations Equation 19-2 and Equation 19-3, the gain and offset calibration constants are:

GCC=(12288-4096)/(11592-3798) = 1.05106492-> 1.05102539 = 0x4388

OCC=12288 - 1.05106492*11592 - 2 = 102.06-> 102 = 0x0066

Table 19-51 shows, for this particular case, examples of how the result values change according to GCC and OCC when result calibration is executed (CAL=1) and when it is not (CAL=0).

Table 19-51. Calibration example

Input Voltage	Raw result (CAL=0)		Calibrated result (CAL=1)	
	Hexadecimal	Decimal	Hexadecimal	Decimal
25% VREF	0x0ED6	3798	0x1000	4095.794
75% VREF	0x2D48	11592	0x3000	12287.486

19.8.6.3 Quantization Error Reduction During Calibration

Figure 19-82 shows how the ADC transfer curve changes due to the addition of two to the MAC output during the calibration - see MAC output equation at Section 19.7.6.6.1, "Overview." The maximum absolute quantization error is reduced by half leading to an increase in accuracy.

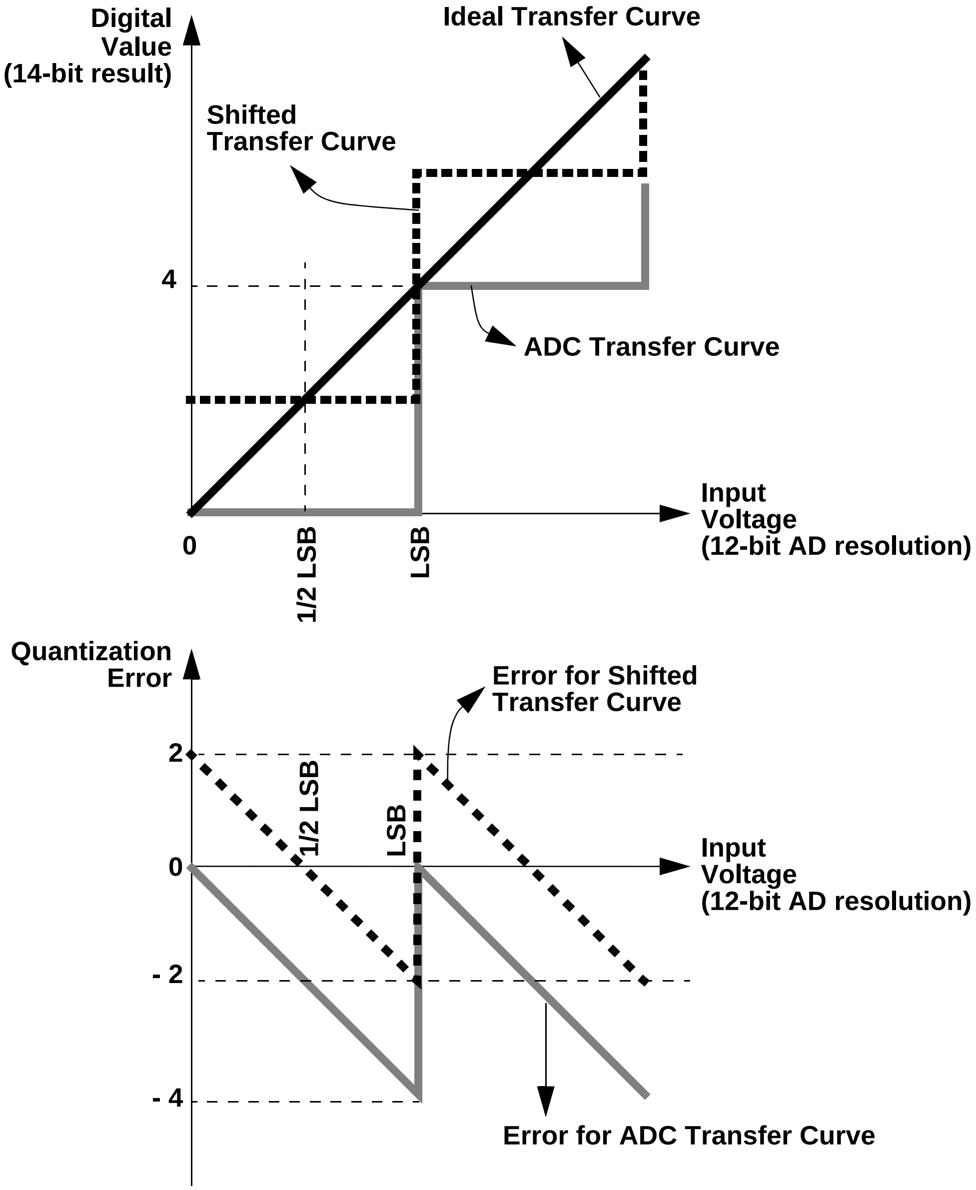


Figure 19-82. Quantization error reduction during calibration

19.8.7 EQADC Versus QADC

This section describes how the EQADC upgrades the QADC functionality. The section also provides a comparison between the EQADC and QADC in terms of their functionality. This section targets the users familiar with terminology in QADC. Figure 19-83 is an overview of a QADC. Figure 19-84 is an overview of the EQADC system.

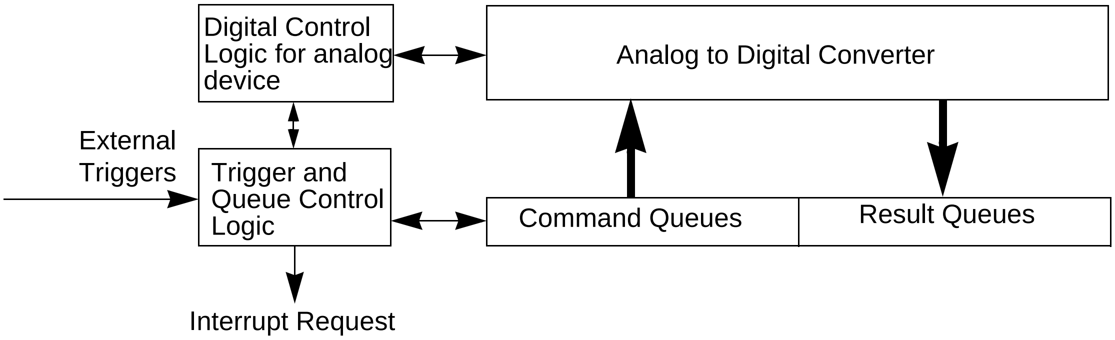


Figure 19-83. QADC Overview

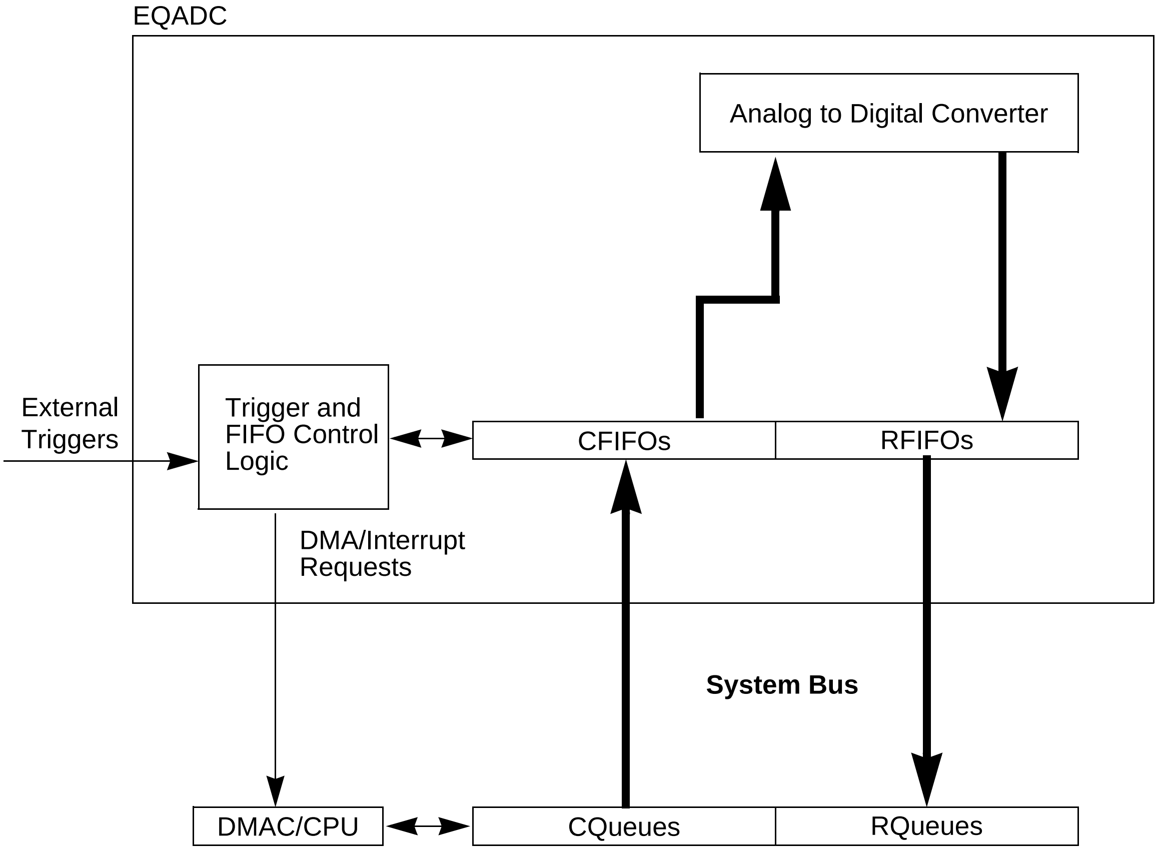


Figure 19-84. EQADC System Overview

The EQADC system consists of these parts: queues in RAM, the EQADC, and on-chip ADCs. As compared with the QADC, the EQADC system requires extra hardware.

1. A DMA or an MCU is required to move data between the EQADC's FIFOs and Queues in the system memory.

Since there are only FIFOs inside the EQADC, much of the terminology or use of the register names, register contents, and signals of the EQADC involve "FIFO" instead of "Queue". These register names, register contents, and signals are functionally equivalent to the "Queue" counterparts in the QADC. Table 19-52 lists how the EQADC register, register contents, and signals are related to QADC.

Table 19-52. Terminology Comparison between QADC and EQADC

QADC Terminology	EQADC Terminology	Function
CCW	Command Message	In the QADC, the hardware only executes conversion command words. In the EQADC, not all commands are conversion commands; some are configuration commands.
Queue Trigger	CFIFO Trigger	In the QADC, a trigger event is required to start the execution of a queue. In the EQADC, a trigger event is required to start command transfers from a CFIFO. When a CFIFO is TRIGGERED and transferring, commands are continuously moved from CQueues to CFIFOs. Thus, the trigger event initiates the "execution of a queue" indirectly.
Current Word Pointer Queue x (CWPQx)	Counter Value of Commands Transferred from Command FIFOx (TC_CFx)	In the QADC, CWPQx allows the last executed command on queue x to be determined. In the EQADC, the TC_CFx value allows the last transferred command on CQueue x to be determined.
Queue Pause Bit (P)	CFIFO Pause Bit	In the QADC, detecting a pause bit in the CCW will pause the queue execution. In the EQADC, detecting a pause bit in the Command will pause command transfers from a CFIFO.
Queue Operation Mode (MQx)	CFIFO Operation Mode (MODEx)	The EQADC supports all queue operation modes in the QADC except operation modes related to a periodic timer. A timer elsewhere in the system can provide the same functionality if it is connected to ETRIGx.
Queue Status (QS)	CFIFO Status (CFSx)	In the QADC, the Queue Status is read to check whether a queue is idle, active, paused, suspended, or trigger pending. In the EQADC, the CFIFO Status is read to check whether a queue is IDLE, WAITING FOR TRIGGER (idle or paused in QADC), or TRIGGERED (suspended or trigger pending in QADC). What CFIFO is currently "active" can be determined by reading the LCFTCBz field on the EQADC_CFSSR registers.

The EQADC and QADC also have similar procedures for the configuration or execution of applications. Table 19-53 shows the steps required for the QADC versus the steps required for the EQADC system.

Table 19-53. Usage Comparison between QADC and EQADC System

Procedure	QADC	EQADC System
Analog Control Configuration	Configure analog device by writing to the QADC registers.	Program configuration commands into command queues.
Prepare Scan Sequence	Program scan commands into command queues.	Program scan commands into command queues.
Queue Control Configuration	Write to the QADC Control Registers.	Write to the EQADC Control Registers.
Data Transferred between Queues and Buffers	Not Required.	Program the DMAC or the CPU to handle the data transfer.
Queue Execution	Require Software or External Trigger events to start queue execution.	Require Software or External Trigger events to start command transfers from a CFIFO.

Chapter 20
Enhanced Serial Communications Interface (eSCI)

20.1 Introduction

The eSCI block is an enhanced SCI block with a LIN master interface layer and DMA support. The LIN master layer complies with the specifications LIN 1.3, LIN 2.0, and SAE J2602/1.

20.1.1 Bibliography

- LIN Specification Package Revision 1.3; December 12, 2002
- LIN Specification Package Revision 2.0; September 23, 2003
- LIN Network for Vehicle Applications, SAE J2602/1, September 1, 2005

20.1.2 Acronyms and Abbreviations

Table 20-1 contains acronyms and abbreviations used in this document.

Table 20-1. Acronyms and Abbreviations

Term	Description
eSCI	Enhanced SCI block with LIN support and DMA support
SCI	Serial Communications Interface
LIN	Local Interconnect Network - A protocol for low-cost automobile networks
LIN PE	LIN Protocol Engine, Finite State Machine to control logic of the LIN hardware.
MCLK	Module Clock, defined in Section 20.4.3.1, “Module Clock”
TCLK	Transmitter Clock, defined in Section 20.4.3.2, “Transmitter Clock”
RCLK	Receiver Clock, defined in Section 20.4.3.3, “Receiver Clock”
RSC	Receiver Sample Counter, defined in Section 20.4.3.3, “Receiver Clock”

20.1.3 Glossary

Table 20-2. Glossary

Term	Definition
Logic level one	The voltage that corresponds to Boolean true (1) state.
Logic level zero	The voltage that corresponds to Boolean false (0) state.

Table 20-2. Glossary (continued)

Term	Definition
Set	To set a bit or bits means to establish logic level one on the bit or bits.
Clear	To clear a bit or bits means to establish logic level zero on the bit or bits.
Asserted	A signal that is asserted is in its active state. An active low signal changes from logic level one to logic level zero when asserted, and an active high signal changes from logic level zero to logic level one.
Preamble	The term preamble describes an idle character which is *transmitted* by the eSCI module.
Bit time	Duration of a single bit in a transmitted byte field or character, equivalent to the duration of one transmitter clock cycle defined in Section 20.4.3.2, “Transmitter Clock”
frame	Entity that consists of the start bit followed by payload bits followed by one ore more stop bits
LIN byte field	Special instance of a frame
SCI frame	Special instance of a frame
LIN frame	Sequence of break character followed by LIN byte fields
LIN TX frame	A LIN frame with the frame header, data byte fields, and checksum field transmitted by the eSCI module
LIN RX frame	A LIN frame with the header field transmitted by the eSCI module and the data byte fields and checksum field received by the eSCI module
module is idle	Module is idle, described in Section 20.1.6.1, “Module Idle Condition”

20.1.4 Overview

The eSCI block allows asynchronous serial communications with peripheral devices and other CPUs. It includes special support to interface to LIN slave devices.

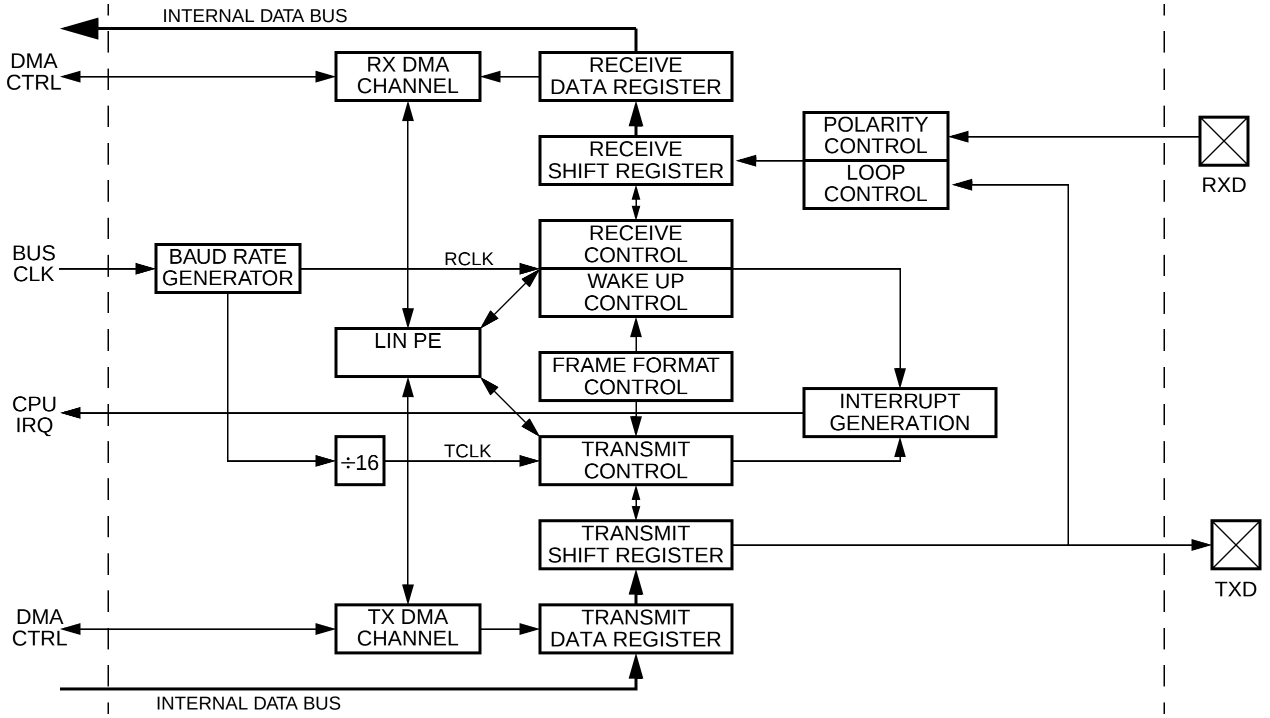


Figure 20-1. eSCI Block Diagram

20.1.5 Features

The eSCI block includes these distinctive features:

- Full-duplex operation
- Standard mark/space non-return-to-zero (NRZ) format
- 13-bit baud rate selection
- Programmable frame, payload, and character format
- Support of 2 stop bits in receiver path
- Hardware parity generation and checking
  - Programmable even or odd parity
- Programmable polarity of RXD pin
- Separately enabled transmitter and receiver
- Two receiver wake-up methods:
  - Idle line wake up
  - Address mark wake up
- Interrupt-driven operation with eight flags:

  - Transmitter empty
  - Transmission complete
  - Receiver full
  - Idle receiver input
  - Receiver overrun
  - Noise error
  - Framing error
  - Parity error
- Receiver framing error detection
- 1/16 bit-time noise detection
- 2 channel DMA interface
- LIN support
  - LIN Master Node functionality (master and slave task)
  - Compatible with LIN slaves from revisions 1.x and 2.0 of the LIN standard
  - Detection of Bit Errors, Physical Bus Errors and Checksum Errors
  - All status bit can generate maskable interrupts
  - Application layer CRC support
  - Programmable CRC polynom
  - Detection and generation of wake-up characters
  - Programmable wake-up delimiter time
  - Programmable slave timeout
  - Can be configured to include header bits in checksum
  - LIN DMA interface

20.1.6 Modes of Operation

The eSCI module has two functional operational modes, SCI and LIN mode, and low power modes. The availability of register bits and fields depends on the selected operational mode.

20.1.6.1 Module Idle Condition

Some modes can only be entered if the module is idle. The module is idle if

- all five active status bits DACT, WACT, LACT, TACT, and RACT in the Interrupt Flag and Status Register 1 (eSCI_IFSR1) are 0, and
- no interrupt request is pending, i.e either the interrupt flag or its related interrupt enable is 0.

To ensure that the module goes idle, the application should clear all interrupt enable bits before triggering the mode change.

20.1.6.2 SCI Mode

The SCI mode is the default functional operational mode and is described in Section 20.4.5, “SCI Mode”.

20.1.6.3 LIN Mode

The LIN mode is the second functional operational mode and is described in Section 20.4.6, “LIN Mode”.

20.1.6.4 Disabled Mode

In the Disabled mode the eSCI module indicates to the clocking system, that all module clocks can be turned off.

The eSCI module is in the Disabled Mode, if the MDIS bit in the Control Register 2 (eSCI_CR2) is set and the module is idle.

20.2 External Signal Description

The eSCI module is connected two a total of two external pins.

20.2.1 Detailed Signal Descriptions

20.2.1.1 eSCI Transmit Pin (TXD)

This pin serves as transmit data output and as the receive data input of eSCI.

20.2.1.2 eSCI Receive Pin (RXD)

This pin serves as receive data input of the eSCI.

20.3 Memory Map and Register Definition

This section provides the memory map and a detailed description of the memory mapped registers.

20.3.1 Memory Map

Table 20-3 provides a 32-bit view of the eSCI’s memory map. This table gives the register offset w.r.t. module base address eSCI_BASE.

Table 20-3. eSCI 32-bit Memory Map

Offset	Register	
0x0000	Baud Rate Register (eSCI_BRR)	Control Register 1 (eSCI_CR1)
0x0004	Control Register 2 (eSCI_CR2)	SCI Data Register (eSCI_SDR)
0x0008	Interrupt Flag and Status Register 1 (eSCI_IFSR1)	Interrupt Flag and Status Register 2 (eSCI_IFSR2)

Table 20-3. eSCI 32-bit Memory Map (continued)

Offset	Register	
0x000C	LIN Control Register 1 (eSCI_LCR1)	LIN Control Register 2 (eSCI_LCR2)
0x0010	LIN Transmit Register (eSCI_LTR)	Reserved
0x0014	LIN Receive Register (eSCI_LRR)	Reserved
0x0018	LIN CRC Polynomial Register (eSCI_LPR)	Control Register 3 (eSCI_CR3)
0x001C	Reserved	

Table 20-4 provides a key for register figures and tables.

Table 20-4. Register Conventions

Convention	Description
▨	Depending on its placement in the read or write row, indicates that the bit is not readable or not writeable.
FIELDNAME	Identifies the field. Its presence in the read or write row indicates that it can be read or written.
Register Field Types	
rwm	A read/write bit that may be modified by a eSCI module in some fashion other than by a reset.
w1c	Write one to clear. A flag bit that can be read, is cleared by writing a one, writing 0 has no effect.
Reset Values	
0	Resets to zero.
1	Resets to one.

20.3.2 Register Descriptions

This section consists of register descriptions in address order. Each description includes a standard register diagram with an associated figure number. Writes to a reserved register location do not have any effect and reads of these locations return a zero. Details of register bit and field function follow the register diagrams, in bit order.

20.3.2.1 Baud Rate Register (eSCI_BRR)

eSCI_BASE + 0x0000 Write: Anytime

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	R			SBR												
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

Figure 20-2. Baud Rate Register (eSCI_BRR)

This register provides the control value for the serial baud rate. The baud rate and clock generation is specified in Section 20.4.3, “Baud Rate and Clock Generation”.

A byte write access to only the upper byte of this register (eSCI_BRR[0:715:8]) will not change the content of the register, instead, the written byte is stored internally into a shadow register. A subsequent byte write access to only the lower byte of this register (eSCI_BRR[8:157:0]) updates the lower byte and copies the content of the shadow register into the upper byte.

A byte write access to only the lower byte of this register (eSCI_BRR[8:157:0]) without a preceding byte write access to only the upper byte copies a value of all zero into the upper byte.

A word write access to this register updates both the lower and upper byte immediately and is the recommended write access type for this register.

Table 20-5. eSCI_BRR Field Descriptions

Field	Description
R	Reserved. These bits are reserved. They are read as 0. Application must not write 1 to these bits.
SBR	Serial Baud Rate. This field provides the baud rate control value SBR.

20.3.2.2 Control Register 1 (eSCI_CR1)

eSCI_BASE + 0x0002 Write: Anytime

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LOOPS	R	RSRC	M	WAKE	0	PE	PT	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
W						▨									rwm	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-3. Control Register 1 (eSCI_CR1)

This register provides bits to configure the functionality of the module, provides the interrupt enable bits for the interrupt flags provided in Interrupt Flag and Status Register 1 (eSCI_IFSR1) and provides the control bits for the transmitter and receiver.

Table 20-6. eSCI_CR1 Field Descriptions

Field	Description
LOOPS	Loop Mode Select. This control bit together with the RSRC control bit defines the receiver source mode. The mode coding is defined in Table 20-7 and the modes are described in Section 20.4.5.3.2, “Receiver Input Mode Selection”.
R	Reserved. This bit is reserved. It is read as 0. Application must not write 1 to these bits.
RSRC	Receiver Source Control. This control bit together with the LOOPS control bit defines the receiver source mode. The mode coding is defined in Table 20-7 and the modes are described in Section 20.4.5.3.2, “Receiver Input Mode Selection”.
M	Frame Format Mode. This control bit together with the M2 bit of the Control Register 3 (eSCI_CR3) controls the frame format used. The supported frame formats and the related settings are defines in Section 20.4.2, “Frame Formats”.
WAKE	Receiver Wake-up Condition. This control bit defines the wake-up condition for the receiver. The receiver wake-up is described in Section 20.4.5.5, “Multiprocessor Communication”. 0 Idle line wake-up. 1 Address mark wake-up
PE	Parity Enable. This control bit enables the parity bit generation and checking. The location of the parity bits is shown in Section 20.4.2, “Frame Formats”. 0 Parity bit generation and checking disabled. 1 Parity bit generation and checking enabled.
PT	Parity Type. This control bit defines whether even or odd parity has to be used. 0 Even parity (even number of ones in character clears the parity bit). 1 Odd parity (odd number of ones in character clears the parity bit).
TIE	Transmitter Interrupt Enable. This bit controls the eSCI_IFSR1[TRDE] interrupt request generation. 0 TDRE interrupt request generation disabled. 1 TDRE interrupt request generation enabled.
TCIE	Transmission Complete Interrupt Enable. This bit controls the eSCI_IFSR1[TC] interrupt request generation. 0 TC interrupt request generation disabled. 1 TC interrupt request generation enabled.
RIE	Receiver Full Interrupt Enable. This bit controls the eSCI_IFSR1[RDRF] interrupt request generation. 0 RDRF interrupt request generation disabled. 1 RDRF interrupt request generation enabled.
ILIE	Idle Line Interrupt Enable. This bit controls theeSCI_IFSR1[IDLE] interrupt request generation. 0 IDLE interrupt request generation disabled. 1 IDLE interrupt request generation enabled.
TE	Transmitter Enable. This control bit enables and disables the transmitter. The control features of the transmitter are described in Section 20.4.5.2.1, “Transmitter States and Transitions”. 0 Transmitter disabled. 1 Transmitter enabled.
RE	Receiver Enable.This control bit enables and disables the receiver. The control features of the receiver are described in Section 20.4.5.3.1, “Receiver States and Transitions”. 0 Receiver disabled. 1 Receiver enabled.

Table 20-6. eSCI_CR1 Field Descriptions (continued)

Field	Description
RWU	Receiver Wake-Up Mode. This bit controls and indicates the receiver wake-up mode, which is described in Section 20.4.5.5, “Multiprocessor Communication”. 0 Normal receiver operation. 1 Receiver is in wake-up mode. **Note:** This bit should be set in SCI mode only.
SBK	Send Break Character. This bit controls the transmission of break characters, which is described in Section 20.4.5.2.7, “Break Character Transmission”. 0 No break characters will be transmitted. 1 Break characters will be transmitted. **Note:** This bit should be set in SCI mode only.

Table 20-7. Receive Source Mode Selection

LOOPS	RSCR	Receiver Input Mode
0	0	Dual Wire Mode
0	1	Reserved
1	0	Loop Mode
1	1	Single Wire Mode

20.3.2.3 Control Register 2 (eSCI_CR2)

eSCI_BASE + 0x0004 Write: Anytime

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MDIS	FBR	BSTP	BERRIE	RXDMA	TXDMA	BRCL	TXDIR	BESM	BESTP	RXPOL	PMSK	ORIE	NFIE	FEIE	PFIE
Reset	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-4. Control Register 2 (eSCI_CR2)

This register provides bits to configure the functionality of the module, and interrupt enable bits for the interrupt flags provided in Interrupt Flag and Status Register 1 (eSCI_IFSR1) and control bits for the transmitter and receiver.

Table 20-8. eSCI_CR2 Field Descriptions

Field	Description
MDIS	Module Disabled Mode. This bit controls the Module Mode of Operation, which is described in Section 20.1.6, “Modes of Operation” 0 Module is not in Disabled Mode. 1 Module is in Disabled Mode, if module is idle.
FBR	Fast Bit Error Detection. This bit controls the Bit Error Detection mode. 0 Standard Bit error detection performed as described in Section 20.4.6.5.3, “Standard Bit Error Detection”. 1 Fast Bit error detection performed as described in Section 20.4.6.5.4, “Fast Bit Error Detection”. **Note:** This bit is used in LIN mode only.

Table 20-8. eSCI_CR2 Field Descriptions (continued)

Field	Description
BSTP	DMA Stop on Bit Error or Physical Bus Error. This bit controls the transmit DMA requests generation in case of bit errors or physical bus errors. Bit errors are indicated by the BERR flag in the Interrupt Flag and Status Register 1 (eSCI_IFSR1) and physical bus errors are indicated by the PBERR flag in the Interrupt Flag and Status Register 2 (eSCI_IFSR2). 0 Transmit DMA requests generated regardless of bit errors or physical bus errors. 1 Transmit DMA requests are *not* generated if eSCI_IFSR1[BERR] flag or eSCI_IFSR2[PBERR] flags are set. **Note:** This bit is used in LIN mode only.
BERRIE	Bit Error Interrupt Enable. This bit controls the BERR interrupt request generation. 0 BERR interrupt request generation disabled. 1 BERR interrupt request generation enabled.
RXDMA	Receive DMA Control. This bit enables the receive DMA feature. When this bit is cleared, a pending receive DMA request is deasserted. 0 Receive DMA disabled. 1 Receive DMA enabled.
TXDMA	Transmit DMA Control. This bit enables the transmit DMA feature. When this bit is cleared, a pending transmit DMA request is deasserted. 0 Transmit DMA disabled. 1 Transmit DMA enabled.
BRCL	Break Character Length. This bit is used to define the length of the break character to be transmitted. The settings are specified in Section 20.4.2.2, “Break Character Formats”.
TXDIR	This bit has no effect. In this device, TXDIR does not control the output driver of the TXD pin in Single Wire Mode. The function is controlled by the relevant Pad Control Register in the System Integration Unit.
BESM	Fast Bit Error Detection Sample Mode. This bit defines the sample point for the Fast Bit Error Detection Mode. 0 Sample point is RS9. 1 Sample point is RS13. **Note:** This bit is used in LIN mode only.
BESTP	Bit Error Transmit Stop. This control bit defines the behavior of the eSCI Transmit Pin TXD while the bit error flag eSCI_IFSR1[BERR] is 1. 0 Application Data Values driven onto TXD pin. 1 Recessive Data Value 1 driven onto TXD pin. **Note:** This bit is used in LIN mode only.
RXPOL	RXD Pin polarity. This bit controls the polarity of the RXD pin. See Section 20.4.2.1.1, “Inverted Data Frame Formats” 0 Normal Polarity. 1 Inverted Polarity.
PMSK	Parity Bit Masking. This bit defines whether the received parity bit is presented in the related bit position in the SCI Data Register (eSCI_SDR). 0 The received parity bit is presented in the bit position related to the parity bit. 1 The value 0 is presented in the bit position related to the parity bit.
ORIE	Overrun Interrupt Enable. This bit controls the eSCI_IFSR1[OR] interrupt request generation. 0 OR interrupt request generation disabled. 1 OR interrupt request generation enabled.
NFIE	Noise Interrupt Enable. This bit controls the eSCI_IFSR1[NF] interrupt request generation. 0 NF interrupt request generation disabled. 1 NF interrupt request generation enabled.

Table 20-8. eSCI_CR2 Field Descriptions (continued)

Field	Description
FEIE	Frame Error Interrupt Enable. This bit controls the eSCI_IFSR1[FE] interrupt request generation. 0 FE interrupt request generation disabled. 1 FE interrupt request generation enabled.
PFIE	Parity Error Interrupt Enable. This bit controls the eSCI_IFSR1[PF] interrupt request generation. 0 PF interrupt request generation disabled. 1 PF interrupt request generation enabled.

20.3.2.4 SCI Data Register (eSCI_SDR)

eSCI_BASE + 0x0006 Write: Anytime

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RN	TN	ERR	0	RD[11:8]				RD[7]	RD[6:0]						
W	▨		▨	▨	▨	▨	▨	▨	TD[7]	TD[6:0]						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-5. SCI Data Register (eSCI_SDR)

In SCI mode this register is used to provide transmit data and retrieve received data. In LIN mode any write access to this register is ignored and any read access returns unspecified data. In case of data transmission this register is used to provide a part of the transmit data. In case of data reception this register provides a part of the received data and related error information.If the application writes to the lower byte of this register (eSCI_SDR[7:08:15]), the internal commit flag iCMT, which is not visible to the application, is set to indicate that the register has been updated and ready to transmit new data.

If the application reads from the lower byte of this register (eSCI_SDR[7:08:15]), a signal is send to the internal receiver unit to indicate that the register was read and is ready to receive new data. The read access will not change the content of any register.

Table 20-9. eSCI_SDR Field Descriptions

Field	Description
RN	Received Most Significant Bit. The semantic of this bit depends on the frame format selected by eSCI_CR3[M2], eSCI_CR1[M], and eSCI_CR1[PE]. [M2=0,M=1,PE=0]: value of received data bit 8 or address bit. [M2=0,M=1,PE=1]: value of received parity bit if eSCI_CR2[PMSK]=0, 0 otherwise. [M2=1,M=0,PE=1]: value of received parity bit if eSCI_CR2[PMSK]=0, 0 otherwise. [M2=1,M=1,PE=1]: value of received parity bit if eSCI_CR2[PMSK]=0, 0 otherwise. It is 0 for all other frame formats.
TN	Transmit Most Significant Bit. The semantic of this bit depends on the frame format selected by eSCI_CR3[M2], eSCI_CR1[M], and eSCI_CR1[PE]. [M2=0,M=1,PE=0]: value to be transmitted as data bit 8 or address bit. It is not used for all other frame formats.

Table 20-9. eSCI_SDR Field Descriptions (continued)

Field	Description
ERR	Receive Error Bit. This bit indicates the occurrence of the errors selected by the Control Register 3 (eSCI_CR3) during the reception of the frame presented in SCI Data Register (eSCI_SDR). In case of an overrun error for subsequent frames this bit is set too. 0 None of the selected errors occured. 1 At least one of the selected errors occured.
RD[11:8]	Received Data. The semantic of this field depends on the frame format selected by eSCI_CR3[M2] and eSCI_CR1[M]. [M2=1,M=1]: value of the received data bits 11:8. (Rx=BITx). It is all 0 for all other frame formats.
RD[7]	Received Bit 7. The semantic of this bit depends on the format selected by eSCI_CR3[M2], eSCI_CR1[M], and eSCI_CR1[PE]. [M2=0,M=0,PE=0]: value of received BIT7 or ADDR BIT. [M2=0;M=0,PE=1]: value of received PARITY BIT if eSCI_CR2[PMSK]=0, 0 otherwise. For all other frame formats it is the value of received BIT7.
TD[7]	Transmit Bit 7. The semantic of this bit depends on the format selected by eSCI_CR3[M2], eSCI_CR1[M], and eSCI_CR1[PE]. [M2=0,M=0,PE=0]: value of transmit BIT7 or ADDR BIT. [M2=0;M=0,PE=1]: not used. PARITY BIT is generated internally before transmission. For all other frame formats it is the value of transmit BIT7.
RD[6:0]	Received bits 6 to 0. Value of received BITx is shown in bit Rx
TD[6:0]	Transmit bits 6 to 0. Value of bit Tx is transmitted in BITx.

20.3.2.5 Interrupt Flag and Status Register 1 (eSCI_IFSR1)

eSCI_BASE + 0x0008 Write: Anytime

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF			DACT	BERR	WACT	LACT	TACT	RACT
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c				w1c				
Reset	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-6. Interrupt Flag and Status Register 1 (eSCI_IFSR1)

This register provides interrupt flags that indicate the occurrence of module events. The related interrupt enable bits are located in Control Register 1 (eSCI_CR1) and Control Register 2 (eSCI_CR2).

Table 20-10. eSCI_IFSR1 Field Descriptions

Field	Description
TDRE	Transmit Data Register Empty Interrupt Flag. This interrupt flag is set when the content of the SCI Data Register (eSCI_SDR) was transferred into internal shift register. **Note:** This flag is set in SCI mode only.
TC	Transmit Complete Interrupt Flag. This interrupt flag is set when a frame, break or idle character transmission has been completed and no data were written into SCI Data Register (eSCI_SDR) after the last setting of the TDRE flag and the SBK bit in Control Register 1 (eSCI_CR1) is 0. This flag is set in LIN mode, if the preamble was transmitted after the enabling of the transmitter.

Table 20-10. eSCI_IFSR1 Field Descriptions (continued)

Field	Description
RDRF	Receive Data Register Full Interrupt Flag. This interrupt flag is set when the payload data of a received frame was transferred into the SCI Data Register (eSCI_SDR) and the receive DMA is disabled. **Note:** This flag is set in SCI mode only.
IDLE	Idle Line Interrupt Flag. This interrupt flag is set when an idle character was detected and the receiver is not in the wake-up state. **Note:** This flag is set in SCI mode only.
OR	Overrun Interrupt Flag. This interrupt flag is set when an overrun was detected as described in Section 20.4.5.3.11, “Receiver Overrun”. **Note:** This flag is set in SCI mode only.
NF	Noise Interrupt Flag. This interrupt flag is set when the receiver has detected noise during the reception of a frame, as described in Section 20.4.5.3.13, “Bit Sampling”.
FE	Framing Error Interrupt Flag. This interrupt flag is set when the payload data of a received frame was transferred into the SCI Data Register (eSCI_SDR) or LIN Receive Register (eSCI_LRR) and the receiver has detected a framing error during the reception of that frame, as described in Section 20.4.5.3.17, “Stop Bit Verification”.
PF	Parity Error Interrupt Flag. This interrupt flag is set when the payload data of a received frame was transferred into the SCI Data Register (eSCI_SDR) and the receiver has detected a parity error for the character, as described in Section 20.4.5.4, “Reception Error Reporting” **Note:** This flag is set in SCI mode only.
DACT	DMA Active. The status bit is set when a transmit or receive DMA request is pending. 0 No DMA request pending 1 DMA request pending.
BERR	Bit Error Interrupt Flag. This flag is set when a bit error was detected as described in Section 20.4.6.5.3, “Standard Bit Error Detection”. **Note:** This flag is set in LIN mode only.
WACT	LIN Wake-Up Active. The status bit is set as long as the LIN wakeup engine receives a LIN wake-up signal. 0 No LIN wakeup signal reception in progress. 1 LIN wakeup signal reception in progress.
LACT	LIN Active. This status bit is set as long as the LIN protocol engine is about to transmit or receive LIN frames. 0 No LIN frame transmission or reception in progress. 1 LIN frame transmission or reception in progress.
TACT	Transmitter Active. This status bit is set as long as the transmission of a frame or special character is ongoing. 0 No transmission in progress. 1 Transmission in progress.
RACT	Receiver Active. This status bit is set as long as the receive is active. The set and clear conditions for the SCI mode are described in Section 20.4.5.3.1, “Receiver States and Transitions”.The set and clear conditions for the LIN mode are described in Section 20.4.6.2.1, “LIN byte field reception”. 0 No reception in progress. 1 Reception in progress.

20.3.2.6 Interrupt Flag and Status Register 2 (eSCI_IFSR2)

eSCI_BASE + 0x000A — Write: Anytime

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RXRDY	TXRDY	LWAKE	STO	PBERR	CERR	CKERR	FRC	0	0	0	0	0	0	UREQ	OVFL
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c							w1c	w1c
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-7. Interrupt Flag and Status Register 2 (eSCI_IFSR2)

This register provides interrupt flags that indicate the occurrence of LIN related events. The related interrupt enable bits are located in LIN Control Register 1 (eSCI_LCR1) and LIN Control Register 2 (eSCI_LCR2). All interrupt flags in this register will be set in LIN mode only.

Table 20-11. eSCI_IFSR2 Field Descriptions

Field	Description
RXRDY	Receive Data Ready Interrupt Flag. This interrupt flag is set when the payload data of a received frame was transferred into the LIN Receive Register (eSCI_LRR) and the receive DMA is disabled.
TXRDY	Transmit Data Ready Interrupt Flag. This interrupt flag is set when a) the content of the LIN Transmit Register (eSCI_LTR) was processed by the LIN PE to generate frame header or frame transmit data, or b) when the module has transmitted a LIN wakeup signal frame.
LWAKE	LIN Wake-up Received Interrupt Flag. This interrupt flag is set when a LIN Wake-up character was received, as described in Section 20.4.6.6, "LIN Wake Up".
STO	Slave Timeout Interrupt Flag. This interrupt flag is set when a Slave-Not-Responding-Error is detected. A detailed description is given in Section 20.4.6.5.5, "Slave-Not-Responding-Error Detection".
PBERR	Physical Bus Error Interrupt Flag. This interrupt flag is set when the receiver input remains unchanged for at least 31 RCLK clock cycles after the start of a byte transmission, as described in Section 20.4.6.5, "LIN Error Reporting".
CERR	CRC Error Interrupt Flag. This interrupt flag is set when an incorrect CRC pattern was detected for a received LIN frame.
CKERR	Checksum Error Interrupt Flag. This interrupt flag is set when a checksum error was detected for a received LIN frame.
FRC	Frame Complete Interrupt Flag. This interrupt flag is set when a LIN TX frame has been completely transmitted or a LIN RX frame has been completely received.
UREQ	Unrequested Data Received Interrupt Flag. This interrupt flag is set when unrequested activity has been detected on the LIN bus, as described in Section 20.4.6.5, "LIN Error Reporting".
OVFL	Overflow Interrupt Flag. This interrupt flag is set when an overflow as described in Section 20.4.6.5.8, "Overflow Detection" was detected.

20.3.2.7 LIN Control Register 1 (eSCI_LCR1)

eSCI_BASE + 0x000C Write: Anytime

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LRES	0	WUD		0	0	PRTY	LIN	RXIE	TXIE	WUIE	STIE	PBIE	CIE	CKIE	FCIE
W		WU														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-8. LIN Control Register 1 (eSCI_LCR1)

This register provides control bits to control and configure the LIN hardware. This register provides the interrupt enable bits for the interrupt flags in Interrupt Flag and Status Register 2 (eSCI_IFSR2).

Table 20-12. eSCI_LCR1 Field Descriptions

Field	Description
LRES	LIN Protocol Engine Stop and Reset. This bit is used to stop and reset the LIN protocol engine as described in Section 20.4.6.7, “LIN Protocol Engine Stop and Reset”. 0 LIN protocol engine is operational. 1 LIN protocol engine is reset and stopped.
WU	LIN Bus Wake-Up Trigger. This bit is used to trigger the generation of a wake-up signal frame on the LIN bus, as described in Section 20.4.6.6, “LIN Wake Up”. 0 Write has no effect. 1 Write triggers the generation of a wake-up signal.
WUD	LIN Bus Wake-Up Delimiter Time. This field determines how long the LIN protocol engine waits after the end of the transmitted wake-up signal, before starting the next LIN frame transmission. 00 4 bit times. 01 8 bit times. 10 32 bit times. 11 64 bit times.
PRTY	Parity Generation Control. This bit controls the generation of the two parity bits in the LIN header. 0 Parity bits generation disabled. 1 Parity bits generation enabled.
LIN	LIN Mode Control. This bit controls whether the device is in SCI or LIN Mode. 0 SCI Mode. 1 LIN Mode.
RXIE	Receive Data Ready Interrupt Enable. This bit controls the eSCI_IFSR2[RXRDY] interrupt request generation. 0 RXRDY interrupt request generation disabled. 1 RXRDY interrupt request generation enabled.
TXIE	Transmit Data Ready Interrupt Enable. This bit controls the eSCI_IFSR2[TXRDY] interrupt request generation. 0 TXRDY interrupt request generation disabled. 1 TXRDY interrupt request generation enabled.
WUIE	LIN Wake-up Received Interrupt Enable. This bit controls the eSCI_IFSR2[LWAKE] interrupt request generation. 0 LWAKE interrupt request generation disabled. 1 LWAKE interrupt request generation enabled.
STIE	Slave Timeout Flag Interrupt Enable. This bit controls the eSCI_IFSR2[STO] interrupt request generation. 0 STO interrupt request generation disabled. 1 STO interrupt request generation enabled.

Table 20-12. eSCI_LCR1 Field Descriptions (continued)

Field	Description
PBIE	Physical Bus Error Interrupt Enable. This bit controls the eSCI_IFSR2[PBERR] interrupt request generation. 0 PBERR interrupt request generation disabled. 1 PBERR interrupt request generation enabled.
CIE	CRC Error Interrupt Enable. This bit controls the eSCI_IFSR2[CERR] interrupt request generation. 0 CERR interrupt request generation disabled. 1 CERR interrupt request generation enabled.
CKIE	Checksum Error Interrupt Enable. This bit controls the eSCI_IFSR2[CKERR] interrupt request generation. 0 CKERR interrupt request generation disabled. 1 CKERR interrupt request generation enabled.
FCIE	Frame Complete Interrupt Enable. This bit controls the eSCI_IFSR2[FRC] interrupt request generation. 0 FRC interrupt request generation disabled. 1 FRC interrupt request generation enabled.

20.3.2.8 LIN Control Register 2 (eSCI_LCR2)

eSCI_BASE + 0x000E Write: Anytime

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	UQIE	OFIE	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-9. LIN Control Register 2 (eSCI_LCR2)

This register provides the interrupt enable bits for the interrupt flags in Interrupt Flag and Status Register 2 (eSCI_IFSR2).

Table 20-13. eSCI_LCR2 Field Descriptions

Field	Description
UQIE	Unrequested Data Received Interrupt Enable. This bit controls the eSCI_IFSR2[UREQ] interrupt request generation. 0 UREQ interrupt request generation disabled. 1 UREQ interrupt request generation enabled.
OFIE	Overflow Interrupt Enable. This bit controls the eSCI_IFSR2[OVFL] interrupt request generation. 0 OVFL interrupt request generation disabled. 1 OVFL interrupt request generation enabled.

20.3.2.9 LIN Transmit Register (eSCI_LTR)

eSCI_BASE + 0x0010 Write: LIN Mode

<table>
<tr><th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>R</td><td colspan="8">DATA</td></tr>
<tr><td>1st W</td><td colspan="2">P</td><td colspan="6">ID</td></tr>
<tr><td>2nd W</td><td colspan="8">LEN</td></tr>
<tr><td>3rd W</td><td>CSM</td><td>CSE</td><td>CRC</td><td>TD (=1)</td><td colspan="4">TO (ignored)</td></tr>
<tr><td>4th+ W</td><td colspan="8">D</td></tr>
<tr><td>Reset</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

Figure 20-10. LIN Transmit Register (eSCI_LTR) - LIN TX frame generation

<table>
<tr><th></th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>R</td><td colspan="8">DATA</td></tr>
<tr><td>1st W</td><td colspan="2">P[1:0]</td><td colspan="6">ID[5:0]</td></tr>
<tr><td>2nd W</td><td colspan="8">LEN</td></tr>
<tr><td>3rd W</td><td>CSM</td><td>CSE</td><td>CRC</td><td>TD (=0)</td><td colspan="4">TO[11:8]</td></tr>
<tr><td>4th W</td><td colspan="8">TO[7:0]</td></tr>
<tr><td>Reset</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

Figure 20-11. LIN Transmit Register (eSCI_LTR) - LIN RX frame generation

This register is used by the application to initiate the LIN frame header generation for both LIN TX frames and LIN RX frames. If a LIN TX frame is generated, this register is used to provide the payload data for the LIN TX frame.

If the LIN PE is in the idle state (eSCI_LCR1[LRES] = 1) or performs a wakeup, each write access to this register is ignored.

In case of an read access, the register provides the last data written into this register in the DATA field.

If the application initiates a LIN TX frame transfer, i.e the TD bit is set to 1, the content and usage shown in LIN Transmit Register (eSCI_LTR) - LIN TX frame generation applies. The initiation and transmit of a TX frame is described in Section 20.4.6.3, “LIN TX Frame generation”.

If the application initiates an LIN RX frame, i.e the TD bit is set to 0, the content and usage shown in LIN Transmit Register (eSCI_LTR) - LIN RX frame generation applies. The initiation and transmit of a RX frame is described in Section 20.4.6.4, “LIN RX frame generation”.

Each successful write access to this register increments the internal write access counter and enables the writing to the next field. The write access counter is reset if

- the LIN PE is in the idle state (eSCI_LCR1[LRES] = 1)
- a LIN TX frame was completely transmitted (eSCI_IFSR1[FRC] was set to 1)

a LIN RX frame was completely received (eSCI_IFSR1[FRC] was set to 1)

Table 20-14. eSCI_LTR Field Descriptions

Field	Description
DATA	Value written in the most recent successful write access.
P	Identifier Parity. This field provides the identifier parity which is used to create the protected identifier if the automatic identifier parity generation is disabled, i.e the PRTY bit in LIN Control Register 1 (eSCI_LCR1) is 0.
ID	Identifier. This field is used for the identifier field in the protected identifier.
LEN	Frame Length. This field defines the number of data bytes to be transmitted or received.
CSM	Checksum Model. This bit controls the checksum calculation model used. 0 Classic Checksum Model (LIN 1.3). 1 Enhanced Checksum Model (LIN 2.0).
CSE	Checksum Enable. This bit control the generation and checking of the checksum byte. 0 No generation and checking of checksum byte. 1 Generation and checking of checksum byte.
CRC	CRC Enable. This bit controls the generation of checking standard or enhanced LIN frames, which are described in Section 20.4.6.2, “LIN frame formats” 0 Standard LIN frame generation and checking. 1 Enhanced LIN frame generation and checking.
TD	Transfer Direction. This bit control the transfer direction of the data, crc, and checksum byte fields. 0 Data, CRC, and Checksum byte fields received, described in Section 20.4.6.4, “LIN RX frame generation”. 1 Data, CRC, and Checksum byte fields transmitted, described in Section 20.4.6.3, “LIN TX Frame generation”.
TO	Timeout Value. The content of the field depends on the transfer direction. RX frame: Defines the time available for a complete RX frame transfer, as described in Section , “To calculate the exact position of the sample point with regard to the RX pin, the delays through the pads and the two Bus Clock cycle delay through the input synchronizer also needs to be taken into account.” TX frame: Must be set to 0.
D	Transmit Data. Data bits for transmission.

20.3.2.10 LIN Receive Register (eSCI_LRR)

eSCI_BASE + 0x0014 Read Only

	7	6	5	4	3	2	1	0
R	D							
W								
Reset	0	0	0	0	0	0	0	0

Figure 20-12. LIN Receive Register (eSCI_LRR)

This register provides the data bytes of received in case of an LIN RX frame was initiated.

Table 20-15. eSCI_LRR Field Descriptions

Field	Description
D	Receive Data. This field provides the data bytes of received LIN RX frames.

20.3.2.11 LIN CRC Polynomial Register (eSCI_LPR)

eSCI_BASE + 0x0018 Write: Anytime

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	P															
Reset	1	1	0	0	0	1	0	1	1	0	0	1	1	0	0	1

Figure 20-13. LIN CRC Polynomial Register (eSCI_LPR)

This register provides the CRC polynom for generation and processing of CRC-enhanced LIN frames.

Table 20-16. eSCI_LPR Field Descriptions

Field	Description
P	Polynomial bit $x^{P[n]}$. Used to define the LIN polynomial. Reset value results in $x^{15} + x^{14} + x^{10} + x^{8} + x^{7} + x^{4} + x^{3} + 1$, which is the polynomial used for the CAN protocol.

20.3.2.12 Control Register 3 (eSCI_CR3)

eSCI_BASE + 0x001A Write: Anytime

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	EROE	ERFE	ERPE	M2	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨					▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 20-14. Control Register 3 (eSCI_CR3)

This register is used to control the frame formats and the generation of the ERR bit in the SCI Data Register (eSCI_SDR).

Table 20-17. eSCI_CR3 Field Descriptions

Field	Description
3 EROE	ERR flag overrun enable. 0 eSCI_SDR[ERR] flag not affected by overrun detection. 1 eSCI_SDR[ERR] flag is set on overrun detection during frame reception.
2 ERFE	ERR flag frame error enable. 0 eSCI_SDR[ERR] flag not affected by frame error detection. 1 eSCI_SDR[ERR] flag is set on frame error detection for the data provided in eSCI_SDR.

Table 20-17. eSCI_CR3 Field Descriptions (continued)

Field	Description
1 ERPE	ERR flag parity error enable. 0 eSCI_SDR[ERR] flag not affected by parity error detection. 1 eSCI_SDR[ERR] flag is set on parity error detection for the data provided in eSCI_SDR.
0 M2	Frame Format Mode 2. This control bit together with the M bit of the Control Register 1 (eSCI_CR1) controls the frame format used. The supported frame formats and the related settings are defines in Section 20.4.2, “Frame Formats”.

20.4 Functional Description

This section provides a complete functional description of the eSCI block, detailing the operation of the design from the end user perspective in a number of subsections.

20.4.1 Module Control

The operational mode of the module is controlled by the MDIS bit in the Control Register 2 (eSCI_CR2). The module can transmit and receives data when it is enabled, i.e MDIS=0.

20.4.2 Frame Formats

The eSCI module uses the standard NRZ mark/space data format. The eSCI supports three basic frame types, which are the data frames, break characters, and idle characters.

20.4.2.1 Data Frame Formats

Each data frame contains a character that is surrounded by a start bit, an optional parity or address bit, and one or two stop bits. The supported data frame formats for transmission and reception are specified in Table 20-18. The supported data frame formats for reception only are specified in Table 20-19.

Table 20-18. Supported Data Frame Formats for RX and TX

Control				Frame Content				
eSCI_CR3	eSCI_CR1			Start Bits	Payload Bits			Stop Bits
M2	M	PE	WAKE		Character Bits	Address Bits[1]	Parity Bits	
LIN byte fields (Figure 20-15)								
0	0	0	0	1	8	0	0	1
SCI Frames (8 payload bits)(Figure 20-16)								
0	0	0	0	1	8	0	0	1
0	0	0	1	1	7	1	0	1
0	0	1	0	1	7	0	1	1
SCI Frames (9 payload bits) (Figure 20-17)								
0	1	0	0	1	9	0	0	1
0	1	0	1	1	8	1	0	1
0	1	1	0	1	8	0	1	1

[1] The address bit identifies the frame as an address character. See Section 20.4.5.5, “Multiprocessor Communication.”

Table 20-19. Supported Data Frame Formats for RX only

Control				Frame Content				
eSCI_CR3	eSCI_CR1			Start Bits	Payload Bits			Stop Bits
M2	M	PE	WAKE		Character Bits	Address Bits	Parity Bits	
SCI Frames (2 stop bits) (see Figure 20-18)								
1	0	1	0	1	8	0	1	2
1	1	1	0	1	12	0	1	2

The structure of the LIN byte fields in normal polarity is shown in Figure 20-15.

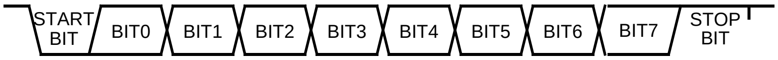


Figure 20-15. LIN Byte Field Format

The structures of the supported SCI frame formats with 8 payload bits are shown in Figure 20-16.

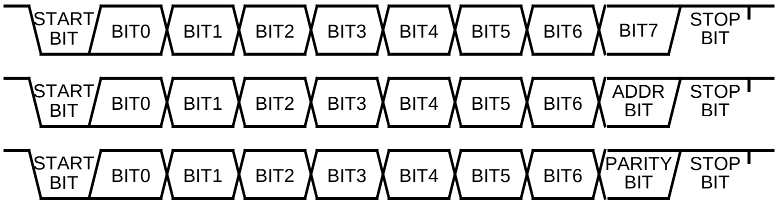


Figure 20-16. SCI Frame Formats (8 payload bits)

The structures of the supported SCI frame formats with 9 payload bits are shown in Figure 20-17.

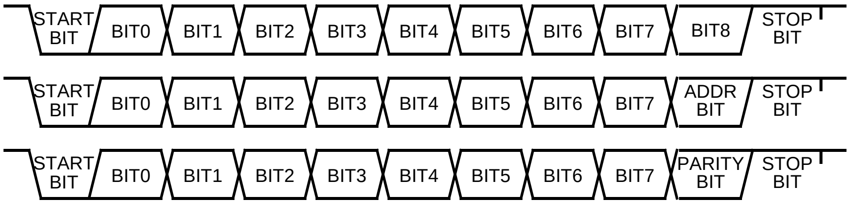


Figure 20-17. SCI Frame Formats (9 payload bits)

The structures of the supported SCI frame formats with 2 stop bits in normal polarity are shown in Figure 20-18. This frame format is supported for reception only.

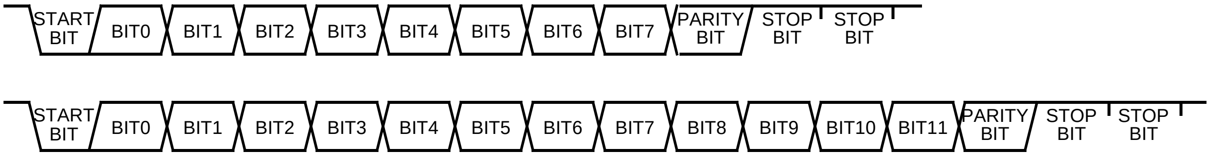


Figure 20-18. SCI Frame Formats (2 stop bits)

20.4.2.1.1 Inverted Data Frame Formats

The structures of the supported data frame formats in inverted polarity are shown in Figure 20-19. These frame types are supported for reception only. The polarity of the RXD pin is controlled by the RXPOL bit in the Control Register 2 (eSCI_CR2).

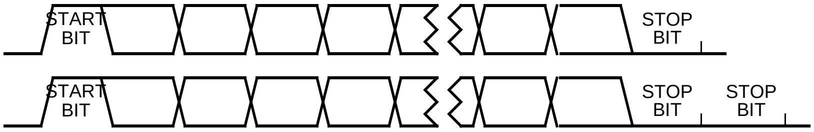


Figure 20-19. Inverted SCI Frame Formats

20.4.2.2 Break Character Formats

The supported break character formats are specified in Table 20-20.

Table 20-20. Supported Break Character Formats

Control[1]			Break Character Content		
eSCI_CR3	eSCI_CR1	eSCI_CR2	Start Bit	Character Bits	Delemit Bits
M2	M	BRCL			
LIN Break Symbol (see Figure 20-20)					
0	0	0	1	9	1
0	0	1	1	12	1
SCI Break Character (see Figure 20-21)					
0	0	0	1	9	0
0	0	1	1	12	0
0	1	0	1	10	0
0	1	1	1	13	0

[1] All codings which are not listed are reserved and must not be used.

The structure and content of the LIN break symbols is shown in Figure 20-20.

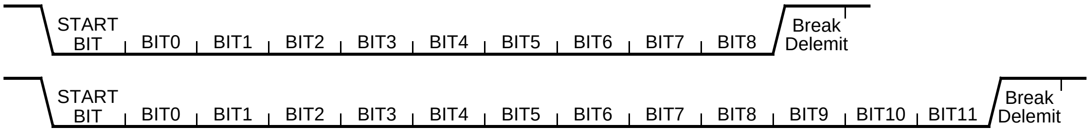


Figure 20-20. LIN Break Symbol Format

The structure and content of the SCI break characters is shown in Figure 20-21.

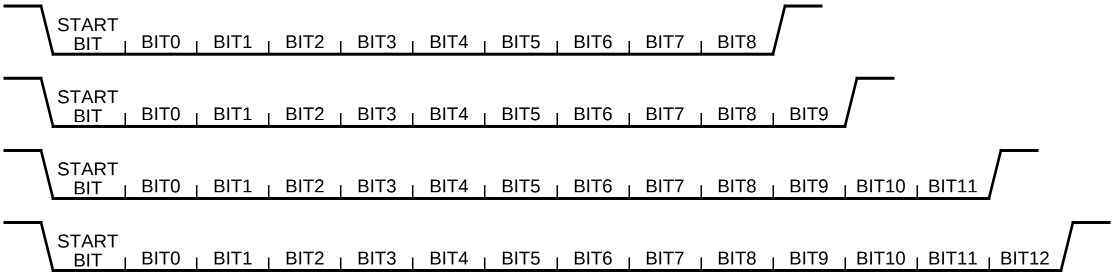


Figure 20-21. SCI Break Character Formats

20.4.2.3 Idle Character Formats

An idle character is a sequence of bits with the value 1. The supported idle character formats are specified in Table 20-21. The preamble has the same structure and content as an idle character.

Table 20-21. Supported Idle Character Formats

Control		Idle Character Length
eSCI_CR3[M2]	**eSCI_CR1[M]**	
Idle Characters (see Figure 20-22)		
0	0	10
0	1	11
1	0	12
1	1	16

The structure and content of the idle characters is shown in Figure 20-22.

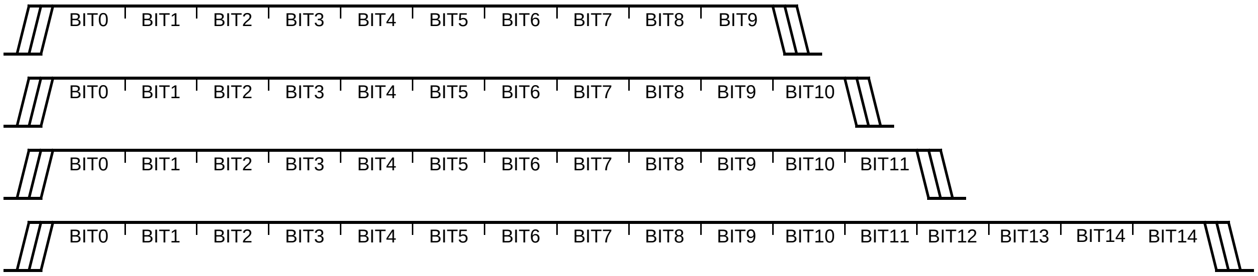


Figure 20-22. Idle Character Formats

20.4.3 Baud Rate and Clock Generation

A 13-bit modulus counter in the baud rate generator derives the baud rate for both the receiver and the transmitter. The value written to the SBR field in the Baud Rate Register (eSCI_BRR) determines the module clock divisor. The baud rate clock is synchronized with the bus clock and drives the receiver. The baud rate clock divided by 16 drives the transmitter. The receiver has an acquisition rate of 16 samples per bit time.

The baud rate generator is enabled when the TE bit or RE bit in the Control Register 1 (eSCI_CR1) is set to 1 for the first time. The baud rate generator is disabled when SBR = 0.

Baud rate generation is subject to one source of error:

- Integer division of the module clock may not give the exact required target baud rate.

Figure 20-22 lists some examples of achieving target baud rates with a module clock frequency of MCLK = 10.2 MHz.

Table 20-22. Baud Rates Error Example (MCLK = 10.2 MHz)

eSCI_BRR[SBR]	RCLK (Hz)	TCLK (Hz)	Target Baud Rate	Error (%)
17	600,000.0	37,500.0	38,400	2.3
33	309,090.9	19,318.2	19,200	.62
66	154,545.5	9659.1	9600	.62
133	76,691.7	4793.2	4800	.14
266	38,345.9	2396.6	2400	.14
531	19,209.0	1200.6	1200	.11
1062	9604.5	600.3	600	.05
2125	4800.0	300.0	300	.00
4250	2400.0	150.0	150	.00
5795	1760.1	110.0	110	.00

20.4.3.1 Module Clock

The module clock MCLK is derived from the system bus clock. It has the same phase and frequency.

20.4.3.2 Transmitter Clock

The transmitter clock TCLK is used to drive the data to the serial bus via the TXD pin. It is derived from the system bus clock by the baud rate generator. The baud rate generator is controlled by the value of the SBR field in the Baud Rate Register (eSCI_BRR). The frequency of the transmitter clock is determined by Equation 20-1 and defines the length of the transmitted bits, which is denoted as the *bit time*.

$$f_{\mathrm{TCLK}} = \frac{f_{\mathrm{MCLK}}}{16 \cdot \mathrm{SBR}}$$ ***Eqn. 20-1***

20.4.3.3 Receiver Clock

The receiver clock RCLK is used to sample the data received on the RXD or TXD pin. It is derived from the system bus clock by the baud rate generator. The baud rate generator is controlled by the value of the SBR field in the Baud Rate Register (eSCI_BRR). The frequency of the receiver sample clock is determined by Equation 20-2.

$$f_{RT} = \frac{f_{MCLK}}{SBR}$$ ***Eqn. 20-2***

The frequency of the receiver clock is 16 times the frequency of the transmitter clock, this each bit is sampled with 16 samples. Each of the 16 samples of a bit has a sample number assigned, which is defined by the receiver sample counter RSC. The n-th sample is denoted by RSn. The receiver sample counter RSC is updated with each rising edge of the receiver clock RCLK.

20.4.4 Baud Rate Tolerance

A transmitting device may be operating at a baud rate below or above the receiver baud rate. Accumulated bit time misalignment can cause one of the three stop bit data samples RS8, RS9, and RS10 to fall outside the actual stop bit. A noise error will occur if the stop bit sample RS8, RS9, and RS10 samples are not all the same logical value 1. A framing error will occur if the receiver clock is misaligned in such a way that the majority of the RS8, RS9, and RS10 stop bit samples are a logic zero.

20.4.4.1 Faster Receiver Tolerance

In this case the receiver has a higher baud rate than the transmitter, thus the stop bit sampling starts already in the last transmitted payload bit. To ensure the correct, noise and framing error free reception of the stop bit, the samples RS8, RS9, and RS10 must be located in the transmitted stop bit as shown in Figure 20-23.

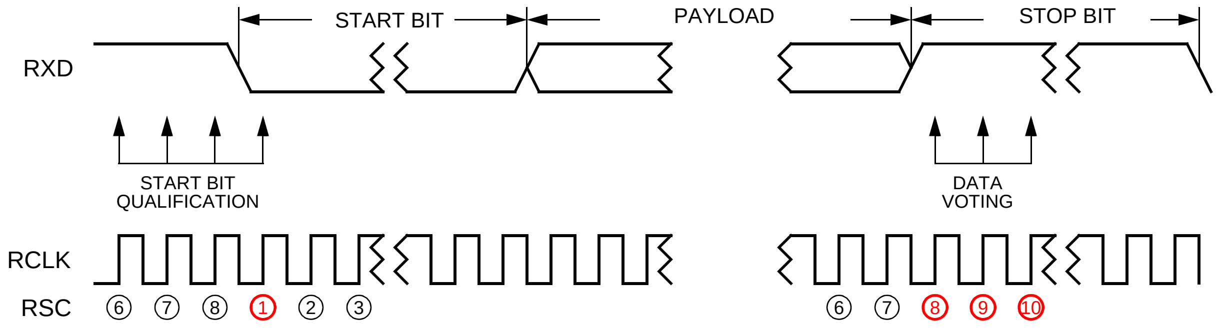


Figure 20-23. Faster Receiver

The maximum tolerance that ensures error free reception can be calculated with the assumption, that RS7 is sampled during the last transmitted payload bit and RS8 is sampled in the stop bit.

For an frame with *n* payload bits the transmitter starts the transmission of the stop bit

$$tx_{STOP} = (n + 1) \cdot 16 \cdot RT_{TR}$$ ***Eqn. 20-3***

after the start of the transmission of the start bit.

For an frame with *n* payload bits the receiver samples the RS8 sample of the stop bit

$$\mathrm{rx}_{\mathrm{STOP}} = (n + 1) \cdot 16 \cdot \mathrm{RT}_{\mathrm{RE}} + 7 \cdot \mathrm{RT}_{\mathrm{RE}}$$ ***Eqn. 20-4***

after the successful qualification of the start bit.

To ensure error free reception of the stop bit, the transmitter must start the transmission of the stop bit before the receiver samples RSC8.

$$\mathrm{tx}_{\mathrm{STOP}} < \mathrm{rx}_{\mathrm{STOP}}$$ ***Eqn. 20-5***

The maximum percent difference between the receiver baud rate and the transmitter baud rate is:

$$\Delta\mathrm{baudrate} \le \left(\frac{\mathrm{rx}_{\mathrm{STOP}} - \mathrm{tx}_{\mathrm{STOP}}}{\mathrm{rx}_{\mathrm{STOP}}}\right) \times 100$$ ***Eqn. 20-6***

The maximum percent differences for the supported frames is given in Table 20-23

Table 20-23. Faster Receiver Maximum Tolerance

payload bits	max baudrate difference	$\mathrm{tx}_{\mathrm{STOP}}$	$\mathrm{rx}_{\mathrm{STOP}}$
8	4.63%	144	151
9	4.19%	160	167
13	3.03%	224	231

20.4.4.2 Slower Receiver Tolerance

In this case the receiver has a slower baud rate than the transmitter, thus the stop bit sampling is still running while the next start bit is already transmitted. To ensure the correct, noise and framing error free reception of the stop bit, the samples RS8, RS9, and RS10 must be located in the transmitted stop bit as shown in Figure 20-24.

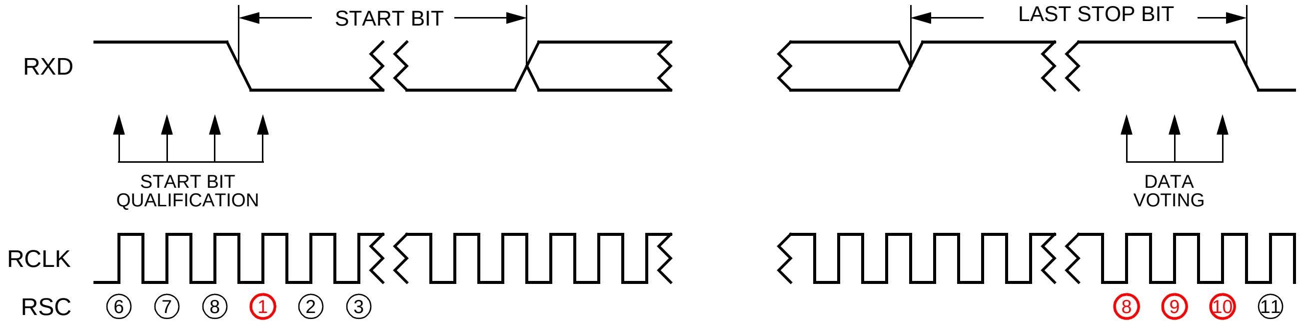


Figure 20-24. Slower Receiver

The maximum tolerance that ensures error free reception can be calculated with the assumption, that RS11 is sampled in the transmitted start bit and RS10 is sampled in the last stop bit.

For an frame with *n* payload bits and *s* stop bits, the transmitter starts the transmission of the next start bit

$$\mathrm{tx}_{\mathrm{START}} = (n + s + 1) \cdot 16 \cdot \mathrm{RT}_{\mathrm{TR}}$$ ***Eqn. 20-7***

after the start of the transmission of the previous start bit.

For an frame with *n* payload bits and s stop bits, the receiver samples the RS10 sample of the last stop bit

$$rx_{STOP} = (n + s) \cdot 16 \cdot RT_{RE} + 9 \cdot RT_{RE}$$ **Eqn. 20-8**

after the successful qualification of the start bit.

To ensure error free reception of the last stop bit, the transmitter must start the transmission of the start bit after the receiver samples RS10.

$$rx_{STOP} < tx_{START}$$ **Eqn. 20-9**

The maximum percent difference between the receiver baud rate and the transmitter baud rate is:

$$\Delta baudrate \le \left(\frac{tx_{START} - rx_{STOP}}{tx_{START}}\right) \times 100$$ **Eqn. 20-10**

The maximum percent differences for the supported frames is given in Table 20-24

Table 20-24. Slower Receiver Maximum Tolerance

payload bits	stop bits	max baudrate difference	rx_{STOP}	tx_{START}
8	1	4.37%	153	160
9	1	3.97%	169	176
9	2	3.57%	185	196
13	2	2.73%	249	256

20.4.5 SCI Mode

20.4.5.1 SCI Mode Configuration

The application must configure the following bits and fields in order to achieve correct SCI operation.

- enable *SCI* Mode
  - LIN Control Register 1 (eSCI_LCR1)[LIN]:= 0
- select *baud rate*
  - Baud Rate Register (eSCI_BRR)[SBR]
- select *receiver input* mode
  - Control Register 1 (eSCI_CR1)[LOOPS]
  - Control Register 1 (eSCI_CR1)[RSRC]
- select *frame format*
  - Control Register 1 (eSCI_CR1)[M]
  - Control Register 1 (eSCI_CR1)[PE]
  - Control Register 1 (eSCI_CR1)[WAKE]
  - Control Register 3 (eSCI_CR3)[M2]
- select *parity type*
  - Control Register 1 (eSCI_CR1)[PT]

20.4.5.2 Transmitter

The transmitter supports the transmission of all frame types defined in Table 20-18, of all break characters defined in Table 20-20, and of all idle characters defined in Table 20-21.

20.4.5.2.1 Transmitter States and Transitions

The transmitter has four basic states which are shown and described in Table 20-25. The state transitions that can triggered by the application commands are shown in Table 20-26. The state transitions that can triggered by the module are shown in Table 20-27. The state diagram of the transmitter is shown in Figure 20-25.

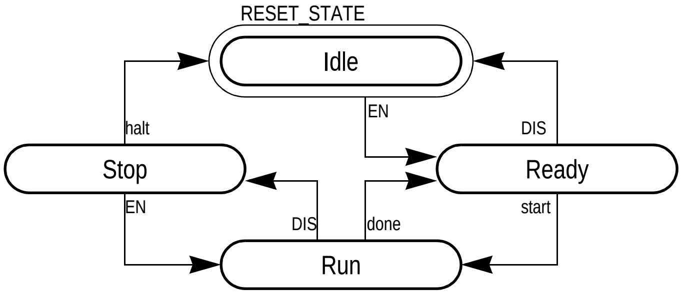


Figure 20-25. Transmitter State Diagram

The current state of the transmitter can be determined by the TE control bit in the Control Register 1 (eSCI_CR1) and the TACT status bit in Interrupt Flag and Status Register 1 (eSCI_IFSR1).

Table 20-25. Transmitter States

State	Indication		Description
	eSCI_CR1[TE]	eSCI_IFSR1[TACT]	
Idle	0	0	Transmitter is *disabled* and *no* transmission is running
Ready	1	0	Transmitter is *enabled* and *no* transmission is running
Run	1	1	Transmitter is *enabled* and transmission is running
Stop	0	1	Transmitter is *disabled* and transmission is running

The application triggers a transition described in Table 20-26 when it issues a command by writing to the TE bit in the Control Register 1 (eSCI_CR1). The transition is triggered only if the conditions are fulfilled. As a result of the transition the state of the transmitter is changed as shown in Figure 20-25 and the action given in Table 20-26 is executed.

Table 20-26. Transmitter Application Transitions

Transition	Command	Precondition	Action	Description
EN	eSCI_CR1[TE]:=1	eSCI_CR1[TE]=0	iPRE:=1	Transmitter is *enabled* by application command.
DIS	eSCI_CR1[TE]:=0	eSCI_CR1[TE]=1		Transmitter is *disabled* by application command

The module transition shown in Table 20-27 are triggered when the described condition or event occurs. The send break bit SBK in the Control Register 1 (eSCI_CR1) is check for the start condition. The internal commit bit iCMT, the transmitter active bit TACT in the Interrupt Flag and Status Register 1 (eSCI_IFSR1), the TDRE, and the TC flag in the Interrupt Flag and Status Register 1 (eSCI_IFSR1) are changed as a action result of the transition.

Table 20-27. Transmitter Module Transitions

Transition	Condition	Action	Description
start	(State=Ready) and (SBK=1 or iPRE=1 or iCMT=1)	TACT:=1	Start of transmission of data frame or special character when data are available or character transmission request is pending.
done	State=Run and last stop bit transmitted	TACT:=0 TC:= (SBK=0 & iPRE=0 & iCMT=0)	Finished transmission of data frame or special character and transmitter still enabled. Transmission is complete if no transmit request is pending.
halt	State=Stop and last stop bit transmitted	TACT:=0 TC:=1 iCMT:=0	Finished transmission of data frame or special character and transmitter was disabled.

20.4.5.2.2 Frame and Character Transmission

The transmitter starts the transmission of a data frame or special character when the condition for the *start* transition as described in Table 20-27 is fulfilled. There are three source for data or character transmission. The priority among these source are specified in Table 20-28. All three sources can be available at one point in time.

Table 20-28. Transmit Source Priority

Priority	Indication	Transmission Source
(highest) 0	iPRE=1	Preamble.
1	eSCI_CR1[SBK]=1	Break character.
(lowest) 2	iCMT=1	SCI Data Register (eSCI_SDR) frame.

20.4.5.2.3 CPU Controlled SCI Data Frame Transmission

The transmission of a data frame is started when the transmitter is in its Ready state and only the commit bit iCMT is set.

As the first step, the content of the SCI Data Register (eSCI_SDR) is transferred into the internal transmitter shift register. When this transfer is finished, the internal commit bit iCMT is cleared and the transmit data register empty flag TDRE in the Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set. If the transmit interrupt enable bit TIE in the Control Register 1 (eSCI_CR1) is also set, the TDRE flag generates a transmitter interrupt request.

The transmitter shift register then shifts a frame out through the TXD output signal, which is prefaced with a start bit and appended with the parity bit, if configured, and the configured number of stop bits.

When the last stop bit has been transmitted and the application has not disabled the transmitter, the transmitter returns to the Ready state via the *done* transition. If no frame or character transmit request is pending, the transfer complete flag TC in the Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set.

If the application has disabled the transmitter while the frame is transmitted and stop bit has been transmitted, the transmitter goes into the Idle state via the *halt* transition. The transfer complete flag TC in the Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set and the internal commit bit iCMT is cleared.

20.4.5.2.4 DMA Controlled SCI Data Frame Transmission

In this mode, the eSCI module handles the generation of Data Frames internally.

When new data required for transmission, the module generates the transmit DMA request and the DMA controller delivers the required data via write accesses to SCI Data Register (eSCI_SDR). The write access to the low byte of SCI Data Register (eSCI_SDR) triggers the transmission of the data. The write access to the high byte of SCI Data Register (eSCI_SDR) triggers no internal operation.

The application request the eSCI module to enter this mode by setting the TXDMA bit in the Control Register 2 (eSCI_CR2). From this point in time, the module start the generation of DMA requests and frame transmission. Before entering this mode, the application should perform the following actions:

1. Configure the module for SCI mode.
2. Enable the transmitter by setting TE in Control Register 1 (eSCI_CR1) to 1.
3. Setup the DMA controller channel and provide frame data in system memory

A block diagram which presents an overview of the DMA Controlled Date Frame Transmission is shown in Figure 20-26.

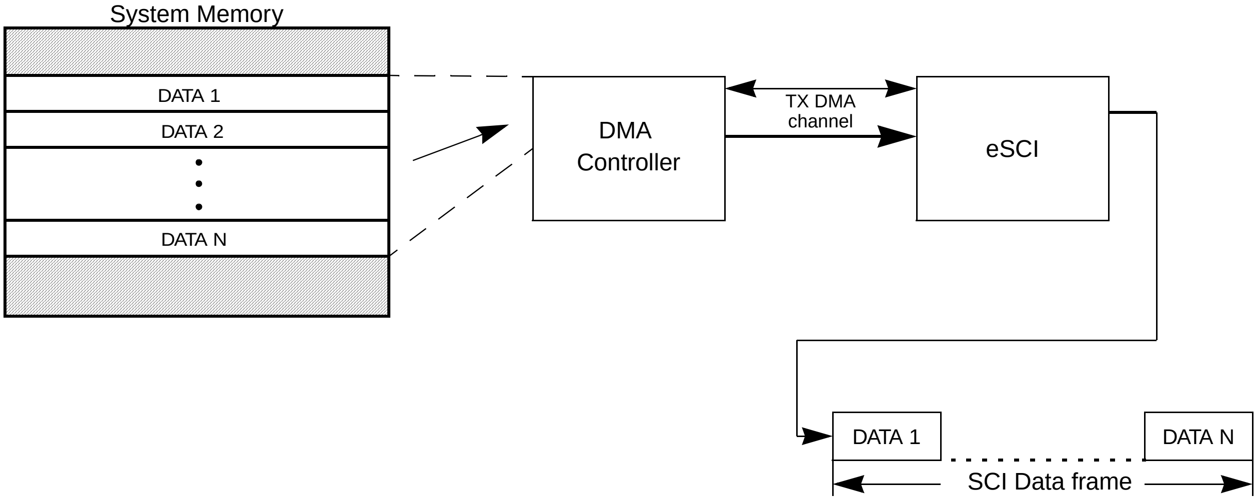


Figure 20-26. DMA Controlled SCI Data Frame generation

20.4.5.2.5 Parity Generation

The eSCI module generates the parity bit in transmitted data frame when the parity enable bit PE in the Control Register 1 (eSCI_CR1) is set. The parity type bit PT in the Control Register 1 (eSCI_CR1) defines whether the odd or even parity is generated.

20.4.5.2.6 Preamble Transmission

The transmission of a preamble is started when the transmitter is in Ready state, the internal iPRE bit, which is not visible to the application, is set, and the SBK in the Control Register 1 (eSCI_CR1) is clear.

After the transmission of the stop bit and if the application has not disabled the transmitter, the transmitter returns to the Ready state via the *done* transition. If no frame or character transmit request is pending, the transfer complete flag TC in the Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set.

If the application has disabled the transmitter while the preamble is transmitted and if the stop bit has been transmitted, the transmitter goes into the Idle state via the halt transition. The transfer complete flag TC in the Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set and the internal commit bit iCMT is cleared.

20.4.5.2.7 Break Character Transmission

The transmission of a break character is started when the transmitter is in Ready state and the send break character bit SBK in the Control Register 1 (eSCI_CR1) is set. After the transmission of the break character and if the application has not disabled the transmitter, the transmitter returns to the Ready state via the *done* transition and restarts the transmission. As long as SBK bit remains set, the transmitter continues to send break characters.

When the application has cleared the SBK bit or has disabled the transmitter, the transmitter continues to transmit the current break character and after it has finished the transmission of this break character it transmits a stop bit. The stop bit at the end of a break character sequence guarantees the recognition of the start bit of the next data frame.

After the transmission of the stop bit and if the application has not disabled the transmitter, the transmitter returns to the Ready state via the *done* transition. If no frame or character transmit request is pending, the transfer complete flag TC in the Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set.

If the application has disabled the transmitter while the break character is transmitted and if the stop bit has been transmitted, the transmitter goes into the Idle state via the *halt* transition. The transfer complete flag TC in the Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set and the internal commit bit iCMT is cleared.

20.4.5.3 Receiver

The receiver supports the reception of all data frame types defined in Table 20-18 and Table 20-19, of all break character defined in Table 20-20, and of all idle characters defined in Table 20-21.

20.4.5.3.1 Receiver States and Transitions

The receiver has four basic states which are shown and described in Table 20-26. The state transitions that can triggered by the application commands are shown in Table 20-26. The state transitions that can triggered by the module are shown in Table 20-27. The state diagram of the transmitter is shown in Figure 20-25.

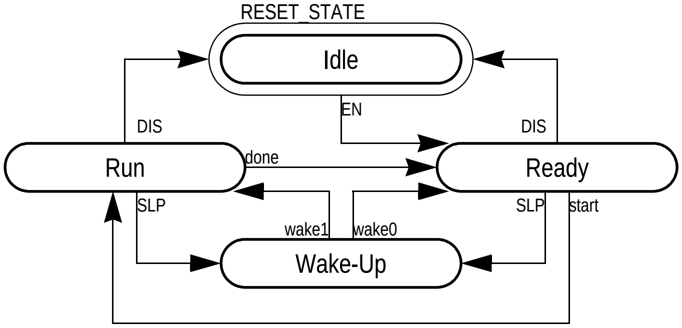


Figure 20-27. Receiver State Diagram

The current state of the receiver can be determined by the RE and RWU bit in the Control Register 1 (eSCI_CR1) and the RACT status bit in Interrupt Flag and Status Register 1 (eSCI_IFSR1).

Table 20-29. Receiver States

State	Indication			Description
	RE	RACT	RWU	
Idle	0	0	0	Receiver is *disabled* and *no* reception is running
Ready	1	0	0	Receiver is *enabled* and *no* reception is running
Run	1	1	0	Receiver is *enabled* and reception is running
Wake-up	1	-	1	Receiver is in wake-up mode

The application triggers a transition described in Table 20-30 when it issues a command by writing to the RE bit in the Control Register 1 (eSCI_CR1). The transition is triggered only if the conditions are fulfilled. As a result of the transition the state of the receiver is changed as shown in Figure 20-27 and the action given in Table 20-30 is executed.

Table 20-30. Receiver Application Transition

Transition	Command	Condition	Action	Description
EN	RE:=1	RE=0		Receiver is *enabled* by application command.
DIS	RE:=0	RE=1		Receiver is *disabled* by application command
SLP	RWU:=1	RE=1		Receiver is set into wake-up mode

The module transitions shown in Table 20-31 are triggered when the described event occurs.

Table 20-31. Receiver Module Transition

Transition	Condition	Action	Description
start	(State=Ready,Run) and (start bit qualified)	RACT:=1	Start of reception of data frame or break character.
done	(State=Run) and (start bit not verified or idle character received)	RACT:=0	Start Bit not Verified or Idle Character received.
wake0	(State=Wake-up) and (idle character received)	RWU:=0	Wake-up Idle Character received.
wake1	(State=Wake-up) and (address frame received)	RWU:=0	Wake-up address frame received.

20.4.5.3.2 Receiver Input Mode Selection

This section describes the three receiver input modes supported by the eSCI module. The modes are selected by the LOOPS and RSRC control bits in the Control Register 1 (eSCI_CR1).

20.4.5.3.3 Dual Wire Mode

In Dual Wire Mode, the eSCI uses the TXD pin for transmitting and the RXD pin for data receiving.

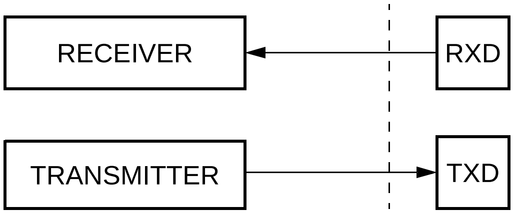


Figure 20-28. Dual Wire Mode

20.4.5.3.4 Single Wire Mode

In Single Wire Mode, the RXD pin is disconnected from the eSCI module and the TXD pin is used for both receiving and transmitting.

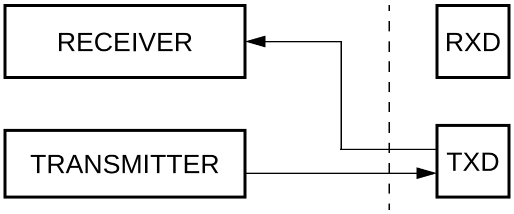


Figure 20-29. Single Wire Mode

NOTE

In this device, the TXDIR bit (eSCI_CR2[1]) does not determine whether the TXD pin is going to be used as an input or an output. The relevant pad control register in the System Integration Unit must be programmed for input or output)

20.4.5.3.5 Loop Mode

In Loop Mode, the input of the receiver is driven by the output of the transmitter. The RXD pin is disconnected from the eSCI module.

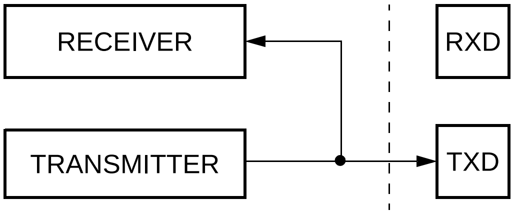


Figure 20-30. Loop Mode

20.4.5.3.6 Frame and Character Reception

The receiver is started when it is in Ready or Wake-up state and on the selected receiver input (see Section 20.4.5.3.2, “Receiver Input Mode Selection”) an active signal is sampled. The receiver enters the Run or Wake-up state. The received bits are recovered by the bit sampling described in Section 20.4.5.3.13, “Bit Sampling”. During the reception, the received bits are shifted into the internal shift register.

20.4.5.3.7 Break Character Detection

The receiver does not provide any means to detect the reception of a break character. Instead, break characters are processed as data frames. Due to the received 0 at the stop bit location, the reception of a break character causes at least a framing error. The error reporting is performed as described in Section 20.4.5.4, “Reception Error Reporting”.

20.4.5.3.8 Idle Character Detection

The Idle character detection starts after the reception of the last stop bit.

20.4.5.3.9 CPU Controlled SCI Data Frame Reception

This section describes the reception process when the receiver is in the Run state.

When the required number of frame bits have been received, the payload bits of the received frame are transferred into SCI Data Register (eSCI_SDR) if the RDRF flag is 0. The receive data register full flag RDRF in Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set. If the receive interrupt enable bit RIE in the Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set, the RDRF interrupt request is generated.

If an idle character has been detected, the IDLE flag in the Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set. If the idle line interrupt enable bit ILIE in the Control Register 1 (eSCI_CR1) is set, the IDLE interrupt request is generated.

If any of the receiver errors described in Section 20.4.5.4, “Reception Error Reporting” have been occurred, that corresponding flags will be set.

If the application disabled the receiver by clearing the receiver enable bit RE in the Interrupt Flag and Status Register 1 (eSCI_IFSR1) the current frame is discarded and no flags will be updated.

20.4.5.3.10 DMA Controlled SCI Data Frames Reception

In this mode, the eSCI module controls the reception of SCI Data frames automatically and utilizes the connected DMA channels. A block diagram which presents an overview of the DMA Controlled SCI Data Frame reception is shown in Figure 20-31. The RX DMA channel is used to transfer the received frame data into the memory.

When new data was received, the module generates the receive DMA request and the DMA controller retrieves the provided data from the SCI Data Register (eSCI_SDR). The read access from the low byte of the SCI Data Register (eSCI_SDR) signals the end of the DMA cycle for the current data and triggers the reception of new data. The read access from the SCI Data Register (eSCI_SDR) triggers no internal action.

The application request the eSCI module to enter this mode by setting the RXDMA bit in the Control Register 2 (eSCI_CR2). From this point in time, the module start the generation of DMA requests and frame transmission and reception. Before entering this mode, the application should perform the following actions:

1. Configure the module for SCI mode.
2. Enable the receiver by setting RE in Control Register 1 (eSCI_CR1) to 1.
3. Setup the DMA controller channel.

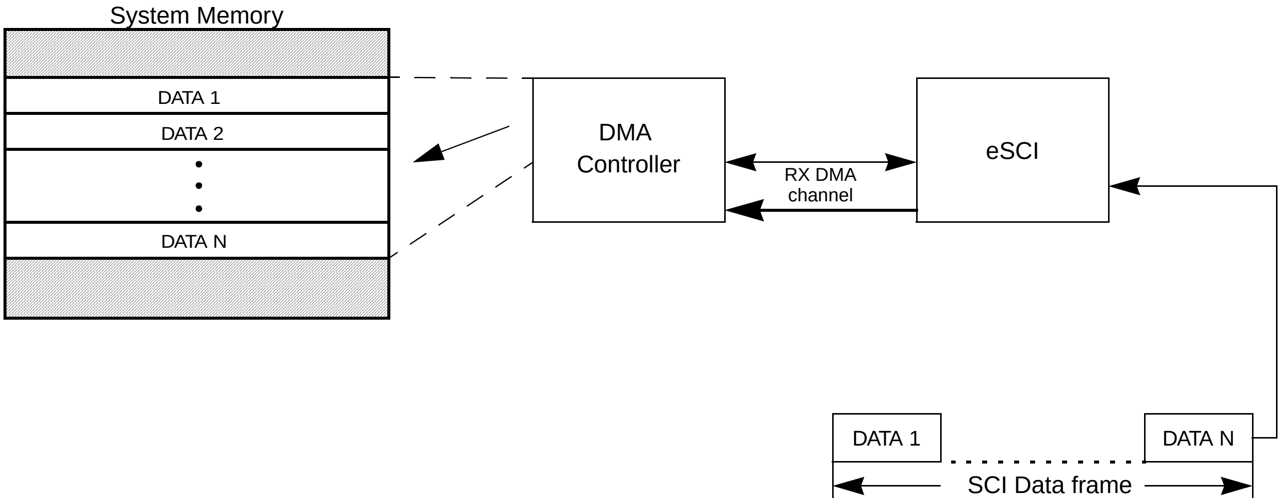


Figure 20-31. DMA Controlled SCI Data Frame Reception

20.4.5.3.11 Receiver Overrun

When the eSCI module has received a frame and attempts to transfer the payload data of the received frame into the SCI Data Register (eSCI_SDR) but neither the application nor the DMA controller has read the SCI Data Register (eSCI_SDR) since its last update, the overrun flag OR in the Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set. The data contained in SCI Data Register (eSCI_SDR) are not changed and the received data are lost.

20.4.5.3.12 Wake-up Frame Reception

This section describes the reception process when the receiver is in the Wake-up state.

When the required number of frame bits have been received, the payload bits of the received frame are transferred into SCI Data Register (eSCI_SDR) if the RDRF flag is 0.

If the *address-mark* wake-up mode is selected and the received frame has the address bit set, the receive data register full flag RDRF in Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set. If the receive interrupt enable bit RIE in the Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set, the RDRF interrupt request is generated. The RWU bit is cleared, and the receiver enters the Run state via the wake1 transition.

If the *idle line* wake-up mode is selected and the receiver has detected an idle character, The RWU bit is cleared, and the receiver enters the Ready state via the wake0 transition.

If any of the receiver errors described in Section 20.4.5.4, “Reception Error Reporting” have been occurred, that corresponding flags will be set.

20.4.5.3.13 Bit Sampling

The receiver samples the selected receiver input (see Section 20.4.5.3.2, “Receiver Input Mode Selection”) with the receiver clock RCLK. The bit sampling for start bit detection is shown in Figure 20-32, the bit sampling for data and stop bit reception is shown in Figure 20-33. The samples indicated by dashed arrows are not used by the receiver. The received data bits are transferred into the internal shift register after the data strobing. If noise or framing errors were detected, this is flagged as described in Section 20.4.5.4, “Reception Error Reporting”

20.4.5.3.14 Bit Synchronization

To adjust for baud rate mismatch, a synchronization of the cyclic receive sample counter RSC is performed during start bit reception as described in Section 20.4.5.3.15, “Start Bit Sampling”.

20.4.5.3.15 Start Bit Sampling

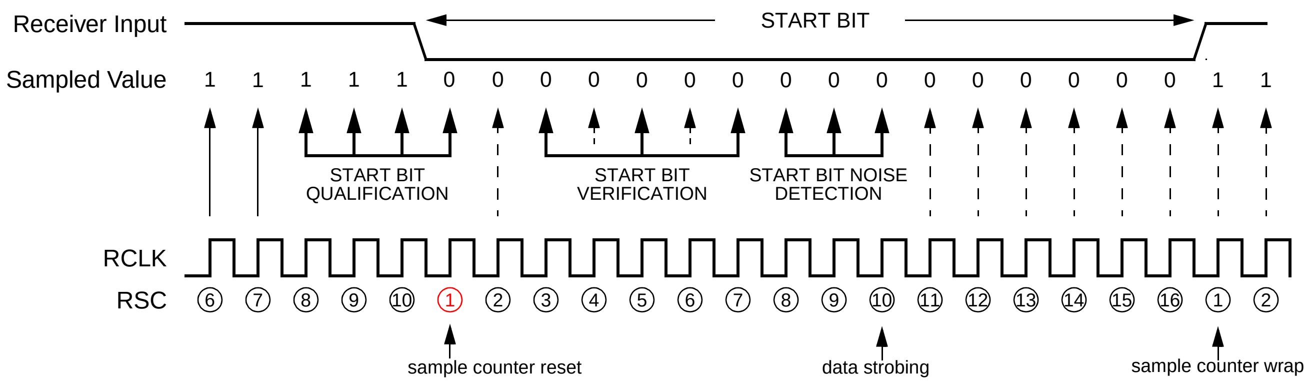


Figure 20-32. Start Bit Sampling and Strobing

The sampling of the start bit consists of three phases, the start bit qualification, the start bit verification, and the start bit noise detection.

Start Bit Qualification

To adjust for baud rate mismatch, the cyclic receive sample counter RSC is re-synchronized after a successful start bit qualification.

A start bit is successfully qualified if

- the start qualification is active, and
- a low sample is read, and
- the low sample was preceded by three consecutive high samples.

The start bit qualification becomes active

- after module reset, or
- after receiver disable and subsequent enable, or
- after the 7-th sample if the start bit verification failed, or
- after the 10-th sample of last stop bit of the preceding frame (example shown in Figure 20-32).

The start bit qualification becomes inactive

- after successful start bit qualification.

Start Bit Verification

After the successful start bit qualification the receiver starts to verify the start bit by a two out of three samples majority voting.

A start bit is verified if at least two out of the three sample RSC3, RSC5, and RSC7 are sampled low. Noise is detected when exactly one out of the three samples is high. In this case, the noise flag eSCI_IFSR1[NF] is set. The result of the start bit verification is summarized in Table 20-32.

Table 20-32. Start Bit Verification Result

[RSC3, RSC5, RSC7]	Start Bit Verified	Verification Noise Detected
000	Yes	No
001	Yes	Yes
010	Yes	Yes
100	Yes	Yes
011	No	No
101	No	No
110	No	No
111	No	No

If the start bit verification was *not successful*, the receiver activates the start bit qualification. If the start bit verification was *successful*, the receiver continues sampling to perform data noise detection on the samples at RSC8, RSC9, and RSC10. The result of the start bit data noise detection is summarized in Table 20-33. If noise is detected, the noise flag eSCI_IFSR1[NF] is set.

Table 20-33. Start Bit Noise Detection

[RSC8, RSC9, RSC10]	Noise Detected
000	No
001	Yes
010	Yes
100	Yes
011	Yes
101	Yes
110	Yes
111	Yes

20.4.5.3.16 Data Bit Sampling

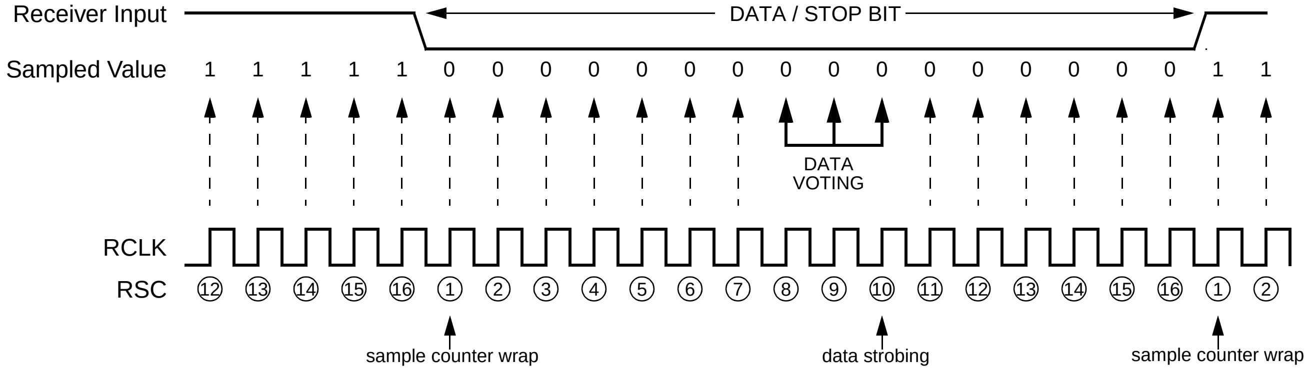


Figure 20-33. Data and Stop Bit Sampling and Strobing

To determine the value of a data bit and to detect noise, a two out of three majority voting is performed on the samples RSC8, RSC9, and RSC10. Table 20-34 summarizes the results of the data bit sample. The receiver detects the number of data bit according to the selected frame format. If noise is detected, the noise flag eSCI_IFSR1[NF] is set.

Table 20-34. Data Bit Sampling

[RSC8, RSC9, RSC10]	Data Bit Value	Noise Detected
000	0	No
001	0	Yes
010	0	Yes
100	0	Yes
011	1	Yes
101	1	Yes
110	1	Yes
111	1	No

20.4.5.3.17 Stop Bit Verification

The reception of a valid stop bit is verified if at least two out of the sample RSC8, RSC9, and RSC10 are sampled high. If this is not that case, a framing error is detected. Noise is detected if not all of the samples are of the same value. In this case, the noise flag eSCI_IFSR1[NF] is set. The result of the stop bit verification is summarized in Table 20-35.

Table 20-35. Stop Bit Verification

[RSC8, RSC9, RSC10]	Stop Bit Verified	Framing Error Detected	Noise Detected
000	No	Yes	No
001	No	Yes	Yes
010	No	Yes	Yes

Table 20-35. Stop Bit Verification (continued)

[RSC8, RSC9, RSC10]	Stop Bit Verified	Framing Error Detected	Noise Detected
100	No	Yes	Yes
011	Yes	No	Yes
101	Yes	No	Yes
110	Yes	No	Yes
111	Yes	No	No

20.4.5.3.18 Parity Checking

The eSCI module calculates the parity of a received character and checks is versus the received parity bit in the received data frame when the parity enable bit PE in the Control Register 1 (eSCI_CR1) is set. The parity type bit PT in the Control Register 1 (eSCI_CR1) defines whether to check for odd or even parity is generated. If an parity error is detected, this is reported as described in Section 20.4.5.4, “Reception Error Reporting”.

20.4.5.4 Reception Error Reporting

The receiver can detect four error types: parity errors, framing errors, noise errors, and the overrun error.

The receiver reports the errors detected during frame reception at the end of the reception of the last stop bit of a frame. For error reporting the receiver utilizes the OR, NF, FE, and PF flags in the Interrupt Flag and Status Register 1 (eSCI_IFSR1).

If the receiver has detected an overrun as described in Section 20.4.5.3.11, “Receiver Overrun”, only the OR flag is set. All other error flags are not updated.

If the receiver has detected noise as described in Section 20.4.5.3.13, “Bit Sampling” the NF flag is set.

If the receiver has *not* detected an overrun and has detected a framing error as described in Section 20.4.5.3.13, “Bit Sampling” the FE flag is set.

If the receiver has *not* detected an overrun and has detected a parity error as described in Section 20.4.5.3.18, “Parity Checking” the PF flag is set.

20.4.5.5 Multiprocessor Communication

The multiprocessor communication allows one processor to send blocks of frames to other processors on the same serial link. To avoid the received data interrupt for frames not intended for the processor, the eSCI receiver can be put into the Wake-up state. If the receiver is in the Wake-up state, the eSCI will still load the received data into the SCI Data Register (eSCI_SDR), but will not set the RDRF flag and consequently not request the RDRF interrupt.

The receiver leaves the Wake-up state and clears the RWU bit in the Control Register 1 (eSCI_CR1) when the wake-up pattern configured by WAKE bit in Control Register 1 (eSCI_CR1) is received. The eSCI module supports two types of wake-up patterns, the idle-line wakup pattern and the address-mark wake-up pattern.

20.4.5.5.1 Idle-Line Wake Up

The idle-line wake-up mode is selected when the WAKE bit in Control Register 1 (eSCI_CR1) is 0. In this mode, the receiver leaves the wake-up state, when an idle character is detected as described in Section 20.4.5.3.8, “Idle Character Detection”. The next received frame is the address frame that contains address information which can be evaluated by the application. If the application decides not to receive the frame block, it can set the RWU bit in the Control Register 1 (eSCI_CR1) and return the receiver to the wake-up state.

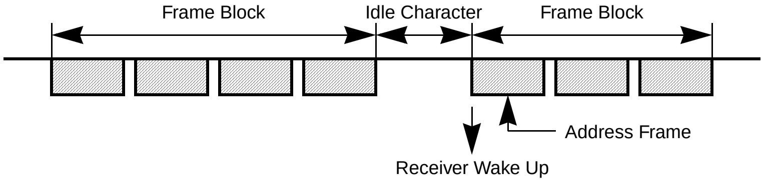


Figure 20-34. Idle-Line Wake Up

20.4.5.5.2 Address-Mark Wake Up

The address-mark wake-up mode is selected when the WAKE bit in Control Register 1 (eSCI_CR1) is 1. If the WAKE bit is set, the address bit is added to the frame format. In this mode, the receiver leaves the wake-up state, when a data frame with the address bit value of 1 was received. This frame is the address frame and contains address information which can be evaluated by the application. If the application decides not to receive the frame block, it can set the RWU bit in the Control Register 1 (eSCI_CR1) and return the receiver to the wake-up state. All data frames that belong to the frame block must have the address bit cleared.

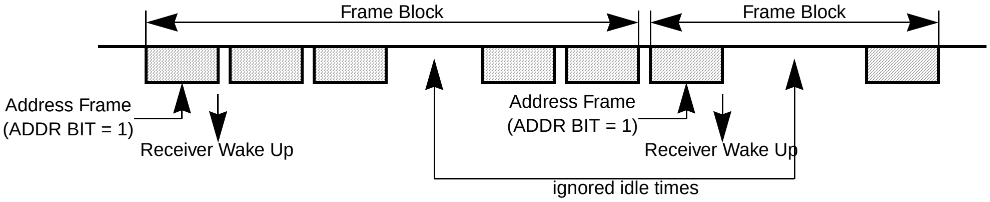


Figure 20-35. Address-Mark Wake Up

20.4.6 LIN Mode

The eSCI provides support for the LIN protocol. It can be used to automate most tasks of a LIN master. In conjunction with the DMA interface it is possible to transmit entire LIN frames and sequences of LIN frames as well as to receive data from LIN slaves without application intervention. There is no special support for LIN slave mode.

20.4.6.1 LIN Mode Configuration

The application must configure the following bits and fields in order to achieve correct LIN operation. The configuration of bits and fields not mentioned in this section depend on the connected LIN slaves and the current application.

- enable *LIN* Mode
  - LIN Control Register 1 (eSCI_LCR1)[LIN]:= 1
- select *RXD* pin as receiver input
  - Control Register 1 (eSCI_CR1)[LOOPS]:= 0
  - Control Register 1 (eSCI_CR1)[RSRC]:= 0
- select *LIN byte fields* as used frame format
  - Control Register 1 (eSCI_CR1)[M]:= 0
  - Control Register 1 (eSCI_CR1)[PE]:= 0
  - Control Register 1 (eSCI_CR1)[WAKE]:= 0
  - Control Register 3 (eSCI_CR3)[M2]:= 0
- select *break character length* of 13 bit as required by LIN 2.0
  - Control Register 2 (eSCI_CR2)[BRCL]:= 1
- select transmission stop on *bit error* detection
  - Control Register 2 (eSCI_CR2)[BESTP]:= 1
- select transmission DMA stop on *bit error* detection
  - Control Register 2 (eSCI_CR2)[BSTP]:= 1
- enable both *transmitter* and *receiver*
  - Control Register 1 (eSCI_CR1)[TE]:= 1
  - Control Register 1 (eSCI_CR1)[RE]:= 1

20.4.6.2 LIN frame formats

The term LIN frame refers to a sequence of LIN byte fields preceded by a break character, both are described in Section 20.4.2, “Frame Formats”. The eSCI module allows to generate LIN frames for LIN slaves of LIN standards 1.3 and 2.0.

20.4.6.2.1 LIN byte field reception

The reception of a LIN byte field starts with the successful start bit qualification and is finished with the reception of the 16-th sample of the stop bit when no start bit start bit qualification pattern has been detected. If a start bit start bit qualification pattern has been detected at or after the 10-th sample of the stop bit, the reception ends at this sample. An ongoing reception is indicated by the RACT status bit in Interrupt Flag and Status Register 1 (eSCI_IFSR1).

The RACT flag is set if all of the following conditions are fulfilled,

1. the receiver is enabled (eSCI_CR1[RE] = 1), and
2. the LIN task is not in reset (eSCI_LCR1[LRES] = 0), and

3. the start bit start bit qualification pattern has been received (see Section 20.4.5.3.15, “Start Bit Sampling”).

The RACT flag is cleared if at least one of the following conditions is fulfilled,

1. the receiver is disabled (eSCI_CR1[RE] = 0), or
2. the LIN task is in reset (eSCI_LCR1[LRES] = 1), or
3. the start bit verification fails at sample 7 according to Table 20-32, or
4. the 16-th sample of the stop bit has been received and no start bit qualification pattern has been detected at or after the 10-th sample.

20.4.6.2.2 Standard LIN frames

A standard LIN frame, shown in Figure 20-36 consists of a break character, a sync field, an ID field, zero or more data fields, and a checksum field. The data fields and the checksum field are generated by the LIN master for TX LIN frames and generated by the LIN slave for RX LIN frames. The header fields will always be generated by the LIN master.

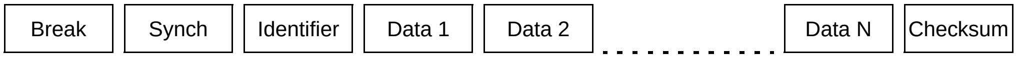


Figure 20-36. Standard LIN frame format

20.4.6.2.3 CRC Enhanced LIN frames

The CRC Enhanced LIN frames shown in Figure 20-37 contain two additional CRC byte fields. These fields are located between the last data field and the Checksum field. The value of the CRC is calculated on the same byte fields as the Checksum is calculated on. The polynom used for the CRC calculation is defined by LIN CRC Polynomial Register (eSCI_LPR). The eSCI module generates the CRC fields for TX frames and checks the CRC fields for RX frames if the CRC bit in the LIN Transmit Register (eSCI_LTR) was written with a value of 1.

Figure 20-37. CRC Enhanced LIN frame format

The CRC Enhanced LIN frames are not part of the LIN standard.

20.4.6.3 LIN TX Frame generation

The eSCI module supports two modes of LIN TX Frame generation, the CPU controlled mode and the DMA controlled mode. In the CPU controlled mode, the application provides the required frame configuration and frame data by subsequent CPU write accesses to the LIN Transmit Register (eSCI_LTR). In the DMA controlled mode, the DMA controller provides the required frame configuration and frame data in response to DMA requests generated by the eSCI module.

20.4.6.3.1 CPU Controlled LIN TX Frame generation

In this mode, the application initiates the generation of an LIN TX Frame and provides the data to be transmitted by a sequence of subsequent CPU write accesses to the LIN Transmit Register (eSCI_LTR). When the eSCI module has processed the data written into LIN Transmit Register (eSCI_LTR), the TXRDY interrupt flag in the Interrupt Flag and Status Register 2 (eSCI_IFSR2) will be set.

The application should clear the TXRDY interrupt flag *before* writing data into the LIN Transmit Register (eSCI_LTR) because the eSCI module will set the TXRDY one clock cycle after the write access.

The first data written to the LIN Transmit Register (eSCI_LTR) provides the Identifier and Identifier Parity fields. The second data written defines the number of data bytes to be transmitted. The third data written defines the CRC and checksum generation. The TD bit has to set to 1 in order to invoke the LIN TX frame generation. The value of the TO field is ignored by the eSCI module for LIN TX frames.

After the third data was written the generation of a LIN TX frame is started. Firstly, a break field is transmitted, then the synch field and the protected identifier field.

All subsequent write accesses to the LIN Transmit Register (eSCI_LTR) provide data bytes to be transmitted via the LIN bus. A data byte field will be transmitted as soon as data are available. After the last data byte, defined by the value written to the LEN field, was send out, the configured CRC and checksum fields will be send out.

After the transmission of the checksum field of the LIN TX frame, the write access counter for the LIN Transmit Register (eSCI_LTR) is reset and the FRC interrupt flag in the Interrupt Flag and Status Register 2 (eSCI_IFSR2) is set.

20.4.6.3.2 DMA Controlled LIN TX Frame generation

In this mode, the eSCI module controls the generation of an LIN TX Frame. When new data required for transmission, the eSCI module generates the transmit DMA request and the DMA controller delivers the required data. The application request the eSCI module to enter this mode by setting the TXDMA bit in the Control Register 2 (eSCI_CR2). From this point in time, the module start the generation of DMA requests and initiates the frame transmission. Before entering this mode, the application should perform the following actions:

1. Configure the module for LIN mode.
2. Enable the transmitter by setting TE in Control Register 1 (eSCI_CR1) to 1.
3. Setup the DMA controller channel and provide frame data in system memory

A block diagram which presents an overview of the DMA Controlled LIN TX Frame is shown in Figure 20-38. The content of the fields in the memory is the same as described in LIN Transmit Register (eSCI_LTR) - LIN TX frame generation.

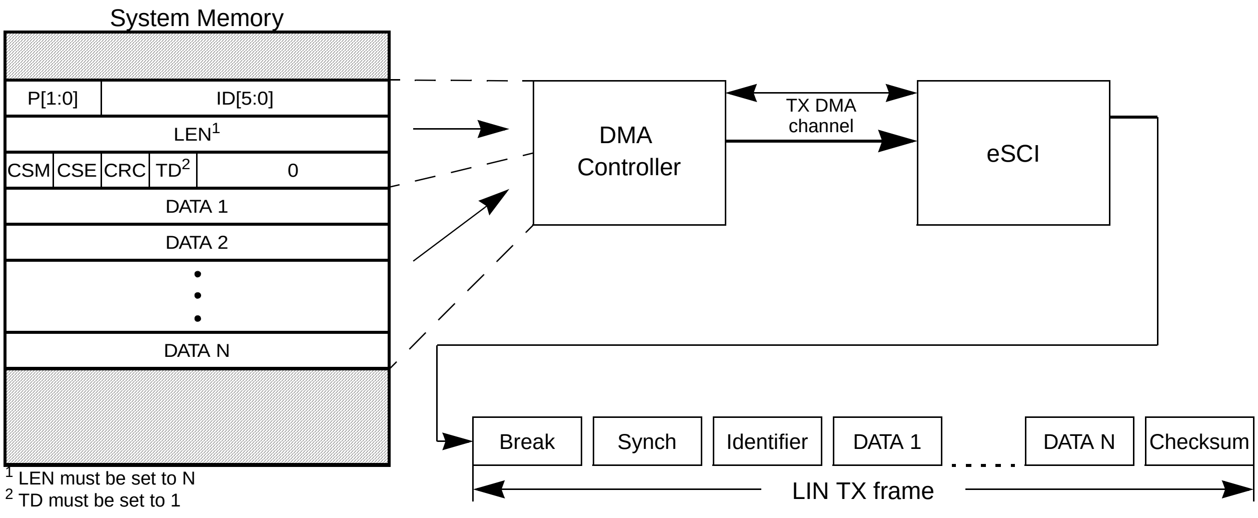


Figure 20-38. DMA Controlled LIN TX Frame generation

20.4.6.4 LIN RX frame generation

The eSCI module supports two modes of LIN RX Frame generation and reception, the CPU controlled mode and the DMA controlled mode. In the CPU controlled mode, the application provides the required data by subsequent CPU write accesses to the LIN Transmit Register (eSCI_LTR) and retrieves the received data by subsequent CPU read accesses to the LIN Receive Register (eSCI_LRR). In the DMA controlled mode, the DMA controller provides the required frame configuration data in response to DMA requests generated by the eSCI module and transfers the received frame data to the memory in response to DMA requests generated by the eSCI module.

20.4.6.4.1 CPU Controlled LIN RX Frames generation

In this mode, the application initiates the generation of an LIN RX Frame by a sequence of subsequent CPU write accesses to the LIN Transmit Register (eSCI_LTR). When the eSCI module has processed the data written into LIN Transmit Register (eSCI_LTR), the TXRDY interrupt flag in the Interrupt Flag and Status Register 2 (eSCI_IFSR2) will be set.

The application should clear the TXRDY interrupt flag *before* writing data into the LIN Transmit Register (eSCI_LTR) because the eSCI module will set the TXRDY one clock cycle after the write access.

The first data written to the LIN Transmit Register (eSCI_LTR) provides the Identifier and Identifier Parity fields. The second data written defines the number of data bytes requested from the LIN slave. The third data written defines the CRC and checksum generation. The TD bit has to set to 0 to invoke the RX frame generation. The TO field defines the upper part of the timeout value. The fourth byte written defines the lower part of the timeout value.

After the fourth byte was written the generation of a LIN RX frame is started. Firstly, a break field is transmitted, then the synch field and the protected identifier field. After the transmission of the protected identifier, the eSCI module starts to receive the frame data transmitted by the LIN slave. When the module has received a complete byte field, the received data are transferred into the LIN Receive Register (eSCI_LRR) and the receive data ready flag RXRDY in the Interrupt Flag and Status Register 2 (eSCI_IFSR2) is set.

The application can retrieve the received data by subsequent read access from LIN Receive Register (eSCI_LRR) after checking the RXRDY flag. The application should clear the RXRDY flag immediately after reading the LIN Receive Register (eSCI_LRR).

After the reception of the configured number of data from the slave, the module starts the reception of the configured CRC and Checksum byte fields. These data are not transferred into the LIN Receive Register (eSCI_LRR). The CRC and Checksum checking is performed internally. In case of errors, they will be reported as described in Section 20.4.6.5, “LIN Error Reporting”

After the reception of the checksum field of the LIN RX frame, the FRC interrupt flag in the Interrupt Flag and Status Register 2 (eSCI_IFSR2) is set.

20.4.6.4.2 DMA Controlled LIN RX Frames generation

In this mode, the eSCI module controls the generation of LIN RX frame header and the reception of the frame data automatically and utilizes the two connected DMA channels. A block diagram which presents an overview of the DMA Controlled LIN RX Frame generation and reception is shown in Figure 20-38. The content of the header fields in the memory is the same as described in LIN Transmit Register (eSCI_LTR) - LIN RX frame generation. The TX DMA channel is used the fetch the LIN RX frame header and control information. The RX DMA channel is used to transfer the received frame data into the memory.

When new data required for transmission, the module generates the transmit DMA request and the DMA controller delivers the required data. When new data was received, the module generates the receive DMA request and the DMA controller retrieves the provided data.

The application request the eSCI module to enter this mode by setting the RXDMA bit in the Control Register 2 (eSCI_CR2). From this point in time, the module start the generation of DMA requests and frame transmission and reception. Before entering this mode, the application should perform the following actions:

1. Configure the module for LIN mode.
2. Enable transmitter and receiver by setting TE and RE in Control Register 1 (eSCI_CR1) to 1.
3. Setup the two DMA controller channels and provide frame header data in system memory.

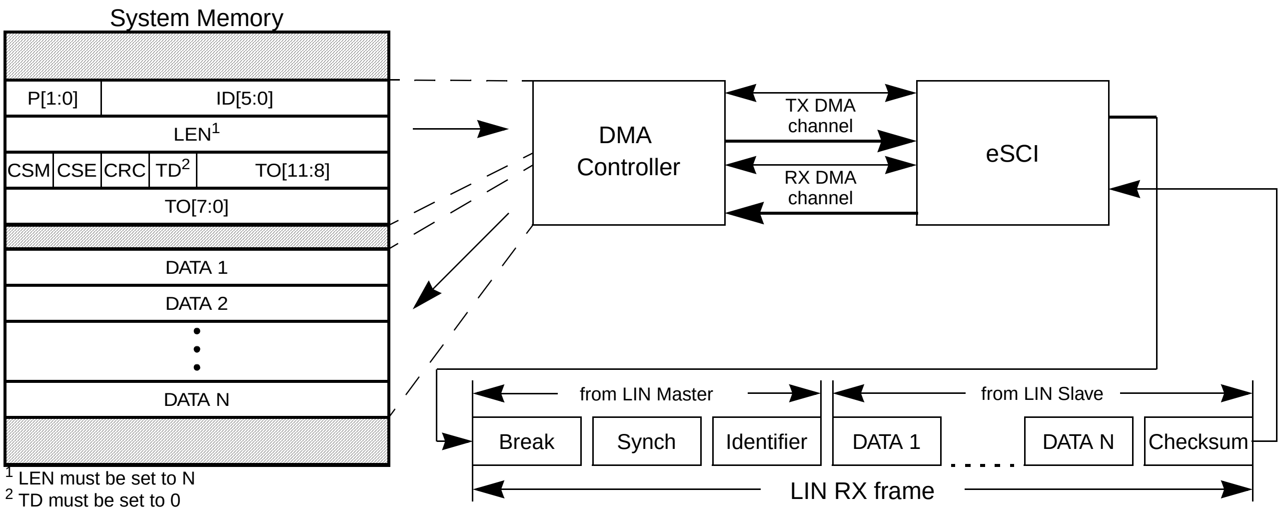


Figure 20-39. DMA Controlled LIN RX Frame generation and reception

20.4.6.5 LIN Error Reporting

This section describes error checking and the signaling of detected errors in LIN mode.

20.4.6.5.1 Physical Bus Error Detection

If the receiver input is sampled 0 for at least 31 sample clock cycles after the start of the transmission of a LIN frame, the physical bus error flag PBERR in the Interrupt Flag and Status Register 2 (eSCI_IFSR2) will be set.

20.4.6.5.2 Unrequested Activity Detection

If an unrequested byte is received (i.e. a byte which is not part of an RX frame) which is not recognized as a wake-up or break character, the bit error flag BERR in the Interrupt Flag and Status Register 2 (eSCI_IFSR2) is set. In addition the RXRDY flag will also be set, the LINRX register must be read before normal operations can proceed.

20.4.6.5.3 Standard Bit Error Detection

The standard bit error detection is enabled when the fast bit error detection control bit FBR in the Control Register 2 (eSCI_CR2) is 0. The standard bit error detection is performed after each LIN byte field transmission.

During the transmission of the LIN frame header and LIN frame data, the receiver is running and receives the signal values on the serial bus. After the complete transmission and the related reception of a LIN byte field, the eSCI compares the data that was transmitted and the data that has been received. If they do not match, the bit error interrupt flag BERR in the Interrupt Flag and Status Register 2 (eSCI_IFSR2) is set.

20.4.6.5.4 Fast Bit Error Detection

Fast Bit Error Detection has been designed to allow flagging of LIN bit errors while they occur, rather than flagging them after a byte transmission has completed (see Figure 20-40).

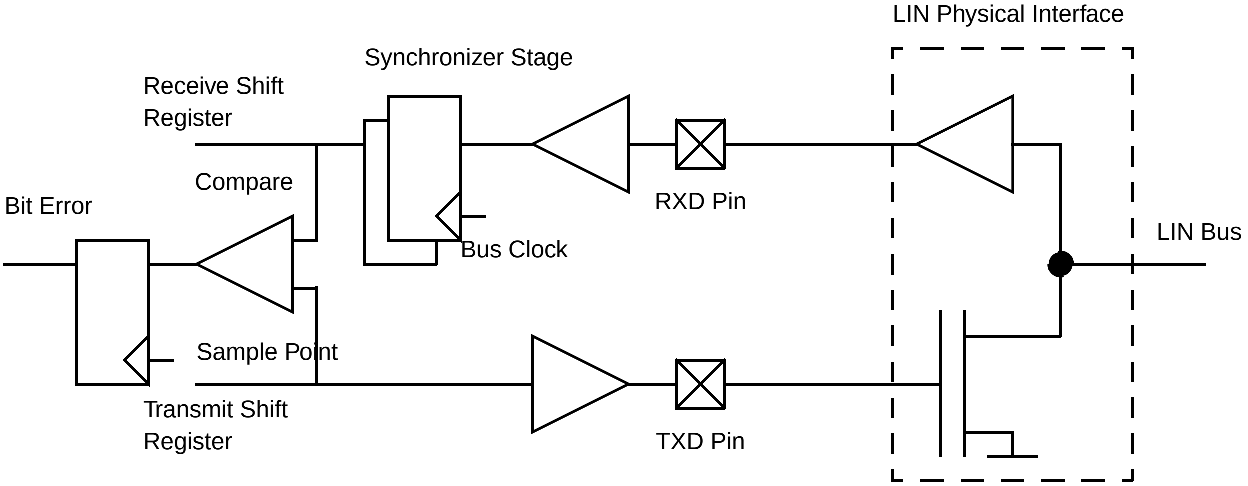


Figure 20-40. Fast Bit Error Detection on a LIN Bus

If fast bit error detection bit FBR in the Control Register 2 (eSCI_CR2) is set the eSCI will compare the transmitted and the received data stream while the transmitter is active (not idle). Once a mismatch between the transmitted data and the received data is detected the following actions are performed the bit error flag BERR will be set.

To adjust to different bus loads the sample point at which the incoming bit is compared to the one which was transmitted can be selected with the BESM bit in the Control Register 2 (eSCI_CR2). If eSCI_CR2[BESM] = 1, the comparison will be performed with sample RS13, otherwise with RS9 (see Figure 20-41) (also see Section 20.4.5.3.13, “Bit Sampling).

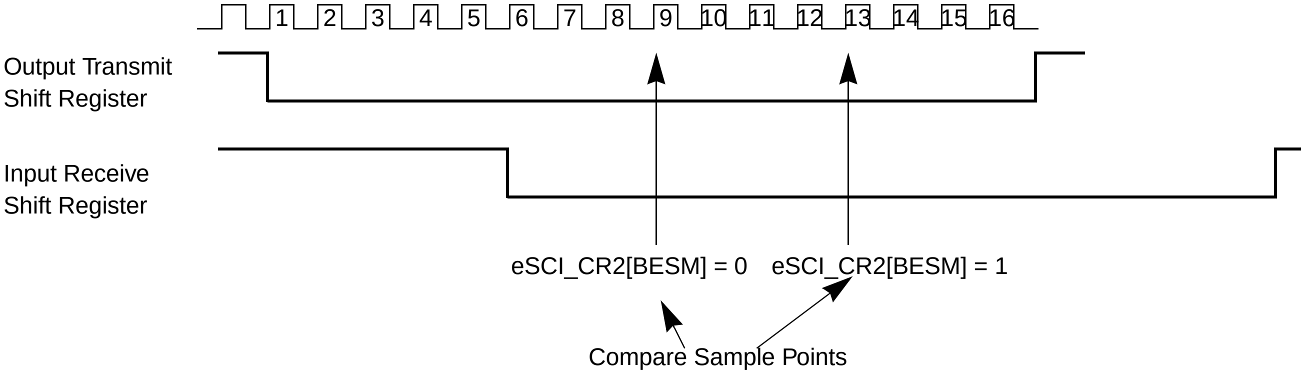


Figure 20-41. Timing Diagram Fast Bit Error Detection

NOTE

To calculate the exact position of the sample point with regard to the RX pin, the delays through the pads and the two Bus Clock cycle delay through the input synchronizer also needs to be taken into account.

20.4.6.5.5 Slave-Not-Responding-Error Detection

The Slave-Not-Responding-Error is defined in LIN Specification Package Revision 1.3; December 12, 2002; 6 ERROR AND EXCEPTION HANDLING. The LIN specification requires that a NO_RESPONSE_ERROR has to be detected if a message frame is not fully completed within the maximum length T_{FRAME_MAX} by any slave task upon transmission of the SYNCH and IDENTIFIER fields. The maximum frame length TFRAME_MAX is defined in LIN Specification Package Revision 1.3; December 12, 2002; 3.3 LENGTH OF MESSAGE FRAME AND BUS SLEEP DETECT, as

$$T_{FRAME\_MAX} = (10 \cdot N_{DATA} + 45) \cdot 1.4 \qquad \textit{Eqn. 20-11}$$

where N_{DATA} is the number of data byte fields of the message frame.

The STO interrupt flag in the Interrupt Flag and Status Register 2 (eSCI_IFSR2) will be set, if an LIN RX frame was not fully received in the amount of time specified in the timeout value field TO in the LIN Transmit Register (eSCI_LTR). The time period starts with the falling edge of the transmitted LIN break character and is specified in units of transmit bits.

To achieve LIN compliant Slave-Not-Responding-Error detection, the timeout value TO in the LIN Transmit Register (eSCI_LTR) field has to be set to T_{FRAME_MAX} when a LIN RX frame is initiated.

20.4.6.5.6 Checksum Error Detection

If the checksum enable bit CSE in the LIN Transmit Register (eSCI_LTR) was set, the checksum checking is performed based on the received checksum byte. The checksum mode is selected by the CSM bit in the LIN Transmit Register (eSCI_LTR). If the value received in the checksum bytes did not match the calculated checksum, the checksum error flag CKERR in the Interrupt Flag and Status Register 2 (eSCI_IFSR2) will be set.

20.4.6.5.7 CRC Error Detection

The CRC checking is performed on the two received CRC bytes CRC1 and CRC2 if the CRC Enhanced LIN frame format was selected by the CRC bit in the LIN Transmit Register (eSCI_LTR). If the value received in the two CRC bytes did not match the calculated CRC pattern, the CRC error flag CERR in the Interrupt Flag and Status Register 2 (eSCI_IFSR2) will be set.

20.4.6.5.8 Overflow Detection

When the receiver has received the next byte field, which should be transferred into the LIN Receive Register (eSCI_LRR), but neither the application nor the RX DMA channel have read data from this register since the last update, the received data overflow flag OVFL in the Interrupt Flag and Status Register 2 (eSCI_IFSR2) will be set. In this case the content of the LIN Receive Register (eSCI_LRR) is not changed. The data received most recently are lost.

20.4.6.6 LIN Wake Up

The section describes the LIN Wake Up behavior of the eSCI module.

20.4.6.6.1 LIN Wake-Up Request Generation

The eSCI module can cause the LIN bus to exit the sleep mode by sending a wake-up signal frame, which consists of a wake-up signal 0x80 (consisting of 8 dominant bits followed by 1 recessive bit), followed by the wake-up delimiter period as defined by the WUD field in the LIN Control Register 1 (eSCI_LCR1).

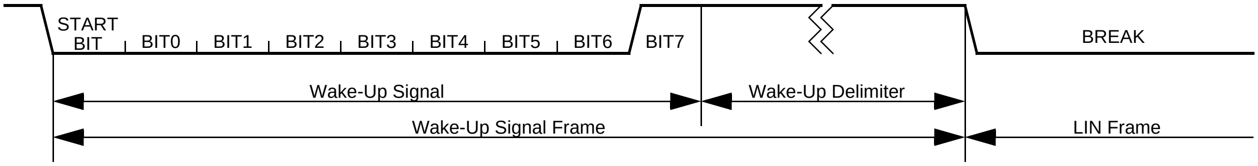


Figure 20-42. LIN Wake-Up Signal Frame

The application triggers the transmission of a wake-up signal frame by writing 1 to the LIN bus wake-up trigger WU in the LIN Control Register 1 (eSCI_LCR1).

The LIN Specification 2.0 requires the generation of LIN wake-up signals as dominant pulses longer than 250 μs and shorter than 5 ms. To achieve this, the eSCI module has to programmed to a baud rate between 32 kBaud and 1.6 kBaud. With each of these baud rate settings, the wake-up signal is transmitted as a dominant pulse longer than 250 μs and shorter than 5 ms.

20.4.6.6.2 LIN Wake-Up Request Detection

The eSCI module detects a LIN wake-up requests when

c) one of the characters 0x00, 0x80, or 0xC0 has been received,
d) followed by zero or more low bits,
e) followed by at least one high bit, and
f) no LIN frame transmission or reception is started or running during the reception above

If a LIN wake-up request has been detected, the LIN wake-up flag LWAKE in the Interrupt Flag and Status Register 2 (eSCI_IFSR2) will be set after the reception of the first high bit.

The LIN Specification 2.0 requires the detection of LIN wake-up requests as dominant pulses longer than 150 μs. To achieve this, the eSCI module has to programmed to the maximum baud rate that is not greater than 43.77 kBaud. With this baud rate setting, any dominant pulse longer than 150 μs is decoded as at least 7 dominant bits (one start and 6 data bits) and consequently as one of the characters 0xC0, 0x80, or 0x00.

20.4.6.7 LIN Protocol Engine Stop and Reset

The LIN protocol engine is stopped and reset when the application set the LRES control bit in the LIN Control Register 1 (eSCI_LCR1) to 1. In this case, the LIN protocol engine will stop immediately. No new transmissions or receptions are initiated, the LIN serial bus is driven with the recessive value 1. Additionally to the stop and reset of the LIN protocol engine the receiver and transmitter modules are stopped and reset as well, and the receive and transmit DMA requests are deasserted.

In order to start the LIN Protocol Engine with idle transmitter and receiver processes, the LRES bit should be asserted until all of the status bits DACT, LACT, TACT, and RACT in the Interrupt Flag and Status

Register 1 (eSCI_IFSR1) are cleared. Theses status bits are cleared within one bit time after assertion of the LRES bit.

20.4.7 Interrupts

This section describes the interrupt sources and interrupt request generation.

20.4.7.1 Interrupt Flags and Enables

All interrupt sources, interrupt flags, and interrupt enable bits are listed in Table 20-36. This table indicates the operational modes, where the interrupt flags can be set by the eSCI module.

Table 20-36. eSCI Interrupt Flags and Interrupt Enable Bits

Interrupt Source	Operational Mode	Interrupt Flag	Interrupt Enable Bit
Transmitter	SCI	eSCI_IFSR1[TDRE]	eSCI_CR1[TIE]
Transmitter	SCI, LIN	eSCI_IFSR1[TC]	eSCI_CR1[TCIE]
Receiver	SCI	eSCI_IFSR1[RDRF]	eSCI_CR1[RIE]
Receiver	SCI	eSCI_IFSR1[IDLE]	eSCI_CR1[ILIE]
Receiver	SCI	eSCI_IFSR1[OR]	eSCI_CR2[ORIE]
Receiver	SCI, LIN	eSCI_IFSR1[NF]	eSCI_CR2[NFIE]
Receiver	SCI, LIN	eSCI_IFSR1[FE]	eSCI_CR2[FEIE]
Receiver	SCI	eSCI_IFSR1[PF]	eSCI_CR2[PFIE]
Receiver	LIN	eSCI_IFSR1[BERR]	eSCI_CR2[BERRIE]
Receiver	LIN	eSCI_IFSR2[RXRDY]	eSCI_LCR1[RXIE]
Transmitter	LIN	eSCI_IFSR2[TXRDY]	eSCI_LCR1[TXIE]
Receiver	LIN	eSCI_IFSR2[LWAKE]	eSCI_LCR1[WUIE]
Receiver	LIN	eSCI_IFSR2[STO]	eSCI_LCR1[STIE]
Receiver	LIN	eSCI_IFSR2[PBERR]	eSCI_LCR1[PBIE]
Receiver	LIN	eSCI_IFSR2[CERR]	eSCI_LCR1[CIE]
Receiver	LIN	eSCI_IFSR2[CKERR]	eSCI_LCR1[CKIE]
Receiver	LIN	eSCI_IFSR2[FRC]	eSCI_LCR1[FCIE]
Receiver	LIN	eSCI_IFSR2[UREQ]	eSCI_LCR2[URIE]
Transmitter, Receiver	LIN	eSCI_IFSR2[OVFL]	eSCI_LCR2[OFIE]

20.4.7.2 Interrupt Request Generation

The eSCI module provides one hardware interrupt request signal to the systems interrupt controller. This interrupt request signal is asserted if and only if at least one of the interrupt flags and the corresponding interrupt enables are set to 1. Otherwise the interrupt line is deasserted.

20.5 Application Information

20.5.1 SCI Data Frames Separated by Preamble

To separate SCI data frame with preambles with minimum idle line time, use this sequence between messages:

1. write to SCI Data Register (eSCI_SDR)
   - this sets the internal iCMT bit which requests the data transmission
2. wait until TDRE in Interrupt Flag and Status Register 1 (eSCI_IFSR1) is set
   - this indicates the start of transmission; the iCMT bit was cleared
3. clear and subsequently set the TE bit in Control Register 1 (eSCI_CR1)
   - this set the internal iPRE bit which requests the preamble transmission
4. write to SCI Data Register (eSCI_SDR)
   - this sets the internal iCMT bit which requests the data transmission

The priority scheme of the transmitter which is described in Table 20-28 ensures, that the preamble is transmitted before the data frame.

Chapter 21
Enhanced Time Processing Unit (eTPU)

21.1 Introduction

eTPU is an intelligent, semi-autonomous co-processor designed for timing control. Operating in parallel with the Host CPU, the eTPU processes instructions, real-time input events, performs output waveform generation, and accesses shared data without Host intervention. Consequently, for each timer event, the Host CPU setup and service times are minimized or eliminated.

High-level assembler, compiler and documentation allows customers to develop their own functions on the eTPU.

eTPU is an enhanced version of the TPU module. Although there is no compatibility at microcode level, eTPU maintains several features of older TPU versions, making it easy to port older applications, at the same time adding several features listed in Section 21.1.2.2, “eTPU Enhancements over TPU3”.

This document also includes the new features belonging to the version of the eTPU known as eTPU2. The new features are summarized in Section 21.1.2.3, “eTPU2 Enhancements over eTPU”.

eTPU architecture aims at high resolution timing capabilities. From a system perspective, high resolution timing is limited by Host CPU overhead required for servicing timing tasks such as period measurement, pulse measurement, pulse width modulated waveform generation, etc. On the eTPU, high resolution timing is achieved by three main capabilities:

- Reduced latency: pin actions are immediate.
- Reduce or eliminate host interrupt service time.
- Double action channel capability reducing the channel request rate.

eTPU provides higher resolution than the Host CPU can achieve and creates no Host overhead for servicing timing tasks.

Latency is the interval from occurrence of an event to the start of event servicing. eTPU can service its own events without interrupting the Host. There are two types of timing events:

- Input pin transition.
- Selected Time Base match, i.e., a selected Time Base counter reached or exceeded a pre-programmed value

Service time is the time spent servicing an event. In general, in microcontrollers the service time is constrained because the instruction set is not optimized for time function synthesis. The eTPU instruction set is optimized, so that time functions can be implemented with much fewer instructions than the Host CPU. Instructions execute faster, service time is reduced and program memory compacted.

Instructions executed by the eTPU are connected directly to the eTPU timing hardware and allow parallelism of hardware related actions.

21.1.1 Overview

Figure 21-1 shows a top-level eTPU A/B Module block diagram. It displays a dual eTPU Engine configuration. The eTPU C Module contains a single eTPU Engine configuration.

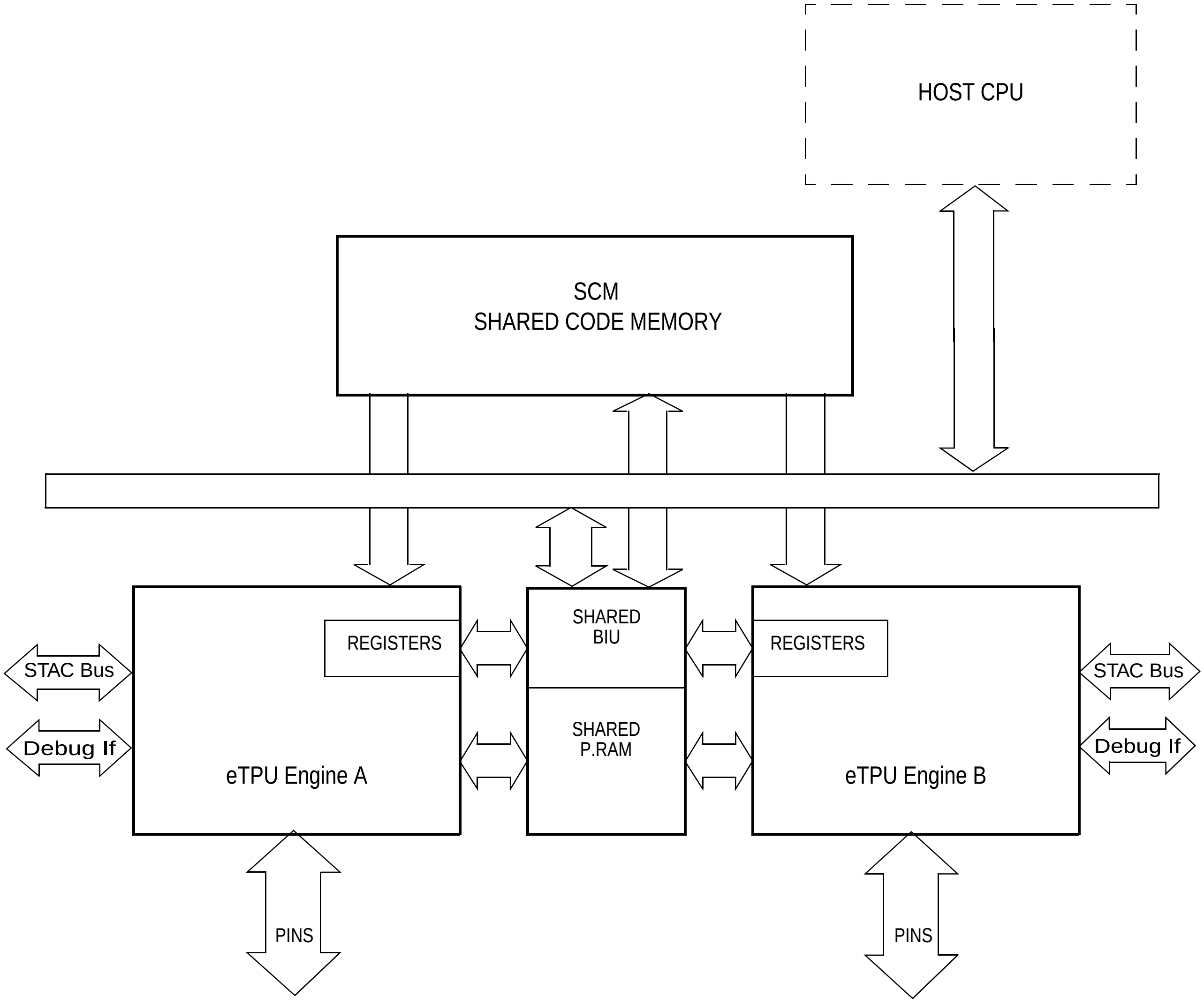


Figure 21-1. eTPU A/B Module Block Diagram

eTPU Engine is responsible for processing input pin transitions and output pin waveform generation based on the **Time Bases**. Each eTPU Engine has its own microprocessor and dedicated hardware for processing signals on I/O pins and can also interface with external time bases through the STAC bus.

Both eTPU Engine CPUs, hereafter called **microengines**, fetch microinstructions from a **Shared Code Memory - SCM**.

Shared Data Memory - SDM - holds eTPU application parameters and work data. It is accessed by Host and both microengines.

Bus Interface Unit - BIU - allows Host to access eTPU registers, SCM and SDM.

Each I/O signal pair is associated with a dedicated **Channel**, which provides hardware for input signal processing and output signal generation, in relationship with selected Time Bases.

The eTPU, as a microprocessed subsystem, works much like a typical real-time system: it runs microengine code from instruction memory (SCM) to handle specific events, accessing data memory (SDM) for parameters, work data and application state info; events may originate from I/O Channels (due to pin transitions and/or time base matches), Host CPU requests or inter-channel requests; events that call for local eTPU processing activate the microengine by issuing a **Service Request**. The Service Request microcode may set an interrupt to the Host CPU. I/O channel events cannot directly interrupt the Host CPU.

Each Channel is associated with a **Function**, which defines its behavior: the Function is a software entity consisting, within the eTPU, of a set of microengine routines that attend to Service Requests. The Function routines are also responsible for Channel configuration. Function routines reside in SCM, which may contain several Functions. A Function may be assigned to several Channels, but a Channel can be associated with just one Function at a given moment. The association between Functions and Channels is defined by Host CPU, and is explained in detail in the eTPU Reference manual.

eTPU hardware supplies resource sharing features that support concurrency:

- a hardware **Scheduler** dispatches the Service Request microengine routines based on a set of priorities defined by the Host CPU. Each Channel has its associated priority;
- a Service Request routine cannot be interrupted until it ends. This sequence of uninterrupted instruction execution is called a **Thread**.
- Channel-specific context (registers and flags) is automatically switched between the end of a Thread and the beginning of the next one.
- SDM arbitration, a dual-parameter coherency controller and semaphores can be used to ensure coherent access to eTPU data shared by both eTPU Engines and Host CPU.

21.1.1.1 eTPU Engine

Each eTPU Engine consists of two 24-bit time bases, 32 independent timer channels, a task scheduler, a microengine, and a Host interface. In addition, each eTPU module has a 32-bit Shared Data Memory (SDM) used for data storage and for passing information between the eTPU Engines and the Host CPU.

Figure 21-2 shows the block diagram for the eTPU Engine.

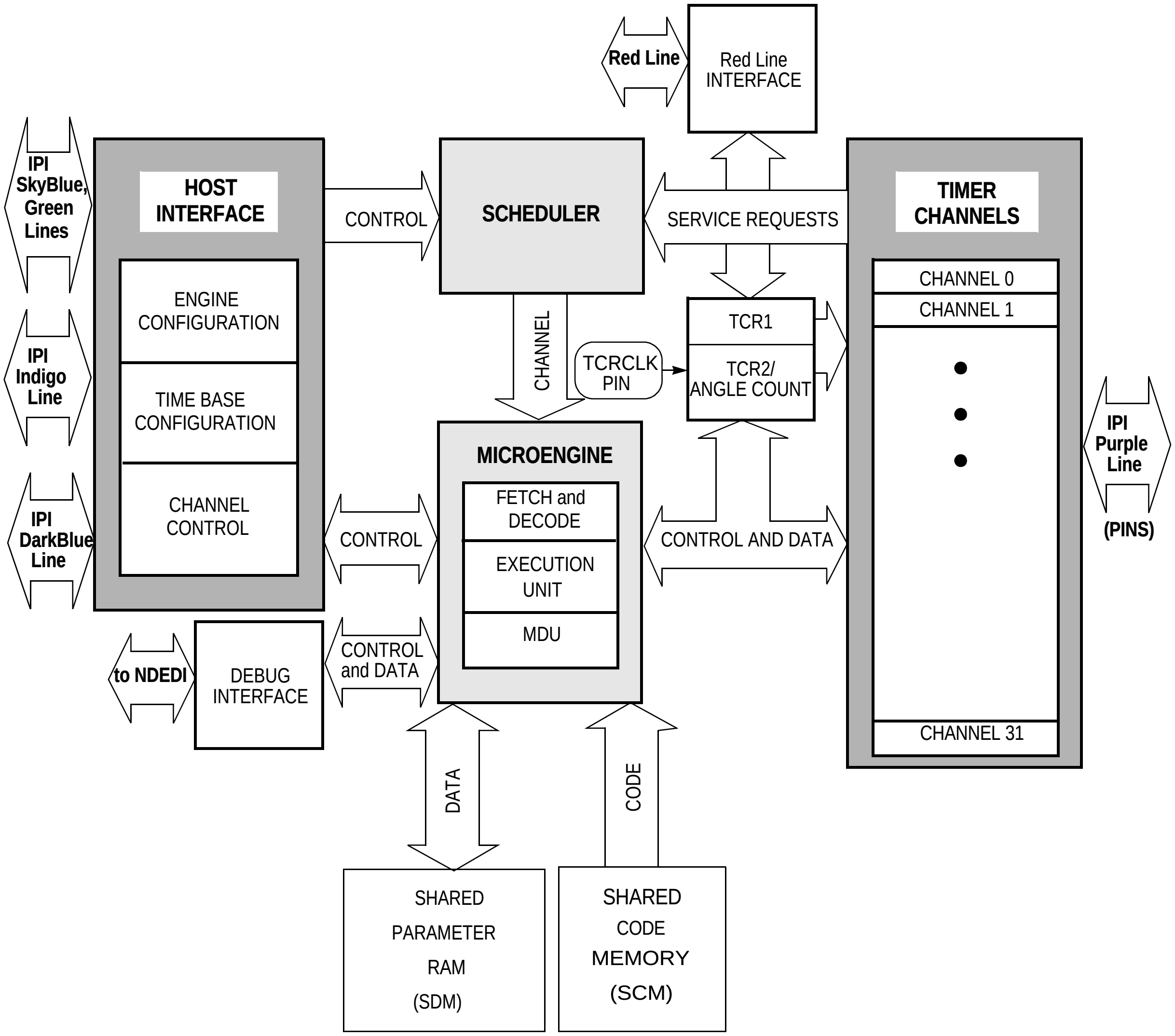


Figure 21-2. eTPU Engine Block Diagram

Throughout this document, the term “eTPU” is sometimes used in place of “eTPU Engine”.

21.1.1.1.1 Time Bases

Two 24-bit counters TCR1 and TCR2 provide reference time bases for all match and input capture events. Prescalers for both time bases are controlled by the Host CPU through bit fields in the eTPU Engine configuration registers. The eTPU is able to export/import time to/from TCR1 or TCR2 in accordance to the Red Line bus specification.

The clock for each of TCR1 and TCR2 clock can be independently derived from the eTPU clock or from an external input via the TCRCLK clock pin. In addition, the TCR2 timebase can be derived from special angle-clock hardware which enables implementing angle-based functions. This feature is added to support advanced angle based engine control applications.

For further details refer to Section 21.3.5, “Time Bases.

21.1.1.1.2 eTPU Timer Channels

Each eTPU Engine has 32 independent channels, each corresponding to an Input/Output signal pair. The channels time resolution is 24 bits, and are all identical.

Each channel consists of logic which supports two events and output controls. The event logic contains two 24-bit capture registers, two 24-bit match registers, greater-equal and equal-only comparators. Supporting two events enables many combinations of double-action functions (for example the channel can handle two events with a single microcode service).

The channel configuration can be changed by the microengine on the fly. Each channel can perform double capture, double match and other capture-match combinations. Channel modes available can do ordered or unordered match. Some modes are also provided that can block one match by the occurrence of the other. Service request can be generated on one or both of the match events.

Input signal can be separated from output signal in each channel. They can, optionally, be combined in a single I/O pin driver. An output buffer enable signal, controlled by microcode, is provided for this case. Digital filters are provided for the input signals, with distinct filtering modes available.

Each channel can use any time base or angle counter for either match or capture operation. For example, a match on TCR1 can capture the value of TCR2. The channels can request service from the microengine due to recognized pin transitions (input events) or timebase matches.

The eTPU channels also support the basic single-action operations found on TPU3 functionality with the exception that time resolution is 24-bits.

Channel configuration combinations:

- Single input capture, no match (TPU3 functionality).
- Single input capture with single match timeout (TPU3 functionality).
- Single input capture with double match timeout with several double match sub-modes.
- Double input capture with single or double match timeout with several double match sub-modes.
- Single output match (TPU3 functionality).
- Double output match with several double match sub-modes.
- Input-dependent output generation.

The double match functionality has various combinations for generation of service request and determining pin actions. For more details refer to the eTPU Reference Manual.

In addition to the predefined channel configurations above, the user can also program its own channel configuration, defining how input captures, matches and service-requests are related.

21.1.1.1.3 Host Interface

The Host interface allows the Host CPU to control the operation of the eTPU. The Host CPU must initialize the eTPU by writing to the appropriate Host interface registers to assign a Function and priority to each channel. In addition, the Host writes to the Host Service Request and channel configuration registers to further define Function operation for each initialized channel. Refer to Section 21.3.2, “Host Interface” for a detailed description.

When the SCM is implemented by RAM, the Host must first initialize it with the proper microcode program prior to enabling any eTPU Function, and then enable eTPU access (which also disables Host access).

21.1.1.1.4 Shared Data Memory - SDM

The SDM works as data RAM which can be accessed by the Host CPU and up to two eTPU Engines. This memory is used for information transfer between the Host CPU and the eTPU, as data storage for the eTPU microcode program or for communication between the two eTPU Engines. SDM width is 32 bits, and is accessible by the Host as byte, 16-bit or 32-bit wide. eTPU can access it as full 32-bits, lower 24-bits or upper byte (8-bit).

The host can also access the SDM space mirrored in other area with Parameter Signal Extension (PSE). Parameter Signal Extension accesses differ from the usual host accesses to the original SDM area as follows:

- Writes are effective only to the lower 3 bytes of a word: the word’s most significant byte is kept unaltered in SDM.
- Reads return the lower 3 bytes of a word sign-extended to 32 bits, i.e.: the most significant bit of the word s 2nd most significant byte is copied in all 8 bits of the most significant read byte.

Each eTPU channel can be associated with a variable number of parameters located in the SDM, according to its selected Function. In addition, the SDM can be fully shared between two eTPU Engines, enabling direct communication between them.

High flexibility of the SDM utilization is achieved as follows:

- Each channel has a programmable base address pointing to the address of its first parameter with two parameter granularity. This way the SDM can be partitioned according to the actual function needs.
- The microcode can access the first 128 parameters of the selected channel in channel relative access mode.
- Each Engine can access all the SDM address space in indirect addressing mode. Blocks of data are easily transferred using stack operation.
- Absolute addressing mode can access the first 256 parameters (TPU3 functionality), implementing a shared pool of parameters holding global variables.

In the Host address space each parameter occupies four bytes. eTPU usage of the upper byte is achieved by having a 32-bit P register which can access the upper byte, the lower 24 bits or all the 32 bits. The microcode can switch between access sizes at any time.

Each Function may require a different number of parameters. During the eTPU initialization the Host has to program channel base addresses, allocating proper parameters for each channel according to its selected Function.

21.1.1.1.5 Scheduler

Out of reset, all channels are disabled. The Host CPU makes a channel active by assigning it one of three priorities: high, middle, or low. The Scheduler determines the order in which channels are serviced based on channel number and assigned priority. The priority mechanism, implemented in hardware, ensures that all requesting channels are serviced. For additional details refer to Section 21.3.3, “Scheduler.”

21.1.1.1.6 Microengine

eTPU microengine executes each instruction in a microcycle of two system clocks, while prefetching the next instruction through an instruction pipeline. Instruction execution time is constant unless it gets wait states from the SDM arbitration. Two eTPU Engines share code memory without having any performance degradation by interleaving their accesses (the Shared Code Memory has one-clock access time).

Instruction width is 32 bits. The microengine instruction set provides basic arithmetic and logic operations, flow control (jumps and subroutine calls), SDM access, and Channel configuration and control. The instruction formats are defined in such a way that allow particular combinations of two or three of these operations with unconflicting resources to be executed in parallel in the same microcycle.

Microengine has also an independent Multiply/Divide/MAC unit that performs these complex operations in parallel with other microengine instructions.

Channel functionality is tightly integrated to the instruction set through Channel Control operations and conditional Branch operations, which support jumps/calls on Channel-specific conditions. This allows quick and terse Channel configuration and control code, contributing to reduced service time.

21.1.1.1.7 Dual eTPU Engine Module

The eTPU A/B implementation includes two eTPU Engines sharing SDM and the same code in SCM.

The two eTPU Engines share the Bus Interface Unit (BIU) and the data memory (SDM) which enable Host-eTPU and eTPU Engine-Engine communication. The shared BIU includes coherency logic which supports dual parameter (8 bytes) coherency in transfers between Host and eTPU, using a temporary parameter area within the SDM. More details on this can be found on Section 21.3.4, “Parameter Sharing and Coherency.”

21.1.1.1.8 Single eTPU Engine Module

The eTPU C implementation includes a single eTPU engine with a private SDM and SCM that is not accessible by the eTPU A/B module. Its BIU and SDM enable Host-eTPU communication. The shared BIU includes coherency logic which supports dual parameter (8 bytes) coherency in transfers between the Host and eTPU, using a temporary parameter area within the SDM.

21.1.2 Features

21.1.2.1 eTPU Feature Summary

The eTPU includes these distinctive features:

- up to 32 channels for each eTPU Engine, each channel is associated with an Input/Output signal pair.
    - enhanced input digital filters on the input pins for improved noise immunity. The eTPU digital filter can use 2 samples, 3 samples or work in continuous mode.
    - identical, orthogonal channels, except for channel 0: each channel can perform any time function. Each time function can be assigned to more than one channel at a given time, so each signal can have any functionality. Channel 0 has the same capabilities of the others, but can also work with special Angle Counter logic (see below).
    - Link Service Request allows activation of a Channel function by request of another channel, even between eTPU Engines.
    - Host Service Request allows activation of a Channel function by Host CPU request
    - each channel has an event mechanism which supports single and double action functionality in various combinations. It includes two 24-bit capture registers, two 24-bit match registers, 24-bit greater-equal and equal-only comparators.
- Two independent 24-bit time bases for channel synchronization:
    - first time base clocked by system clock with programmable prescaler division from 1 to 512 (in steps of 2), or by output of second time base prescaler.
    - first time base can also be clocked by external signal with programmable prescaler division of 1 to 256.
    - second time base clocked by external signal with programmable prescaler division from 1 to 64.
    - second time base external clock source can be replaced by system clock divided by 8.
    - both time bases can be exported or imported via Shared Time and Counter bus.
    - second time base counter can work as an Angle counter, enabling angle based applications to match angle instead of time.
    - second time base can also be used as a pulse accumulator gated by external signal.
- Event-Triggered VLIW processor (microengine):
    - 2 stage pipeline implementation (fetch and execution), with separate instruction memory - SCM - and data memory - SDM (Harvard architecture)
    - fixed-length instruction execution in two system clock microcycle
    - interleaved SCM access in dual eTPU Engine avoids contention in time for instruction memory
    - SCM address space of up to 16K positions (64 KB)
    - SDM with interleaved access in dual eTPU Engine avoids contention for data memory
    - SDM address space of up to 8 KB (both Engines).

  - instruction set with embedded Channel support, including specialized Channel control subinstructions and conditional branching on Channel-specific flags.
  - channel-oriented addressing: channel-bound address mode with Host configured Channel Base Address allows channel data isolation, independent of microengine application code.
  - channel-bound data address space of up to 128 32-bit parameters (512 bytes)
  - global parameter address mode allows access to common Channel data of up to 256 32-bit parameters (1024 bytes)
  - support for indirect and stacked data access schemes.
  - parallel execution of: data access, ALU, Channel control and flow control subinstructions in selected combinations.
  - 32-bit microengine registers and 24-bit resolution ALU, with 1 microcycle addition and subtraction, absolute value, bitwise logical operations on 24-bit, 16-bit, or byte operands; single bit manipulation, shift operations, sign extension and conditional execution.
  - additional 24-bit Multiply/MAC/Divide unit which supports all signed/unsigned Multiply/MAC combinations, and unsigned 24-bit Divide. The MAC/Divide unit works in parallel with the regular microcode commands.
- Resource sharing features support channel sharing of channel registers, memory and microengine time:
  - hardware Scheduler works as a “task management” unit, dispatching event service routines by predefined, Host-configured priority.
  - automatic Channel context switch when a “task switch” occurs, i.e., one Function Thread ends and another begins to service a request from other Channel: Channel-specific registers, flags and parameter base address are automatically loaded for the next serviced channel.
  - individual channel priority setting in 3 levels: high, middle and low.
  - Scheduler priority scheme allows calculation of worst case latency for event servicing and ensures servicing all channels by preventing permanent blockage.
  - SDM shared between Host CPU and both eTPU Engines, supporting communication either between Channels and Host or inter-channel.
  - hardware implementation of 4 Semaphores supports resource sharing between both eTPU Engines.
  - Hardware semaphores directly supported by the microengine instruction set.
  - dual parameter coherency hardware support allows atomic (to host) access to 2 parameters by microengine(s) in back-to-back accesses.
  - coherent dual-parameter controller allows atomic (to microengines) accesses to 2 parameters by the host.
- Development support features:
  - Nexus class 3 debug support (optional, associated with the eTPU-Nexus Block NDEDI).
  - Software breakpoints.
  - Debug interface supporting single-step execution, forced microinstruction execution, Hardware breakpoints and watchpoints on several conditions.

- Safety support features:
  - SCM (code memory) continuous signature-check built-in self test (MISC - Multiple Input Signature Calculator), runs concurrently with eTPU normal operation.
- (more on Section 21.1.2.3, “eTPU2 Enhancements over eTPU”).

21.1.2.2 eTPU Enhancements over TPU3

- 32 orthogonal channels with enhanced functionality. Full support for double action with double match and double transition sub-mode combinations.
- Input and Output features separated in channel logic and microinstructions, allowing input and output signals to be processed separately or combined.
- Increased time resolution and execution unit to 24 bits.
- Increased linear code memory, shared by two eTPU Engines, configurable up to 16K positions (64 KB).
- Increased SDM address range (8 KB each Engine) and width (32 bits per parameter). The SDM can be dynamically allocated to support variable number of parameters for each channel. Each channel can have access to at least 256 parameters.
- The SDM is fully shared by two eTPU Engines (SDM), supporting direct inter-engine communication with the help of hardware semaphores.
- Enhanced arithmetic operations, including add/subtract with carry, absolute value, multiple shift and rotate, conditional execution with variable operand widths.
- Enhanced logic operations, including bitwise operations (and, or, xor) and bit manipulation, with conditional execution. Support for read-modify-write of any bit in the SDM.
- Hardware for Multiply/MAC/Divide, running in parallel to execution of other operations. The 24-bit divide result is available after 13 other unrelated instructions. Multiplication supports any data width of both operands (8, 16 or 24 bits), signed or unsigned. A 24x24 Multiply/MAC result is available after four other unrelated instructions. A 24x8 Multiply/MAC result is available after one other unrelated instruction.
- Supports export/import of time bases from other sources through the real time bus (STAC - Shared Time and Counter bus). This internal bus is used for sharing real time data between multiple peripherals.
- Contains angle clock hardware, supported by microcode, which can provide a 24-bit angle bus instead of time bus. This feature enables the eTPU to run angle based engine control applications.
- More interrupt types. Each eTPU channel can generate a data transfer request interrupt, in addition to regular interrupts, and one global exception interrupt. Data Transfer requests can be used either as interrupt sources or DMA requests. This feature takes advantage of DMA peripherals which off-load the Host. Interrupt Overflow status is also provided.
- Improved visibility to the Host (pin states, time bases, serviced channel).
- An edge case of priority inversion on TPU3 Scheduler was resolved.
- Supports channel link requests between eTPU Engines.

21.1.2.3 eTPU2 Enhancements over eTPU

- TCR1, channel logic and digital filters (both channel and TCRCLK) now have an option to run at divisions of full system clock speed, besides eTPU clock / 2.
- Channels support unordered transitions: transition B can now be detected before transition A. Related to this enhancement, TDLA and TDLB can now be independently negated by microcode. Refer to the eTPU Reference Manual for details.
- Channel logic can now work on full eTPU clock, allowing faster response with slow clocks.
- Added a new User Programmable Channel Mode: the blocking, enabling, service request and capture characteristics of this channel mode can be programmed via microcode. Refer to the eTPU Reference Manual for details.
- Microinstructions now provide an option to issue Interrupt and Data Transfer requests selected by CHAN. They can also be requested simultaneously at the same instruction. Refer to the eTPU Reference Manual for details.
- Channel Flags 0 and 1 can now be tested for branching, besides selecting the entry point. Refer to the eTPU Reference Manual for details.
- Channel digital filters can be bypassed, and can be clocked by full eTPU clock (Section 21.3.4.4.4, “Bypass Mode”).
- Scheduler priority-passing mechanism can now be disabled (Section 21.3.3.2.2, “Priority Passing Disabling”).
- New Watchdog mechanism kills threads over a programmable timeout (Section 21.3.1, “Watchdog”).
- New counter allows microengine load information collection for performance analysis (Section 21.3.7.1, “Idle Counter”).
- Channels 1 and 2 (besides channel 0) can now be selected to control the EAC (Section 21.2.6.1, “ETPUTBCR - eTPU Time Base Configuration Register”).
- Timebase prescalers are now reset when the GTBE input is negated, guaranteeing synchronization with eMIOS in all cases (Section 21.3.5.4, “GTBE - Global Time Base Enable”).
- New MISC flag indicates when an SCM signature calculation round is completed. This allows measuring of the average MISC scan period in a real application situation (Section 21.2.5.1, “ETPUMCR - eTPU Module Configuration Register”).
- New channel TCCEA flag allows continuous capture even after TDLA is set, making it fully compatible with TPU behavior. Refer to the eTPU Reference Manual for details.
- New branch condition PRSS tells the pin state at the time when a channel (match or transition) service request occurred. Refer to the eTPU Reference Manual for details.
- MRLEA/B can now be negated independently by microcode. Refer to the eTPU Reference Manual for details.
- New Engine Relative address mode allows a function to access SDM address space common to one engine, but distinct between engines. Refer to the eTPU Reference Manual for details.
- Memory error support for Code (SCM) and Data (SDM) memories: error detection, correction, and hardware-assisted soft-error fixing (Section 21.3.6.2, “Memory Error Support”).

NOTE

All changes above are backward compatible with the classic eTPU, so that legacy object code (both Host and microcode) runs on eTPU2 without modification.

21.1.3 Modes of Operation

eTPU can be seen as capable of working in the following modes:

- **User Configuration Mode**

  User has the ability to program the eTPU Cores with User Time Functions, having access to the Shared Code Memory (SCM).
- **User Mode**

  User does not access the eTPU Shared Code Memory:
  - Use of predefined eTPU Functions
  - No need for eTPU Core programming ability
- **Debug Mode**

  User debugs eTPU code, accessing special Trace/Debug features via Nexus interface:
  - hardware breakpoint/watchpoint setting
  - access to internal registers
  - single-step execution
  - forced instruction execution
  - software breakpoint insertion and removal.
- **Module Disable Mode**

  eTPU Engine clocks are stopped through a register write to ETPUECR bit MDIS, saving power. Input sampling stops. eTPU Engines can be in Module Disable Mode independently. Module Disable Mode stops only the Engine clock, so that the Shared BIU, and Global Channel registers can be accessed, and interrupts and DMAs can be cleared and enabled/disabled. An Engine only enters Module Disable Mode when any currently running thread is finished. Refer to the eTPU Reference Manual for details.
- **Stop Mode**

  Stop Mode is entered when eTPU answers device stop request assertion with stop acknowledge. Refer to the SIU_HLT and SIU_HLTACK register descriptions for details on how to place this module in Stop mode.

These modes are loosely selected: there is no unique register field or signals to choose between them. Some features of one mode can be used with features of other mode(s). More on this subject can be found on Section 21.1.3.1, “eTPU Mode Selection” below.

NOTE

Throughout this document, an engine is said to be “stopped” if it is either in Module Disable mode or Stop mode.

21.1.3.1 eTPU Mode Selection

User and User Configuration are the normal operating modes, and differ from each other only in access to SCM. User programmability is only possible with a RAM SCM.

Debug Mode is characterized by the use of the debug interface features. Specifically, Nexus blocks provide Nexus class 3 debug features. For more information on debug features, refer to the eTPU Reference Manual.

Module Disable Mode is entered by setting ETPUECR register bit MDIS. eTPU Engines can be individually stopped going into Module Disable Mode (there is one ETPUECR register for each Engine). Each engine can leave Module Disable Mode by writing MDIS=0 (which can only be done if VIS=0).

Stop Mode is activated by IP-Bus (device stop request). In this case, the eTPU waits for both eTPU Engines to enter in stop mode, and then asserts the stop acknowledge line. eTPU leaves Stop Mode when device stop request is negated, but only if VIS=0. If device stop request is negated and VIS=1, eTPU will leave Stop Mode as soon as VIS=0.

NOTE

An Engine can stay in Module Disable mode when it leaves Stop Mode if its bit MDIS=1, even if the other leaves it.

21.2 External Signal Description

21.2.1 Overview

There are 69 external signals associated with each eTPU Engine: 32 channel input signals, 32 channel output signals, 4 output disable inputs, and TCRCLK clock input, totalling 138 in a Dual Engine system.

The TCRCLK signal is used to clock TCR1/2 counters or gate the TCR2 clock. In Angle Mode it is used as tooth signal input.

21.2.2 Detailed Signal Descriptions

21.2.2.1 eTPU Channel Output Signals [0-31]

Each channel output signal is associated with a channel. The microcode may affect the logic level of an output signal[1] by implementing one of two actions:

- Specify the logic level output to the signal when there is a match or a transition.
- Immediately force a logic level.

The output signal may also be forced to a logic level, independently of the output value from the channel logic, by one of the four (each Engine) output disable signals (see Section 21.2.2.4, “eTPU Channel Output Disable Signals”).

---

[1] Note that the minimum pulse width is one microcycle (two eTPU clocks), and slow 5V pads may not be able to transfer it on time. For generation of very short pulses the eTPU pads have to be programmed by the system integration for fast operation mode with the voltage levels defined for fast pad operation in the MCU technology.

21.2.2.2 eTPU Channel Input Signals [0-31]

Each channel input signal is associated with a channel. The microcode can directly control the effect of the transition edge. Each channel can be programmed to sense a transition when a rising and/or falling edge is detected. The channel logic can also process two transition events, and relate these events to each other and to other programmed timer events. The edge sensitivities of the two transition events are configured independently by microcode.

Each channel input signal has an associated synchronizer made of two flip-flops sampling the signal on every other eTPU clock[1], followed by a digital filter. This digital filter can work in three sub-modes, whose purpose is to filter out noise pulses that have width less then a programmed value of eTPU clocks, preventing these transitions from being input to the transition detect logic. The synchronizer and digital filter are guaranteed to pass pulses that are greater than a programmed value. All channel input filters in one Engine work on the same mode and sampling clock. For more information on channel input filters, refer to Section 21.3.4.4, “Enhanced Digital Filter - EDF”. In one of the Angle Modes, the output of the digital filter of channel 0 is replaced by the output of TCRCLK signal digital filter. Refer to the eTPU Reference Manual for details.

21.2.2.3 Time Base Clock Signal — TCRCLK

TCRCLK is an input signal used to control the Time Bases TCR1 and TCR2. There is one independent TCRCLK input for each Engine. For pulse accumulator operations TCRCLK can be used as a gate for a counter based on the eTPU clock divided by eight. For Angle operations TCRCLK can be used to get the tooth transition indications in Angle Mode.

Like the channel input signals, the TCRCLK signal has an associated synchronizer followed by a digital filter. This digital filter can work in two sub-modes, whose purpose is to filter out noise pulses that have width less then a programmed value of eTPU clocks, preventing these transitions from being input to the transition detect logic. The synchronizer and digital filter are guaranteed to pass pulses that are greater than a programmed value. The clock and operation sub-mode of the TCRCLK filter is configured independently of the other channel input filters, through the field TCRCF in register ETPUTBCR. For more information on filter sub-modes, refer to Section 21.3.5.5, “TCRCLK Digital Filter”. In one of the Angle Modes, the output of the digital filter of channel 0 is replaced with the output of TCRCLK signal digital filter.

21.2.2.4 eTPU Channel Output Disable Signals

Each eTPU engine has 4 input disable signals to force the outputs of a group of 8 channels to an inactive level. When an input disable signal is active, all the channels in its group of 8 that have their ODIS bits set to 1 in ETPUCxCR register have their outputs forced to the opposite of the value specified in bit OPOL of the same register. Therefore, channels can be individually selected to be affected by the input disable signals, as well as their disabling forced polarity.

The source of the disable signals and their mapping to eTPU channel groups is given in Table 21-2.

---

[1] Derived from system clock described in Chapter 7, “Clocking”.

Table 21-2. Output Disable Channel Groups

Disable Signal Source	eTPU	
eMIOS Channel Flag	Engine	Channels
11	A	0 to 7
10		8 to 15
9		16 to 23
8		24 to 31
20	B	0 to 7
21		8 to 15
22		16 to 23
23		24 to 31
11	C	0 to 7
10		8 to 15
9		16 to 23
8		24 to 31

21.2.3 Memory Map/Register Definition

The guideline for the description of all bits and fields throughout this section is to provide only a brief explanation (without examples or method of use) of the features, since it will be used mainly as a reference for the reader that is studying Section 21.3, “Functional Description, where those features are explained in detail.

21.2.4 Memory Map

The eTPU System simplified memory map is shown in Table 21-4. Each of the register areas shown may have their own reserved address areas.

Table 21-5 and Table 21-6 show detailed memory maps. Offsets are relative to the eTPU Base addresses given in Table 21-3.

Table 21-3. eTPU Module Base Addresses

eTPU Module	Base address
A/B	0xC3FC_0000
C	0xC3E2_0000

SCM unused area is decoded and returns a fixed opcode defined in the register ETPUSCMOFFDATAR.

Table 21-4. High Level Memory Map

Offset	Use
0x00-0x1F	System Configuration Registers
0x20-0x2F	eTPU A and eTPU C Time Base Registers
0x30-0x3F	RESERVED[1]
0x40-0x4F	eTPU B Time Base Registers

Table 21-4. High Level Memory Map

0x50-0x5F	RESERVED[1]
0x60-0x6F	eTPU A and eTPU C Extra Engine Registers
0x70-0x7F	eTPU B Extra Engine Registers
0x80-0xFF	RESERVED[1]
0x100-0x13F	Memory Error Support Registers
0x140-0x1FF	RESERVED[1]
0x200-0x2FF	eTPU A/B and eTPU C Global Channel Registers
0x300-0x3FF	RESERVED[1]
0x400-0x7FF	eTPU A and eTPU C Channel Registers
0x800-0xBFF	eTPU B Channel Registers
0xC00-0xFFF	RESERVED[1]
0x1000-0x3FFF	RESERVED[1]
0x4000-0x7FFF	RESERVED[1]
0x8000 - 0xBFFF[2]	eTPU A/B and eTPU C SDM - Shared Data Memory
0xC000 - 0xFFFF[2]	eTPU A/B and eTPU C SDM PSE mirror [3]
0x10000 - 0x1FFFF[2]	eTPU A/B and eTPU C SCM - Shared Code Memory[4]

[1] Reserved addresses must not be used. Access to these memory positions complete with 0-wait-states, but the behavior is unspecified.

[2] Actual sizes of SCM and SDM are MCU-dependent, see the memory map of the SoC for details.

[3] Parameter Sign Extension access area, see Section 21.3.2.3, “Parameter Access”.

[4] SCM access is available only when bit VIS=1 on register ETPUMCR, under certain conditions (see Section 21.2.5.1, “ETPUMCR - eTPU Module Configuration Register”).

Table 21-5. Detailed Memory Map eTPU A/B

Offset	Use
0x00	ETPUMCR - eTPU Module Configuration Register
0x04	ETPUCDCR - eTPU Coherent Dual-Parameter Controller Register
0x08	RESERVED
0x0C	ETPUMISCCMPR - eTPU MISC Compare Register
0x10	ETPUSCMOFFDATAR - eTPU SCM Off-range Data Register
0x14	ETPUECR_A - eTPU A Engine Configuration Register
0x18	ETPUECR_B - eTPU B Engine Configuration Register
0x1C	RESERVED
0x20	ETPUTBCR_A - eTPU A Time Base Configuration Register
0x24	ETPUTB1R_A - eTPU A Time Base 1
0x28	ETPUTB2R_A - eTPU A Time Base 2
0x2C	ETPUREDCR_A - eTPU A STAC Configuration Register

Table 21-5. Detailed Memory Map eTPU A/B

Offset	Use
0x30	RESERVED
0x34	RESERVED
0x38	RESERVED
0x3C	RESERVED
0x40	ETPUTBCR_B - eTPU B Time Base Configuration Register
0x44	ETPUTB1R_B - eTPU B Time Base 1
0x48	ETPUTB2R_B - eTPU B Time Base 2
0x4C	ETPUREDCR_B - eTPU B STAC Configuration Register
0x50	RESERVED
0x54	RESERVED
0x58	RESERVED
0x5C	RESERVED
0x60	ETPUWDTR_A - eTPU A Watchdog Timer Register
0x64	RESERVED
0x68	ETPUIDLER_A - eTPU A Idle Counter Register
0x6C	RESERVED
0x70	ETPUWDTR_B - eTPU B Watchdog Timer Register
0x74	RESERVED
0x78	ETPUIDLER_B - eTPU B Idle Counter Register
0x7C	RESERVED
0x80 - 0xFF	RESERVED
0x100	ETPUMECR - eTPU Memory Error Control Register
0x104	ETPUDEIAR - eTPU Data Error Injection Address Register
0x108	ETPUDEIDPR - eTPU Data Error Injection Data Pattern Register
0x10C	ETPUDEIPPR - eTPU Data Error Injection Parity Pattern Register
0x110	ETPUDERAR - eTPU Data Error Report Address Register
0x114	ETPUDERDR - eTPU Data Error Report Data Register
0x118	ETPUDERSR - eTPU Data Error Report Syndrome Register
0x11C	RESERVED
0x120	ETPUMESR - eTPU Memory Error Status Register
0x124	ETPUCEIAR - eTPU Code Error Injection Address Register
0x128	ETPUCEIDPR - eTPU Code Error Injection Data Pattern Register
0x12C	ETPUCEIPPR - eTPU Code Error Injection Parity Pattern Register
0x130	ETPUCERAR - eTPU Code Error Report Address Register
0x134	ETPUCERDR - eTPU Code Error Report Data Register
0x138	ETPUCERSR - eTPU Code Error Report Syndrome Register
0x13C	ETPUCEFR - eTPU Code Error Fix Register
0x140 - 0x1FF	RESERVED
0x200	ETPUCISR_A - eTPU A Channel Interrupt Status Register
0x204	ETPUCISR_B - eTPU B Channel Interrupt Status Register
0x208	RESERVED
0x20C	RESERVED

Table 21-5. Detailed Memory Map eTPU A/B

Offset	Use
0x210	ETPUCDTRSR_A - eTPU A Channel Data Transfer Request Status Register
0x214	ETPUCDTRSR_B - eTPU B Channel Data Transfer Request Status Register
0x218	RESERVED
0x21C	RESERVED
0x220	ETPUCIOSR_A - eTPU A Channel Interrupt Overflow Status Register
0x224	ETPUCIOSR_B - eTPU B Channel Interrupt Overflow Status Register
0x228	RESERVED
0x22C	RESERVED
0x230	ETPUCDTROSR_A - eTPU A Channel Data Transfer Request Overflow Status Register
0x234	ETPUCDTROSR_B - eTPU B Channel Data Transfer Request Overflow Status Register
0x238	RESERVED
0x23C	RESERVED
0x240	ETPUCIER_A - eTPU A Channel Interrupt Enable Register
0x244	ETPUCIER_B - eTPU B Channel Interrupt Enable Register
0x248	RESERVED
0x24C	RESERVED
0x250	ETPUCDTRER_A - eTPU A Channel Data Transfer Request Enable Register
0x254	ETPUCDTRER_B - eTPU B Channel Data Transfer Request Enable Register
0x258–0x25C	RESERVED
0x260	ETPUWDSR_A - eTPU A Watchdog Status Register
0x264	ETPUWDSR_B - eTPU B Watchdog Status Register
0x268–0x27F	RESERVED
0x280	ETPUCPSSR_A - eTPU A Channel Pending Service Status Register
0x284	ETPUCPSSR_B - eTPU B Channel Pending Service Status Register
0x288	RESERVED
0x28C	RESERVED
0x290	ETPUCSSR_A - eTPU A Channel Service Status Register
0x294	ETPUCSSR_B - eTPU B Channel Service Status Register
0x298	RESERVED
0x29C	RESERVED
0x300 - 0x3FF	RESERVED
0x400	ETPUC0CR_A - eTPU A Channel 0 Configuration Register
0x404	ETPUC0SCR_A - eTPU A Channel 0 Status and Control Register
0x408	ETPUC0HSRR_A - eTPU A Channel 0 Host Service Request Register
0x40C	RESERVED
0x410	ETPUC1CR_A - eTPU A Channel 1 Configuration Register
0x414	ETPUC1SCR_A - eTPU A Channel 1 Status and Control Register
0x418	ETPUC1HSRR_A - eTPU A Channel 1 Host Service Request Register
0x41C	RESERVED
	. .

Table 21-5. Detailed Memory Map eTPU A/B

Offset	Use
0x5F0	ETPUC31CR_A - eTPU A Channel 31 Configuration Register
0x5F4	ETPUC31SCR_A - eTPU A Channel 31 Status and Control Register
0x5F8	ETPUC31HSRR_A - eTPU A Channel 31 Host Service Request Register
0x5FC - 0x7FF	RESERVED
0x800	ETPUC0CR_B - eTPU B Channel 0 Configuration Register
0x804	ETPUC0SCR_B - eTPU B Channel 0 Status and Control Register
0x808	ETPUC0HSRR_B - eTPU B Channel 0 Host Service Request Register
0x80C	RESERVED
0x810	ETPUC1CR_B - eTPU B Channel 1 Configuration Register
0x814	ETPUC1SCR_B - eTPU B Channel 1 Status and Control Register
0x818	ETPUC1HSRR_B - eTPU B Channel 1 Host Service Request Register
0x81C	RESERVED
	. . .
0x9F0	ETPUC31CR_B - eTPU B Channel 31 Configuration Register
0x9F4	ETPUC31SCR_B - eTPU B Channel 31 Status and Control Register
0x9F8	ETPUC31HSRR_B - eTPU B Channel 31 Host Service Request Register
0x9FC - 0x7FFF	RESERVED
0x8000 - 0xBFFF[1]	Shared Parameter RAM - SPRAM
0xC000 - 0xFFFF[1]	Shared Parameter RAM - SPRAM - PSE mirror [2]
0x10000–1FFFF[3]	Shared Code Memory - SCM[4]

[1] The actual SPRAM size is MCU-dependent. Please refer to the SoC Memory Map for details.

[2] Parameter Sign Extension access area, see Section 21.3.2.3, “Parameter Access”.

[3] The actual SCM size is MCU-dependent. When the size not the maximum, the unused SCM address range returns the value of the register ETPUSCMOFFDATAR.

[4] SCM access is available only when bit VIS=1 on register ETPUMCR, under certain conditions (see Section 21.2.5.1, “ETPUMCR - eTPU Module Configuration Register”). SCM can only be written in 32 bit accesses.

Table 21-6. Detailed Memory Map eTPU C

Offset	Use
0x00	ETPUMCR - eTPU Module Configuration Register
0x04	ETPUCDCR - eTPU Coherent Dual-Parameter Controller Register
0x08	RESERVED
0x0C	ETPUMISCCMPR - eTPU MISC Compare Register
0x10	ETPUSCMOFFDATAR - eTPU SCM Off-range Data Register
0x14	ETPUECR_C - eTPU C Engine Configuration Register
0x18	RESERVED
0x1C	RESERVED

Table 21-6. Detailed Memory Map eTPU C

Offset	Use
0x20	ETPUTBCR_C - eTPU C Time Base Configuration Register
0x24	ETPUTB1R_C - eTPU C Time Base 1
0x28	ETPUTB2R_C - eTPU C Time Base 2
0x2C	ETPUREDCR_C - eTPU C STAC Configuration Register
0x30	RESERVED
0x34	RESERVED
0x38	RESERVED
0x3C	RESERVED
0x40	RESERVED
0x44	RESERVED
0x48	RESERVED
0x4C	RESERVED
0x50	RESERVED
0x54	RESERVED
0x58	RESERVED
0x5C	RESERVED
0x60	ETPUWDTR_C - eTPU C Watchdog Timer Register
0x64	RESERVED
0x68	ETPUIDLER_C - eTPU C Idle Counter Register
0x6C	RESERVED
0x70	RESERVED
0x74	RESERVED
0x78	RESERVED
0x7C	RESERVED
0x80 - 0xFF	RESERVED
0x100	ETPUMECR - eTPU Memory Error Control Register
0x104	ETPUDEIAR - eTPU Data Error Injection Address Register
0x108	ETPUDEIDPR - eTPU Data Error Injection Data Pattern Register
0x10C	ETPUDEIPPR - eTPU Data Error Injection Parity Pattern Register
0x110	ETPUDERAR - eTPU Data Error Report Address Register
0x114	ETPUDERDR - eTPU Data Error Report Data Register
0x118	ETPUDERSR - eTPU Data Error Report Syndrome Register
0x11C	RESERVED
0x120	ETPUMESR - eTPU Memory Error Status Register
0x124	ETPUCEIAR - eTPU Code Error Injection Address Register
0x128	ETPUCEIDPR - eTPU Code Error Injection Data Pattern Register
0x12C	ETPUCEIPPR - eTPU Code Error Injection Parity Pattern Register
0x130	ETPUCERAR - eTPU Code Error Report Address Register
0x134	ETPUCERDR - eTPU Code Error Report Data Register
0x138	ETPUCERSR - eTPU Code Error Report Syndrome Register
0x13C	ETPUCEFR - eTPU Code Error Fix Register
0x140 - 0x1FF	RESERVED

Table 21-6. Detailed Memory Map eTPU C

Offset	Use
0x200	ETPUCISR_C - eTPU C Channel Interrupt Status Register
0x204	RESERVED
0x208	RESERVED
0x20C	RESERVED
0x210	ETPUCDTRSR_C - eTPU C Channel Data Transfer Request Status Register
0x214	RESERVED
0x218	RESERVED
0x21C	RESERVED
0x220	ETPUCIOSR_C - eTPU C Channel Interrupt Overflow Status Register
0x224	RESERVED
0x228	RESERVED
0x22C	RESERVED
0x230	ETPUCDTROSR_C - eTPU C Channel Data Transfer Request Overflow Status Register
0x234	RESERVED
0x238	RESERVED
0x23C	RESERVED
0x240	ETPUCIER_C - eTPU C Channel Interrupt Enable Register
0x244	RESERVED
0x248	RESERVED
0x24C	RESERVED
0x250	ETPUCDTRER_C - eTPU C Channel Data Transfer Request Enable Register
0x254-0x25C	RESERVED
0x260	ETPUWDSR_C - eTPU C Watchdog Status Register
0x264-0x27F	RESERVED
0x280	ETPUCPSSR_C - eTPU C Channel Pending Service Status Register
0x284	RESERVED
0x288	RESERVED
0x28C	RESERVED
0x290	ETPUCSSR_C - eTPU C Channel Service Status Register
0x294	RESERVED
0x298	RESERVED
0x29C	RESERVED
0x300 - 0x3FF	RESERVED
0x400	ETPUC0CR_C - eTPU C Channel 0 Configuration Register
0x404	ETPUC0SCR_C - eTPU C Channel 0 Status and Control Register
0x408	ETPUC0HSRR_C - eTPU C Channel 0 Host Service Request Register
0x40C	RESERVED
0x410	ETPUC1CR_C - eTPU C Channel 1 Configuration Register
0x414	ETPUC1SCR_C - eTPU C Channel 1 Status and Control Register
0x418	ETPUC1HSRR_C - eTPU C Channel 1 Host Service Request Register
0x41C	RESERVED

Table 21-6. Detailed Memory Map eTPU C

Offset	Use
	. .
0x5F0	ETPUC31CR_C - eTPU C Channel 31 Configuration Register
0x5F4	ETPUC31SCR_C - eTPU C Channel 31 Status and Control Register
0x5F8	ETPUC31HSRR_C - eTPU C Channel 31 Host Service Request Register
0x5FC - 0x7FF	RESERVED
0x800	RESERVED
0x804	RESERVED
0x808	RESERVED
0x80C	RESERVED
0x810	RESERVED
0x814	RESERVED
0x818	RESERVED
0x81C	RESERVED
	. . .
0x9F0	RESERVED
0x9F4	RESERVED
0x9F8	RESERVED
0x9FC - 0x7FFF	RESERVED
0x8000 - 0xBFFF[1]	Shared Parameter RAM - SPRAM
0xC000 - 0xFFFF[1]	Shared Parameter RAM - SPRAM - PSE mirror [2]
0x10000–1FFFF[3]	Shared Code Memory - SCM[4]

[1] The actual SPRAM size is MCU-dependent. Please refer to the SoC Memory Map for details.

[2] Parameter Sign Extension access area, see Section 21.3.2.3, “Parameter Access”.

[3] The actual SCM size is MCU-dependent. When the size not the maximum, the unused SCM address range returns the value of the register ETPUSCMOFFDATAR.

[4] SCM access is available only when bit VIS=1 on register ETPUMCR, under certain conditions (see Section 21.2.5.1, “ETPUMCR - eTPU Module Configuration Register”). SCM can only be written in 32 bit accesses.

21.2.5 System Configuration Registers

21.2.5.1 ETPUMCR - eTPU Module Configuration Register

In the eTPUA/B module, this register is shared between eTPU A and eTPU B engines. In the eTPU C module, it is exclusive to eTPU C engine. ETPUMCR gathers configuration and status in the eTPU system, including a Global Exception. It is also used for configuring the SCM (Shared Code Memory) operation and test.

Base + 0x000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	SDM ERR	WDTO 1	WDTO 2[1]	MGE 1	MGE 2[1]	ILF1	ILF2[1]	0	0	0	SCMSIZE				
W	GEC	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	SCMSIZE				

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	SCM MISC	SCM MISF	SCM MISE N	SCM ERR	0	VIS	0	0	0	0	0	GTBE
W	▨	▨	▨	▨	SCM MISC C	▨		▨	▨		▨	▨	▨	▨	▨	
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

[1] Not present in eTPU C.

Figure 21-3. ETPUMCR Register

GEC— Global Exception Clear

This write-only bit negates Global Exception request and clears Global Exception status bits MGE1, MGE2, ILF1, ILF2 and SCMMISF.

1 = Negate Global Exception, clear status bits ILF1, ILF2, MGE1, MGE2, and SCMMISF.
0 = Keep Global Exception request and status bits ILF1, ILF2, MGE1, MGE2, and SCMMISF as is.

GEC works the same way with either one or both Engines in Module Disable Mode.

SDMERR — SDM Read Error

This flag indicates that an SDM non-correctable read error occurred on a microengine or host (including CDC) read, generating a Global Exception. This bit is cleared by writing 1 to GEC.

1 = Global Exception requested by SDM read error is pending.
0 = No Global Exception pending because of SDM read error.

WDTO1,2 — Watchdog Timeout

Flags WDTO1 and WDTO2 indicate that a Watchdog Timeout occurred in eTPU engine A or C and eTPU engine B respectively, generating a Global Exception. These bits are cleared by writing 1 to GEC.

1 = Global Exception requested by Watchdog timeout
0 = No Global Exception pending because of Watchdog timeout.

MGE1,2— Microcode Global Exception - Engine A/C, B

These bits indicate that a Global Exception was asserted by microcode executed on the respective Engine. The determination of the reason why the Global Exception was asserted is application dependent: it can be coded in an SDM status parameter, for instance. This bit is cleared by writing 1 to GEC.

1 = Global Exception requested by microcode is pending
0 = No microcode-requested Global Exception pending.

ILF1,2— Illegal Instruction Flag - eTPU A/C, B

The ILF1/2 bit is set by the microengine to indicate that an illegal instruction was decoded in Engine 1/2. This bit is cleared by host writing 1 to GEC. See the eTPU Reference Manual for more details.

1 = Illegal Instruction detected by eTPU A/C, B.
0 = Illegal Instruction not detected.

SCMSIZE[0:4] - SCM Size

This read-only field holds the number of 2 KB SCM Blocks minus 1. This value is 11 for eTPU A/B and 5 for eTPU C, meaning SCM sizes of 24 KB and 12 KB, respectively.

SCMMISC, SCMMISCC — SCM MISC Complete, SCM MISC Complete Clear

Flag SCMMISC indicates that MISC has completed the evaluation of the SCM signature since reset or the since the last time it was cleared. SCMMISC is cleared by writing 1 to SCMMISCC (at same bit position), and is not cleared when MISC is disabled (SCMMISEN=0). SCMMISC asserts at the end of the SCM memory scan, either if the signature matches or not.

1 = MISC completed at least one SCM signature calculation and compare since the last time SCMMISC was cleared.
0 = MISC has not yet completed an SCM signature calculation and compare since the last time SCMMISC was cleared.

SCMMISF— SCM MISC Flag

The SCMMISF bit is set by the SCM MISC (Multiple Input Signature Calculator) logic to indicate that the calculated signature does not match the expected value, at the end of a MISC iteration. See Section 21.3.6.1, “SCM Test - Multiple Input Signature Calculator” for more details.

1 = MISC has read entire SCM array and the expected signature in ETPUMISCCMPR does not match the value calculated.
0 = Signature mismatch not detected.

This bit is cleared when Global Exception is cleared by writing 1 to GEC.

SCMMISEN — SCM MISC Enable

The SCMMISEN bit is used for enabling/disabling the operation of the MISC logic. SCMMISEN is readable and writable at any time. The MISC logic will only operate when this bit is set to 1. When the bit is reset the MISC address counter is set to the initial SCM address. When enabled, the MISC will continuously cycle through the SCM addresses, reading each and calculating a CRC. In order to save power, the MISC can be disabled by clearing the SCMMISEN bit. See Section 21.3.6.1, “SCM Test - Multiple Input Signature Calculator” for more details.

1 = MISC operation enabled.
0 = MISC operation disabled. The MISC logic is reset to its initial state.

SCMMISEN resets automatically when MISC logic detects an error, i.e., when SCMMISF transitions from 0 to 1, disabling the MISC operation.

SCMERR — SCM Read Error

This flag indicates that an SCM non-correctable read error occurred on a host or microengine (including MISC) read, generating a Global Exception. This bit is cleared by writing 1 to GEC.

1 = Global Exception requested by SCM read error is pending.
0 = No Global Exception pending because of SCM read error.

VIS — SCM Visibility Bit

VIS bit turns SCM visible to the IP-Bus and resets MISC state (but SCMMISEN keeps its value).
1 = SCM is visible to the slave bus. MISC state is reset.
0 = SCM is not visible to the slave bus. Accessing SCM address space issues a bus error, writes are protected and reads are meaningless.

This bit is write protected when any of the Engines is not halted or stopped[1]. When VIS=1, the ETPUECR MDIS bits are write protected. The value written to SCM is unspecified if other transfer sizes are used.

GTBE - Global Time Base Enable

GTBE enables time bases in both Engines, allowing them to be started synchronously.
1 = time bases in both Engines are enabled to run.
0 = time bases in both Engines are disabled to run.

NOTE

Global Time Base Enable action may also depend on other blocks, as explained in Section 21.3.5.4, "GTBE - Global Time Base Enable".

NOTE

When GTBE is turned off with Angle Mode enabled, the EAC must be reinitialized before GTBE is turned on again. The EAC reinitialization procedure is described in the eTPU Reference Manual.

---

[1] Engine is stopped in Module Disable or Stop Modes.

21.2.5.2 ETPUCDCR - eTPU Coherent Dual-Parameter Controller Register

ETPUCDCR configures and controls dual-parameter coherent transfers. For more info, see Section 21.3.4.2, “Coherent Dual-parameter Controller - CDC.”

Base + 0x004

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	STS	CTBASE					PBBASE									
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	PWIDTH	PARM0							WR	PARM1						
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-4. ETPUCDCR Register

STS — Start Bit

This bit is set by the host in order to start the data transfer between the parameter buffer pointed by PBBASE and the target addresses selected by the concatenation of fields CTBASE and PARM0/1. The host receives wait-states until the data transfer is complete, when this bit is reset by the coherency logic (see Section 21.3.4.2, “Coherent Dual-parameter Controller - CDC). Therefore, host always reads STS as 0.

1 = (write) starts a coherent transfer.
0 = (write) does not start a coherent transfer.

CTBASE[0:4] — Channel Transfer Base

This field concatenates with fields PARM0/PARM1 to determine the absolute word offset (from the SDM base) of the parameters to be transferred:

Parameter 0 word address = {CTBASE, PARM0} + SDM base word address

Parameter 1 word address = {CTBASE, PARM1} + SDM base word address

PBBASE[0:9] — Parameter Buffer Base Address

This field points to the base address of the parameter buffer location, with granularity of 2 parameters (8 bytes). The host (byte) address of the first parameter in the buffer is PBBASE*8 + SDM Base Byte Address. The microengine absolute (word) address of the first parameter in the buffer is PBBASE*2.

PWIDTH — Parameter Width Selection

This bit selects the width of the parameters to be transferred between the PB and the target address.

1 = Transfer 32-bit parameters. All 32 bits of the parameters are written in the destination address.
0 = Transfer 24-bit parameters. The upper byte remains unchanged in the destination address.

WR — Read/Write selection

This bit selects the direction of the coherent data transfer.

1 = Write operation. Data transfer is from the PB to the selected SDM address.
0 = Read operation. Data transfer is from the selected SDM address to the PB.

PARM0[0:6], PARM1[0:6] — Channel Parameter number 0/number 1

These fields in concatenation with CTBASE[0:4] determine the word address offset (from the SDM base) of the parameters that are destination or source (defined by WR) of the coherent transfer. The word SDM address offset of the parameters are {CTBASE, PARM0/1}.Note that PARM0 and PARM1 allow non-contiguous parameters to be transferred coherently. The parameter pointed by {CTBASE, PARM0} is the first transferred.

21.2.5.3 ETPUMISCCMPR - eTPU MISC Compare Register

ETPUMISCCMPR holds the 32-bit signature expected from the whole SCM array. This register must be written by the host with the 32-bit word to be compared against the calculated signature at the end of the MISC cycle. This register is global to both eTPU Engines. For more detail see Section 21.3.6.1, “SCM Test - Multiple Input Signature Calculator.

Base + 0x00C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ETPUMISCCMP[0:15]															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ETPUMISCCMP[16:31]															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-5. ETPUMISCCMPR Register

ETPUMISCCMP[0:31] — Expected Multiple Input Signature Register value

See Section 21.3.6.1, “SCM Test - Multiple Input Signature Calculator.

21.2.5.4 ETPUSCMOFFDATAR - eTPU SCM Off-range Data Register

ETPUSCMOFFDATAR holds the 32-bit value returned when SCM array is accessed at non implemented addresses, either by the Host or by the microengine. The non implemented address for eTPU AB is 0xc3fd6000 to 0xc3fd7FFC and for eTPU C is 0xc3e33000 to 0xc3e33ffc. This register can be written by the host with the 32-bit instruction to be executed by the microengine to recover from runaway code. This register is global to both eTPU Engines. The reset value is 0xf3775ffb. For more detail see Section 21.3.2.6.3, "SCM Off-range Data.

Base + 0x010

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ETPUSCMOFFDATA[0:15]															
W																
RESET:									[1]							

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ETPUSCMOFFDATA[16:31]															
W																
RESET:																

[1] The reset value is 0xf3775ffb, an instruction that clears MRLEs, MRLs and TDLs, disables channel service requests, ends the thread and generates an illegal instruction Global Exception when accessed by eTPU.

Figure 21-6. ETPUSCMOFFDATAR Register

ETPUSCMOFFDATA[0:31] — SCM Off-range read data value

See Section 21.3.2.6.3, "SCM Off-range Data.

21.2.5.5 ETPUECR - eTPU Engine Configuration Register

Each Engine has its own ETPUECR register. ETPUECR holds configuration and status fields that are programmed independently in each Engine.

eTPU A/C: Base + 0x014 / eTPU B: Base + 0x018

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FEND	MDIS	0	STF	0	0	0	0	HLTF	0	0	0	FCSS	FPSCK		
W			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨				
RESET:	0	0	0	0	0	0	0	0	0(1[1])	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CDFC		0	ERBA					SPPD IS	0	0	ETB				
W			▨							▨	▨					
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

[1] Engine may go to Debug state (halted) soon after reset, depending on the NDEDI configuration (see NDEDI Block Guide).

Figure 21-7. ETPUECR Register

FEND — Force End

FEND assertion terminates any current running thread as if an END instruction have been executed (see eTPU Reference Manual for details).

1 = Ends any ongoing thread.
0 = Normal operation.

This bit is self-negating when the thread ends[1]. Writing FEND=1 is ignored and FEND stays 0 when the microengine is in TST, halted, stopped, or idle (no thread executing).

MDIS — Module Disable Bit

When MDIS is set, the Engine shuts down its internal clocks, going into Module Disable Mode. TCR1 and TCR2 cease to increment, and input sampling stops. The Engine asserts the stop flag (STF) bit to indicate that it has stopped However, the BIU continues to run, and the Host can access all registers except for the channel registers[2] (see list of channel registers on Section 21.2.11, "Channel Configuration and Control Registers"). After MDIS is set, even before STF asserts, data read from the channel registers is not meaningful and writes are ineffective, issuing a Bus Error. When the MDIS bit is asserted while the microcode is executing, the eTPU will stop when the thread is complete.

1 = Commands Engine to stop its clocks.
0 = eTPU Engine runs.

Stop completes on the next eTPU clock after the stop condition is valid. The MDIS bit is write-protected when VIS=1.

NOTE

Once MDIS is switched from 1 to 0 or vice versa, it must not be written a different value until STF changes accordingly.

STF — Stop Flag Bit

Each Engine asserts its stop flag (STF) to indicate that it has stopped. Only then the host can assume that the Engine has actually stopped. The eTPU system is fully stopped when the STF bits of both eTPU Engines are asserted. In case of IP-Bus stop, the eTPU system responds by acknowledging stop only after both eTPU 1 and eTPU 2 have been stopped. Engine only stops when any ongoing thread is complete also in this case.

1 = Engine has stopped (after the local MDIS bit has been asserted, or after the IP-Bus stop line has been asserted).
0 = Engine is operating.

Summarizing Engine stop conditions, which STF reflects:

STF_1:= (after stop completed) MDIS_1 | device stop request

STF_2:= (after stop completed) MDIS_2 | device stop request

STF_1 and STF_2 mean STF bit from engine 1 and STF bit from engine 2 respectively.

---

[1] Only on rare occasions (e.g., during a long stall, (see eTPU Reference Manual) FEND can be read as 1, because it negates as soon as the end begins execution.

[2] The Timebase registers can still be read with MDIS=1, but writes are ineffective and a Bus Error is issued. Global Channel Registers and SDM can be accessed normally.

HLTF — Halt Mode Flag

If eTPU Engine entered halt state, this flag is asserted. The flag remains asserted while the microengine is in halt state, even during a single-step or forced instruction execution.

1 = eTPU Engine is halted
0 = eTPU Engine is not halted.

FCSS - Filter Clock Source Selection

Speeds up the filter clock source before the prescaler, allowing more input capture resolution at minimum prescaling.

1 = use eTPU clock as EDF clock source before prescaler
0 = use eTPU clock / 2 as EDF clock source before prescaler.

NOTE

FCSS=1 also makes the channel work on T2/T4 timing mode (see eTPU Reference Manual).

FPSCK[0:2] — Filter Prescaler Clock Control

FPSCK controls the prescaling of the clocks used in digital filters for the channel input signals and TCRCLK input, as shown in Table 21-7. Filtering can be controlled independently by Engine, but all input digital filters in the same Engine have same clock prescaling. For more details see Section 21.3.4.4.5, “Filter Clock Prescaler.”

Table 21-7. Filter Prescaler Clock Control

Filter Control	Sample on eTPU clock Divided by:
000	2
001	4
010	8
011	16
100	32
101	64
110	128
111	256

A new value written to FPSCK only becomes effective when the filter prescaler finishes the current count.

CDFC[0:1] — Channel Digital Filter Control

These bits select a digital filtering mode for the channels when configured as inputs for improved noise immunity (refer to Table 21-8). The eTPU has three digital filtering modes for the channels which provide programmable trade-off between signal latency and noise immunity (see Section 21.3.4.4, “Enhanced Digital Filter - EDF”). Changing CDFC during eTPU normal input channel operation is not recommended since it changes the behavior of the transition detection logic while executing its operation.

Table 21-8. Channel Digital Filter Control

CDFC	Selected Digital Filter
00	TPU2/3 Two Sample Mode: Using the filter clock which is the eTPU clock divided by (2, 4, 8,.., 256) as a sampling clock (selected by FPSCK field in ETPUECR), comparing two consecutive samples which agree with each other sets the input signal state. This is the default reset state.
01	eTPU bypass mode: the input signal is taken unfiltered, also making the channels work on T2/T4 timing mode[1].
10	eTPU Three Sample Mode: Similar to the TPU2/3 two sample mode, but comparing three consecutive samples which agree with each other sets the input signal state.
11	eTPU Continuous Mode: Signal need to be stable for the whole filter clock period. This mode compares all the values at the rate of eTPU clock (FCSS=1) or eTPU clock divided by two (FCSS=0), between two consecutive filter clock pulses. Signal needs to be continuously stable for the entire period. If all the values agree with each other, input signal state is updated.

[1] see eTPU Reference Manual

ERBA — Engine Relative Base Address

This field value is concatenated with the AID instruction field in engine relative address mode to form the SDM address (see eTPU Reference Manual).

SPPDIS — Schedule Priority Passing Disable

SPPDIS is used to disable the priority passing mechanism of the microengine scheduler (see Section 21.3.3.2.1, “Primary Scheme - Priority Among Channels on Different Levels).

1 = Scheduler priority passing mechanism disabled.
0 = Scheduler priority passing mechanism enabled.

NOTE

SPPDIS bit must not be changed while any channel is enabled.

ETB[0:4] — Entry Table Base

The field determines the location of the microcode entry table for the eTPU functions in SCM (see eTPU Reference Manual for details of how to use this field).

21.2.6 Time Base Registers

Time Base registers allows configuration and visibility of internally-generated time bases TCR1 and TCR2. There is one of each of these registers for each eTPU Engine.

NOTE

Writes to this register issue bus error and are ineffective when MDIS=1. Reads are always allowed.

21.2.6.1 ETPUTBCR - eTPU Time Base Configuration Register

This register configures several timebase options.

eTPU A/C: Base + 0x020 / eTPU B: Base + 0x040

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TCR2CTL			TCRCF		AM		0	0	0	TCR2P					
W								▨	▨	▨						
RESET:	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TCR1CTL		TCR1 CS	0	0	0	0	0	TCR1P							
W				▨	▨	▨	▨	▨								
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-8. ETPUTBCR Register

TCR2CTL[0:2] — TCR2 Clock/Gate Control

These bits are part of the TCR2 clocking system (see Section 21.3.5, “Time Bases). They determine the clock source for TCR2 before the prescaler. TCR2 can count on any detected edge of the TCRCLK signal or use it for gating eTPU clock divided by 8. After reset - TCRCLK signal rising edge is selected. TCR2 can also be clocked by the eTPU clock divided by 8. TCR2CTL also determines the TCRCLK edge selected for angle tooth detection in angle mode. See Table 21-9.

Table 21-9. TCR2 Clock Source

TCR2CTL	AM = 0 **TCR2 Clock before prescaler**	AM = 1 Angle Tooth detection
000	Gated DIV8 clock (eTPU clock / 8). In this case, when the external TCRCLK signal is low, the DIV8 clock is blocked, preventing it from incrementing the TCR2 prescaler. When the external TCRCLK signal is high, TCR2 prescaler is incremented at the frequency of the eTPU clock divided by 8.	do not use with AM=1
001	Rise transition on TCRCLK signal increments the TCR2 prescaler.	rise edge
010	Fall transition on TCRCLK signal increments the TCR2 prescaler.	fall edge
011	Rise or Fall transition on TCRCLK signal increments the TCR2 prescaler.	both edges
100	DIV8 clock (eTPU clock / 8)	do not use with AM=1
101	do not use with AM=0	
110		no edge[1]
111	TCR2 frozen, except as STAC client	do not use with AM=1

[1] TCRCLK edges are not detected by the EAC logic, but they can still be detected by the channel 0 logic if AM=01.

TCRCF[0:1] — TCRCLK Signal Filter Control

This field controls the TCRCLK digital filter (see Section 21.3.5.5, "TCRCLK Digital Filter), determining whether the TCRCLK signal input (after a synchronizer) is filtered with the same filter clock as the channel input signals (see Section 21.3.4.4, "Enhanced Digital Filter - EDF") or uses the eTPU clock divided by 2, and also whether the TCRCLK digital filter works in integrator mode or two sample mode (see Table 21-10).

Table 21-10. TCRCLK Filter Clock/Mode

TCRCF	Filter Clock	Filter Mode
00	eTPU clock divided by 2	two sample
01	filter clock of the channels	two sample
10	eTPU clock divided by 2	integrator
11	filter clock of the channels	integrator

AM — Angle Mode Selection

This field enables the Enhanced Angle Counter logic to generate angle information (see eTPU Reference Manual for details), and also select the tooth signal input and the channel used to process it, as shown in Table 21-11. When EAC is not disabled by AM and neither TCR1 nor TCR2 are STAC Clients, the EAC (eTPU Angle Clock) hardware provides angle information to the channels using the TCR2 bus. When AM is reset (non-angle mode), the EAC operation is disabled, and its internal registers can be used as general purpose. For more information, see eTPU Reference Manual.

Table 21-11. AM - Angle Mode Selection

Value	TCR2 Value	Tooth signal	Tooth processing channel
0 0	Timebase (EAC operation disabled)	not applicable	
0 1	Angle Ticks	TCRCLK input	0
1 0		channel 1 input	1
1 1		channel 2 input	2

If TCR1 or TCR2 is a STAC Bus Client (see Section 21.3.5.3, "STAC Interface"), the EAC operation is not allowed, and if AM is set the Angle Logic does not work properly.

NOTE

Changing AM may cause spurious transition detections on the channel selected by AM, depending on the channel mode and state (see eTPU Reference Manual for details). If AM must be changed with GTBE=1, the recommended procedure is described in the eTPU Reference Manual.

TCR2P[0:5] — Timer Count Register 2 Prescaler Control

These bits are part of the TCR2 clocking system (see Section 21.3.5, "Time Bases). TCR2 is clocked from the output of a prescaler. The prescaler divides its input by (TCR2P+1) allowing frequency divisions from 1 to 64. The prescaler input is the eTPU clock divided by 8 (in gated or non-gated clock mode), or Internal Timebase input, or TCRCLK filtered input. This field has no effect on TCR2 in Angle Mode.

TCR1CTL[0:1] — TCR1 Clock/Gate Control

TCR1CTL is part of the TCR1 clocking system (see Section 21.3.5, “Time Bases). It determines, together with TCR1CS, the clock source for TCR1. TCR1 can count on detected rising edge of the TCRCLK signal, the eTPU clock, or the eTPU clock divided by 2 (see Table 21-12). After reset TCRCLK signal is selected

TCR1CS - TCR1 Clock Source

TCR1CS provides the option to double the TCR1 incrementing speed, using eTPU clock as its clock source instead of eTPU clock / 2.

1 = use eTPU clock as TCR1 clock source before the prescaler; can only be set in specific combinations with TCR1CTL (see Table 21-12).

0 = use eTPU clock / 2 as TCR1 clock source before the prescaler, if that clock source is selected by TCR1CTL.

NOTE

TCR1CS=1 also makes the channel work on T2/T4 timing mode (see eTPU Reference Manual for details).

NOTE

The clock source of the EAC angle tick generator will still be an even division of eTPU clock if TCR1CS=1, obeying to the fields TCR1P as if TCR1CS=0 (see eTPU Reference Manual for details).

Table 21-12. TCR1 Clock Source

TCR1CTL	TCR1CS[1]	TCR1 Clock before prescaler
00	0	selects TCRCLK as clock source for the TCR1 prescaler[2]
10	0	selects eTPU clock divided by 2 as clock source for the TCR1 prescaler
10	1	selects eTPU clock as clock source for the TCR1 prescaler
11	0	TCR1 frozen, except as a STAC client;

[1] All other combinations of TCR1CTL and TCR1CS are reserved.

[2] This selection must not be used in Angle Mode.

TCR1P[0:7] — Timer Count Register 1 Prescaler Control

TCR1 is clocked from the output of a prescaler. The input to the prescaler is the internal eTPU clock divided by 2, eTPU clock, or the output of TCRCLK filter. The prescaler divides this input by (TCR1P+1) allowing frequency divisions from 1 up to 256.

21.2.6.2 ETPUTB1R - eTPU Time Base 1 (TCR1) Visibility Register

This register provides visibility of the TCR1 time base for host read access (see Section 21.3.5, “Time Bases). This register is read-only. The value of the TCR1 time base shown can be driven by the TCR1 counter or imported from STAC bus, depending on the configuration set in ETPUREDCR.

eTPU A/C: Base + 0x024 / eTPU B: Base + 0x044

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	TCR1[0:7]							
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TCR1[8:23]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-9. ETPUTB1R Register

TCR1[0:23] — TCR1 value

TCR1 value used on matches and captures. See Section 21.3.5, “Time Bases.

21.2.6.3 ETPUTB2R - eTPU Time Base 2 (TCR2) Visibility Register

This register provides visibility of the TCR2 time base for host read access (see Section 21.3.5, “Time Bases). This register is read-only. The value of the TCR2 time base shown can be driven by the TCR2 counter, the Angle Mode logic, or imported from STAC, depending on Angle Mode and STAC configurations set in registers ETPUTBCR and ETPUREDCR.

eTPU A/C: Base + 0x028 / eTPU B: Base + 0x048

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	TCR2[0:7]							
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TCR2[8:23]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-10. ETPUTB2R Register

TCR2[0:23] — TCR2 value

TCR2 value used on matches and captures. See Section 21.3.5, “Time Bases.

21.2.6.4 ETPUREDCR - eTPU STAC Configuration Register

This register configures the eTPU STAC bus operation as a STAC Server/Client module (see Section 21.3.5.3, “STAC Interface”).

eTPU A/C: Base + 0x02C / eTPU B: Base + 0x04C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	REN1	RSC1	0	0	SERVER_ID1				0	0	0	0	SRV1			
W			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨				
RESET:	0	0	0	0	SERVER_ID1				0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	REN2	RSC2	0	0	SERVER_ID2				0	0	0	0	SRV2			
W			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨				
RESET:	0	0	0	0	SERVER_ID2				0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-11. ETPUREDCR Register

REN1,2 — TCR1/2 Resource[1] Client/Server Operation Enable Bits

These bits enable or disable Client/Server operation to eTPU STAC resources. REN1 enables TCR1 and REN2 enables TCR2 STAC bus operations.

1 = Server/Client Operation for resource 1/2 is enabled.
0 = Server/Client Operation for resource 1/2 is disabled.

RSC1,2 — TCR1/2 Resource Server/Client Assignment Bits

These bits select the eTPU data resource assignment to be used as Servers or Clients. RSC1 selects the functionality of TCR1 and RSC2 selects the functionality of TCR2. For Server mode, external plugging determines the unique server address assigned to each TCR. For a Client mode, the SRV1 and SRV2 fields determine the Server address to which the Client listens.

1 = Resource Server operation.
0 = Resource Client operation.

NOTE

When TCR1 or TCR2 is configured as a STAC Bus Client (REN2=1, RSC2=0) the eTPU Angle Clock hardware cannot be used.

NOTE

RSC1,2 must not be changed when the respective REN1,2 bit is asserted.

SERVER_ID1,2 — STAC Ids 1,2

STAC Server Ids (read-only plug values) used for TCR1 and TCR2, respectively, when STAC servers.

---

[1] **resource** identifies any parameter that changes along the time and can be exported / imported from other device. In eTPU context, a resource can be TCR1 or TCR2 (either Time or Angle values).

Table 21-13. Server IDs

	eTPU_A	eTPU_B	eTPU_C
SERVER_ID1	0x0	0x1	0x4
SERVER_ID2	0x2	0x3	0x5

SRV1,2 — TCR1/2 Resource Server

These bits select the address of the specific STAC Server to which the local TCR1 or TCR2 listens when configured as a STAC Client. SRV1 and SRV2 select the STAC Server of TCR1 and TCR2, respectively.

21.2.7 Engine Related Registers

This section gathers registers that are engine-related, other than ETPUECR (see Section 21.2.5.5, "ETPUECR - eTPU Engine Configuration Register)".

21.2.7.1 ETPUWDTR - eTPU Watchdog Timer Register

This register configures the watchdog timer for the engine.

eTPU A/C: Base + 0x060 / eTPU B: Base + 0x070

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	WDM		0	0	0	0	0	0	0	0	0	0	0	0	0	0
W			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	WDCNT[15:0]															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-12. ETPUWDTR Register

WDM — Watchdog Mode

WDM selects the Watchdog operation mode, as shown in Table 21-14. For more information on the Watchdog operation, see Section 21.3.1, "Watchdog".

Table 21-14. WDM - Watchdog Mode

Value	Watchdog Mode
0 0	disabled
0 1	reserved

Table 21-14. WDM - Watchdog Mode

Value	Watchdog Mode
1 0	thread length
1 1	busy length

NOTE

Before a new mode is configured, all conditions below must apply:

1- all channels must be disabled (ETPUCxCR field CPR=00).

2- no thread must be executing (register ETPUCSSR reads 0).

3- the watchdog must be disabled (WDM=00).

WDCNT[15:0] — Watchdog Count

This field indicates the maximum number of microcycles allowed for a thread (in thread length mode) or a sequence of threads (in busy length mode) before the current running thread is forced to end. For more information on Watchdog operation, see Section 21.3.1, “Watchdog.

NOTE

The TST microcycles are also counted by the watchdog.

21.2.7.2 ETPUIDLER - eTPU Idle Register

This register counts the microcycles in which the microengine is idle (see Section 21.3.7.1, “Idle Counter).

eTPU A/C: Base + 0x068 / eTPU B: Base + 0x078

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	IDLE_CNT[31:16]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	IDLE_CNT[15:0]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	ICLR
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-13. ETPUIDLER Register

IDLE_CNT[31:0] — Idle Count

This is a freerunning count of the number of idle microcycles in the microengine. For more information on idle counter operation, see Section 21.3.7.1, “Idle Counter”.

ICLR — Idle Clear

This write-only bit is used to clear the idle count IDLE_CNT:

1 = clear the idle count IDLE_CNT
0 = do not clear idle count IDLE_CNT

21.2.8 Memory Error Support Registers

21.2.8.1 ETPUMECR - eTPU Memory Error Control Register

This register gathers control bits of both SCM and SDM error detection and correction support.

Base + 0x100

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FME	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MECIE
W		▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	HDEIE	MDEIE	0	0	0	DTEND	DRDIS	DEDD	CEIE	0	0	0	0	CTEND	CRDIS	CEDD
W			▨	▨	▨					▨	▨	▨	▨			
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-14. ETPUMECR Register

FME — Field Modification Enable

Negation of this bit disables the modification of all fields in the register, except MECIE, DRDIS, CRDIS and FME itself:

1 = all fields can be modified in a write access
0 = fields cannot be modified in a write access, except MECIE, DRDIS, CRDIS and FME

Write accesses do not cause a bus error, regardless of FME state.

MECIE — Memory Error Correction Interrupt Enable

This bit enables the Memory Error Correction Interrupt (see Section 21.3.2.2.1, “Interrupt Types and Sources”):

1 = Memory Error Correction Interrupt enabled
0 = Memory Error Correction Interrupt disabled

HDEIE — Host Data Error Injection Enable

This bit enables error injection into the SDM host read path, including CDC transfers:

1 = SDM error injection enabled for host reads
0 = SDM error injection disabled for host reads

Register ETPUDEIAR specifies the address where the error is injected, while registers ETPUDEIDPR and ETPUDEIPPR define the bit flips on data and parity, respectively.

NOTE

Data error injection is also performed when data error detection is disabled (DEDD=1).

MDEIE — Microengine(s) Data Injection Enable

This bit enables error injection into the SDM microengine read path:
1 = SDM error injection enabled for microengine reads
0 = SDM error injection disabled for microengine reads

Register ETPUDEIAR specifies the address where the error is injected, while registers ETPUDEIDPR and ETPUDEIPPR define the bit flips on data and parity, respectively.

NOTE

Data error injection is also performed when data error detection is disabled (DEDD=1).

DTEND — Thread End on Data Non-correctable Error

This bit controls the behavior of the microengine when a non-correctable data error occurs. For more details, see Section 21.3.6.2, “Memory Error Support”.
1 = thread ends on non-correctable data read access by the microengine
0 = thread continues on non-correctable data read access by the microengine

NOTE

DTEND must only be changed when the engine is stopped or halted, or when all channels are disabled.

DRDIS — Data Error Report Disable

This bit disables the sampling of SDM error address, data and syndrome into the registers ETPUDRAR, ETPUDRDR, ETPUDRSR. It does not, however, disable the setting of ETPUMESR bits DCERR and DNCERR
1 = SDM error report sampling disabled
0 = SDM error report sampling enabled

NOTE

To guarantee coherent reading of registers ETPUDRAR, ETPUDRDR and ETPUDRSR, DRDIS must be asserted beforehand if SDM concurrent accesses can happen, by host or microengine. After the registers are read, DRDIS may be negated again to allow new samplings.

DEDD — Data Error Detection Disable

This bit disables SDM error detection:
1 = SDM error detection disabled
0 = SDM error detection enabled

The disabled state turns ineffective any action due to error detection, including consequently error correction.

CEIE — Code Error Injection Enable

This bit enables error injection into the SCM read path, either on microengine accesses (ETPUMCR bit VIS=0) or host accesses (VIS=1):
1 = SCM error injection enabled
0 = SCM error injection disabled

Register ETPUCEIAR specifies the address where the error is injected, while registers ETPUCEIDPR and ETPUCEIPPR define the bit flips on data and parity, respectively.

NOTE

Code error injection is also performed when code error detection is disabled (CEDD=1).

CTEND — Thread End on Code Non-correctable Error

This bit controls the behavior of the microengine when a non-correctable code error occurs. For more details, see Section 21.3.6.2, “Memory Error Support”.

1 = thread ends on instruction fetch non-correctable error
0 = thread continues on instruction fetch non-correctable error

NOTE

CTEND must only be changed when the engine is stopped or halted, or when all channels are disabled.

CRDIS — Code Error Report Disable

This bit disables the sampling of SCM error address, data and syndrome into the registers ETPUCRAR, ETPUCRDR, ETPUCRSR. It does not, however, disable the setting of ETPUMESR bits CCERR and CNCERR.

1 = SCM error report sampling disabled
0 = SCM error report sampling enabled

NOTE

To guarantee coherent reading of registers ETPUCRAR, ETPUCRDR and ETPUCRSR, CRDIS must be asserted beforehand if SCM concurrent accesses can happen, by host or microengine. After the registers are read, CRDIS may be negated again to allow new samplings.

CEDD — Code Error Detection Disable

This bit disables SCM error detection:

1 = SCM error detection disabled
0 = SCM error detection enabled

The disabled state turns ineffective any action due to error detection, including consequently error correction.

21.2.8.2 ETPUDEIAR - eTPU Data Error Injection Address Register

This register holds the address where error is to be injected into the SDM read data path. The two least significant bits are 0 (read-only), so that the address is always word-aligned.

NOTE

> This register must only be modified when ETPUMECR bits HDEIE and MDEIE are both negated.

Base + 0x104

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	INJ_ADDR[13:2]												0	0
W	▨	▨													▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-15. ETPUDEIAR Register

INJ_ADDR[13:2] — Injection Address

INJ_ADDR holds the (word) address where an error is injected in the SDM read data path, as specified by registers ETPUDEIDPR and ETPUDEIPPR. For more information on error injection, see Section 21.3.6.2.2, “Error Injection”.

21.2.8.3 ETPUDEIDPR - eTPU Data Error Injection Data Pattern Register

This register is used to specify the bits in which error is injected into the SDM read data path.

NOTE

> This register must only be modified when ETPUMECR bits HDEIE and MDEIE are both negated.

Base + 0x108

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DFLIP[31:16]															
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DFLIP[15:0]															
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-16. ETPUDEIDPR Register

DFLIP[31:0] — Data Flip bits

Each bit corresponds to a bit in the actual data read from the SDM address defined in the ETPUDEIAR register, specifying whether it should be inverted or not. The inversion only occurs if the SDM error injection is enabled by the ETPUMECR register bits HDEIE and/or MDEIE. For more information on error injection, see Section 21.3.6.2.2, “Error Injection”.

1 = read data bit is inverted
0 = read data bit is not inverted

21.2.8.4 ETPUDEIPPR - eTPU Data Error Injection Parity Pattern Register

This register is used to specify the bits in which error is injected into the SDM read parity path.

NOTE

This register must only be modified when ETPUMECR bits HDEIE and MDEIE are both negated.

Base + 0x10C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	PFLIP[19:15]					0	0	0	PFLIP[14:10]				
W	▨	▨	▨						▨	▨	▨					
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	PFLIP[9:5]					0	0	0	PFLIP[4:0]				
W	▨	▨	▨						▨	▨	▨					
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-17. ETPUDEIPPR Register

PFLIP[19:0] — Parity Flip bits

Each bit corresponds to a bit in the actual parity read from the SDM address defined in the ETPUDEIAR register, specifying whether it should be inverted or not. The inversion only occurs if the SDM error injection is enabled by the ETPUMECR register bits HDEIE and/or MDEIE. Bits PFLIP[19:15] corresponds to the parity bits for the most significant byte, PFLIP[14:10] to the next most significant byte and so on until PFLIP[4:0] for the least significant byte. For more information on error injection, see Section 21.3.6.2.2, “Error Injection”.

1 = parity data bit is inverted
0 = parity data bit is not inverted

21.2.8.5 ETPUDERAR - eTPU Data Error Report Address Register

This register holds the address where the last non-correctable error was detected from the SDM. It also captures the address of the last correctable error detected, as long as the ETPUMESR bit DNCERR is 0. The two least significant bits are 0 (read-only), so that the address is always word-aligned.

Base + 0x110

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ERR_ACC		0	ERR_CHANNEL					0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	ERR_ADDR[13:2]												0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-18. ETPUDERAR Register

ERR_ACC[1:0] — Error Access Source Indication

> This two-bit field indicates the source of the access when the error occurred, as indicated by Table 21-15.

Table 21-15. ERR_ACC - SDM Error Access source indication

Value	Access Source
0 0	Host - direct access
0 1	Engine 1
1 0	Engine 2
1 1	CDC

ERR_CHANNEL[5:0] — Error Channel Access Indication

> This field holds the number of the channel being serviced (not necessarily CHAN) when the error occurred. Its content is meaningful only when the field ERR_ACC indicates Engine 1 or Engine 2.

ERR_ADDR[13:2] — Error Address

> ERR_ADDR holds the (word) address where the last captured SDM read error was detected. The last captured error is a non-correctable one if ETPUMESR register bit DNCERR=1, or a correctable one if DNCERR=0 and DCERR=1. The field is meaningless if both DNCERR and DCERR are 0. For more information on error detection, see Section 21.3.6.2, “Memory Error Support”.

21.2.8.6 ETPUDERDR - eTPU Data Error Report Data Register

This register holds the data read with the last non-correctable error detected from the SDM. It also captures the data of the last correctable error read detection, as long as the ETPUMESR bit DNCERR is 0. In both cases, the data is captured after the error correction logic. The data is meaningless if both DNCERR and DCERR are 0. For more information on error detection, see Section 21.3.6.2, “Memory Error Support”.

NOTE

Data is sampled before the error correction logic.

Base + 0x114

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DATA[31:16]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DATA[15:0]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-19. ETPUDERDR Register

NOTE

Individual data bytes are not meaningful when their respective BE bits in ETPUDERSR are not asserted.

21.2.8.7 ETPUDERSR - eTPU Data Error Report Syndrome Register

This register holds the syndrome read with the last non-correctable error detected from the SDM. It also captures the syndrome of the last correctable error read detection, as long as the ETPUMESR bit DNCERR is 0.

Base + 0x118

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BE3	0	0	SYND3[4:0]					BE2	0	0	SYND2[4:0]				
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BE1	0	0	SYND1[4:0]					BE0	0	0	SYND0[4:0]				
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-20. ETPUDERDR Register

BE[0-3] — Byte Enables (0 to 3)

Each BE bit corresponds to a report byte in a 32-bit aligned data word. BE3 corresponds to the most significant one. BE qualifies the error report for its respective byte:

1 = byte was selected for read, and the error report for the respective byte is meaningful
0 = byte was not selected for read, therefore its respective error report fields are not meaningful

SYND[0-3][4:0] — Error Syndrome (0 to 3)

SYNDn holds the syndrome of an error detected in the SDM read data path, as defined in Section 21.3.6.2.1, “Error Correction Code (ECC) and Syndrome Definition”. The field is meaningless if both DNCERR and DCERR are 0. For more information on error detection, see Section 21.3.6.2, “Memory Error Support”.

NOTE

Syndrome fields which have their respective BE bits negated are not meaninful.

21.2.8.8 ETPUMESR - eTPU Memory Error Status Register

This register gathers status bits of both SCM and SDM error detection and correction support.

Base + 0x120

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DCERR	0	0	0	0	0	0	DCOVR	CCERR	0	0	0	0	0	0	CCOVR
W	DCERRC	▨	▨	▨	▨	▨	▨	DCOVRC	CCERRC	▨	▨	▨	▨	▨	▨	CCOVRC
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DNCERR	0	0	0	0	0	0	DNCOVR	CNCERR	0	0	0	0	0	0	CNCOVR
W	DNCERRC	▨	▨	▨	▨	▨	▨	DNCOVRC	CNCERRC	▨	▨	▨	▨	▨	▨	CNCOVRC
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-21. ETPUMESR Register

DCERR — Data Correctable Error flag

This bit flags a correctable error:
1 = an SDM correctable error occurred
0 = no SDM correctable error occurred

DCERR asserts even if DNCERR is asserted beforehand, but in this case the address and data of the erroneous access is not captured (see Section 21.2.8.5, "ETPUDERAR - eTPU Data Error Report Address Register" and following sections). DCERR is cleared by writing a 1 at the same position (DCERRC).

NOTE

A correctable error is not considered for flagging if accompanied of a non-correctable one. Therefore, if both correctable and non-correctable data errors occur on the same access (in distinct bytes), the DNCERR flag asserts, but not the DCERR flag.

DCERRC — Data Correctable Error flag clear

Writing 1 to this bit clears the DCERR status flag:
1 = clears DCERR flag
0 = do not clear DCERR flag

DCOVR — Data Correctable Overrun flag

This bit asserts when an SDM correctable error occurs and the DCERR flag is already asserted. Assertion of this bit causes no interrupt.
1 = an SDM correctable error overrun occurred
0 = no SDM correctable error overrun occurred

DCOVR is cleared by writing a 1 at the same position (DCOVRC).

NOTE

A correctable error is not considered for flagging if accompanied of a non-correctable one. Therefore, if both correctable and non-correctable data errors occur on the same access (in distinct bytes), the DCOVR does not assert, even if DCERR is asserted. The DNCOVR flag, on the other hand, asserts if DNCERR is asserted.

DCOVRC — Data Correctable Overrun flag clear

Writing 1 to this bit clears the DCOVR status flag:
1 = clear DCOVR flag
0 = do not clear DCOVR flag

CCERR — Code Error flag

This bit flags a correctable (single-bit) error:
1 = an SCM correctable error occurred
0 = no SCM correctable error occurred

CCERR asserts even if CNCERR is asserted beforehand, but in this case the address and data of the erroneous access is not captured (see Section 21.2.8.12, “ETPUCERAR - eTPU Code Error Report Address Register” and following sections). CCERR is cleared by writing a 1 at the same position (CCERRC).

CCERRC — Code Error flag clear

Writing 1 to this bit clears the CCERR status flag:
1 = clears CCERR flag
0 = do not clear CCERR flag

CCOVR — Code Correctable Overrun flag

This bit asserts when an SCM correctable error occurs and the CCERR flag is already asserted. Assertion of this bit causes no interrupt.
1 = an SCM correctable error overrun occurred
0 = no SCM correctable error overrun occurred

CCOVR is cleared by writing a 1 at the same position (CCOVRC).

CCOVRC — Code Correctable Overrun flag clear

Writing 1 to this bit clears the CCOVR status flag:
1 = clear CCOVR flag
0 = do not clear CCOVR flag

DNCERR — Data Non-Correctable Error flag

This bit flags a non-correctable error:
1 = an SDM non-correctable error occurred; registers ETPUDERAR and ETPUDERDR are not updated by correctable error detections
0 = no SDM non-correctable error occurred

DNCERR is cleared by writing a 1 at the same position (DNCERRC). DNCERR must be cleared to allow correctable error updates to registers ETPUDERAR and ETPUDERDR.

DNCERRC — Data Non-Correctable Error flag clear

Writing 1 to this bit clears the DNCERR status flag:
1 = clear DNCERR flag
0 = do not clear DNCERR flag

DNCOVR — Data Non-Correctable Overrun flag

This bit asserts when an SDM non-correctable error occurs and the DNCERR flag is already asserted. Assertion of this bit causes no interrupt.
1 = an SDM non-correctable error overrun occurred
0 = no SDM non-correctable error overrun occurred

DNCOVR is cleared by writing a 1 at the same position (DNCOVRC).

DNCOVRC — Data Non-Correctable Overrun flag clear

Writing 1 to this bit clears the DNCOVR status flag:
1 = clear DNCOVR flag
0 = do not clear DNCOVR flag

CNCERR — Code Non-Correctable Error flag

This bit flags a non-correctable error:
1 = an SCM non-correctable error occurred; registers ETPUCERAR and ETPUCERDR are not updated by correctable error detections
0 = no SCM non-correctable error occurred

CNCERR is cleared by writing a 1 at the same position (CNCERRC). CNCERR must be cleared to allow correctable error updates to registers ETPUCERAR and ETPUCERDR.

CNCERRC — Code Error flag clear

Writing 1 to this bit clears the CNCERR status flag:
1 = clear CNCERR flag
0 = do not clear CNCERR flag

CNCOVR — Code Non-Correctable Overrun flag

This bit asserts when an SCM non-correctable error occurs and the CNCERR flag is already asserted. Assertion of this bit causes no interrupt.
1 = an SCM non-correctable error overrun occurred
0 = no SCM non-correctable error overrun occurred

CNCOVR is cleared by writing a 1 at the same position (CNCOVRC).

CNCOVRC — Code Non-Correctable Overrun flag clear

Writing 1 to this bit clears the CNCOVR status flag:
1 = clear CNCOVR flag
0 = do not clear CNCOVR flag

21.2.8.9 ETPUCEIAR - eTPU Code Error Injection Address Register

This register holds the address where error is to be injected into the SCM read data path. The two least significant bits are 0 (read-only), so that the address is always word-aligned.

NOTE

This register must only be modified when ETPUMECR bit CEIE is negated.

Base + 0x124

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	INJ_ADDR[15:2]														0	0
W															▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-22. ETPUCEIAR Register

INJ_ADDR[15:2] — Injection Address

INJ_ADDR holds the (word) address where an error is injected in the SCM read data path, as specified by registers ETPUCEIDPR and ETPUCEIPPR. For more information on error injection, see Section 21.3.6.2.2, “Error Injection”.

21.2.8.10 ETPUCEIDPR - eTPU Code Error Injection Data Pattern Register

This register is used to specify the bits in which error is injected into the SCM read data path.

NOTE

This register must only be modified when ETPUMECR bit CEIE is negated.

Base + 0x128

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DFLIP[31:16]															
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DFLIP[15:0]															
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-23. ETPUCEIDPR Register

DFLIP[31:0] — Data Flip bits

Each bit corresponds to a bit in the actual data read from the SCM address defined in the ETPUCEIAR register, specifying whether it should be inverted or not. The inversion only occurs if the SCM error injection is enabled by the ETPUMECR register bit CEIE. For more information on error injection, see Section 21.3.6.2.2, “Error Injection”.

1 = read data bit is inverted
0 = read data bit is not inverted

21.2.8.11 ETPUCEIPPR - eTPU Code Error Injection Parity Pattern Register

This register is used to specify the bits in which error is injected into the SCM read parity path.

NOTE

This register must only be modified when ETPUMECR bit CEIE is negated.

Base + 0x12C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	PFLIP[6:0]						
W	▨	▨	▨	▨	▨	▨	▨	▨	▨							
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-24. ETPUCEIPPR Register

PFLIP[6:0] — Parity Flip bits

Each bit corresponds to a bit in the actual parity/code read from the SCM address defined in the ETPUCEIAR register, specifying whether it should be inverted or not. The inversion only occurs if the SCM error injection is enabled by the ETPUMECR register bit CEIE. For more information on error injection, see Section 21.3.6.2.2, “Error Injection”.

1 = parity data bit is inverted
0 = parity data bit is not inverted

21.2.8.12 ETPUCERAR - eTPU Code Error Report Address Register

This register holds the address where the last non-correctable error was detected from the SCM. It also captures the address of the last correctable error detected, as long as the ETPUMESR bit CNCERR is 0. The two least significant bits are 0 (read-only), so that the address is always word-aligned.

Base + 0x130

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ERR_ACC		0	ERR_CHANNEL					0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ERR_ADDR[15:2]														0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-25. ETPUCERAR Register

ERR_ACC[1:0] — Error Access Source Indication

This two-bit field indicates the source of the access when the error occurred, as indicated by Table 21-16.

Table 21-16. ERR_ACC - SCM Error Access source indication

Value	Access Source
0 0	Host
0 1	Engine 1
1 0	Engine 2
1 1	MISC

ERR_CHANNEL[5:0] — Error Channel Access Indication

This field holds the number of the channel being serviced (not necessarily CHAN) when the error occurred. Its content is meaningful only when the field ERR_ACC indicates Engine 1 or Engine 2.

ERR_ADDR[15:2] — Error Address

ERR_ADDR holds the (word) address where the last captured SCM read error was detected. The last captured error is a non-correctable one if ETPUMESR register bit CNCERR=1, or a correctable one if CNCERR=0 and CCERR=1. The field is meaningless if both CNCERR and CCERR are 0. For more information on error detection, see Section 21.3.6.2, “Memory Error Support”.

21.2.8.13 ETPUCERDR - eTPU Code Error Report Data Register

This register holds the data read with the last non-correctable error detected from the SCM. It also captures the data of the last correctable error read detection, as long as the ETPUMESR bit CNCERR is 0. In both cases, the data is captured after the error correction logic. The data may be meaningless if both CNCERR and CCERR are 0. For more information on error detection, see Section 21.3.6.2, “Memory Error Support”.

NOTE

Code is sampled before the error correction logic.

Base + 0x134

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CODE[31:16]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CODE[15:0]															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-26. ETPUCERDR Register

21.2.8.14 ETPUCERSR - eTPU Code Error Report Syndrome Register

This register holds the syndrome and global parity read with the last non-correctable error detected from the SCM. It also captures the syndrome of the last correctable error read detection, as long as the ETPUMESR bit CNCERR is 0.

Base + 0x138

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	SYND[6:0]						
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-27. ETPUCERSR Register

SYND[6:0] — Error Syndrome

SYND holds the syndrome of an error detected in the SCM read data path, as defined in Section 21.3.6.2.1, “Error Correction Code (ECC) and Syndrome Definition”. The field is meaningless if both CNCERR and CCERR are 0. For more information on error detection, see Section 21.3.6.2, “Memory Error Support”.

21.2.8.15 ETPUCEFR - eTPU Code Error Fix Register

This register is used to fix single-bit error in a specified SCM position. The fix is done automatically, concurrently with code execution, by the MISC logic (see Section 21.3.6.2.3, “Error Fixing).

Base + 0x13C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CFIXM		0	0	0	0	0	0	0	0	0	0	0	0	0	0
W			▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CFIX_ADDR[15:2]														0	0
W															▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-28. ETPUCEFR Register

CFIXM — Code Fix Mode

This field specifies the SCM MISC fix logic operation mode, as defined in Table 21-17.

NOTE

The CFIXM field can only be changed to/from disabled mode.

Table 21-17. CFIXM - Code Fix Mode

Value	Mode	Description
0 0	Disabled	SCM error fix disabled.
0 1	Reserved	Reserved value, must not be used.
1 0	Reserved	Reserved value, must not be used.
1 1	Automatic	The MISC logic rewrites the code word value read (already fixed by the error correction mechanism) into the same address whenever it finds a correctable error.

CFIX_ADDR[13:2] — Code Fix Address

This field is used for hardware debug purposes. CFIX_ADDR may change its contents and must not be written in automatic mode. For more information on error detection and correction, see Section 21.3.6.2, “Memory Error Support”.

21.2.9 Channel Registers Layout

The channel registers area is shown in Figure 21-29 and detailed in next sections, for eTPU systems of 32 channels per Engine. Reserved areas are placed to allow doubling the number of channels to 64 for each eTPU Engine.

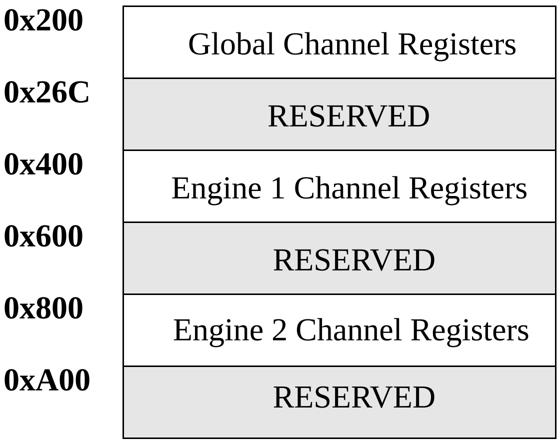


Figure 21-29. Channel Registers Area

21.2.10 Global Channel Registers

The registers in this section group, by type, the interrupt status and enable bits from all the channels. This organization eases management of all channels or groups of channels by a single interrupt handler routine. These bits, except the service and watchdog status, are mirrored in the individual channel registers, grouped by channel.

21.2.10.1 ETPUCISR - eTPU Channel Interrupt Status Register

Host interrupt status (see Section 21.3.2.2, “Interrupts and Data Transfer Requests) from all channels are grouped in ETPUCISR. Their bits are mirrored from the Channel Status/Control registers (see Section 21.2.11, “Channel Configuration and Control Registers) and Host must write 1 to clear a status bit.

eTPU A/C: Base + 0x200 / eTPU B: Base + 0x204

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIS31	CIS30	CIS29	CIS28	CIS27	CIS26	CIS25	CIS24	CIS23	CIS22	CIS21	CIS20	CIS19	CIS18	CIS17	CIS16
W	CIC31	CIC30	CIC29	CIC28	CIC27	CIC26	CIC25	CIC24	CIC23	CIC22	CIC21	CIC20	CIC19	CIC18	CIC17	CIC16
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CIS15	CIS14	CIS13	CIS12	CIS11	CIS10	CIS9	CIS8	CIS7	CIS6	CIS5	CIS4	CIS3	CIS2	CIS1	CIS0
W	CIC15	CIC14	CIC13	CIC12	CIC11	CIC10	CIC9	CIC8	CIC7	CIC6	CIC5	CIC4	CIC3	CIC2	CIC1	CIC0
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-30. ETPUCISR Register

CISx — Channel x Interrupt Status
1 = indicates that channel x has a pending interrupt to the Host CPU.
0 = indicates that channel x has no pending interrupt to the Host CPU.

CICx — Channel x Interrupt Clear
1 = clear interrupt status bit.
0 = keep interrupt status bit unaltered.

For details about interrupts see the eTPU Reference Manual for details.

21.2.10.2 ETPUCDTRSR - eTPU Channel Data Transfer Request Status Register

Data Transfer request status (see Section 21.3.2.2, “Interrupts and Data Transfer Requests) from all channels are grouped in ETPUCDTRSR. Their bits are mirrored from the Channel Status/Control registers (see Section 21.2.10.9, “ETPUCSSR - eTPU Channel Service Status Register).

eTPU A/C: Base + 0x210 / eTPU B: Base + 0x214

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DTRS 31	DTRS 30	DTRS 29	DTRS 28	DTRS 27	DTRS 26	DTRS 25	DTRS 24	DTRS 23	DTRS 22	DTRS 21	DTRS 20	DTRS 19	DTRS 18	DTRS 17	DTRS 16
W	DTRC 31	DTRC 30	DTRC 29	DTRC 28	DTRC 27	DTRC 26	DTRC 25	DTRC 24	DTRC 23	DTRC 22	DTRC 21	DTRC 20	DTRC 19	DTRC 18	DTRC 17	DTRC 16
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DTRS 15	DTRS 14	DTRS 13	DTRS 12	DTRS 11	DTRS 10	DTRS 9	DTRS 8	DTRS 7	DTRS 6	DTRS 5	DTRS 4	DTRS 3	DTRS 2	DTRS 1	DTRS 0
W	DTRC 15	DTRC 14	DTRC 13	DTRC 12	DTRC 11	DTRC 10	DTRC 9	DTRC 8	DTRC 7	DTRC 6	DTRC 5	DTRC 4	DTRC 3	DTRC 2	DTRC 1	DTRC 0
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-31. ETPUCDTRSR Register

DTRSx — Channel x Data Transfer Request Status

These bits mimic the corresponding ETPU DMA requests. DTRSx can be cleared by software (writing 1 to DTRCx) or by the assertion of corresponding DMA completion acknowledge line.

1 = indicates that channel x has a pending data transfer request.
0 = indicates that channel x has no pending data transfer request.

DTRCx — Channel x Data Transfer Request Clear

1 = clear status bit.
0 = keep status bit unaltered

For details about interrupts see the eTPU Reference Manual for details.

21.2.10.3 ETPUCIOSR - eTPU Channel Interrupt Overflow Status Register

Interrupt Overflow status (see Section 21.3.2.2, "Interrupts and Data Transfer Requests) from all channels are grouped in ETPUCIOSR. Their bits are mirrored from the Channel Status/Control registers (see Section 21.2.11.2, "ETPUCxSCR - eTPU Channel x Status Control Register) and one must write 1 to clear a status bit.

eTPU A/C: Base + 0x220 / eTPU B: Base + 0x224

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIOS 31	CIOS 30	CIOS 29	CIOS 28	CIOS 27	CIOS 26	CIOS 25	CIOS 24	CIOS 23	CIOS 22	CIOS 21	CIOS 20	CIOS 19	CIOS 18	CIOS 17	CIOS 16
W	CIOC 31	CIOC 30	CIOC 29	CIOC 28	CIOC 27	CIOC 26	CIOC 25	CIOC 24	CIOC 23	CIOC 22	CIOC 21	CIOC 20	CIOC 19	CIOC 18	CIOC 17	CIOC 16
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CIOS 15	CIOS 14	CIOS 13	CIOS 12	CIOS 11	CIOS 10	CIOS 9	CIOS 8	CIOS 7	CIOS 6	CIOS 5	CIOS 4	CIOS 3	CIOS 2	CIOS 1	CIOS 0
W	CIOC 15	CIOC 14	CIOC 13	CIOC 12	CIOC 11	CIOC 10	CIOC 9	CIOC 8	CIOC 7	CIOC 6	CIOC 5	CIOC 4	CIOC 3	CIOC 2	CIOC 1	CIOC 0
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-32. ETPUCIOSR Register

CIOSx — Channel x Interrupt Overflow Status

1 = indicates that interrupt overflow occurred in the channel.
0 = indicates that no interrupt overflow occurred in the channel.

CIOCx — Channel x Interrupt Overflow Clear

1 = clear status bit.
0 = keep status bit unaltered.

For details about interrupt overflow, see Section 21.3.2.2.2, "Interrupt and Data Transfer Request Overflow.

21.2.10.4 ETPUCDTROSR - eTPU Channel Data Transfer Request Overflow Status Register

Data Transfer Request Overflow status (see Section 21.3.2.2, “Interrupts and Data Transfer Requests) from all channels are grouped in ETPUCDTROSR. Their bits are mirrored from the Channel Status/Control registers (see Section 21.2.11.2, “ETPUCxSCR - eTPU Channel x Status Control Register) and one must write 1 to clear a status bit.

eTPU A/C: Base + 0x230 / eTPU B: Base + 0x234

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DTROS31	DTROS30	DTROS29	DTROS28	DTROS27	DTROS26	DTROS25	DTROS24	DTROS23	DTROS22	DTROS21	DTROS20	DTROS19	DTROS18	DTROS17	DTROS16
W	DTROC31	DTROC30	DTROC29	DTROC28	DTROC27	DTROC26	DTROC25	DTROC24	DTROC23	DTROC22	DTROC21	DTROC20	DTROC19	DTROC18	DTROC17	DTROC16
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	DTROS15	DTROS14	DTROS13	DTROS12	DTROS11	DTROS10	DTROS9	DTROS8	DTROS7	DTROS6	DTROS5	DTROS4	DTROS3	DTROS2	DTROS1	DTROS0
W	DTROC15	DTROC14	DTROC13	DTROC12	DTROC11	DTROC10	DTROC9	DTROC8	DTROC7	DTROC6	DTROC5	DTROC4	DTROC3	DTROC2	DTROC1	DTROC0
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-33. ETPUCDTROSR Register

DTROSx — Channel x Data Transfer Request Overflow Status

1 = indicates that data transfer request overflow occurred in the channel.
0 = indicates that no data transfer request overflow occurred in the channel.

DTROCx — Channel x Data Transfer Request Overflow Clear

1 = clear status bit.
0 = keep status bit unaltered.

For details about data transfer request overflow, see Section 21.3.2.2.2, “Interrupt and Data Transfer Request Overflow.

21.2.10.5 ETPUCIER - eTPU Channel Interrupt Enable Register

Host interrupt enable (see Section 21.3.2.2, “Interrupts and Data Transfer Requests) from all channels are grouped in ETPUCIER. Their bits are mirrored from the Channel Configuration registers (see Section 21.2.11.2, “ETPUCxSCR - eTPU Channel x Status Control Register).

eTPU A/C: Base + 0x240 / eTPU B: Base + 0x244

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	CIE 31	CIE 30	CIE 29	CIE 28	CIE 27	CIE 26	CIE 25	CIE 24	CIE 23	CIE 22	CIE 21	CIE 20	CIE 19	CIE 18	CIE 17	CIE 16
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	CIE 15	CIE 14	CIE 13	CIE 12	CIE 11	CIE 10	CIE 9	CIE 8	CIE 7	CIE 6	CIE 5	CIE 4	CIE 3	CIE 2	CIE 1	CIE 0
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-34. ETPUCIER Register

CIEx — Channel x Interrupt Enable

1 = interrupt enabled for channel x

0 = interrupt disabled for channel x.

For details about interrupts see the eTPU Reference Manual for details.

21.2.10.6 ETPUCDTRER - eTPU Channel Data Transfer Request Enable Register

Data Transfer request enable (see Section 21.3.2.2, “Interrupts and Data Transfer Requests) from all channels are grouped in ETPUCDTRER. These bits are mirrored from the Channel Configuration registers (see Section 21.2.11.2, “ETPUCxSCR - eTPU Channel x Status Control Register).

eTPU A/C: Base + 0x250 / eTPU B: Base + 0x254

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DTRE 31	DTRE 30	DTRE 29	DTRE 28	DTRE 27	DTRE 26	DTRE 25	DTRE 24	DTRE 23	DTRE 22	DTRE 21	DTRE 20	DTRE 19	DTRE 18	DTRE 17	DTRE 16
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DTRE 15	DTRE 14	DTRE 13	DTRE 12	DTRE 11	DTRE 10	DTRE 9	DTRE 8	DTRE 7	DTRE 6	DTRE 5	DTRE 4	DTRE 3	DTRE 2	DTRE 1	DTRE 0
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-35. ETPUCDTRER Register

DTREx — Channel x Data Transfer Request Enable
1 = Data Transfer request enabled for channel x.
0 = Data Transfer request disabled for channel x.

For details about interrupts see the eTPU Reference Manual for details.

21.2.10.7 ETPUWDSR - eTPU Watchdog Status Register

ETPUWDSR indicates the watchdog influence in each of the engine channels.

eTPU A/C: Base + 0x260 / eTPU B: Base + 0x264

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	WDS31	WDS30	WDS29	WDS28	WDS27	WDS26	WDS25	WDS24	WDS23	WDS22	WDS21	WDS20	WDS19	WDS18	WDS17	WDS16
W	WDSC31	WDSC30	WDSC29	WDSC28	WDSC27	WDSC26	WDSC25	WDSC24	WDSC23	WDSC22	WDSC21	WDSC20	WDSC19	WDSC18	WDSC17	WDSC16
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	WDS15	WDS14	WDS13	WDS12	WDS11	WDS10	WDS9	WDS8	WDS7	WDS6	WDS5	WDS4	WDS3	WDS2	WDS1	WDS0
W	WDSC15	WDSC14	WDSC13	WDSC12	WDSC11	WDSC10	WDSC9	WDSC8	WDSC7	WDSC6	WDSC5	WDSC4	WDSC3	WDSC2	WDSC1	WDSC0
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-36. ETPUWDSR Register

WDSx — Channel x Watchdog Status
1 = watchdog forced end of the channel thread and disabled it.
0 = no watchdog force on the channel.

WDSCx — Channel x Watchdog Status Clear
1 = clear watchdog status bit.
0 = keep watchdog status bit unaltered.

For details about Watchdog mechanism, see Section 21.3.1, “Watchdog”

21.2.10.8 ETPUCPSSR - eTPU Channel Pending Service Status Register

ETPUCPSSR is a read-only register that holds the status of the pending Channel Service Requests (see the eTPU Reference Manual for details).

eTPU A/C: Base + 0x280 / eTPU B: Base + 0x284

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SR31	SR30	SR29	SR28	SR27	SR26	SR25	SR24	SR23	SR22	SR21	SR20	SR19	SR18	SR17	SR16
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SR15	SR14	SR13	SR12	SR11	SR10	SR9	SR8	SR7	SR6	SR5	SR4	SR3	SR2	SR1	SR0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-37. ETPUCPSSR Register

SRx - Pending Service Request x

Indicates a pending Service Request for channel x.

1 = pending Service Request for channel x
0 = no Service Request pending for channel x

Pending SR status is a logic OR of all service requests pending: if only HSR is active, SRx clears only at the end of the thread. SRx clear due to the other request sources is microcode dependent.

NOTE

The pending service status bit for a channel is 1 when a Service Request is pending, even if the Channel is disabled (CPRx = 0).

NOTE

There can be a delay of one clock between writing HSR > 0 in register ETPUCxHSRR of a channel and its respective bit being asserted in ETPUCPSSR.

21.2.10.9 ETPUCSSR - eTPU Channel Service Status Register

ETPUCSSR holds the current channel service status on whether it is being serviced or not (see the eTPU Reference Manual for details). Only one bit may be asserted in this register at a given time. When no channel is being serviced the register read value is 0x00000000. ETPUCSSR is a read-only register. The register can be read during normal eTPU operation for monitoring the scheduler activity.

NOTE

Channel Service Status does not always reflect decoding of the CHAN register, since the later can be changed by the service thread microcode.

eTPU A/C: Base + 0x290 / eTPU B: Base + 0x294

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SS31	SS30	SS29	SS28	SS27	SS26	SS25	SS24	SS23	SS22	SS21	SS20	SS19	SS18	SS17	SS16
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SS15	SS14	SS13	SS12	SS11	SS10	SS9	SS8	SS7	SS6	SS5	SS4	SS3	SS2	SS1	SS0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-38. ETPUCSSR Register

SSx - Service Status x

Indicates that channel x is currently being serviced. It is updated at the 1st microcycle of a Time Slot Transition (see the eTPU Reference Manual for details), or when the microengine ends the thread.

1 = channel x is currently being serviced

0 = channel x is not currently being serviced

21.2.11 Channel Configuration and Control Registers

Each channel has a group of 3 registers used to control, configure and check status of that channel as shown in Table 21-18. This organization eases individual channel management.

NOTE

> A bus error is issued on read or write accesses to these registers when ETPUECR bit MDIS=1. Writes are ineffective on bus error.

Table 21-18. Channel Registers Structure

Channel Offset	Register Name
0x00	ETPUCxCR - eTPU Channel Configuration Register
0x04	ETPUCxSCR - eTPU Channel Status/Control Register
0x08	ETPUCxHSRR - eTPU Channel Host Service Request Register
0x0C	RESERVED

One contiguous area is used to map all channel registers of each eTPU engine as shown inTable 21-19.

Table 21-19. Channel Registers Map

Offset	Registers Structure
0x400	eTPU A/C Channel 0 Registers Structure
0x410	eTPU A/C Channel 1 Registers Structure
0x420	eTPU A/C Channel 2 Registers Structure
0x430	. .
0x5E0	eTPU A/C Channel 30 Registers Structure
0x5F0	eTPU A/C Channel 31 Registers Structure
0x600	RESERVED
0x800	eTPU B Channel 0 Registers Structure
0x810	eTPU B Channel 1 Registers Structure
0x820	. .
0x9E0	eTPU B Channel 30 Registers Structure
0x9F0	eTPU B Channel 31 Registers Structure
0xA00	RESERVED

There are 64 structures defined, one for each available channel in the eTPU System (32 for each Engine). The base address for the structure presented can be calculated by using the following equation:

$$Channel\_Register\_Base = ETPU\_Engine\_Channel\_Base + (channel\_number * 0x10)$$

where:

ETPU_Engine_Channel_Base = *ETPU_Base* + *0x400* for Engine 1

ETPU_Engine_Channel_Base = *ETPU_Base* + *0x800* for Engine 2

21.2.11.1 ETPUCxCR - eTPU Channel x Configuration Register

ETPUCxCR gathers configurations set individually per channel.

Channel_Register_Base + 0x0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIE	DTRE	CPR		0	0	ETPD	ETCS	0	0	0	CFS				
W					▨	▨			▨	▨	▨					
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ODIS	OPOL	0	0	0	CPBA										
W			▨	▨	▨											
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-39. ETPUCxCR Register

NOTE

> The fields ETCS, CFS and CPBA must only be changed while the channel is disabled (field CPR=00).

CIE — Channel Interrupt Enable

(this bit is mirrored from ETPUCIER - see Section 21.2.10.5, “ETPUCIER - eTPU Channel Interrupt Enable Register)

1 = Enable interrupt for this channel.
0 = Disable interrupt for this channel.

See the eTPU Reference Manual for details.

DTRE — Channel Data Transfer Request Enable

(this bit is mirrored from ETPUCDTRER - see Section 21.2.10.6, “ETPUCDTRER - eTPU Channel Data Transfer Request Enable Register)

1 = Enable data transfer request for this channel.
0 = Disable data transfer request for this channel.

See the eTPU Reference Manual for details.

CPR[0:1] — Channel Priority

This field defines the priority level for the channel, used by the Hardware Scheduler (See Section 21.3.3, “Scheduler”).

Table 21-20. Priority level Bits

CPR	Priority
00	Disabled
01	Low
10	Middle

Table 21-20. Priority level Bits

CPR	Priority
11	High

ETPD — Entry Table Pin Direction

This bit selects which channel signal, input or output, is used in the Entry Point selection. The ETPD value has to be compatible with the function chosen for the channel, selected in the field CFS. For details about Entry Table and condition encoding schemes, refer to the eTPU Reference Manual.

1 = use PSTO for Entry Point selection.
0 = use PSTI for Entry Point selection.

ETCS — Entry Table Condition Select

This bit determines the channel condition encoding scheme that selects, according to channel conditions, the Entry Point to be taken in an Entry Table. ETCS value has to be compatible with the function chosen for the channel, selected in field CFS. Two condition encoding schemes are available. For details about Entry Table and condition encoding schemes, refer to the eTPU Reference Manual.

1 = select Alternate Entry Table Condition encoding scheme.
0 = select Standard Entry Table Condition encoding scheme.

CFS[0:4] — Channel Function Select

This field defines the function to be performed by the channel (see the eTPU Reference Manual for details). The Function assigned to the channel has to be compatible with the channel condition encoding scheme, selected by field ETCS.

ODIS — Output Disable

This bit enables the channel to have its output forced to the value opposite to OPOL when the output disable input signal corresponding to the channel group that it belongs is active. See Section 21.2.2.4, “eTPU Channel Output Disable Signals” and the eTPU Reference Manual for details.

1 = turns on the output disable feature for the channel
0 = turns off the output disable feature for the channel.

OPOL - Output Polarity

Determines the output signal polarity. The activation of the output disable signal forces, when enabled by the ODIS bit, the channel output signal to the opposite of this polarity (see the eTPU Reference Manual for details).

1 = output active high (output disable drives output to low)
0 = output active low (output disable drives output to high)

CPBA[0:10] — Channel x Parameter Base Address

The value of this field times 8 specifies the SDM parameter base host (byte) address for channel x (2-parameter granularity; see Section 21.3.2.4, “SDM Organization). As seen by the Host, the channel parameter base (byte) address is:

without parameter sign extension: ETPU_Base + 0x8000 + CPBA*8

with parameter sign extension: ETPU_Base + 0xC000 + CPBA*8

21.2.11.2 ETPUCxSCR - eTPU Channel x Status Control Register

ETPUCxSCR gathers the interrupt status bits of the channel, and also the Function Mode definition (read-write). Bits CIS, CIOS and DTRS for each channel can be also accessed from ETPUCISR, ETPUCIOSR and ETPUCDTRSR registers respectively (see Section 21.2.10, "Global Channel Registers). Host must write 1 to clear a status bit.

Channel_Register_Base + 0x4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CIS	CIOS	0	0	0	0	0	0	DTRS	DTROS	0	0	0	0	0	0
W	CIC	CIOC	▨	▨	▨	▨	▨	▨	DTRC	DTROC	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	IPS	OPS	OBE	0	0	0	0	0	0	0	0	0	0	0	FM	
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-40. ETPUCxSCR Register

CIS — Channel Interrupt Status

1 = channel has a pending interrupt to the Host CPU.
0 = channel has no pending interrupt to the Host CPU.

CIC — Channel Interrupt Clear

1 = clear interrupt status bit.
0 = keep interrupt status bit unaltered.

These bits are mirrored in ETPUCISR - see Section 21.2.10.1, "ETPUCISR - eTPU Channel Interrupt Status Register). See also the eTPU Reference Manual for details.

CIOS — Channel Interrupt Overflow Status

1 = interrupt overflow asserted for this channel
0 = interrupt overflow negated for this channel

CIOC — Channel Interrupt Overflow Clear

1 = clear status bit.
0 = keep status bit unaltered.

These bits are mirrored in ETPUCIOSR - see Section 21.2.10.3, "ETPUCIOSR - eTPU Channel Interrupt Overflow Status Register). See also Section 21.3.2.2.2, "Interrupt and Data Transfer Request Overflow.

DTRS — Data Transfer Request Status

1 = Channel has a pending data transfer request.
0 = Channel has no pending data transfer request.

DTRC — Data Transfer Request Clear

1 = clear status bit.
0 = keep status bit unaltered

These bits are mirrored in ETPUCISR - see Section 21.2.10.2, “ETPUCDTRSR - eTPU Channel Data Transfer Request Status Register). See also the eTPU Reference Manual for details.

DTROS — Data Transfer Request Overflow Status

1 = data transfer request overflow asserted for this channel
0 = data transfer request overflow negated for this channel

DTROC — Data Transfer Request Overflow Clear

1 = clear status bit.
0 = keep status bit unaltered.

These bits are mirrored in ETPUCDTROSR - see Section 21.2.10.4, “ETPUCDTROSR - eTPU Channel Data Transfer Request Overflow Status Register). See also Section 21.3.2.2.2, “Interrupt and Data Transfer Request Overflow.

IPS — Channel Input Pin State

This bit shows the current value of the filtered channel input signal state

OPS — Channel Output Pin State

This bit shows the current value driven in the channel output signal, including the effect of the external output disable feature (see Section 21.2.2.4, “eTPU Channel Output Disable Signals). If the channel input and output signals are connected to the same pad, OPS reflects the value driven to the pad (if OBE=1). This is not necessarily the actual pad value, which drives the value in the bit IPS.

OBE — Output Buffer Enable

This bit shows the state of the channel output buffer enable signal, controlled by microcode.

FM[0:1] — Channel Function Mode[1]

Each function uses this field for specific configuration. These bits can be tested by microengine code (see the eTPU Reference Manual for details).

---

[1] These bits are equivalent to the TPU/TPU2/TPU3 Host Sequence (HSQ) bits.

21.2.11.3 ETPUCxHSRR - eTPU Channel x Host Service Request Register

ETPUCxHSRR is used by the Host to issue service requests to the channel.

Channel_Register_Base + 0x8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	HSR		
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨			
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 21-41. ETPUCxHSRR Register

HSR[0:2] — Host Service Request

This field is used by the Host CPU to request service to the channel (see Section 21.3.2.5, "Host Service Requests)

HSR = 000: no Host Service Request pending

HSR > 000: function-dependent Host Service Request pending.

HSR value turns to 000 automatically at the end of microengine service for that channel, but only if the thread started due to an HSR. Host should write HSR>0 only when HSR=0. Writing HSR=000 withdraws a pending request if scheduler did not begin to resolve the Entry Point yet, but it does not abort the service thread from that point on. For more details, see the eTPU Reference Manual and Section 21.3.2.5, "Host Service Requests.

21.3 Functional Description

21.3.1 Watchdog

Each engine has a watchdog mechanism to prevent a thread or a sequence of threads from running too long, impacting the latency of the other channel services. The watchdog is configured through the register ETPUWDTR (see Section 21.2.7.1, "ETPUWDTR - eTPU Watchdog Timer Register"). When the watchdog is enabled, an internal counter increments on each microcycle when a thread is executing. If the count is greater than the value specified in the ETPUWDTR field WDCNT and a thread is still executing, the watchdog:

1. forces an END of the thread
2. sets the WDS status bit of the serviced channel in the ETPUWDSR register (see Section 21.2.10.7, "ETPUWDSR - eTPU Watchdog Status Register"). The channel is disabled, not initiating any thread until its WDS bit is cleared (its CPR field in ETPUCxCR is not changed, however).
3. issues a Global Exception and sets the ETPUMCR bit WDTO (see Section 21.2.5.1, "ETPUMCR - eTPU Module Configuration Register").

The watchdog can be configured in one of the following modes, defining how the internal watchdog count is reset:

- **Thread Length Mode**: the watchdog count is reset at the end of each thread.
- **Busy Length Mode**: the watchdog count is reset when the microengine goes idle. A sequence of threads, one right after another, keeps the count running. The counter is also reinitialized when a thread is forced to end, so that a new count begins if another TST initiates at the following microcycle.

The following applies to the watchdog mechanism:

- microcycles during TST and SDM access wait-states (on TST or instruction execution) are counted.
- if the watchdog count equals WDCNT in the last microinstruction (with SDM wait-states or not) of a thread servicing a channel, its WDS bit is not set.
- if the watchdog count expires (gets greater than WDCNT) during the TST, the thread is forced end on its first instruction.
- the watchdog count does not wrap, so that a thread (in thread length mode) or a thread sequence (in busy length mode) that lasts for more than the maximum value of WDCNT does get a forced end.

NOTE

Watchdog must not be enabled when the microengine enters halt mode. The counter does not run when the engine is stopped, and resets when the watchdog is disabled.

21.3.2 Host Interface

21.3.2.1 System Configuration

System Configuration Registers are described in Section 21.2.5, “System Configuration Registers”. Detailed explanation on the configured functionalities is found throughout Section 21.3, “Functional Description.

21.3.2.2 Interrupts and Data Transfer Requests

21.3.2.2.1 Interrupt Types and Sources

Each one of the eTPU channels can be a source of two requests: **Channel Interrupt** request and **Data Transfer Request**. Channel Interrupts are targeted to a Host CPU. Data Transfer Requests may be targeted to a data transfer module (e.g., a DMA controller). Interrupt and Data Transfer registers are used by the Host to enable interrupts and data transfer requests, indicate their status and service them. Interrupt and Data Transfer requests have the same sets of registers and external signals, and are handled in the same way. They differ only by the fact that Data Transfer Requests are also cleared by the assertion of respective DMA completion acknowledge line. Data Transfer Requests can be used as another source for Host interrupts at MCU integration if not used with a DMA.

NOTE

Interrupt and Data Transfer requests can be cleared even when Engines are in Module Disable Mode, through the Global Channel Registers, and also DMA completion for Data Transfer requests.

Channel Interrupts and Data Transfer Requests can only be issued by eTPU microcode, through one of the Channel Control instruction fields (see the eTPU Reference Manual for details).

Both Channel Interrupt and Data Transfer requests can be individually enabled for each channel.

eTPU Interrupt and Data Transfer Registers are mirrored in two organizations: grouped by Channel and grouped by type (interrupt status, interrupt enable, data transfer status, data transfer enable). This allows either “channel-oriented” or “bundled channel” Host interrupt service schemes, or a combination of them. For a detailed description, refer to Section 21.2.9, “Channel Registers Layout” and Section 21.2.10, “Global Channel Registers.”

eTPU can also assert a **Global Exception** interrupt indicating a global illegal state. There are five possible sources for a Global Exception:

- Execution of an illegal instruction by the microengine (see the eTPU Reference Manual for details). This Global Exception source is flagged by the bits ILF1 and ILF2 in register ETPUMCR.
- An SCM signature mismatch detected by the Multiple Input Signature Calculator - MISC. See Section 21.3.6.1, “SCM Test - Multiple Input Signature Calculator. This source is flagged by the bit SCMMISF in register ETPUMCR.
- Microcode request, through microinstruction field CIRC (see the eTPU Reference Manual for details). This Global Exception source is flagged by bits MGE1(Engine A/C) and MGE2(Engine B) in register ETPUMCR. The cause of this illegal state is application-dependent. The microcode may write an error code into the SDM to indicate the cause of the exception, for instance.
- An SDM or SCM non-correctable error on a microengine or host access, flagged by the ETPUMCR bits SDMERR and SCMERR, respectively (see Section 21.3.6.2, “Memory Error Support”).

Global Exceptions cannot be directly disabled within eTPU, except by disabling its sources (Memory Error, MISC and microcode), and it is cleared by writing 1 to the GEC bit in ETPUMCR. Clearing Global Exception clears all Global Exception source status bits (ILF1, ILF2, SCMMISF, MGE1, MGE2, SCMERR, SDMERR). If GEC is written 1 at the same time any of the sources issues a Global Exception, both the interrupt and the status bit of that source remains asserted. The assertion of Global Exception by one of the sources above does not prevent the others from asserting it too, so any number of them, in any combination, can be flagged.

NOTE

There can be a race between the clear of a Global Exception and occurrence of a new set condition, such that the set happens just before the clear and cannot be sensed by the Host. Therefore, Global Exception cannot be used as a normal interrupt source: it should only be used for emergency procedures.

21.3.2.2.2 Interrupt and Data Transfer Request Overflow

If a Channel Interrupt was issued, its status bit is still set, and microcode issues another Channel Interrupt, the Interrupt Overflow status bit is set for that channel. Interrupt Overflow status can be checked by the Host in Channel Status register ETPUCxSCR bit CIOS (Section 21.2.11.2, "ETPUCxSCR - eTPU Channel x Status Control Register), mirrored in register ETPUCIOSR (Section 21.2.10.3, "ETPUCIOSR - eTPU Channel Interrupt Overflow Status Register). Interrupt Overflow status is not cleared automatically when Interrupt Status is cleared. The same mechanism and respective registers (ETPUCDTROSR) are available for Data Transfer Requests.

If interrupt is set and cleared at the same time, set prevails and overflow is not altered (keeps the same state as it was before, asserted or not).

Global Exception has no overflow status.

21.3.2.3 Parameter Access

21.3.2.3.1 Parameter Access Widths

From the Host side the SDM address space is mapped in bytes, and each 32-bit parameter occupies 4 contiguous, aligned bytes. The Host can read/write the SDM by 8-, 16-, or 32-bit accesses in aligned addresses.

In 32-bit access, Host can access all 32 bits or only the lower 24 bits with an automatic sign extension (see Section 21.3.2.3.4, "Parameter Sign Extension Area).

21.3.2.3.2 Parameter Addresses and Endianness

To access parameter number *xxx*, eTPU Microengine(s) would select address xxx. The Host would add *(xxx*4)* to the SDM base address to access the same parameter. For example, parameter 0x101 is seen by the Host in *(SDM base address +0x404)*. An example of SDM memory map is shown in Figure 21-42. The Host can access the SDM with a 32-bit-wide bus cycle to a four-byte aligned address, 16-bit-wide bus cycle to a two-byte aligned address, or 8-bit wide bus cycle to any byte address.

The address of the 24-bit parameters and the most significant byte depends on the endianness of the MCU.

21.3.2.3.3 Parameter Concurrency

Host accesses to parameters may occur in parallel with eTPU Microengine accesses. Readings taken from a group of parameters while they are being simultaneously updated may lack coherency. eTPU provides mechanisms to ensure parameter coherency in accesses from both Host side and Microengine side, including the use of a coherent dual-parameter transfer mechanism, described in detail on Section 21.3.4, "Parameter Sharing and Coherency."

21.3.2.3.4 Parameter Sign Extension Area

The SDM address space to the Host is mirrored in a Parameter Sign Extension - PSE - area (see Section 21.2.4, "Memory Map"). Accesses from the Host to the PSE area differ from accesses to the standard SDM address space as follows:

- **Writes**: the most significant byte of the parameters is not written, and the SDM retains the old byte value, regardless of the Host access size.
- **Reads**: the most significant bit of the 24-bit parameter (that is, the msbit of the second most significant 32-bit parameter byte) is repeated in the 8 most significant bits of the read value on all 32-bit reads and most significant 16- and 8-bit reads.

The same parameters written in the standard SDM address space are read from the PSE area with the same offsets, and vice-versa.

This feature reliefs the Host from extending the signal of 24-bit eTPU parameters before calculations, and from read-modify-write accesses to modify 24-bit parameters at the SDM.

21.3.2.4 SDM Organization

The SDM internal partition for channel allocation is dynamic and programmed in the Channel Registers (see Section 21.2.11.2, “ETPUCxSCR - eTPU Channel x Status Control Register).

The Host application is responsible for allocating a different parameter base address to each channel during the initial eTPU configuration, and to allocate enough parameters for the selected Function, with no unintentional overlapping between parameters of different functions.

Besides channel parameters, global areas may have to be allocated for parameters that are shared by more than one channel, in one or both Engines. Also, temporary parameter areas should be reserved to be used by the coherent parameter transfer mechanisms described in Section 21.3.4, “Parameter Sharing and Coherency, if necessary.

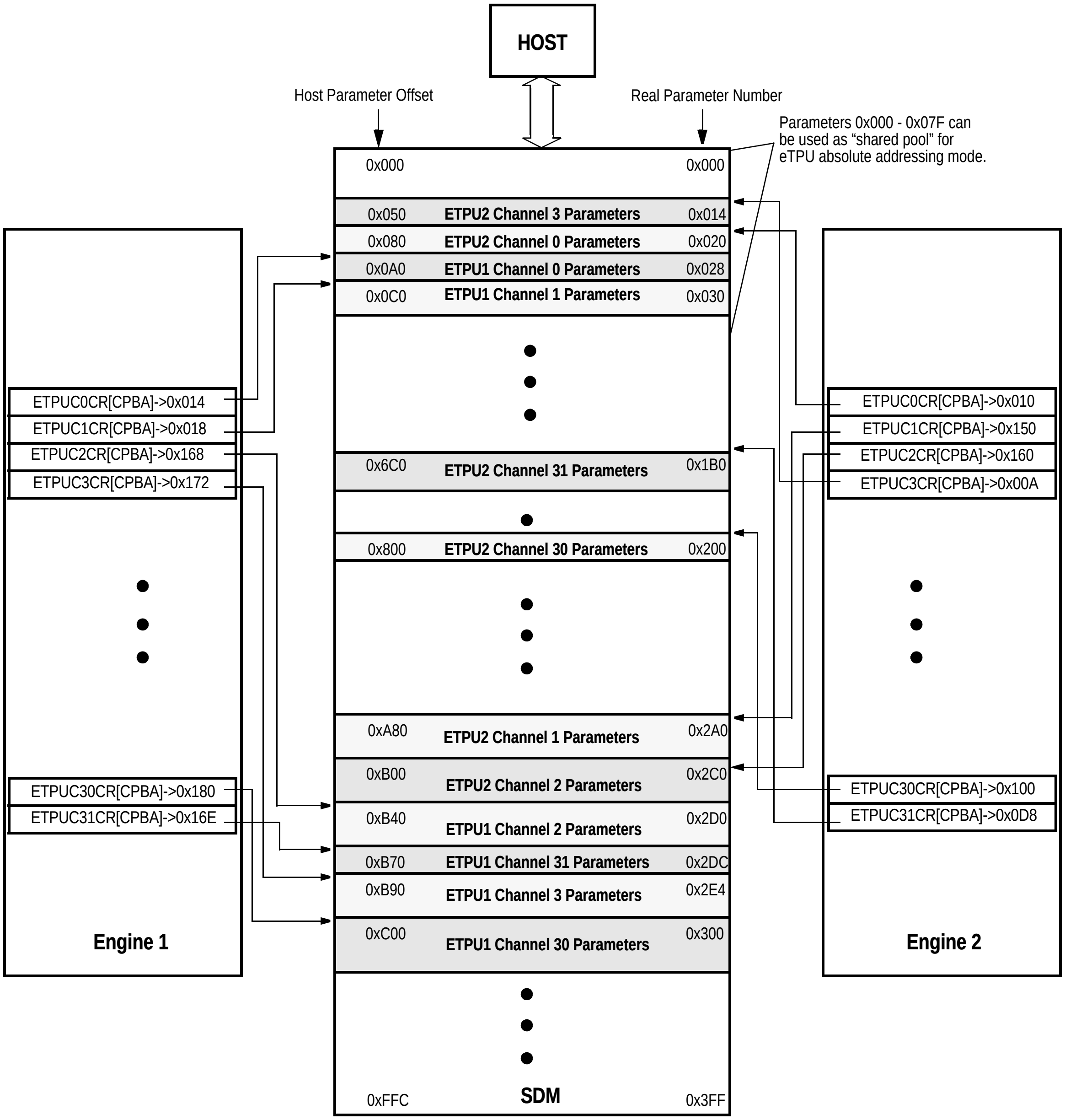


Figure 21-42. SDM Organization Example (4 KB)

Single-engine eTPU or dual eTPU system may require less parameters than the maximum number provided by the SDM. Since the SDM partition is fully dynamic, there is no limitation of fixed channel addresses, and the reduced array can be fully utilized.

21.3.2.5 Host Service Requests

Host CPU can request immediate service from a channel by writing a non-zero value to the Host Service Request register field - HSR (see Section 21.2.11.3, "ETPUCxHSRR - eTPU Channel x Host Service Request Register). There is one HSR field for each channel, so that writing to it generates a Service Request to the respective channel only. A zero value in HSR means no Host Service Request is pending for the channel.

HSR value turns to 000 automatically at the end of microengine service for that channel, but only if the thread started due to an HSR.

The meaning of a non-zero HSR value depends on the Function assigned for the channel. These bits are part of the conditions which select the Function entry point, and cannot be tested by microcode. For more details, refer to the eTPU Reference Manual.

If Host writes HSR=000 when a thread for the same channel is already running, the thread runs until the end and is not aborted. If Host writes HSR>000 when an HSR thread for the same channel is already running, HSR value resets at the end of the thread, and no new HSR will be pending. If HSR is written before its value is resolved by the scheduler during TST, the entry point will obey the new HSR value, and if this new value is 000, no service thread is executed for the HSR.

The scheduling of HSRs is completely asynchronous with Host accesses, and there is no race-free manner to change an HSR value before service thread execution, so generally the safe way is: write HSR>0 only when HSR=0. Error recovery or emergency host procedures may require one to the safely abort service and reset channel state when an HSR is already pending or executing. In these cases, the procedure below should be followed:

1. Disable the channel, writing CPR=00 in register ETPUCxCR. That will prevent any pending HSR to be serviced.
2. Check if the channel is currently being serviced, reading its service status bit in register ETPUCSSR. If it is, wait for the time necessary to finish the service pending, or check again until HSR == 0, or channel service bit in ETPUCSSR is cleared.
3. Write HSR with the error recover value. This value should, possibly combined with other host-defined flags in SDM or FM bits, initiate a channel reset or error recovery procedure.
4. Re-enable the channel, writing CPR value > 0 in register ETPUCxCR.

21.3.2.6 SCM access

Only Host can access SCM as data. Depending on the specific device, SCM may be implemented as a RAM or ROM. This determines Host accesses to the SCM as shown below.

21.3.2.6.1 SCM RAM Implementations

When SCM is implemented as RAM, the Host may read or write to SCM by setting ETPUMCR bit VIS=1. If VIS=0 and Host tries to access SCM space, a bus error is issued, writes are ineffective and read data is meaningless. Both Engines must be stopped or halted to set VIS=1.

NOTE

It is necessary to turn VIS bit on to set software breakpoints (see the eTPU Reference Manual for details).

21.3.2.6.2 SCM Low Power

SCM turns off its internal clocks when both Engines are stopped (ETPUECR bit STF asserted), VIS=0 at ETPUMCR, and MISC is not enabled (SCMMISEN=0). The SCM clocks are automatically turned on if either one of the STF bits is negated or VIS turns to 1, or SCMMISEN turns to 1.

SCM clocks are not turned off if any of the Engines is not stopped, even if they are both halted.

The conditions for SCM Clocks and MISC activation are summarized in Table 21-21.

Table 21-21. SCM Clocks and MISC activation

ETPUECR_1 STF	ETPUECR_2 STF	ETPUMCR VIS	ETPUMCR SCMMISEN	SCM Clocks	MISC
0	x	0[1]	1	On	On
0	x	0[1]	0	On	off
x	0	0[1]	1	On	On
x	0	0[1]	0	On	off
1	1	0	0	off	off
1	1	0	1	On	On
1[2]	1[2]	1	0	On	off
1[2]	1[2]	1	1	On	off[3]
0	0	x	0	On	off
0	0	x	1	On	On

[1] VIS cannot be written 1 if ETPUECR_1 bit STF=0 or ETPUECR_2 bit STF=0, and both HLTF bits are 0.

[2] If VIS=1, neither MDIS can be written 0 nor the Engine leave Stop Mode, regardless of device stop reques.

[3] MISC resets and stays so when VIS=1, restarting automatically when VIS goes 0 if SCMMISEN=1.

21.3.2.6.3 SCM Off-range Data

When read accesses are made, either by the Host or by a microengine, to addresses above the limit corresponding to the SCMSIZE value in ETPUMCR, the value read comes from the register ETPUSCMOFFDATAR. The Host can program the register at initialization with an opcode value with operations that try to protect or recover the system from runaway code, for instance: terminate the thread, clear channel flags, disable match and transition service requests, issue an interrupt, jump to an error recovery procedure[1]. Writes to unimplemented addresses do not return error and can write on unspecified mirror addresses, so they should be avoided.

---

1. only part of these suggested operations can be parallelized in a single instruction, see the eTPU Reference Manual for details.

21.3.2.7 Bus Error Conditions

Follows a summary of the possible causes of bus errors on host accesses to the eTPU memory space:

- read or write access to the SCM with ETPUMCR bit VIS=0.
- read or write access to the channel registers with ETPUECR bits MDIS=1, or bit STF=1.
- write access to the Time Base Registers with ETPUECR bits MDIS=1, or bit STF=1.
- non-correctable errors in the SCM on reads of any size or writes less than 32-bits wide, when ETPUMECR bit CEDD=0.
- non-correctable errors in the SDM on reads, when ETPUMECR bit DEDD=0.

21.3.3 Scheduler

Every Function is composed of one or more Threads. A Thread consists of a group of instructions that, once begins execution, cannot be interrupted by host or channel events. Each active channel intents to be serviced, being granted time for Thread execution. Since one microengine handles several channels operating concurrently, the Function threads must be executed serially.

The task of the Scheduler is to recognize and prioritize the channels needing service and to grant execution time to each channel. The time given to an individual Thread for execution or service is called a **Time Slot**. The duration of a time slot is determined by the number of instructions executed in the Thread plus SDM wait-states received, and varies in length.

At any time, an arbitrary number of channels can require service. To request service, channel logic, eTPU microcode or Host application notifies the Scheduler by issuing a Service Request.

21.3.3.1 Channel Enabling and Priority Assignment

Every channel is assigned one of three priority levels - high, middle, or low - by the Host CPU, through the Channel Configuration Register field CPR (see Section 21.2.11.1, “ETPUCxCR - eTPU Channel x Configuration Register”). These registers are also used to disable the channel, which is equivalent to assigning it a “null” priority. In this case, the Scheduler does not grant any of its Service Requests.

It is possible to change the channel priority level or disable it dynamically. If the Host disables a channel when it is currently being serviced, channel service thread will complete. This means that it is possible for the output level of a channel signal to change, or a Host interrupt occur, even after its priority register was written to “null”. For instance, if an output transition is scheduled, the transition will occur even after the channel is disabled.

Service requests previously pending or that occur while a channel is disabled remain asserted while the channel is disabled, and are serviced if the channel is enabled again, in due time determined by the priority scheme and concurrent requests from other channels. Channels are disabled after reset, and it is recommended to configure a Host Service Request for initialization of a channel before that channel is enabled to active priority (see Section 21.4, “Initialization/Application Information”).

21.3.3.2 Channel Priority Schemes

The Scheduler holds a Service Grant register with one bit for each channel. Once the Scheduler grants a time slot to channel, the Service Grant bit for that channel is asserted in the Service Grant register. When the Service Grant bit of a channel is set, the channel may request new service but is not serviced again before its Service Grant bit is cleared.

When all channels in a same priority level are serviced, their Service Grant bits are cleared at the end of the thread, one eTPU clock before the next serviced channel is calculated, according to the scheme below[1]:

- Clear all grant bits of priority High if all channels of that priority that are requesting have their grant bits in 1.
- Clear all grant bits of priority Medium if all channels of that priority that are requesting have their grant bits in 1.
- Clear all grant bits of priority Low if all channels of that priority that are requesting have their grant bits in 1.
- Clear all grant bits of disabled channels.

This scheme assures that no channel is left with its grant bit forever asserted (preventing it from being serviced again), even if the channel priorities are reassigned during the execution.

Priority level is determined based on the maximum latency desired for each channel. A channel having a Function that requires the most frequent or more immediate service should be allocated a high priority level.

The eTPU employs a **primary** and a **secondary priority scheme**. These two schemes ensure frequent servicing of high-demand Functions and ensure a minimum time allocation to all channels requesting service, regardless of their priority level. The primary scheme prioritizes requesting channels that have different priority levels; the secondary scheme prioritizes requesting channels that have the same priority level.

Initially, a channel requests service and is granted a time slot by the Scheduler: Service Grant bit is asserted. If only high-level channels constantly receive service first because of their priority level, middle- and low-level channels would only be serviced by default, i.e., if no high-level channels request service. To ensure that each priority level receives an opportunity for servicing, every time slot has a fixed priority level that the Scheduler honors first. Divided into sets of seven, time slots are numbered from one to seven. Figure 21-43 illustrates the numbered time slots in sets of seven (fields A and B) and identifies their assigned default priority level. The high level has more time slots than the middle and low levels. Out of every seven time slots available, four are assigned to honor high-level channels first, two are assigned to honor middle-level channels first, and one is assigned to honor low-level channels first. Only one request (in each Engine) is serviced per time slot. When no channel requests service and the microengine is idle the priority scheme is initialized to time slot one, to prevent priority inversion on the next request[2].

---

[1] grant bits are also cleared in the next clock, when the service channel is chosen, or when the microengine is idle, using the same scheme.

[2] Priority inversion would occur in the following situation: no channel is requesting service, and the current time slot is primarily assigned to a low-priority channel. If the Scheduler was not reset to time slot one and two channels requested service at the same time, one with high priority and the other with low priority, the channel to be serviced would be the low-priority channel.

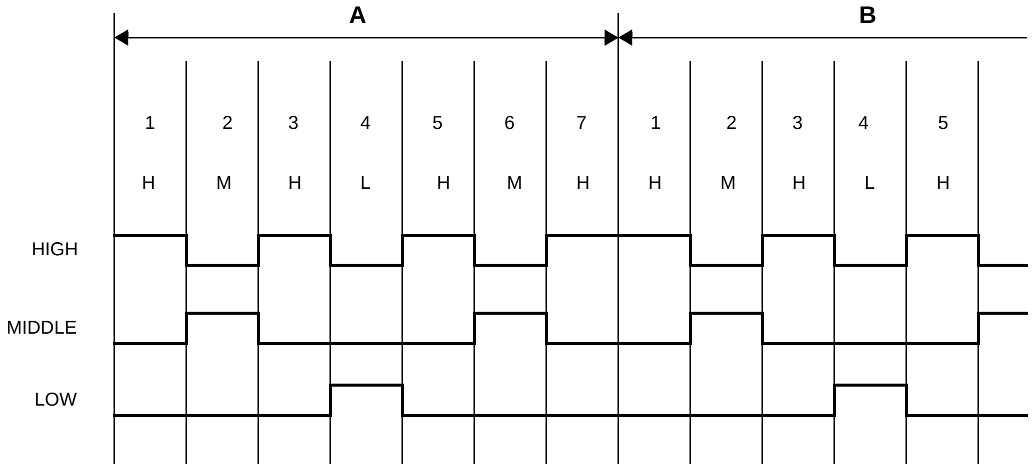


Figure 21-43. Time Slot Priority levels

21.3.3.2.1 Primary Scheme - Priority Among Channels on Different Levels

Although time slot priority assignment is fixed, the servicing priority is not. The primary scheme acknowledges the priority level assigned to a time slot, granting service first to a channel having the same priority. In Figure 21-43, time slot one has a high-level assignment; therefore, a high-level channel requesting service is recognized first. However, if no high-level channel requests service, the Scheduler recognizes a requesting middle-level channel. If this level has no request, the Scheduler continues to the low-level. If no requests occur, the Scheduler truncates the seven state cycle and starts a new cycle at time slot one, waiting for the first request. Granting service to a different-level channel is called priority passing. The order of passing always gives the highest priority to the assigned level, and the second priority to the higher of the remaining requesting priority levels as shown in Table 21-22.

Table 21-22. Priority Passing

Assigned Priority Level	Next Priority Level	Next Priority Level
High	→ Middle	→ Low
Middle	→ High	→ Low
Low	→ High	→ Middle

When priority is passed to another level, that level is serviced and the fixed-priority-level sequence is resumed with the next time slot.

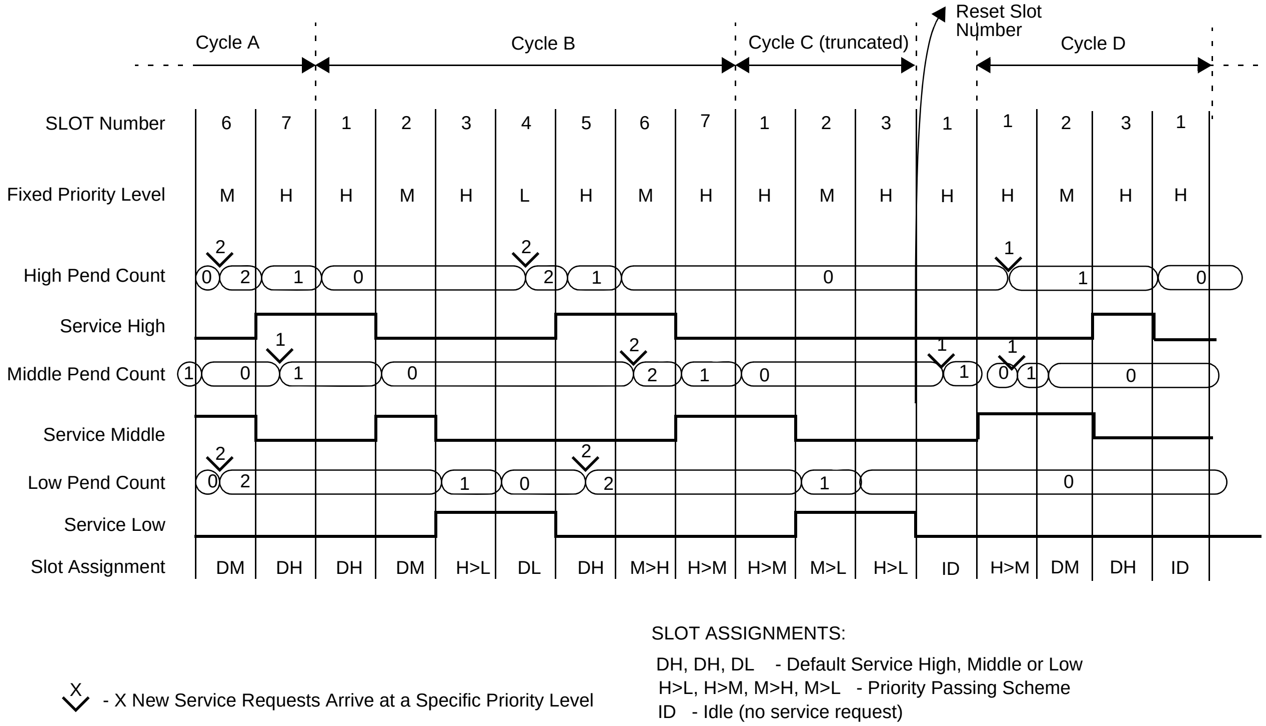


Figure 21-44. Priority Passing Example

Examples of priority passing are shown in Figure 21-44. Each cycle contains seven time slots (or less if no service request exist). In cycle B, no high-level or middle-level service requests are present before time slot three which is assigned by default to high-level priority. Thus, time slot three is passed to the low level. In cycle B there are also no middle-level service requests before time slot six, so it passes the priority to a requesting high-level channel. During time slot six no more high level requests are left, but two new middle-level requests arrive, and there are also three low level pending service requests. Thus, time slot seven of cycle B and time slot one of cycle C are passed to the middle-level which is the next priority level after high. Time slots two and three of cycle C are passed to the low level which contains the three remaining channel service requests. At time slot three of cycle C the last low level request is serviced, and the Scheduler passes to idle state. At this point the cycle C is truncated and the Scheduler passes to time slot one of cycle D.

21.3.3.2.2 Priority Passing Disabling

The priority passing scheme allows a case where a high priority channel looses to a lower priority one right after another lower priority has been serviced, exemplified in the Cycle D on Figure 21-44. A middle priority channel wins time slot 1 due to priority passing from high to middle. While it is being serviced, two new service requests arrive, one high and one middle priority. The high priority request looses to the middle one on next time slot 2 by default priority assignment.

This priority inversion can be avoided by setting the ETPUECR register bit SPPDIS (see Section 21.2.5.5, "ETPUECR - eTPU Engine Configuration Register"), which disables the priority passing mechanism. When priority passing is disabled, at the end of the thread the slot number is incremented until a time slot that matches the priority of one of the requesting channel(s). The time slot advance takes no extra clocks. If no channel requests service, the time slot counter stays at time slot 1. The priority selection scheme with disabled priority passing is summarized in Table 21-23.

Table 21-23. Priority Passing Disabling

At the end of time slot	servicing priority	if any request of priority	service it on time slot	else if any request of priority	service it on time slot	else if any request of priority	service it on time slot
1	High	Medium	2	High	3	Low	4
2	Medium	High	3	Low	4	Medium	6
3	High	Low	4	High	5	Medium	6
4	Low	High	5	Medium	6	Low	4
5	High	Medium	6	High	7	Low	4
6	Medium	High	7	Medium	2	Low	4
7	High	High	1	Medium	2	Low	4

An example of the priority passing disabling scheme is illustrated in Figure 21-45. The sequence of service requests is the same as in the example of Figure 21-44, and although the time slot incrementing differs, the priorities granted are the same for cycle B. Cycle C has one of the low priority channels serviced before the second middle one. Cycle D, however, no longer has the priority inversion.

In cycle B, after the time slot 2 only a low priority request remains, so the time slot count advances directly to 4, which has a low priority assigned. Time slot keeps on 4 for the next service, as only a low priority request remains also, and only time slot 4 is assigned to low. Two high priority services contend for the next time slot 5 (assigned to High). The second high priority channel is serviced on the next time slot, jumped to 7 because there is no middle request, ending cycle B. Cycle C starts with time slot 2, as there are no high priority requests and two middle and two low ones. After the first middle service, time slot count skips 3 assigned to high (no high requests), and services a low priority channel on time slot 4. It follows the same scheme until there are no other requests and cycle C is truncated, resetting the time slot counter to 1.

Cycle D begins with a middle request, jumping to time slot 2. During this service two requests arrive, one high and one middle. Unlike what happened with priority passing, the next serviced is the high priority channel, as the time slot increments to 3. The second middle priority channel request in cycle D is finally serviced next, on time slot 6.

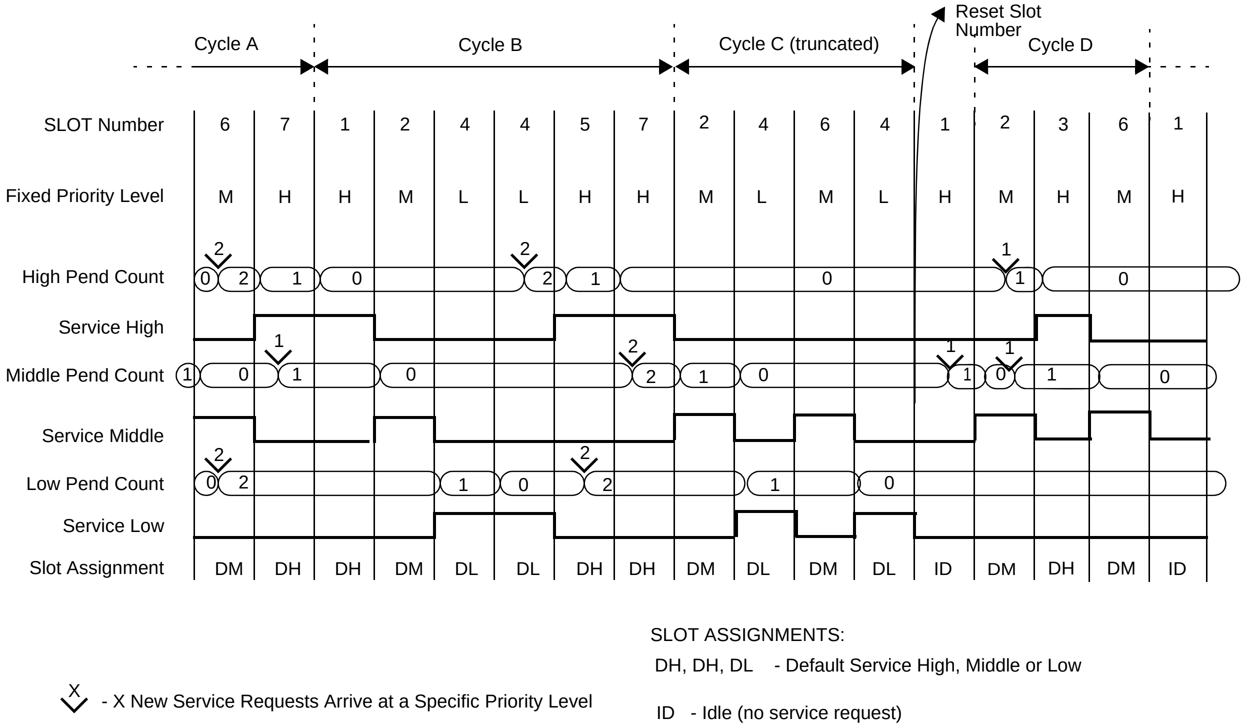


Figure 21-45. Priority Passing Disabling Example

21.3.3.2.3 Secondary Scheme - Priority Among Channels on the Same Level

Because channels can randomly request service, channels having the same priority level will inevitably request service simultaneously. A secondary scheme prioritizes these requests. The Scheduler services channels on each of the three priority levels, beginning with the lowest numbered channel on that level.

21.3.3.2.4 Priority Scheme Example

The overall priority scheme simultaneously incorporates both primary and secondary schemes. Combining both schemes in the following example conveys their correlation.

1. One high-priority and one low priority channels request service, while the Scheduler is in time slot one. Having its service request bit asserted, a single high-level channel is granted the time slot, which has high-level priority (primary scheme) and its service grant bit is asserted. At the end of the thread, the service grant bit is negated (no more requests of high priority level channels).
2. The Scheduler proceeds to time slot two, which has middle-level priority; however, no middle-level channel is requesting service. Priority is passed to the high level, but no high-level channel is requesting service; therefore, priority is passed again, and service is granted to the single requesting low-level channel. Once serviced, this channel's grant bit is negated (no more low-level requests).
3. The Scheduler resumes with the fixed-priority sequence on time slot three; however, no channels are requesting service. The Scheduler returns to time slot one, waiting for requests.
4. Two high-level and two middle-level channels simultaneously request service. Being in time slot one which is assigned high priority, the Scheduler finds the lowest numbered high-level channel (secondary scheme) and selects it for service. This channel's service grant bit is asserted.
5. The Scheduler continues to time slot two, which has middle priority (primary scheme), and allocates the slot to the lowest numbered middle-level channel requesting service (secondary scheme). The Scheduler notes the still unserviced middle-level channel and proceeds to time slot three.
6. Time slot three is allocated for high priority. The slot is allocated to the remaining unserviced high-priority channel, and the channel's service grant bit is asserted. The Scheduler checks again at the end of the thread. All service grant bits of high-level requested channels are asserted; therefore, all high-priority channels that requested have been allocated execution time. Under this condition, all service grant bits of the high-level serviced channels are negated. The Scheduler proceeds to time slot four.
7. Time slot four is allocated for low-priority channel; however, no low-level channel is requesting service. Priority is passed to the high level, but no high-level channel is requesting service; therefore, priority is passed again, and service is granted to the remaining middle-level channel which requests service. This channel's service grant bit is asserted. The Scheduler checks again at the end of the thread. All grant bits of middle-level requested channels are asserted; therefore, all middle-priority channels have been allocated execution time. Under this condition, all service grant bits of the middle-level serviced channels are negated. The Scheduler proceeds to time slot five. Meanwhile a low priority channel requests service.
8. Time slot five is allocated for high-priority channels, but there are no more requests from high-priority or middle priority channels. The single low-level channel which required service is granted time slot five. Once serviced, the channel's service grant bit is asserted. Next, the service grant bit is negated (no more requests of low priority level channels).
9. The Scheduler resumes with the fixed-priority sequence on time slot six; however, no channels are requesting service. The Scheduler returns to time slot one and waits for requests.

21.3.3.3 Time Slot Latency

Latency is the amount of time between a service request and the beginning of service on that channel. The following factors affect latency:

- Number of active channels
- Number of channels on a priority level
- Number of available time slots on a priority level
- Number of microcycles required to execute a thread of a Function
- Number of SDM accesses during execution of a Function thread
- eTPU clock frequency.

Each time slot may require a different number of microcycles, depending on the thread of a Function to be executed. This variation is shown in Figure 21-46.

For more details on latency evaluation, see Section 21.4.2, “Estimating Worst Case Latency.”

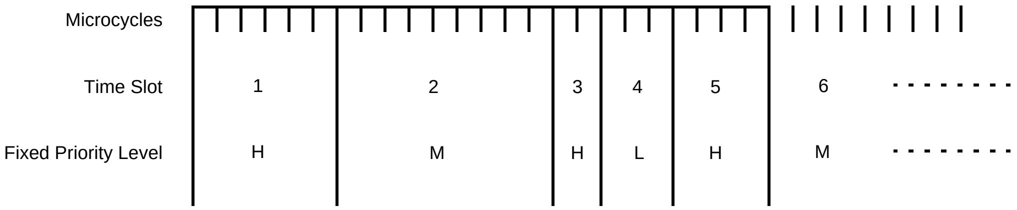


Figure 21-46. Time-Slot Variation

21.3.4 Parameter Sharing and Coherency

SDM can be concurrently accessed by Host and Microengines (two in a Dual eTPU Engine system). In general, there is no guaranteed order by which a group of parameters is accessed, which may lead to a lack of internal consistency if two or more related parameters are read when only part of them is updated.

eTPU provides mechanisms to guarantee parameter coherency. The most generic mechanisms for Host-eTPU coherency, suitable for any number of parameters, are:

- the use of Transfer Service Thread mechanism.
- the mailbox (or “software semaphore”) mechanism.

These mechanisms, described in Section 21.4.1, “Multiple Parameter Coherency Methods,” use microcode to transfer parameters from temporary buffers in SDM to their definitive locations (or vice-versa). These methods have the disadvantage of wasting processing and code memory resources.

eTPU also provides a **Coherent Dual-parameter Controller - CDC** - mechanism. It is used by Host to coherently transfer pairs of parameters from/to a parameter buffer located on SDM to/from the locations on SDM where parameters are accessed directly by the channels. Coherency is guaranteed by SDM access arbitration. Although limited to two parameters only, it has lower latency and wastes no microengine resources[1]. CDC usage is described in Section 21.3.4.2, “Coherent Dual-parameter Controller - CDC.”

For parameters shared by both Engines, eTPU provides **hardware semaphores**. Coherency is assured given the semaphores are used to prevent concurrent access to the changing parameters. Microengine can request semaphores using specific microinstructions.

Neither Host nor CDC have access to the hardware semaphores, but they can be combined with microcode transfer mechanisms if Host must coherently access parameters which are also shared by both Engines.

21.3.4.1 Host Side Atomic Access

Host side atomic accesses can be achieved by either of following ways:

- for one parameter, the SDM should be accessed by 32-bit-wide data transfers to ensure coherency
- for two parameters only, using the Coherent Dual Parameter Controller.
- indirectly, for any number of parameters, by requesting microcode to coherently access SDM in its behalf. The host side atomicity problem becomes, then, a microengine side atomicity problem. Some methods that use this approach to achieve coherency are described in Section 21.4.1, "Multiple Parameter Coherency Methods."

21.3.4.2 Coherent Dual-parameter Controller - CDC

Dual-parameter coherency is supported by a Coherent Dual-parameter Controller hardware - CDC, which contends with microengine for SDM access. CDC atomically transfers, upon Host's command, two parameters from one area of the SDM to another. One area is a temporary (buffer) area, where the two parameters are directly read or written by the Host. This temporary area has to begin in an SDM address multiple of 2 words, and the two parameters must be sequential. The other area is the channel parameter area where the microcode normally accesses the parameters, usually with the channel relative address mode (see the eTPU Reference Manual for details). In this area, the parameters transferred by CDC don't have to be sequential. A transfer from the temporary area to the channel area, when the Host sends data to the channel, is called a **write transfer**. Inversely, in a **read transfer** the parameters are copied from the channel area to the temporary area (channel to Host).

Coherency is guaranteed by the SDM access contention rules implemented in the SDM arbiter. CDC transfers are coherent in respect to the two Engines, so the target parameters in the channel area may be shared by channels on them both. During CDC operation, the Host may suffer from 3 up to 11 eTPU clocks wait states[1], and the Microengine(s) may suffer up to 2 microcycle wait-states[2]. CDC accesses are atomic with respect to Microengine(s) accesses to the SDM. Even when neither engine is in TST, CDC may suffer up to 4 eTPU clock internal wait-states from SDM arbiter, meaning 9 slave wait-states to Host, so that it does not break atomic back-to-back accesses from microengine(s). CDC also cannot break TST preload accesses. Host can initiate CDC back-to-back transfers: there is no need of idle slave cycles between two transfers.

---

[1] a microengine access to the SDM in the moment CDC is performing the transfer may suffer a maximum of two wait-states.

[1] The maximum number of Host wait states on CDC occurs when both microengines overlap their TSTs, delayed 3 eTPU clocks from each other.

[2] One microcycle takes two eTPU clocks. Microengines get wait-states in multiples of microcycles, while Host and CDC wait-states are multiples of eTPU clocks.

NOTE

SDM memory error detection and correction does not affect the number of wait-states to host due to CDC (see Section 21.3.6.2, “Memory Error Support”).

CDC can be used to atomically fix SDM soft errors. For more details, see Section 21.3.6.2.3, “Error Fixing”.

21.3.4.2.1 CDC Programming

The Coherent Dual-parameter Controller Register (see Section 21.2.5.2, “ETPUCDCR - eTPU Coherent Dual-Parameter Controller Register) is used to configure and initiate CDC transfers between the temporary area and channel parameter area. Host asserts STS bit in order to start the data transfer. CDC then contends for the SDM and starts the transfer. When the data transfer is complete, STS returns to 0. Host receives wait-states for writing STS=1 while CDC contends for SDM and during the transfer. The write access ends when CDC finishes the transfer. Host receives wait-states during the CDC transfer. If Host writes ETPUCDCR with STS=0 or does not write the STS byte, the CDC transfer does not occur. CDC programming can be summarized as follows:

1. if it is a write transfer, i.e., from Host to channel, write the two parameters into temporary area.
2. write ETPUCDCR with STS=1 and the remaining CDC programming parameters: parameter width (32 or 24 bits, field PWIDTH), transfer direction (read or write, field WR), temporary parameter area base address (field PBBASE), and the absolute addresses of the parameters to be transferred (concatenation of the fields CTBASE and PARM0/1).
3. if it is a read transfer, i.e., from channel to host, read the two parameters from the temporary area into Host memory/registers.

21.3.4.3 SDM Arbitration

Up to four entities can access SDM:

- two Microengines (in a dual eTPU Engine system)
- the Coherent Dual-parameter Controller (CDC)
- the Host CPU (direct memory-mapped access)

The following rules specify the access priorities for contended access. They keep compatibility with the TPU3 dual parameter access atomicity, but only between the microengine and CDC (not Host accesses through slave bus).

1. Microengine accesses from the two eTPU Engines are interleaved between each other, but not with Host or CDC accesses;
2. The eTPU microengine(s) gives priority for SDM accesses to either the Host CPU or the CDC under any of the following conditions:
    a. the microengine has completed accessing the second parameter in a back-to-back SDM access[1].

---

[1] if microengine tries to access the SDM in the following microcycles, the third and fourth consecutive accesses are considered the first and second of a new back-to-back dual access.

  b. the SDM was not accessed during the last arbitration slot for the microengine and the host does not loose the access to the other engine in the current arbitration slot[1].
  c. CDC is transferring data, after its first (read) access. Note that the CDC can be in middle of a data transfer of another pair of parameters, unrelated to the ones that microengine tries to access.
3. The eTPU microengine takes priority for SDM accesses under either of the following conditions:
  a. the Host CPU or CDC has done a data transfer during the last access arbitration slot for the engine[1]. Also, the Host CPU does not hold a pending access against the other eTPU microengine.
  b. the microengine is arbitrating for the access of its second parameter in a back-to-back access[1]. All pairs of back-to-back parameter accesses are coherent with respect to Host and CDC (not to the other microengine).

The direction (read or write) of any individual access by Host or microengine is irrelevant to the arbitration. The use of Normal or PSE SDM area by the Host is also irrelevant to the arbitration.

The first parameter preloading in a TST is considered first access by the arbiter, regardless of any access made at the END microinstruction of the previous thread, i.e.: the last access of a thread and the first preload are never considered a back-to-back access. On the other hand, the TST preload accesses are considered back-to-back and are, therefore, atomic with respect to Host or CDC.

NOTE

> The Zero SDM operation (see the eTPU Reference Manual for details) is considered an SDM access for arbitration purposes both on writes and reads; the fact that read SDM data is discarded is irrelevant for arbitration.

21.3.4.4 Enhanced Digital Filter - EDF

The EDF eliminates passing of signal transitions which are caused by noise. Its purpose is to eliminate false transition service requests caused by noise pulses which are shorter than a programmed width.

The EDF has three modes of operations, selected by the CDFC field in the ETPUECR register (see Section 21.2.5.5, “ETPUECR - eTPU Engine Configuration Register). These modes offer selections of trade-off between noise immunity and signal latency. CDFC also allows the filter to be bypassed. Table 21-24 gives an example of minimum detected signal pulse and maximum filtered noise pulse in the three EDF operation modes. In Angle Mode, if AM=01, the EDF in channel 0 is replaced with the digital filter and synchronizer of the TCRCLK signal. In this mode, channel 0 works in combination with the Angle Counter logic, and their operation is fully synchronized.

Following subsections provide the functional description of the eTPU channel digital filter.

21.3.4.4.1 Two-Sample Mode

In this mode the EDF works like the TPU2/3 digital filter. It uses the filter clock which is the eTPU clock divided by (2, 4, 8,.., 256) as a sampling clock. The filter clock is selected by the FPSCK field in the

---

[1] The microengine access slot is between its own T4 and T2 edges.

ETPUECR - Engine Configuration Register (see Section 21.2.5.5, “ETPUECR - eTPU Engine Configuration Register). The EDF compares two consecutive samples. If both samples have the same value, the input signal state is updated. Note that when the FPSCK field selects the eTPU clock divided by two, the EDF works like the TPU1 four-clock digital filter.

21.3.4.4.2 Three-Sample Mode

In this mode, like in the TPU2/3 mode, the EDF uses the filter clock as a sampling clock. The EDF compares three consecutive samples. If all three samples have the same value, the input signal state is updated.

The Three-Sample mode gives more signal latency than the Two-Sample mode, but also better noise immunity and better ratio between minimum detected signal pulse to maximum filtered noise pulse. When a certain filter clock frequency is selected for Two-sample mode, double filter clock frequency can be selected to get better latency in Three-sample mode.

21.3.4.4.3 Continuous Mode

In this mode the EDF compares all the values sampled at the rate of eTPU clock divided by two, between two consecutive filter clock pulses. If the signal is continuously stable for the entire digital filter clock period (i.e all the samples have the same signal value), the input signal state is updated.

This method gives the same latency and the same ratio between minimum detected signal pulse to maximum filtered noise pulse, as the Two-Sample mode, as long as there is no noise. Each sampled noise delays the signal transition detection by at least a whole digital filter clock period.

The Continuous mode gives the best noise immunity by comparing multiple samples of the noise. On the other hand, when a short noise pulse appears in the middle of the filter clock period at the same time of a real signal transition, the Continuous mode may reject a real signal transition and delay the response to the first filter clock period in which the signal is continuously stable. This may add to the latency and also to the minimum detected signal pulse in a noisy environment.

21.3.4.4.4 Bypass Mode

In bypass mode the signal that feeds the edge detection comes directly from the output of the synchronizer, not filtered.

21.3.4.4.5 Filter Clock Prescaler

The TCRCLK signal and each channel configured as an input have an associated synchronizer followed by a digital filter connected to the signal that samples signal transitions. After reset, the digital filter filters out high and low pulse widths smaller than the period of two eTPU clocks with ETPUECR bit FCSS=0, or 1 eTPU clock with FCSS=1, preventing these transitions from being input to the transition detect logic. For FPSCK=0 and FCSS=0, the synchronizer and digital filter are guaranteed to pass pulses that are as wide as or wider than four eTPU clocks, meaning a minimum period of 8 eTPU clocks. These figures are halved by setting FCSS=1. By changing the FPSCK field in register ETPUECR the user can select a lower clock rate for the filter signal to define wider valid pulses and filter out wider noise pulses. The filter prescaler clock control is a division of the eTPU clock. To guarantee pulse detection by the digital filter, the pulse must cover at least the stated number of samples at the filter clock rate. For example, a two sample digital filter must sample two points in the pulse to detect it. Table 21-24 shows the minimum guaranteed detected pulse width and the maximum filtered noise pulse width. The table refers only to the digital filter operation. The external pulses may have to be wider (to ensure detection) or narrower (to ensure filtering) depending on the rise/fall delay differences in the MCU receivers and internal logic. Delays introduced by synchronizer, filter and edge detection logic are explained in the eTPU Reference Manual.

Table 21-24. Pulse Widths and Delays

Filter Control (FPSCK)		Filter Clock Period[1]	Min. Width Guaranteed Detected / Max. Width filtered (Min. Filter Delay / Max. Filter Delay)[1]	
FCSS = 0	FCSS = 1		Two-Sample or Continuous Mode	Three-Sample or Integrator[2] Mode
not avail.	000	1	2 / 1 (2 / 3)	3 / 2 (3 / 4)
000	001	2	4 / 2 (3 / 3)	6 / 4 (5 / 5)
001	010	4	8 / 4 (5 / 7)	12 / 8 (9 / 11)
010	011	8	16 / 8 (9 / 15)	24 / 16 (17 / 23)
011	100	16	32 / 16 (17 / 31)	48 / 32 (33 / 47)
100	101	32	64 / 32 (33 / 63)	96 / 64 (65 / 95)
101	110	64	128 / 64 (65 / 127)	192 / 128 (129 / 191)
110	111	128	256 / 128 (129 / 255)	384 / 256 (257 / 383)
111	not avail.	256	512 / 256 (257 / 511)	768 / 512 (513 / 767)

[1] This table shows pulse widths and delays in number of periods of the eTPU clock.

[2] Integrator mode is available for TCRCLK filtering only, see Section 21.3.5.5, “TCRCLK Digital Filter.”

NOTE

If the ETPUTBCR field TCRCF selects the filter clock of the channels (see Section 21.2.6.1, “ETPUTBCR - eTPU Time Base Configuration Register), the TCRCLK filter will be clocked as if FCSS=0, always dividing eTPU clock /2 using FPSCK, regardless if FCSS is 0 or 1.

21.3.5 Time Bases

Each eTPU engine has two Time Counter Registers, TCR1 and TCR2. They provide 24-bit time bases, shared by all 32 channels. TCR1 can also work at full-speed eTPU clock, when ETPUTBCR[TCR1CS]=1 Both TCR1 and TCR2 values can be imported from or exported to the STAC bus. For information on STAC bus protocol and definition of STAC modules refer to IPI STAC and Section 21.3.5.3, “STAC Interface”.

The TCR2 counters between the two Engines are out of phase by 1 eTPU clock, even when Time Bases are shared between them through STAC.

21.3.5.1 Timer Count Register 1 - TCR1

TCR1 can be used in the following modes:

- Internally Clocked Mode

- Externally Clocked Mode
- STAC Bus Client Mode

The host program can read TCR1 time base through the ETPUTB1R (see Section 21.2.6.2, “ETPUTB1R - eTPU Time Base 1 (TCR1) Visibility Register).

The TCR1 bus runs through all the local engine channels

21.3.5.1.1 Externally Clocked Mode

TCR1 can be driven externally by the TCRCLK input, after the digital filter. The TCR1 clock source is configured by the TCR1CTL bit, as shown in Figure 21-47. For more information on clock source selection, please refer to Section 21.2.6.1, “ETPUTBCR - eTPU Time Base Configuration Register.”

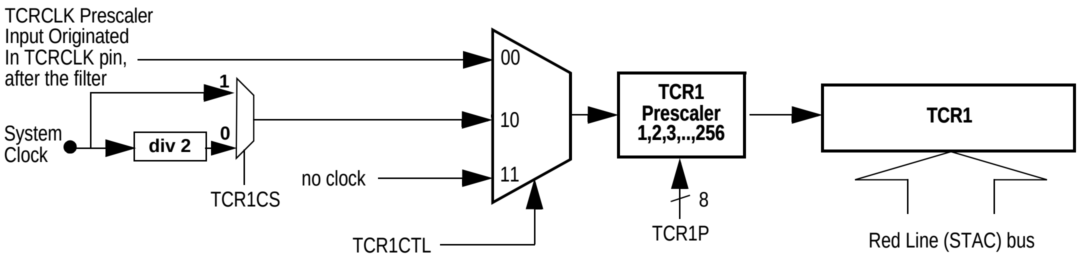


Figure 21-47. TCR1 Clock Selection

21.3.5.1.2 Internally Clocked Mode

TCR1 can be driven by the eTPU clock or eTPU clock divided by 2, before the prescaler. TCR1CTL can also be used to freeze TCR1 clock independently of TCR2 (unlike GTBE).

21.3.5.1.3 TCR1 Clock Prescaling

Any clock source selected by TCR1CTL is prescaled by a factor of 1 to 256, selected by ETPUTBCR field TCR1P. For more information on prescaler configuration refer to Section 21.2.6.1, “ETPUTBCR - eTPU Time Base Configuration Register.” The TCR1 Prescaler resets when etpu_gtbe_in is negated. After reset, it starts counting up to TCR1P when etpu_gtbe_in is asserted. When TCR1 increments (etpu_gtbe_in=1), the prescaler starts a new count and the new TCR1P becomes effective. When TCR1 is written by microcode, the prescaler is reloaded with TCR1P and it becomes effective, if etpu_gtbe_in is asserted.

21.3.5.1.4 STAC Bus Client Mode

In this mode the TCR1 register is continuously updated from the STAC bus, and the clock selection and prescaling logic becomes ineffective. It is not writable by the microcode, and when read, it reflects the STAC bus imported value. The use of EAC is not allowed in client mode. This mode is configured through the register ETPUREDCR (see Section 21.2.6.4, “ETPUREDCR - eTPU STAC Configuration Register).

21.3.5.1.5 STAC Bus Server Mode

TCR1 bus can be exported to the STAC bus as a server, providing time information to other peripherals. This mode is configured through the register ETPUREDCR (see Section 21.2.6.4, “ETPUREDCR - eTPU STAC Configuration Register).

21.3.5.2 Timer Count Register 2 - TCR2

The TCR2 is a 24-bit counter which can be used in the following modes:

- Pin Transition Mode: Count the rise, fall or both transitions of TCRCLK signal.
- Angle Clock Mode: Count internal tooth angle in combination with the eTPU Angle Counter (EAC) hardware which implements an Angle PLL, and generates angle information to the channels. This mode is targeted for angle based applications.
- STAC (STAC) Bus Client Mode: TCR2 is driven by an external source (see Section 21.3.5.2.4, “STAC Bus Client ModeSection , “When eTPU clock divided by two is selected, the synchronizer and the digital filter are guaranteed to pass pulses that are wider than four eTPU clocks (two filter clocks). Otherwise the TCRCLK is filtered with the same filter clock as the channel input signals. For details on TCRCLK and channels digital filter control refer to Section 21.2.6.1, “ETPUTBCR - eTPU Time Base Configuration Register and Section 21.3.4.4, “Enhanced Digital Filter - EDF”.).
- Gated Mode: Count with rate derived from the eTPU clock divided by eight. The TCRCLK signal is used to gate this count, enabling pulse accumulator operations.
- Internally Clocked Modes: TCR2 is driven by internal clock, with count rate of eTPU clock divided by eight.

All clock sources pass through a prescaler. In addition, the TCR2 count can be originated from the EAC which is a hardware angle clock and angle counter. Figure 21-48 shows the diagram for TCR2 clock control. When TCR2 is not driven by the EAC or STAC, the ETPUTBCR field TCR2CTL selects the clock source, also allowing TCR2 to be frozen independently of TCR1 (see Section 21.2.6.1, “ETPUTBCR - eTPU Time Base Configuration Register). When in Angle Mode, TCR2CTL selects the TCRCLK edge sensitivity.

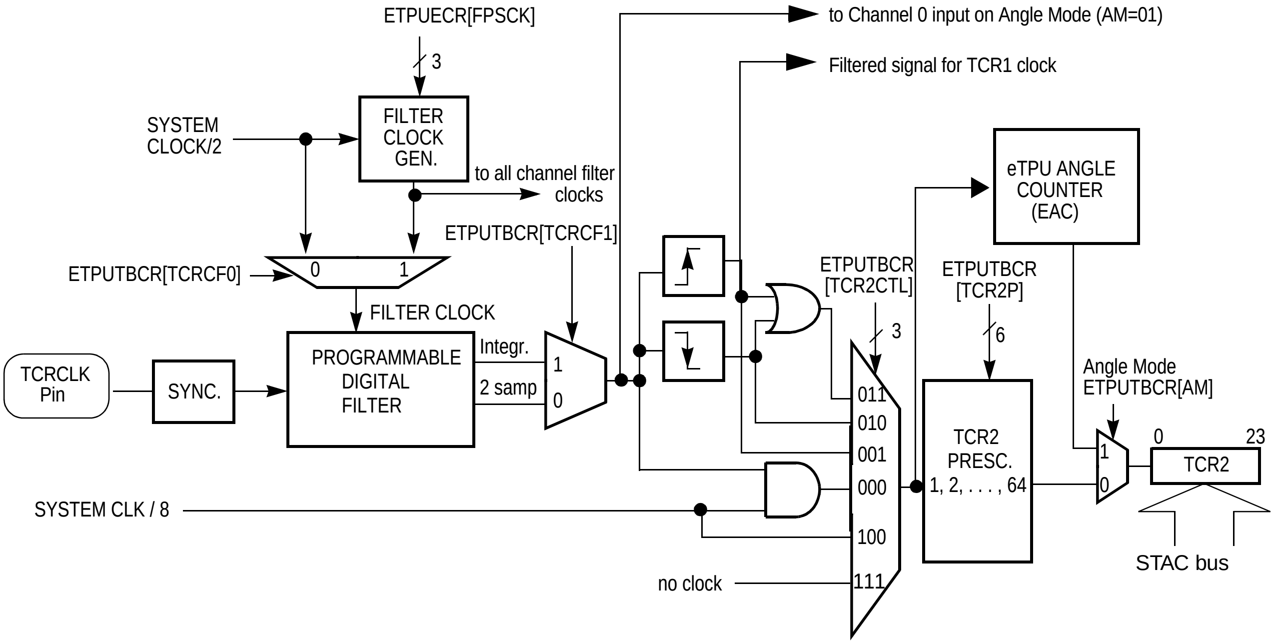


Figure 21-48. TCR2 Clock Control

The TCRCLK signal input is passed through a synchronizer and a programmable digital filter. In Angle Mode with AM=01, synchronizer and filter are also used in Channel 0, replacing its input synchronizer and filter, to get the same timing in the EAC and Channel 0. The TCRCLK synchronizer is an improved filter that provides best latency while maintaining proper noise filtering (see Section 21.2.6.1, "ETPUTBCR - eTPU Time Base Configuration Register field TCRCF[0:1] — TCRCLK Signal Filter Control).

The TCR2 bus runs through all the local engine channels

The TCR2 value is readable to the host through the ETPUTB2R register (refer to Section 21.2.6.3, "ETPUTB2R - eTPU Time Base 2 (TCR2) Visibility Register). When the TCR2 bus value is imported from the STAC bus (STAC client mode), TCR2 is not writable by the microcode, and read access from the microcode or from the host reflect the imported TCR2 value.

21.3.5.2.1 TCR2 Clock Prescaling

Except in Angle Mode, any clock source selected by TCR2CTL is prescaled by a factor of 1 to 64, selected by ETPUTBCR field TCR2P. For more information on prescaler configuration refer to Section 21.2.6.1, "ETPUTBCR - eTPU Time Base Configuration Register." The TCR2 Prescaler resets when etpu_gtbe_in is negated. After reset, it starts counting up to TCR2P when etpu_gtbe_in is asserted. When TCR2 increments (etpu_gtbe_in=1), the prescaler starts a new count and the new TCR2P becomes effective.

The counter that divides the eTPU clock by 8 before the prescaler also resets when etpu_gtbe_in is negated, or when TCR2 is written by microcode.

21.3.5.2.2 TCR2 Gated Mode

TCR2 Gated mode is selected in field TCR2CTL of register ETPUTBCR. In this mode the TCRCLK signal enables or disables transfer of the eTPU clock divided by 8 to the TCR2 prescaler. By programming the prescaler, TCR2 can run at rates from eTPU clock divided by eight down to eTPU clock divided by 512, in steps of eight eTPU clock divisions. For more information refer to Section 21.2.6.1, “ETPUTBCR - eTPU Time Base Configuration Register.

21.3.5.2.3 TCR2 Signal Transition Modes

These modes are selected when the TCR2CTL field in ETPUTBCR is set to rise, fall or “rise-and-fall”. In these modes the TCRCLK signal is the TCR2 clock source, and its maximum transition rate depends on the TCRCLK digital filter mode of operation. The TCRCLK digital filter can be programmed to use the eTPU clock divided by two, or use the same filter clock of the channels, controlled by the TCRCF field in ETPUTBCR. It contains an up-down counter which operates as a digital integrator, optimizing signal latency in the selected mode and clock rate.

When eTPU clock divided by two is selected, the synchronizer and the digital filter are guaranteed to pass pulses that are wider than four eTPU clocks (two filter clocks). Otherwise the TCRCLK is filtered with the same filter clock as the channel input signals. For details on TCRCLK and channels digital filter control refer to Section 21.2.6.1, “ETPUTBCR - eTPU Time Base Configuration Register and Section 21.3.4.4, “Enhanced Digital Filter - EDF”.

21.3.5.2.4 STAC Bus Client Mode

In this mode the TCR2 register is continuously updated from the STAC bus, and the clock selection and prescaling logic becomes ineffective. It is not write accessible for the microcode, and when read, it reflects the STAC bus imported value. The use of EAC is not allowed in client mode. This mode is configured through the register ETPUREDCR (see Section 21.2.6.4, “ETPUREDCR - eTPU STAC Configuration Register).

21.3.5.2.5 STAC Bus Server Mode

When TCR2 bus is exported to the STAC bus as a server, it can provide either time or angle bus to other peripherals, according to its operation mode. This mode is configured through the register ETPUREDCR (see Section 21.2.6.4, “ETPUREDCR - eTPU STAC Configuration Register). To provide sequential update of the STAC clients, the Angle tick rate must not be faster than the STAC programmed update rate. This requirement puts a limitation on the angle clock count rate on high rate mode. In this case the Angle and Angle Fraction accumulator (see the eTPU Reference Manual for details) are advanced at rate of eTPU clock divided by eight. Therefore, the STAC update rate for the Angle Bus must not be slower than eight eTPU clocks.

21.3.5.2.6 TCR2 Bus in Angle Clock Mode

In this mode the TCR2 counter operates as part of the eTPU Angle Counter (EAC). The TCR2 bus value reflects this angle representation in which it counts Angle Ticks. Angle Mode is selected when the AM bit is set in ETPUTBCR.

Note that when TCR2 works in Angle Mode, it does not count directly from the TCR2 clock input which indicates tooth signal transition. Its Angle counter is controlled by the Count Control and High Rate logic (see the eTPU Reference Manual for details), which provides the interpolated pin position, and handle cases of missing tooth, acceleration, de-acceleration and mechanical corrections.

The EAC uses the TCRCLK signal to get the tooth transition indications. The TCR2CTL field in ETPUTBCR has to be set for the appropriate tooth edge detection rise, fall, “rise-and-fall” or none. TCR2 count clock comes from the EAC control and not directly from the physical tooth. This way the EAC control processes the signal transitions and handles missing teeth and flywheel mechanical corrections. Note that when TCR2CTL selects “none” for tooth edge selection, the TCR2 is not necessarily frozen, but can still be incremented by the EAC logic.

In Angle Mode, eTPU channel 0, 1 or 2 operation is combined with the EAC operation. When channel 0 is selected for EAC operation, the TCRCLK digital filter is used both by the EAC and by channel 0 to get full synchronization between the two logics.

The eTPU Angle Counter (EAC) logic runs continuously and updates the TCR2 Angle counter, eliminating the microcode latency in updating the TCR2 value.

21.3.5.3 STAC Interface

Both time bases TCR1 and TCR2 can be shared between the Engines and with other blocks in the same MCU. Each one of both eTPU engines can drive their time bases to the STAC (Shared Time and Count) bus, acting as a server, while any other block can capture the value into its resources and behave like a client. For further reference about the STAC bus operation refer to Section 21.2.6.4, “ETPUREDCR - eTPU STAC Configuration Register.

The eTPU can export to the STAC bus or import from the STAC bus the following internal resources:

- TCR1: Can be exported to or imported from the STAC bus. TCR1 can only be imported from STAC bus when the engine is not in Angle Mode. When TCR1 is imported from the STAC bus, it becomes read-only for the microcode and reflects the imported values. For details refer to Section 21.3.5.1, “Timer Count Register 1 - TCR1.
- TCR2: Can be exported to or imported from the STAC bus. TCR2 can only be imported from the STAC bus when engine is not in Angle Mode. When TCR2 is imported from the STAC bus, it becomes read-only for the microcode, and reflects the imported values. When exported to the STAC bus, TCR2 can work in either Angle Mode or as a free running counter associated with the TCRCLK signal. For details refer to the eTPU Reference Manual.

Proper configuration of the following bits is necessary to determine what can drive the STAC bus: ETPUTBCR[AM] and ETPUREDCR[REN2, RSC2], according to Figure 21-25.

Table 21-25. STAC Bus and Host Read Sources

AM (ETPUTBCR)	REN2,RSC2 (ETPUREDCR)	TCR2 Bus Source (Host read of ETPUTB2R)	STAC Bus Driver
00	0x (disabled)	TCR2/Time	x
01, 10 or 11	0x (disabled)	TCR2/Angle	x
00	11 (Server)	TCR2/Time	TCR2/Time

Table 21-25. STAC Bus and Host Read Sources

01, 10 or 11	11 (Server)	TCR2/Angle	TCR2/Angle
01, 10 or 11	10 (Client)	reserved[1]	

[1] **STAC** client configuration in Angle Mode is also not allowed for TCR1.

Note that Angle Mode is not available for STAC bus clients: configuring both at the same time brings unspecified results. When TCR2 is a stand-alone counter or a STAC Bus server, the same value that is driven to the internal TCR2 bus is also exported to the STAC bus (either Time Count or Angle).

STAC bus configuration is provided by the ETPUREDCR bits REN1/2 and RSC1/2. REN1/2 enable the STAC interface to interact with the resource (either TCR1 or TCR2 bus). RSC1/2 configure the resource (either TCR1 or TCR2 bus) as Server or Client.

Each time base / angle count resource from each engine receives a unique 4-bit hard-wired address that identifies it as a potential server. This address is used by the STAC Controller to coordinate which resource will drive the bus at a given STAC time-slot. For any time-slot there is a server driving the bus upon selection of the STAC Controller, and there may be a client linked to that server by the ETPUREDCR bits SRV1/2 on each Engine. When the server address on the STAC bus matches the value in SRV1/2, the client will load the STAC information into the appropriate resource. For information on eTPU STAC Bus configuration refer to Section 21.2.6.4, “ETPUREDCR - eTPU STAC Configuration Register.

The eTPU does not include a STAC Controller module, which is instantiated once in the system integration.

NOTE

Setting a timebase as client of itself is not allowed.

21.3.5.4 GTBE - Global Time Base Enable

GTBE bit in ETPUMCR register enables time bases in both engines, allowing them to be started synchronously. GTBE is divided in two block interface signals: etpu_gtbe_out and etpu_gtbe_in. GTBE bit sets etpu_gtbe_out, and etpu_gtbe_in enables time bases to start. The etpu_gtbe_out signal can be used for synchronization between eTPU time bases and time bases from other modules. If the GTBE bit in ETPUMCR must enable only the eTPU time bases, etpu_gtbe_out is simply connected to etpu_gtbe_in. These two cases are shown in Figure 21-49. Synchronization logic can be as simple as an OR or an AND logic gate.

Once etpu_gtbe_in transitions to 1, the Engine 1 Time Bases start 1 eTPU clock earlier than Time Bases in Engine 2, except when TCRCLK is selected as clock source or TCR1 when ETPUTBCR[TCR1CS]=1. This happens independently of prescaler values as long as they are the same for both engines, because the prescalers also freeze when etpu_gtbe_in = 0. Microcode can always write to TCR1/2 registers, with either value of etpu_gtbe_in.

NOTE

The timebase prescalers are reset when the GTBE input is negated.

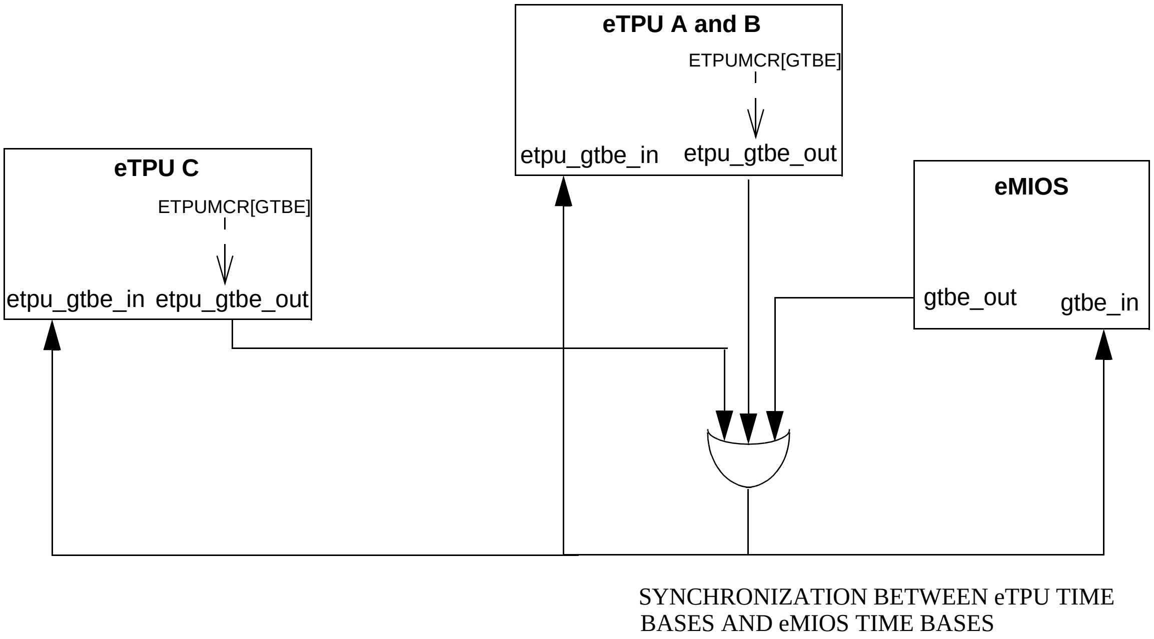


Figure 21-49. Time Base Synchronization

21.3.5.5 TCRCLK Digital Filter

The TCRCLK signal has an improved integrating digital filter with a 2-bit up-down counter. The counter counts up to 3 when a high signal level is detected, or down to 0 when a low level is detected. The signal state is updated to one when the counter stops at 3, or zero when the counter stops at 0. The field TCRCF in register ETPUTBCR (see Section 21.2.6.1, “ETPUTBCR - eTPU Time Base Configuration Register”) determines whether the TCRCLK signal input (after a synchronizer) is filtered with the same filter clock as the channel input signals (see Section 21.3.4.4, “Enhanced Digital Filter - EDF”) or uses the eTPU clock divided by 2, and also whether the TCRCLK digital filter works in integrator mode or the same two sample mode as the channel filters (see Table 21-10).

The TCRCLK filter delay and prescaling determines the minimum detectable TCRCLK pulse widths and, therefore, its maximum frequency, as shown in Section 21.3.4.4.5, “Filter Clock Prescaler and Table 21-24. The TCRCLK signal delay from the module input to TCR1/TCR2 incrementing or detection in the EAC logic is explained in the eTPU Reference Manual.

21.3.6 Safety Features

This section describes the Multiple Input Signature Calculator and memory error support features.

The Multiple Input Signature Calculator - MISC - is an SCM test feature accessible through registers ETPUMCR and ETPUMISCCMPR (see Section 21.2.5, “System Configuration Registers). MISC allows SCM test “on the fly”, i.e., while eTPU is running, with no impact on eTPU functionality or performance.

Memory Error support features comprises SCM and/or SDM error detection, correction, report, and soft error fix.

21.3.6.1 SCM Test - Multiple Input Signature Calculator

The Multiple Input Signature Calculator (MISC) comprises special hardware that sequentially reads all SCM positions and calculates, in parallel, a 32-bit signature from a 32-input CRC signature calculator with the following polynomial:

$1 + x^{1} + x^{2} + x^{22} + x^{31}$

A complete description of the signature calculation procedure can be found in Section 21.4.4, “MISC Algorithm.

Once started by the Host the MISC runs continuously, restarting after the completion of each cycle, when it sets the ETPUMCR register flag SCMMISC (see Section 21.2.5.1, “ETPUMCR - eTPU Module Configuration Register). The average time for a MISC calculation can be measured by checking SCMMISC state at regular intervals, incrementing a counter and clearing SCMMISC if it is set.

MISC accesses to the SCM array are executed if none of the engines is accessing the SCM, to avoid degradation of the microengine performance: it happens while no channel is being serviced. An ongoing MISC operation can be aborted by writing 0 to SCMMISEN.

The Host must load the register ETPUMISCCMPR (see Section 21.2.5.3, “ETPUMISCCMPR - eTPU MISC Compare Register) with the expected value to be found at the end of the MISC cycle, and then start the signature calculation writing bit SCMMISEN=1 in register ETPUMCR (see Section 21.2.5.1, “ETPUMCR - eTPU Module Configuration Register). MISC zeroes the signature accumulator and starts reading SCM data and calculating the signature. After last SCM position is read, MISC compares the value in signature accumulator against the value in ETPUMISCCMPR: if there is a mismatch MISC stops, a Global Exception is issued and the bit SCMMISF in register ETPUMCR assumes value 1. If no mismatch is found, MISC repeats the procedure automatically. When signature is being calculated, SCM address starts at the last SCM address and counts down to 0. The conditions for executing a MISC operation are (see also Table 21-21):

- Both microengines in idle state (no channel is being serviced) or stopped, in any combination (e.g., engine 1 idle with engine 2 stopped).
- ETPUMCR bit VIS = 0.
- ETPUMCR bit SCMMISEN=1.

Note that MISC can run regardless of SCM implementation type (RAM or ROM).

If SCMMISEN=0 or VIS=1, the MISC logic stays at its initial state, with address counter pointing to the last SCM position and accumulator reset.

MISC can also be used to perform single-bit error fixes in the SCM. For more details, see Section 21.3.6.2.3, “Error Fixing”).

21.3.6.2 Memory Error Support

Error Correction Code (ECC) support for SDM and SCM provides guaranteed single bit error correction and double bit error detection. Error detection and correction is performed on read accesses and on write accesses less than 32-bit wide to the SCM.

SDM ECC is calculated per byte, 5 bits wide each, totalling 20 ECC bits for each 32-bit word. That allows the ECC to be updated without a read-modify-write operation on writes of any size.

NOTE

> Error is not checked on bytes not selected on SDM read accesses, either by the Host, CDC or microengine. For instance, the most significant byte is not checked for errors on a microengine 24-bit read access. Accesses to the PSE area (see Section 21.3.2.3.4, “Parameter Sign Extension Area”) are exceptions to this rule, see note further below.

Microengine SDM accesses do not take any wait-state due to ECC support. Host SDM read accesses take two wait-states minimum[1], regardless if an error was detected or not, if the SDM error detection is enabled (ETPUMECR bit DEDD=0). Host SDM read accesses take a minimum of one wait-state if the SDM error detection is disabled (ETPUMECR bit DEDD=1). Host SDM write accesses take one wait-state minimum, regardless of ECC support presence or enabling.

SCM ECC is 7 bits wide, comprising the whole 32-bit word. Read-modify-write is automatically made, with error detection and taking one additional wait-state (two wait-states total), when 16-bit or 8-bit wide writes are made into the SCM in a host access. SCM errors are also detected on Entry Table accesses during TST (see the eTPU Reference Manual for details).

Host SCM read and write accesses have two wait-states[1], regardless if an error was detected or not, and regardless of the SCM error detection disabling (ETPUMECR bit CEDD).

SDM single (correctable) and non-correctable bit errors are flagged in the ETPUMESR register bits DCERR and DNCERR respectively (see Section 21.2.8.1, “ETPUMECR - eTPU Memory Error Control Register”). SCM correctable and non-correctable errors are flagged in the ETPUMESR register bits CCERR and CNCERR, respectively. Double bit errors are guaranteed to be flagged, while three or more simultaneous bit errors in an SDM byte or SCM word are not guaranteed to be flagged. An error flagged as non-correctable can be in two or more bits.

The address, syndrome and data of a non-correctable SDM access can be sampled into the registers ETPUDERAR, ETPUDERSR and ETPUDERDR, respectively. The register ETPUDERAR also captures the SDM read access originator, whether the host, CDC, or specific eTPU channel service (see Section 21.2.8.5, “ETPUDERAR - eTPU Data Error Report Address Register”, Section 21.2.8.7, “ETPUDERSR - eTPU Data Error Report Syndrome Register” and Section 21.2.8.6, “ETPUDERDR - eTPU Data Error Report Data Register”). Similarly, non-correctable SCM accesses address, syndrome and data are stored in the registers ETPUCERAR, ETPUCERSR and ETPUCERDR, respectively. The register ETPUCERAR also captures the SCM read access originator, whether the host, MISC, or specific eTPU channel service (see Section 21.2.8.12, “ETPUCERAR - eTPU Code Error Report Address Register”, Section 21.2.8.14, “ETPUCERSR - eTPU Code Error Report Syndrome Register” and Section 21.2.8.13,

---

[1] one wait-state more, compared with eTPU versions before eTPU2 and previous eTPU2 without ECC support.

"ETPUCERDR - eTPU Code Error Report Data Register"). The occurrence of a non-correctable error overwrites the report register values from a correctable error, either on SDM (DCERR=1) or SCM (CCERR=1).

The last address, syndrome and data of a correctable SDM access can also be sampled into registers ETPUDERAR, ETPUDERSR and ETPUDERDR, but only if the ETPUMESR bit DNCERR is 0. That prevents access address, syndrome and data from a correctable error to overwrite the same information from a non-correctable access. Similarly, the last address, syndrome and data from a correctable SCM access are stored in registers ETPUCERAR, ETPUCERSR and ETPUCERDR only if ETPUMESR bit CNCERR is 0. For either SDM or SCM, a new correctable error overwrites the report registers' values from an earlier correctable error, but report values from a non-correctable error are not overwritten, as long as bit DNCERR/CNCERR is asserted, neither by a new correctable error nor by a non-correctable one.

The sampling of SDM and SCM error report registers can be disabled through the ETPUMECR bits DRDIS and CRDIS, respectively. When DRDIS (CRDIS) is asserted, the update of the SDM (SCM) error report registers is disabled, but not of ETPUMESR bits DCERR, DNCERR (CCERR, CNCERR). DRDIS, CRDIS must be set before reading the respective report registers to assure their coherent reading, preventing them to be updated between reads.

For **microengine accesses**, the Global Exception interrupt asserts whenever a non-correctable error is detected, either on SCM or SDM (see Section 21.3.2.2, "Interrupts and Data Transfer Requests"). These errors are flagged by ETPUMCR bits SCMERR and SDMERR, respectively (see Section 21.2.5.1, "ETPUMCR - eTPU Module Configuration Register"). Single bit errors are corrected and do not generate Global Exceptions.

For **host accesses** a bus error is issued whenever a non-correctable error is detected, either on SCM or SDM (but not through the CDC) and it also generates a Global Exception. Single bit errors are corrected (if the error correction feature is available) and do not cause bus transfer errors.

NOTE

> Error detection also works for host accesses to the PSE area (see Section 21.3.2.3.4, "Parameter Sign Extension Area"). In this case, error is not detected in the most significant byte (bits 0:7). Also, because of sign extension, error is always detected on the bits 8:15, regardless if they are selected, whenever the most significant byte is read.

Correctable errors, either on SDM or SCM, can also be monitored by polling of the ETPUMESR bits DCERR and CCERR.

The error detection logic in SDM and SCM can be disabled by setting the ETPUMECR bits DEDD and CEDD, respectively. The disabled state turns ineffective any action due to error detection, including error correction.

SCM error detection and correction also works for MISC accesses, which are considered microengine accesses for all purposes.

The executing thread is affected by a non-correctable error as follows:

- **if ETPUMECR bit DTEND=1**: on SDM errors due to a read operation, if the instruction is not also an END, the following instruction automatically executes an END, and its operations may be executed, totally or partially. On SDM errors during the TST preload, the thread automatically ends on the first instruction, if the error occurs on a preload parameter selected by the entry point bit PP (see the eTPU Reference Manual for details). The first instruction may be executed totally or partially. Data read into P or DIOB may be corrupted in either case.

- **if ETPUMECR bit DTEND=0**: the TST or thread continues the execution flow normally. Data read into P or DIOB may be corrupted in either case.
- **if ETPUMECR bit CTEND=1**: on SCM errors, the corrupted instruction is not executed, instead replaced by a simple END. Due to instruction prefetching, an error may be detected and yet not affect the thread execution. However, an error detected in a prefetched instruction may cause error report and global interrupts, even if not executed. If a non-correctable error occurs on the entry table read during TST (see the eTPU Reference Manual for details), the first instruction in the thread is replaced with a simple END, a global exception is issued and the reports and flags are normally updated.
- **if ETPUMECR bit CTEND=0**: the execution flow continues normally, and the results depend on the particular instruction corruption.

21.3.6.2.1 Error Correction Code (ECC) and Syndrome Definition

SDM ECC is composed of 5 Hsiao code bits for each data byte. **SCM ECC** is composed of 7 Hsiao code bits for each 32-bit word.

Table 21-26 shows the encoding of the SDM syndrome and global parity fields as they read on the ETPUDERSR register (see Section 21.2.8.7, “ETPUDERSR - eTPU Data Error Report Syndrome Register”). There is one syndrome and one global parity for each byte in an SDM word, numbered SYND0 to SYND3 from the least to the most significant.

Table 21-26. SDM Syndrome Definition

SYNDn[4:0] (hex)	Result	SYNDn[4:0] (hex)	Result
10	error on **code** bit 8n+4	02	error on **code** bit 8n+1
08	error on **code** bit 8n+3	01	error on **code** bit 8n
04	error on **code** bit 8n+2		
0E	error on data bit 8n+7	19	error on data bit 8n+3
15	error on data bit 8n+6	13	error on data bit 8n+2
07	error on data bit 8n+5	16	error on data bit 8n+1
1A	error on data bit 8n+4	1C	error on data bit 8n
00	no error	any other	non-correctable error

Table 21-27 shows the definition of the SCM syndrome field as it reads on the ETPUCERSR register (see Section 21.2.8.14, “ETPUCERSR - eTPU Code Error Report Syndrome Register”).

Table 21-27. SCM Syndrome Definition

SYND[6:0] (hex)	Result	SYND[6:0] (hex)	Result
40	error on **code** bit 6	04	error on **code** bit 2
20	error on **code** bit 5	02	error on **code** bit 1
10	error on **code** bit 4	01	error on **code** bit 0
08	error on **code** bit 3		
49	error on data bit 31	1A	error on data bit 15
0D	error on data bit 30	23	error on data bit 14
0E	error on data bit 29	2A	error on data bit 13
38	error on data bit 28	32	error on data bit 12
4C	error on data bit 27	46	error on data bit 11
1C	error on data bit 26	4A	error on data bit 10
58	error on data bit 25	52	error on data bit 9
0B	error on data bit 24	62	error on data bit 8
54	error on data bit 23	13	error on data bit 7
15	error on data bit 22	29	error on data bit 6
16	error on data bit 21	31	error on data bit 5
34	error on data bit 20	43	error on data bit 4
25	error on data bit 19	45	error on data bit 3
26	error on data bit 18	19	error on data bit 2
64	error on data bit 17	51	error on data bit 1
2C	error on data bit 16	61	error on data bit 0
00	no error	any other	non-correctable error

21.3.6.2.2 Error Injection

Errors can be injected into the SDM read data path through the registers ETPUDEIAR, ETPUDEIDPR and ETPUDEIPPR (see Section 21.2.8.2, “ETPUDEIAR - eTPU Data Error Injection Address Register”, Section 21.2.8.3, “ETPUDEIDPR - eTPU Data Error Injection Data Pattern Register” and Section 21.2.8.4, “ETPUDEIPPR - eTPU Data Error Injection Parity Pattern Register”). They define bit flips on accesses at the specific address specified in ETPUDEIAR.

SDM error injection is enabled independently for host and microengine accesses by the ETPUMECR bits HDEIE and MDEIE, respectively. The user must program ETPUDEIAR, ETPUDEIDPR and ETPUDEIPPR with HDEIE=MDEIE=0, and only then enable error injection by setting HDEIE=1 and/or MDEIE=1 to avoid races (see Section 21.2.8.8, “ETPUMESR - eTPU Memory Error Status Register”).

Similarly, SCM errors can be injected into the SCM read data path through the registers ETPUCEIAR, ETPUCEIDPR and ETPUCEIPPR, enabled by the ETPUMECR bit CEIE (see Section 21.2.8.9, “ETPUCEIAR - eTPU Code Error Injection Address Register”, Section 21.2.8.10, “ETPUCEIDPR - eTPU Code Error Injection Data Pattern Register” and Section 21.2.8.11, “ETPUCEIPPR - eTPU Code Error Injection Parity Pattern Register”). Errors are injected on host accesses to the address specified in ETPUCEIAR when ETPUMCR bit VIS=1, or on microengine and MISC accesses when VIS=0.

21.3.6.2.3 Error Fixing

Single bit soft errors can be fixed by rewriting the right value, as corrected by the ECC logic, into the same address of the erroneous data. eTPU provides means to perform such fixes safely, coherently and concurrently with the normal operation.

SDM fixes can be done using the CDC (see Section 21.3.4.2, “Coherent Dual-parameter Controller - CDC”), limited to the lower 8 KB of SDM. The Host application simply has to program a transfer with both origin and destination positions pointing to the address of the data to be fixed. Because the PBBASE field has an address granularity of 8 bytes, a word adjacent to the one to be fixed must also be rewritten. The CDC transfer is atomic, therefore the fix is done coherently and does not interfere with the application code or data flows.

SCM fixes can be done by programming the ETPUCEFR (see Section 21.2.8.14, “ETPUCERSR - eTPU Code Error Report Syndrome Register”). The fixes are performed by the MISC logic, which must be enabled (see Section 21.3.6.1, “SCM Test - Multiple Input Signature Calculator). The only mode available, selected by the ETPUCEFR field CFIXM, is:

- **Automatic Fix Mode (CFIXM=11)**: the MISC logic automatically rewrites the contents of an address whenever it finds a correctable error; the CFIXM field keeps its value if a fix is made.

21.3.7 Performance Monitoring Features

21.3.7.1 Idle Counter

The Idle Counter Register ETPUIDLER (see Section 21.2.7.2, “ETPUIDLER - eTPU Idle Register”) continuously counts microcycles in which the microengine is not busy with channel service. It can be used to measure the microengine utilization by rating the count measured during a period of time to the number of microcycles contained in the period. The Idle counter does not count microcycles when the engine is stopped, or is in TST or halt states.

21.4 Initialization/Application Information

21.4.1 Multiple Parameter Coherency Methods

Follows a description of two methods for coherent transfer of multiple parameters between Host and eTPU. Both methods involve the use of two parameter areas: the Transfer Parameter Area (hereafter called TPA), which is the SDM area directly accessed by the Host for reads and writes, and the Permanent Parameter Area (hereafter called PPA), which are the SDM positions where channel parameters are normally accessed by the Function microcode. Note that parameters in either TPA or PPA do not have to be in sequential addresses. TPAs and PPAs allocation are completely defined by the application, and there may be any number of them, independently of the channels.

The methods described here are not the only solutions for the coherent transfer problem, and both can coexist in eTPU and even used in combination. Also note that for transfers of a pair of parameters, the Coherent Dual-parameter Controller is faster and have less impact on both eTPU and Host performance. That said, the methods are:

- **Transfer Service**: a microengine thread transfers, upon Host Service Request, data from/to a TPA to/from a PPA.Coherency is guaranteed by the fact that a thread is atomic with respect to other threads in the same Engine, and so are its transfers. If parameters in PPA are shared by both Engines, hardware semaphores have to be used to access them.
- **Mailbox**: for Host to eTPU transfers, the microcode checks a flag, set by the host, indicating the existence of new parameter data in the TPA. It can, then, either access TPA data directly or copy it to the PPA. For eTPU to Host transfers, when microcode changes PPA, it copies them to the TPA and flags updated TPA data to Host, possibly using an Interrupt or a Data Transfer Request. The Mailbox flag is reset when data is copied: by the eTPU microcode, when it transfers TPA to PPA (possibly followed by an Interrupt); by the Host, when it reads data from the TPA. This indicates that TPA is free for another transfer.

Transfer Service has the advantage of separating the task of data transfer from the functional service thread that accesses the parameters, with less impact to the latter. Compared to the Mailbox method, however, it has bigger average latency, because the Transfer Service thread has to contend for a time slot to execute. This latency can be minimized if Transfer Service thread is assigned to a separate channel with higher priority, but even so it does not guarantee that PPA is updated before the next execution of the functional thread that uses it.

Mailbox method, on the other hand, makes the functional thread check for the existence of new data (Host to eTPU). It does not have to be responsible for the transfer, though: it may access the TPA directly, and a Transfer Service can then be used to copy data from TPA to PPA.

21.4.2 Estimating Worst Case Latency

Reliable systems are designed to work under worst-case conditions. This section explains how to estimate worst-case latency (WCL) for any eTPU function in any system. The appendix covers the following topics:

- Introduction to Worst-Case Latency
- Using Worst-Case Latency Estimates to Evaluate Performance

- Priority Scheme Details used in WCL Analyses
- First-Pass WCL Analysis
- Second-Pass WCL Analysis

The first-pass WCL analysis is based on a deterministic, generalized formula that is easy to apply. Because of the generalizations in the formula, the first analysis result is almost always much worse than the real worst case. If the desired system performance is within the limits of this first analysis, then no further analysis is required; the system is well within the performance limits of the eTPU. If the desired system performance exceeds that indicated by the first analysis, the second-pass WCL analysis should be applied. The second-pass analysis is not a generalized formula, but rather uses specific system details for a realistic worst-case estimation.

21.4.2.1 Introduction to Worst-Case Latency

NOTE

In this Appendix the latency calculation and examples refer to old TPU functions such as PWM, DIO etc. These functions use single action channels which have single transition and single match functionality. They are not optimized for the eTPU hardware enhancement which support various double action modes. These examples are for reference only. New eTPU functions which are optimized for the new hardware will impose different latency calculations.

Worst-case latency for a channel is the longest amount of time that can elapse between the execution of any two function threads on that channel. For example, if in a particular system, channel 5 is running PWM, the worst-case latency for channel 5 is the longest possible time between the execution of two PWM threads. The worst case time includes the time the execution unit takes to execute threads for other active channels, and other delays described later in this section. Refer to Figure 21-50.

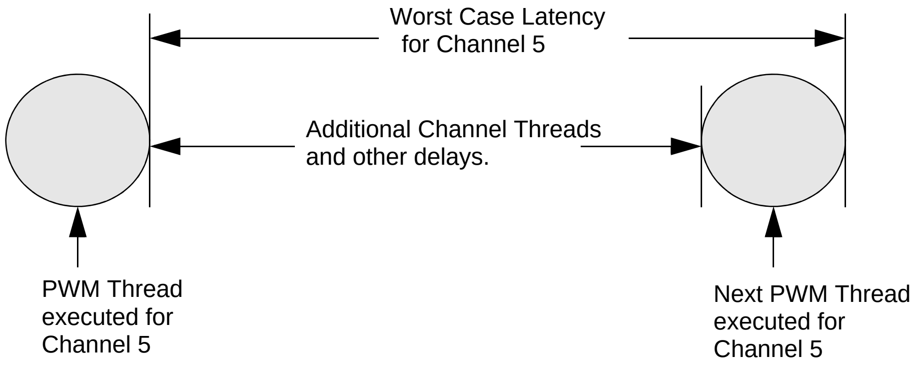


Figure 21-50. Worst-Case Latency for PWM

Worst-case latency for a channel depends both on the function running on that channel and on the activity on other channels. Since the 32 eTPU channels must all share the same execution unit, execution speed of a particular function varies with each system. The PWM thread response is faster if there are no other active channels than if other channels are also active. In addition, changing the priority scheme and channel number assignments can change performance for a function even if the same set of functions are still active.

Each function is divided into treads, as shown in Figure 21-51 (see also the eTPU Reference Manual for details). The eTPU Microengine executes one thread of a function at a time. For example, the Microengine might execute thread 1 of PWM, then thread 3 of DIO, then thread 2 of PWM, then thread 2 of SM, and so on. The amount of time the eTPU Microengine grants a function to execute a thread varies with the number of microcode instructions in the thread.

Since there is only one eTPU Microengine (in each eTPU Engine), the eTPU cannot actually execute the software for multiple functions simultaneously. However, the hardware for each of the channels is independent. This means that, for example, all 32 channel signals can change thread at the same moment, provided that the function software sets up the channel hardware to do so beforehand.

With Host CPU code, the system designer assigns functions to channels and initializes the functions. After initialization, functions typically run without Host intervention, except for eTPU channel interrupts to the Host to give or receive information. Most functions can run continuously with periodic servicing from the eTPU Microengine. As required, the channels request service from the eTPU Microengine, and the eTPU Scheduler determines the order in which the channels are serviced. Worst-case latency for a channel can be derived from the details of the priority scheme that the scheduler uses (see Section 21.3.3, “Scheduler).

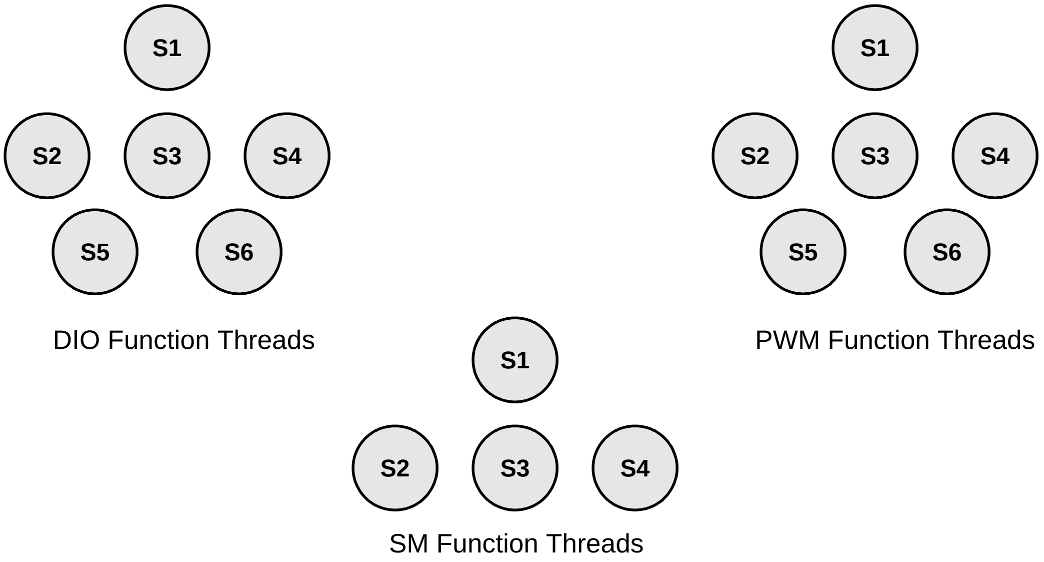


Figure 21-51. Function Threads

21.4.2.2 Using Worst-Case Latency Estimates to Evaluate Performance

Once WCL is found for a channel, the user must determine how to use this number to analyze performance. To analyze the performance of a channel running the PWM function, for example, some information about what happens in each thread is necessary.

The following example refers to old TPU PWM function, which is not optimized to the eTPU enhanced hardware. For PWM, thread 1 is the initialization thread, and threads 2 and 3 are used during normal function execution. (PWM threads 4, 5, and 6 are for special modes and will be assumed to be unused on channel 5). Thread 2 writes a time into the channel 5 match register and performs other operations that will cause the channel 5 signal to go from low to high at the time indicated in the match register (match time). At match time, the signal goes high and channel 5 requests service from the eTPU Microengine to execute thread 3. Thread 3 writes a time into the channel 5 match register and performs other operations that will cause the channel 5 signal to go from high to low at match time. At match time, the signal goes low and channel 5 requests service from the eTPU Microengine to execute thread 2. A PWM wave is kept running on the system by the eTPU executing thread 2, then thread 3, then thread 2, then thread 3, and so on.

Since the definition of worse-case latency assumes a fully loaded running system, initialization threads are not part of worst-case calculations. For the channel 5 example, the two PWM threads in Figure 21-50 are thus the two normal running threads, threads 2 and 3.

Figure 21-50 does not define which thread is thread 2 and which is thread 3. Since the worst-case latency derived from the first-pass analysis is the worst case between any2 threads (not counting initialization threads), it is safe to say that the worst-case latency shown in Figure 21-51 represents both the worst-case high time and the worst-case low time.

Notice in Figure 21-50 that worst-case latency is drawn from the end of the execution of the first PWM thread to the end of the execution of the next PWM thread. It is drawn from end to end because the microcode instructions that make up the threads control the channel hardware. To make sure that all the microcode instructions needed to change the pin thread have been executed, it is necessary to include the execution time of the second thread.

Thread information for each function is found in the programming notes for individual TPU functions. Refer to Freescale Programming Note TPUPN00/D, Using the TPU Function Library and TPU Emulation Mode, for a list of available programming notes. Similar documentation will we provided for the eTPU new functions.

21.4.2.3 Priority Scheme Details Used in WCL Analysis

The user assigns functions to channel numbers and gives each active channel a priority level of high, middle, or low. The Scheduler uses the channel number and channel priority level to determine the order in which to grant service.

The scheduler allocates time slots to specific priority levels of high, middle, or low. One function thread is executed in each time slot. The length of a time slot varies according to the length of the executing thread. When fully loaded, the scheduler always assigns time slots in a seven-slot sequence (see Figure 21-52). After a seven-slot sequence is completed, another seven-slot sequence begins (see Figure 21-53). Note that in eTPU, when no service request exists, the scheduler goes to thread 1, but WCL calculation considers full load.

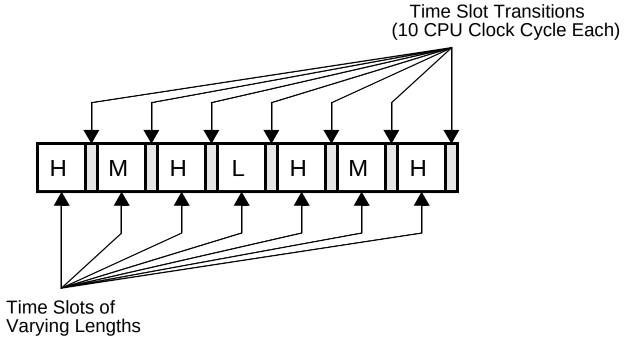


Figure 21-52. Time-Slot Sequence

This sequence scheme gives higher-priority channels more service time than lower-priority channels. High-priority channels are allocated four of seven time slots, middle-priority channels are allocated two of seven time slots, and low-priority channels are allocated one of seven time slots.

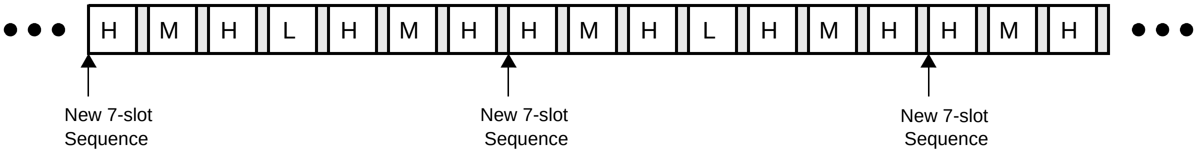


Figure 21-53. Multiple Time-Slot Sequences

21.4.2.3.1 Priority Passing

If no channel of the priority level assigned to the time slot is requesting service, the eTPU scheduler can pass priority to other levels. If no high-level channel is requesting service during a high level time slot, a middle-level channel is granted service; or, if no middle level-channel is requesting service, a low-level channel is granted service. If no middle-level channel is requesting service during a middle-level time slot, a high-level channel is granted service; or, if no high-level channel is requesting service, a low-level channel is granted service. If no low-level channel is requesting service during a low-level time slot, a high-level channel is granted service; or, if no high-level channel is requesting service, a middle-level channel is granted service. If no channel is requesting service, the time slot sequence is reset to state 1 and the scheduler idles until a request is received.

Priority passing is implemented in hardware and does not contribute to worst-case latency.

21.4.2.3.2 Time-Slot Transition

After each time slot, the eTPU must prepare for the next time slot. This preparation time between each time slot is called a time-slot transition. See the eTPU Reference Manual for details. Time-slot transitions can take from six up to ten eTPU clocks.

21.4.2.3.3 Channel Number Priority

If more than one channel of a priority level is requesting service, the lowest numbered channel is granted service first. For example, if channels 0, 5, and 9 are all high-level channels requesting service during a high time slot, channel 0 is granted service first. Continuing this example, if channel 0 requests service again immediately after being serviced, it is not serviced again until channels 5 and 9 are serviced. This scheme is implemented so that continuously-requesting low numbered channels do not take all the time on the eTPU execution unit and leave no time for other channels.

The scheduler uses registers to keep track of which channels have been serviced and which require servicing. Each channel has two register bit: a service request register (SRR) and a service grant register (SGR). The SRR is set when a channel requests service. After the channel has been granted service, the SGR is set and the SRR is cleared.

SGRs are not cleared individually by channel, but rather as priority level groups. The clearing of a group of SGRs begins a new cycle for that priority level. An SGR group is cleared on the condition that a channel of that priority level has just been serviced, and no other channel of that priority level is requesting service (has a set SRR) and has not been granted service (has a clear SGR).

For example, if a middle-priority channel has just been serviced (either in a middle-priority time slot or a high or low-priority time slot gained by priority passing), the SRRs and SGRs of all middle-priority channels are compared. If there is no middle-priority channel with its SRR set and SGR cleared, the scheduler clears all middle-level SGRs. If there is a middle-level channel with its SRR set and SGR cleared, the scheduler does not clear the SGR group, and the requesting middle-level channel is serviced on the next middle-level time slot (or possibly sooner by priority passing).

21.4.2.3.4 SDM Collision Rate

Most function threads read or write to the eTPU SDM at least once. Because both the eTPU Microengine and Host can access the SDM but not at the same time, the Microengine may suspend execution during the SDM access while waiting for the Host to finish accessing the SDM. At other times the Host may wait for the Microengine. Wait states can take up to two eTPU clocks, when the Host accesses the SDM directly, without using CDC. Microengine(s) wait-states must be added into the worst-case latency calculation. The system designer should estimate the percentage of SDM accesses in the system that will result in Microengine wait-states. This percentage is called the RAM collision rate (RCR). In each collision with direct Host accesses to the SDM the Microengine(s) wait for two eTPU clocks.

In eTPU the Coherent Dual-parameter Controller (CDC) may also access the SDM for atomic transfers of two parameters. eTPU Microengine may wait on this operation (if it is in service time) until the transfer is complete. CDC always transfers two parameters, making four consecutive accesses (read, write, read, write) of one eTPU clock each. The system designer should estimate the percentage of SDM accesses in the system that will result in a Microengine wait due to coherent transfer, and multiply it with the average number of eTPU clocks the Microengine waits for each transfer. This percentage is called Coherent Parameter Collision Rate (CPCR).

In addition, Microengine to Microengine multiple parameter coherent communication, using the hardware semaphores, may hold one Microengine which waits to lock the semaphore while the other Microengine is holding it. This waiting is due to a software loop, not hardware wait-states. Note that single parameter access of one Microengine does not affect the timing of the other Microengine due to SDM time interlace.

This implies that single parameter Microengine to Microengine communication does not affect the performance. The Microengine which waits for the semaphore will loop until it is freed by the other Microengine. This time depends on the eTPU application. The system designer should estimate the percentage of Microengine to Microengine coherent parameter communication that will result in eTPU semaphore loops, and multiply it with the average number of eTPU clocks the Microengine loops for each such transfer. This percentage is called CCR (Communication Collision Rate).

A 100% collision rate for a system is the theoretical worst case. In many systems, however, the RCR, CPCR and CCR would be very low, sometimes even near 0%. This is because the eTPU is an independent processor capable of servicing most function needs, so that the Host rarely needs to access the eTPU SDM. Also coherent Microengine to Microengine communication of more than one parameter may be rare. To find a realistic RCR, CPCR the system designer should evaluate the Host code and find the percentage of time it accesses the eTPU SDM with or without using the CDC. This percentage gives a good RCR and CPCR. The eTPU application provides a good estimation of CCR.

NOTE

> The programming practice of polling a flag in the eTPU SDM causes a very high RCR and should be avoided in high-performance systems.

After the collision rate for a system is found, it can be applied to the WCL calculations for each channel. The system designer can use the collision percentage and the number of SDM accesses (with and without semaphores) to estimate the eTPU loop time for a function. Note that in old TPU functions CPCR and CCR are both zero.

The estimation of eTPU wait time is as follows:

Variables:

N1 = Number of simple RAM accesses in the longest thread

RCRWait = Maximal eTPU clocks wait time for simple RAM collision = 2

CPCRWait = Average eTPU clocks for Coherent Parameter Transfer (using CDC).

N2 = Number of eTPU-eTPU semaphore RAM accesses in the longest thread

CCRWait = Average eTPU clocks for Microengine-Microengine communication transfer.

Estimated Wait Time:

Function eTPU maximal wait time =

N1 *(RCR * RCRWait + CPCR * CPCRWait) + N2* CCR * CCRWait

21.4.2.4 First-Pass Worst-Case Latency Analysis

Following is the first-pass calculation of worst-case latency for a channel. Remember that this analysis uses generalizations that usually produce a result much worse than the real worst case. If the worst-case result from the first analysis is too long for the desired performance, use the second analysis for a more realistic worst-case analysis.

21.4.2.4.1 Worst-Case Assumptions and Formula

To estimate worst-case latency for a channel, assume this worst-case condition: the channel has just been serviced in a time slot of its priority level, and all other channels in the system are continuously requesting service and have cleared SGRs. The worst-case latency is the time from the end of the channel's service until the end of the channel's next service. See Figure 21-54.

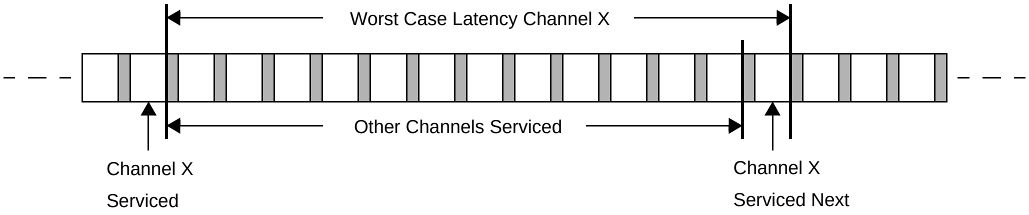


Figure 21-54. First-Pass Worst-Case Latency

To estimate worst-case latency:

- Find the worst-case service time for each active channel.
- Using the H-M-H-L-H-M-H time-slot sequence, map the channels that are granted for each time slot.
- Add time for six-clock time-slot transitions.

Finding the Worst-Case Service Time for Each Active Channel

A table for eTPU functions should list the longest threads (not counting initialization threads) for the functions, and the number of eTPU SDM accesses in the longest thread (semaphored and non semaphored). These figures will be used for estimating Microengine wait time. Table 21-28 is an example for old TPU functions in which there are only simple SDM accesses. It does not take into consideration the CDC operation and Microengine to Microengine communication.

The worst-case service time for each channel is: (CPCR=CCR=0)

Longest thread + ((number of RAM accesses in longest thread+1) * RCR * 2 clocks). Note that the formula adds 1 RAM accesses for the parameter preload that occurs during TST. There are actually three accesses during TST, but only the first one can receive wait-states.

Table 21-28. Longest Threads and RAM Accesses for old TPU Functions

Function	Longest Thread	RAM Accesses
DIO	10	4
ITC	40 (no linking) 42 (linking)	7
OC	40	7
PWM	24	4

Table 21-28. Longest Threads and RAM Accesses for old TPU Functions

Function	Longest Thread	RAM Accesses
SPWM Mode 0 Mode1 Mode 2	 14 18 20 (no linking) 22 (linking	 4 4 4 4
PMA	94	8
PMM	94	8
PSP Angle-Angle Mode Angle-Time Mode	 76 50	 6 3
SM[1]	160	21
PPWA Mode 0 Mode 1 Mode 2 Mode 3	 44 50[2] 44 50	 9 10 9 10

[1] Assumes one master and one slave. For each additional slave
a) Add 32 clocks and 2 RAM accesses, and
b) Add (STEP_RATE_CNT ∗ two clocks)

[2] With one channel linked. Add two clocks for each additional channel linked.

Mapping the Channels for Each Time Slot

To determine when a channel will be serviced again, it is necessary to determine which other channels will be serviced first. Do this by assuming all channels are continuously requesting service and mapping the channels into the time-slot sequence.

Adding Time for Time-Slot Transitions

Add six eTPU clocks for time-slot transitions which occur after each time slot.

21.4.2.4.2 First-Pass Analysis Worst-Case Latency Examples

The examples in this section assume the system configuration shown in Table 21-29.

Table 21-29. System Configuration Example

Channel	Priority	Function[1, 2]
0	High	PWM (driving a DC motor)
1	Middle	PPWA (Mode 0, measuring the DC motor speed)
2	Low	DIO (Input)

[1] 9% RAM Collision Rate (RCR)

[2] CPU clock rate = 40 MHz, or 25 ns per clock period

Finding the WCL for PWM on Channel 0

The following shows how to find the WCL for PWM on channel 0.

1. Find the worst-case service time for each active channel.
   a. Longest thread of PWM is 24 CPU clocks with four RAM accesses.

      24 + ((4 RAM accesses+1) * 0.09 * 2 CPU clock waits) = 24.9 CPU clocks, rounded up to 25 CPU clocks (since there are no partial clock periods)

      Channel 0 worst-case service time = 25 CPU clocks.
   b. Longest thread of PPWA in mode 0 is 44 CPU clocks with nine RAM accesses.

      44 + ((9 RAM accesses+1) * 0.09 * 2 CPU clock waits) = 45.8 CPU clocks, rounded up to 46 CPU clocks

      Channel 1 worst-case service time = 46 CPU clocks.
   c. Longest thread of DIO is ten CPU clocks with four RAM accesses.

      10 + ((4 RAM accesses+1) * 0.09 * 2 CPU clock waits) = 10.9 CPU clocks, rounded up to 11 CPU clocks

      Channel 2 worst-case service time = 11 CPU clocks.
2. Assume channel 0 has just been serviced and that channels 1 and 2 are continuously requesting service. Using the H-M-H-L-H-M-H time-slot sequence, map the channels that are granted for each time slot. See Figure 21-55.

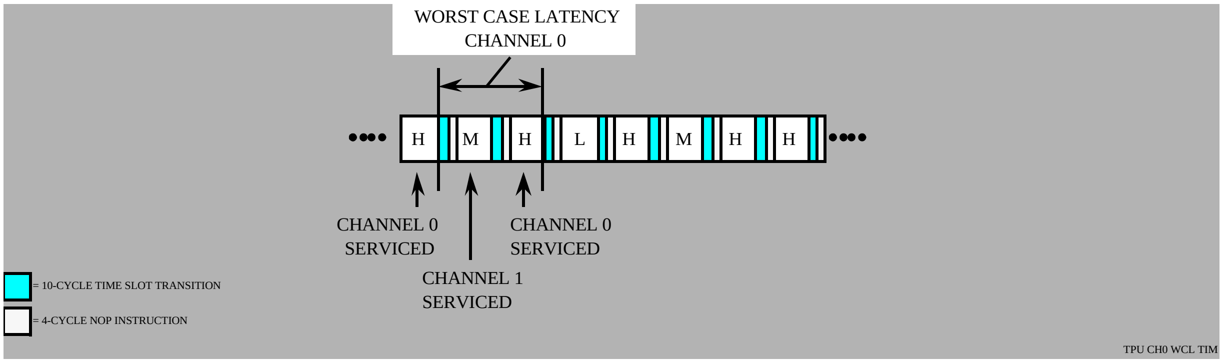


Figure 21-55. Next Servicing for Channel 0

Channel 1 will be serviced in the middle-priority time slot before channel 0 is serviced again.

3. Add time for the six-clock CPU time-slot transitions. See Figure 21-55 and Table 21-30.

A four-clock NOP occurs after each channel is serviced since there is one channel in each priority level, i.e., a new cycle for a priority level is started after each channel is serviced. Time-slot transitions occur after each time slot.

Table 21-30. Worst-Case Latency for Channel 0

Channel 0 worst-case service time	25 clocks
Channel 1 worst-case service time	46 clocks
Two 6-clock time-slot transitions	12 clocks
Total clocks	83 clocks

83 clocks * 25 ns/clock = 2075 ns

Conclusion: in this system configuration PWM can run with a minimum high time or low time of 2075 ns.

Note that in double match eTPU system the PWM can be serviced once in each period, and there is no latency for minimum high time. The latency in eTPU PWM function will represent the minimum PWM period.

Finding the WCL for PPWA on Channel 1

The following shows how to find the WCL for PPWA on channel 1.

1. Find the worst-case service time for each active channel. See step 1 of previous example.
2. Assume channel 1 has just been serviced and that channels 0 and 2 are continuously requesting service. Using the H-M-H-L-H-M-H time-slot sequence, map the channels that are granted for each time slot. See Figure 21-56.

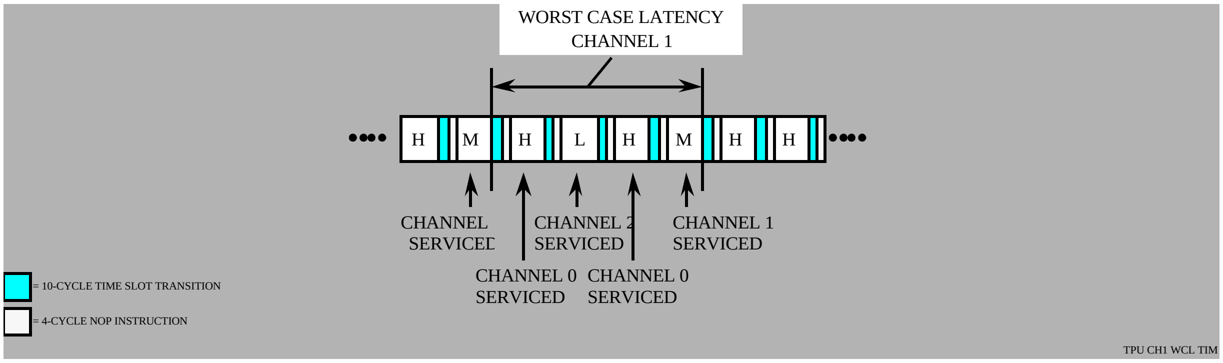


Figure 21-56. Next Servicing for Channel 1

Channel 0 will be serviced twice and channel 2 once before channel 1 is serviced again.

3. Add time for the six-clock CPU time-slot transitions. See Figure 21-56 and Table 21-31.

Table 21-31. Worst Case Latency for Channel 1

Two Channel 0 worst-case service times	50 clocks
Channel 1 worst-case service time	46 clocks
Channel 2 worst-case service time	11 clocks
Four 6-clock time-slot transitions	24 clocks

Table 21-31. Worst Case Latency for Channel 1

Total clocks	131 clocks

131 clocks * 25 ns/clock = 3275 ns

Conclusion: in this system configuration PPWA can measure a period or pulse of minimum 3275 ns.

Note that PPWA function optimized for eTPU hardware can use double transition mode to measure very narrow pulses with one service after the second transition, and latency will affect only the minimum gap between two input pulses. Also the function threads would have more efficient coding.

Finding the WCL for DIO on Channel 2

The following shows how to find the WCL for DIO on channel 2.

1. Find the worst-case service time for each active channel. See step 1 of previous examples.
2. Assume channel 2 has just been serviced and that channels 0 and 1 are continuously requesting service. Using the H-M-H-L-H-M-H time-slot sequence, map the channels that are granted for each time slot. See Figure 21-57.

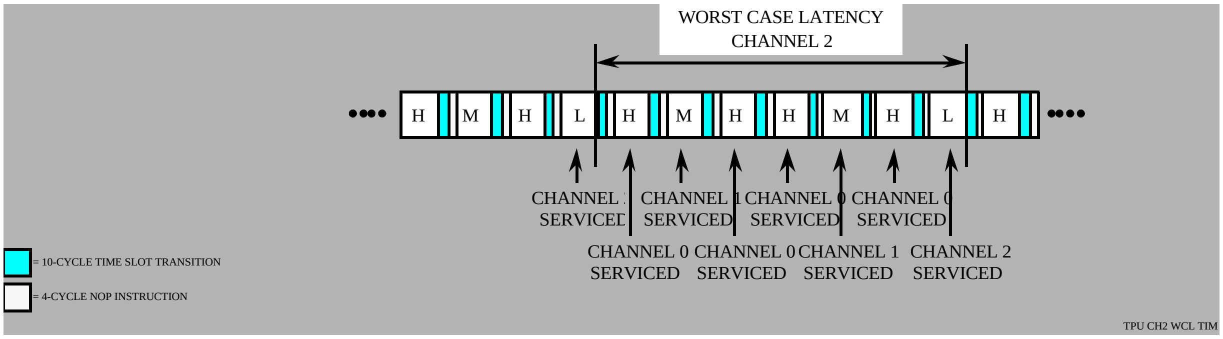


Figure 21-57. Next Servicing for Channel 2

Channel 0 will be serviced four times and channel 1 twice before channel 2 is serviced again.

3. Add time for the ten-clock CPU time-slot transitions and the four-clock NOPs. See Figure 21-57 and Table 21-32.

Table 21-32. Worst Case Latency for Channel 2

Four Channel 0 worst-case service times	100 clocks
Two Channel 1 worst-case service time	92 clocks
Channel 2 worst-case service time	11 clocks
Seven 6-clock time-slot transitions	42 clocks
Total clocks	245 clocks

245 clocks * 25 ns/clock = 6125 ns

Conclusion: in this system configuration DIO can keep track of the input level at a minimum of every 6125 ns.

Note that DIO function optimized for eTPU hardware can use double transition mode to measure two pin transitions at a time and reduce the service time, improving the overall system performance and latency.

21.4.2.5 Second-Pass Worst-Case Latency Analysis

Following is an example of a second-pass analysis for calculating worst-case latency for a channel. The second-pass analysis is useful for higher-performance systems, since it gives a more realistic worst-case latency result than first-pass analysis.

This example uses a relatively simple system in order to illustrate the basic principles of second-pass analysis. For a more complex example of second-pass analysis, refer to Multiphase Motor Commutation TPU Function (COMM)(TPUPN09/ D).

21.4.2.5.1 Second-Pass Analysis Guidelines

Rather than use a fixed formula, a second-pass analysis relies on the application of the following guidelines.

1. The first-pass analysis makes the assumption that all channels in the system are continually requesting service. For many systems this is an unrealistic assumption. For example, if TCR1 is counting at a rate of 2 MHz (500 ns per count) and a channel is running the DIO function with a match rate of 20,000 TCR1 counts, the DIO will request service every 10 ms (20,000 * 500 ns = 10,000,000 ns or 10 ms). It is therefore unrealistic to assume that the channel running this DIO function is continuously requesting service. Figure out a realistic service request rate for each channel. Time slots can then be mapped to each channel at the real rate of request.
2. If a function is active during system initialization but not during the high-speed running mode of the system, then that system does not need to be included in the high-speed worst-case latency calculations.
3. Use a realistic SDM collision rate.
4. Be careful when assigning functions priority levels and channel numbers. Decide which function or functions will be most difficult to make perform at the desired level. Assign those channels high priority and low channel numbers. Try different priority and channel assignments to see how it affects the system.
5. The seven-slot sequence of || H | M | H | L | H | M | H || is asymmetrical when put back-to-back with other seven-slot sequences. Note that in the following sequence there are two high-priority slots next to each other:

   || H | M | H | L | H | M | H |||| H | M | H | L | H | M | H ||
6. Make sure that when mapping out channels to the sequence, you choose a worst-case slot to start the mapping. For example, when estimating WCL for a high-priority channel, do not start the mapping in the last high-priority slot in a seven-slot sequence, as that is a best case for a high-priority channel since another high-priority time slot is next.
7. Instead of always using the longest thread in the function as the worst-case thread, evaluate the threads in the function that will be used in the system and use the appropriate worst-case threads. For example, in the preceding example of first-pass analysis, the PWM was shown to be able to

achieve a high time and low time of 2475 ns under worst-case conditions. This was derived using the longest PWM thread of 24 CPU clocks. This longest thread is actually thread 2, the thread that is entered after the pin has just gone high. Thread 3, the thread that is entered after the pin has just gone low, requires only 2 CPU clocks. Therefore, in the first-pass example, the high time was correctly derived, but the low time is actually shorter than was estimated.

21.4.2.5.2 Second-Pass Analysis Example

This example requires three 50% PWM waveforms: one 5 kHz (200 ms/period) and two 50 kHz (20 ms/period), each running DC motors. (Remember that the PWM function requests service from the eTPU after each high time and after each low time, so the eTPU must handle a request every 100 ms for the 5 kHz PWM and every 10 ms for the 50 MHz PWM.)

NOTE

This example uses square waves for simplicity. Notice that to use a PWM waveform in the typical way, in which the pulse is modulated, the pulse must not be modulated in a way that violates the worst-case latency requirements.

This example also uses one DIO channel monitoring a signal level every millisecond and one PPWA channel in mode 0 monitoring the speed of the 5-kHz DC motor. The PPWA must measure periods of 5 kHz (200 ms/period).

The CPU is interrupted by the channel running the PPWA function after measuring 200 periods (every 40 ms). The interrupt service routine performs an averaging of the period accumulation and checks it against a known parameter. The interrupt service time is so short and infrequent that it is a tiny fraction of total system time. The interrupt service routine contains no polling of the SDM. Therefore a realistic RCR = 0%.

First-Try System Configuration

Try a system configuration that seems likely to work. If it does not, change priority levels or channel numbers.

The 5 kHz and 50 kHz PWMs are the most time-critical functions. Those are assigned high priority. PPWA is assigned middle priority. The DIO is low performance and is assigned low priority. Refer to Table 21-33.

Table 21-33. First-Try System Configuration

Channel	Priority	Function[1], [2]
0	High	PWM at 50 kHz (needs a 4-µs WCL)
1	High	PWM at 50 kHz (needs a 4-µs WCL)
2	High	PWM at 5 kHz (needs a 40-µs WCL)
8	Middle	PPWA at 5 kHz (needs a 80-µs WCL)
15	Low	DIO as input at rate of 1 ms

[1] 0% RAM collision rate

[2] CPU clock rate = 40 MHz, or 60 ns per clock period

With this system configuration, worst-case service time for each active channel is determined as follows:

a. Longest thread of PWM is 24 CPU clocks with four RAM accesses.

   24 + ((4 RAM accesses+1) * 0 * 2 CPU clock waits) = 24 CPU clocks

   Channels 0-2 worst-case service time = 24 CPU clocks.

b. Longest thread of PPWA in mode 0 is 44 CPU clocks with nine RAM accesses.

   44 + ((9 RAM accesses +1)* 0 * 2 CPU clock waits) = 44 CPU clocks

   Channel 8 worst-case service time = 44 CPU clocks.

c. Longest thread of DIO is ten CPU clocks with four RAM accesses.

   10 + ((4 RAM accesses+1) * 0 * 2 CPU clock waits) = 10 CPU clocks

   Channel 15 worst-case service time = 10 CPU clocks.

   To find the WCL for channel 0, assume channel 0 has just finished service.

   Map the channels in the H-M-H-L-H-M-H sequence. See Figure 21-58.

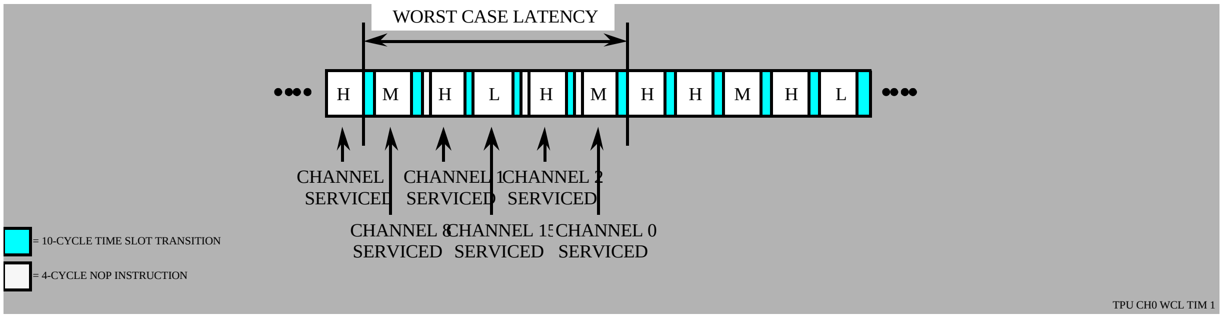


Figure 21-58. Worst-Case Latency for Channel 0 (First Try)

Conclusion: with this system configuration, worst-case latencies for channels 0 and 1 are too high (WCL for channel 1 is the same as WCL for channel 0). Try a different system configuration.

Second-Try System Configuration

The second-try system configuration is shown in Table 21-34.

Table 21-34. Second-Try System Configuration

Channel	Priority	Function[1, 2]
0	High	PWM at 50 kHz (needs a 4-µs WCL)
1	High	PWM at 50 kHz (needs a 4-µs WCL)
2	Middle	PWM at 5 kHz (needs a 40-µs WCL)
8	Middle	PPWA at 5 kHz (needs a 80-µs WCL)
15	Low	DIO as input at rate of 1 ms

[1] 0% RAM collision rate

[2] CPU clock rate = 40 MHz, or 60 ns per clock period

To find the WCL for channel 0, assume channel 0 has just finished service. Map the channels in the H-M-H-L-H-M-H sequence. See Figure 21-59.

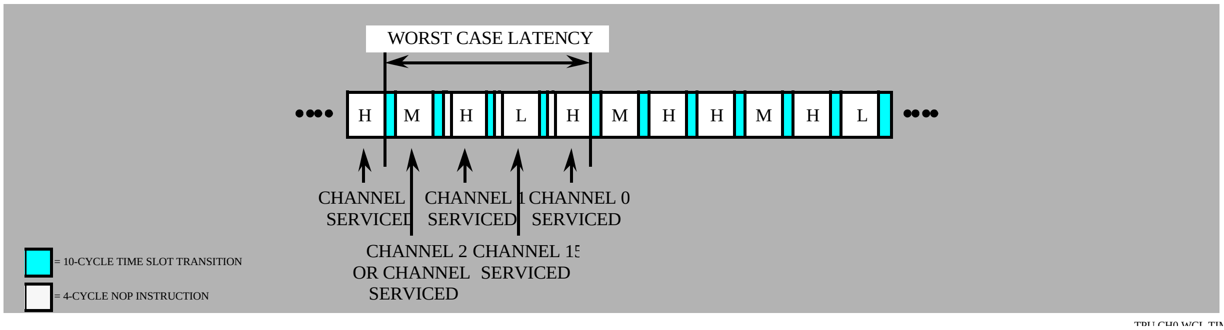


Figure 21-59. Worst-Case Latency for Channel 0 (Second Try)

Conclusion: with this system configuration, the WCL of both channel 0 and channel 1 is 3.85 ms, which is within the limit of 4 ms needed for a 50-kHz PWM.

Next, find the WCL for channel 2. Assume channel 2 has just finished service. Map the channels in the H-M-H-L-H-M-H sequence. See Figure 21-60.

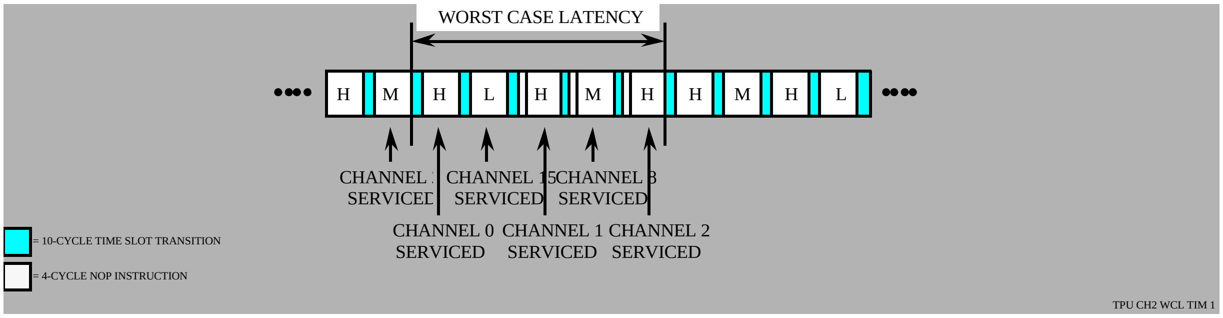


Figure 21-60. Worst-Case Latency for Channel 2

Conclusion: with this system configuration, the WCL for channels 2 and 8 is 4.7 ms, which is within the 40 and 80 ms WCL requirements.

Notice that channels 2 and 8 are well within their WCL requirements. The system could be reconfigured as shown in Table 21-35 to give channels 0 and 1 a larger margin while still keeping channels 2, 8 and 15 within their WCL requirements.

Table 21-35. Second-Try System with Channel 0 and 1 Reconfigured

Channel	Priority	Function[1], [2]
0	High	PWM at 50 kHz (needs a 10-µs WCL)
1	High	PWM at 50 kHz (needs a 10-µs WCL)
2	Middle	PWM at 5 kHz (needs a 40-µs WCL)
8	Low	PPWA at 5 kHz (needs a 80-µs WCL)
15	Low	DIO as input at rate of 1 ms

[1] 0% RAM collision rate

[2] CPU clock rate = 40 MHz, or 60 ns per clock period

21.4.3 Memory Error Service Procedures

Follows separate procedures for correctable and non-correctable error interrupt service routines. It is assumed that the non-correctable error ISR has higher priority, and cannot be preempted by the correctable error ISR. For memory error support details, see Section 21.3.6.2, “Memory Error Support”.

NOTE

The memory error support hardware does not (and is not intended to) guarantee that no error occurred will go unreported. However, combined with the procedures below, the information loss is minimized to statistical irrelevance on most scenarios.

21.4.3.1 Non-correctable Error Service Procedure

The basic procedure below can be followed in a Global Exception or a bus error machine check service routine. It handles the service for both SCM and SDM errors, as they are symmetrical at this level. However, a real implementation is likely to split the procedure in two, one for each memory.

1. confirm/identify the error source, checking assertion of the ETPUMCR bits SCMERR and SDMERR.
2. save the state(s) of ETPUMECR bit(s) CRDIS/DRDIS.
3. set the ETPUMECR bit(s) CRDIS/DRDIS=1 to disable further updates to the respective report registers, guaranteeing coherency.
4. check whether the report registers were updated by a correctable or non-correctable error; to do this, check the ETPUCERSR syndrome (for SCM) and/or the ETPUDERSR syndromes (for SDM, only those with the respective BE flags asserted). If it is a correctable error report, skip the steps 5 and 6 for the respective memory.

5. identify the error source (host, microengine or MISC/CDC) by checking the field ERR_ACC in the register(s) ETPUCERAR/ETPUDERAR; take or schedule the appropriate application-dependent recover actions.
6. update error statistics as needed; it may include recording the access source, address, data and syndrome from the registers ETPUCERAR/ETPUDERAR, ETPUCERDR/ETPUDERDR, ETPUCERSR/ETPUDERSR.
7. save the status of non-correctable error overflows flagged in the ETPUMESR bit(s) CNCOVR/DNCOVR.
8. set the ETPUMECR bit(s) CRDIS/DRDIS to the state(s) saved in the step 2. That assures the access coherency of a preempted Correctable Error Service Procedure (which may have set CRDIS/DRDIS before this procedure) is not disturbed.
9. clear ETPUMESR bits CNCOVR/DNCOVR and CNCERR/DNCERR at once, in a single write to ETPUMESR.
10. use the overrun status saved on step 7 to update the error statistics, as needed.
11. check the other Global Exception flags in ETPUMCR; take or schedule the appropriate (application dependent) recover actions.
12. clear the Global Exception, writing ETPUMCR bit GEC=1.

21.4.4 MISC Algorithm

The MISC generator is based on the following polynomial:

`G(x) = 1 + x`^{1}` + x`^{2}` + x`^{22}` + x`^{31}` (equivalent to feedback mask = 0x80400007)`

The MISC signature generation starts by clearing the MISC Accumulator value to 0 and preloading the MISC Counter with the highest SCM address. It then steps through each address decrementing the counter, reading 32 bit values and following the algorithm below:

```
If the least significant bit in MISC is 1 then
    MISC = MISC right shifted by 1 bit
    MISC = MISC XOR 0x80400007
else
    MISC = MISC right shifted by 1 bit
end if
MISC = MISC XOR RAM data
```

The code example below shows an excerpt of C code that calculates the MISC signature for a given array of data, based on the previous algorithm:

```
#define SCM_size   (MAX_SCM_ADDRESS / 4)   /* last byte address - converted to 32-bit word */
#define POLY       0x80400007              /* G(x) = 1 + x1 + x2 + x22 + x31 */

/*******************************************************************************
 FUNCTION : void calc_misc()
 PURPOSE : This function calculates the MISC value.
 INPUTS NOTES : none
 RETURNS NOTES : MISC value
 GENERAL NOTES : the array 'unsigned int data[]' represents the actual memory
                 array, organized in 32-bit words.
*******************************************************************************/
unsigned int calc_misc (void)
{
  int j; /* loop counter */

  unsigned int misc = 0;

  for (j = (SCM_size-1); j >= 0 ; j--) {   /* SCM_size has the number of 32-bit words in SCM */

      if (misc & 0x1) {
          misc >>= 1;
          misc ^= POLY;
      }
      else {
          misc >>= 1;
      }
     misc ^= data[j];         /* data[j] is the actual 32-bit word taken from the SCM array */
  }

  return (misc);          /* final signature calculated */

};
```

The value calculated by this algorithm must be loaded into register ETPUMISCCMPR prior to activating the SCM MISC calculator in eTPU. Once the MISC calculator is activated (bit SCMMISEN in register

ETPUMCR is written to 1) eTPU itself will start this procedure[1] reading the SCM whenever allowed by microengine. At the end of the cycle, when all the array has been read and the SCM signature is calculated, the Host CPU can be notified via Global Exception if the MISC Accumulator does not match the value in ETPUMISCCMPR.

The average time taken by MISC to complete the signature of the whole SCM can be given by the formula:

```
Average MISC period = S / (4 * f * (1 - L))
```

where f is clock frequency, S is SCM size in bytes and L is eTPU load (as a percentage of execution clocks over a period of time, including TST clocks).

Further detail on MISC calculation can be found on Section 21.3.6.1, “SCM Test - Multiple Input Signature Calculator.” The application note **AN2192 - Detecting Errors in the Dual Port RAM (DPTRAM) Module** is also a good source of information (although it refers to TPU) on MISC signature.

---

[1] eTPU MISC hardware is optimized to read 32-bit words from memory and to calculate this CRC in parallel, rather than shifting one bit at a time. The actual implementation inside eTPU, although bringing to the same results, does not match exactly the algorithm shown here.

Chapter 22
Error Correction Status Module (ECSM)

22.1 Introduction

The error correction status module (ECSM) provides a set of registers that configure and report ECC errors for the device including accesses to RAM and flash memory. The application may configure the device for the types of memory errors to be reported, and then query a set of read-only status and information registers to identify any errors that have been signaled.

There are two types of ECC errors: correctable and non-correctable. A correctable ECC error is generated when only one bit is wrong in a 64-bit doubleword. In this case, it is corrected automatically by hardware and no flags or other indication is set that the error occurred. A non-correctable ECC error is generated when two or more bits in a 64-bit doubleword are incorrect. Non-correctable ECC errors cause an interrupt, and if enabled, additional error details are available in the ECSM.

Error correction is implemented on 64 bits of data at a time, using eight bits for ECC for every 64-bit doubleword. ECC is checked on reads and calculated on writes per the following:

1. 64 bits containing the desired byte / halfword / word or doubleword in memory is read and ECC checked.
2. If the access is a write, then
   - The new byte / halfword / word / doubleword is merged into the 64 bits.
   - New ECC bits are calculated.
   - The 64 bits and the new ECC bits are written back.

NOTE

To use ECC with SRAM, the SRAM memory must be written to before ECC is enabled.

22.1.1 Features

The ECSM has this major feature:

- Registers for capturing information on memory errors.

22.2 Memory Map and Registers

This section provides a detailed description of all ECSM registers.

22.2.1 Module Memory Map

The ECSM memory map is shown in Table 22-1. The address of each register is given as an offset to the ECSM base address. Registers are listed in address order, identified by complete name and mnemonic, and lists the type of accesses allowed.

Table 22-1. ECSM Memory Map

Offset from ECSM_BASE_ADDR (0xFFF4_0000)	Register	Size	Access	Reset Value[1]	Section/Page
0x0000	ECSM_PCT—Processor Core Type	16 bits	RO	0xE759	22.2.2.1/22-3
0x0002	ECSM_REV—Revision	16 bits	RO	0x00	22.2.2.2/22-3
0x0004	PLAMC—Platform AXBS Master Configuration	16 bits	RO	0xFF	22.2.2.3/22-4
0x0006	PLASC—Platform AXBS Slave Configuration	16 bits	RO	0x80CF	22.2.2.4/22-5
0x0008	ECSM_IMC—Peripheral On-Platform Module Config	32 bits	RO	4847EC00	22.2.2.5/22-5
0x000F	ECSM_MRSR—Miscellaneous Reset Status	8 bits	RO	0x40	22.2.2.6/22-6
0x0010-0x0023	Reserved				
0x0024	MUDCR—Miscellaneous User-Defined Control Register	32 bits	R/W	0x40000000	22.2.2.7/22-7
0x0043	ECSM_ECR—ECC configuration register	8 bits	R/W	0x00	22.2.2.8/22-7
0x0047	ECSM_ESR—ECC status register	8 bits	R/W	0x00	22.2.2.9/22-8
0x004A	ECSM_EEGR—ECC error generation register	16 bits	R/W	0x0000	22.2.2.10/22-11
0x0050	ECSM_FEAR—Flash ECC address register	32 bits	RO	U	22.2.2.11/22-13
0x0056	ECSM_FEMR—Flash ECC master register	8 bits	RO	0x0U	22.2.2.12/22-15
0x0057	ECSM_FEAT—Flash ECC attributes register	8 bits	RO	U	22.2.2.13/22-15
0x0058	ECSM_FEDRH—Flash ECC data register high	32 bits	RO	U	22.2.2.14/22-16
0x005C	ECSM_FEDRL—Flash ECC data register low	32 bits	RO	U	22.2.2.14/22-16
0x0060	ECSM_REAR—RAM ECC address register	32 bits	RO	U	22.2.2.15/22-17
0x0065	ECSM_RESR—RAM ECC syndrome register	8 bits	RO	U	22.2.2.16/22-18
0x0066	ECSM_REMR—RAM ECC master register	8 bits	RO	0x0U	22.2.2.17/22-19
0x0067	ECSM_REAT—RAM ECC attributes register	8 bits	RO	U	22.2.2.18/22-20
0x0068	ECSM_REDRH—RAM ECC data register high	32 bits	RO	U	22.2.2.19/22-21
0x006C	ECSM_REDRL—RAM ECC data register low	32 bits	RO	U	22.2.2.19/22-21
0x0007–0x3FFF	Reserved				

[1] Please refer to the register definition. U = undefined at reset.

22.2.2 Register Descriptions

This section lists the ECSM registers in address order and describes the registers and their bit fields. Attempted accesses to reserved addresses result in an error termination; however, attempted writes to read-only registers are ignored and do not terminate with an error.

NOTE

> Unless noted otherwise, reads and writes to the programming model must match the size of the register, e.g., an *n*-bit register only supports *n*-bit writes, etc. Attempted writes of a different size than the register width produce an error termination of the bus cycle and no change to the targeted register.

22.2.2.1 Processor Core Type (ECSM_PCT)

The ECSM_PCT is a 16-bit read-only register specifying the architecture of the processor core in the device. The state of this register is defined by a module input signal; it can only be read from the peripheral programming model. Any attempted write is ignored.

See Figure 22-1 and Table 22-2 for the Processor Core Type definition.

Offset: MCM Base + 0x0000 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PCT[15:0]															
W																
Reset	1	1	1	0	0	1	1	1	0	1	0	1	1	0	0	1

Figure 22-1. Processor Core Type (ECSM_PCT) Register

Table 22-2. Processor Core Type (ECSM_PCT) Field Descriptions

Field	Description
PCT[15:0]	Processor Core Type 0xE759

22.2.2.2 Revision (ECSM_REV)

The ECSM_REV is a 16-bit read-only register specifying a revision number. The state of this register is defined by an input signal; it can only be read from the peripheral programming model. Any attempted write is ignored.

See Figure 22-2 and Table 22-2 for the Revision definition.

Offset: MCM Base + 0x0002 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	REV[15:0]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-2. Revision (ECSM_REV) Register

Table 22-3. Revision (ECSM_REV) Field Descriptions

Field	Description
REV[15:0]	Revision The REV[15:0] field is specified by an input signal to define a software-visible revision number.

22.2.2.3 Platform AXBS Master Configuration (PLAMC)

The PLAMC is a 16-bit read-only register identifying the presence/absence of bus master connections to the platform’s AMBA-AHB Crossbar Switch (AXBS). The state of this register is defined by a module input signal; it can only be read from the peripheral programming model. Any attempted write is ignored.

See Figure 22-3 and Table 22-4 for the Platform AXBS Master Configuration definition.

Offset: MCM Base + 0x0004 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	AMC[7:0]							
W																
Reset	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Figure 22-3. Platform AXBS Master Configuration (PLAMC) Register

Table 22-4. Platform AXBS Master Configuration (PLAMC) Field Descriptions

Field	Description
AMC[7:0]	AXBS Master Configuration AMC[n] = 0 if a bus master connection to AXBS input port “n” is absent AMC[n] = 1 if a bus master connection to AXBS input port “n” is present

22.2.2.4 Platform AXBS Slave Configuration (PLASC)

The PLASC is a 16-bit read-only register identifying the presence/absence of bus slave connections to the platform’s AMBA-AHB Crossbar Switch (AXBS), plus a 1-bit flag defining the internal platform datapath width (DP64). The state of this register is defined by a module input signal; it can only be read from the peripheral programming model. Any attempted write is ignored.

See Figure 22-4 and Table 22-5 for the Platform AXBS Slave Configuration definition.

Offset: MCM Base + 0x0006 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DP64	0	0	0	0	0	0	0	ASC[7:0]							
W																
Reset	1	0	0	0	0	0	0	0	1	1	0	0	1	1	1	1

Figure 22-4. Platform AXBS Master Configuration (PLAMC) Register

Table 22-5. Platform AXBS Master Configuration (PLAMC) Field Descriptions

Field	Description
DP64	64-bit Datapath DP64 = 0 if the platform datapath width is 32 bits DP64 = 1 if the platform datapath width is 64 bits
ASC[7:0]	AXBS Slave Configuration ASC[n] = 0 if a bus slave connection to AXBS output port “n” is absent ASC[n] = 1 if a bus slave connection to AXBS output port “n” is present

22.2.2.5 Peripheral Module Configuration (ECSM_IMC)

The ECSM_IMC is a 32-bit read-only register identifying the presence/absence of the 32 low-order peripheral modules connected to the primary slave bus controller. The state of this register is defined by a module input signal; it can only be read from the peripheral programming model. Any attempted write is ignored.

A ‘0’ indicates a peripheral module connection to decoded slot “n” is absent. A ‘1’ indicates a peripheral module connection to decoded slot “n” is present.

See Figure 22-4 and Table 22-5 for the Platform AXBS Slave Configuration definition.

Offset: ECSM_BASE_ADDR + 0x0008 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MC[0:15]															
W																
Reset	0	1	0	0	1	0	0	0	0	1	0	0	0	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	MC[16:31]															
W																
Reset	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0

Figure 22-5. Peripheral Module Configuration (ECSM_IMC) Register

Table 22-6. Peripheral Module Configuration (ECSM_IMC) Field Descriptions

Field	Description
MC[0:31]	Module Configuration MC[n] = 0 if a module connection to decoded slot “n” is absent MC[n] = 1 if a module connection to decoded slot “n” is present

22.2.2.6 Miscellaneous Reset Status Register (ECSM_MRSR)

The ECSM_MRSR is an 8-bit register that contains a bit for each of the reset sources to the device. An asserted bit indicates the last type of reset that occurred. Only one bit is set at any time in the ECSM_MRSR, reflecting the cause of the most recent reset as signalled by device reset input signals. The ECSM_MRSR can only be read from the peripheral programming model. Any attempted write is ignored.

Address: ECSM Base + 0x000F Access: User read/write

	0	1	2	3	4	5	6	7
R	POR	DIR	SWTR	0	0	0	0	0
W								
Reset	0	1	0	0	0	0	0	0

Figure 22-6. Miscellaneous Reset Status Register (ECSM_MRSR)

Table 22-7. Miscellaneous Reset Status Register (ECSM_MRSR) Field Descriptions

Field	Description
0 POR	Power-on Reset 1 Last recorded event was caused by a power-on reset (based on a device input signal)
1 DIR	Device-input Reset 1 Last recorded event was a reset caused by a device input reset.
2 SWTR	Platform Software Watchdog Timer Reset 1 Last recorded event was a reset caused by the platform’s software watchdog timer.
3–7	Reserved

22.2.2.7 Miscellaneous User-Defined Control Register (MUDCR)

The MUDCR provides a program-visible register for user-defined control functions.

Offset: ECSM_BASE_ADDR + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	SRAM_WS	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-7. Miscellaneous User-Defined Control Register (MUDCR)

Table 22-8. Miscellaneous User-Defined (MUDCR) Control Descriptions

Field	Description
SRAM_WS	Sram Wait State Control This bit is used to select whether the sram controller will insert 1-wait state into every read access made to the ram arrays. 1 SRAM controller is a 1-wait state controller 0 SRAM controller is a 0-wait state controller

NOTE

The SRAM_WS bit must remain at 1 when the platform, flash, and peripheral clock frequency is greater than 60 MHz.

22.2.2.8 ECC Configuration Register (ECSM_ECR)

The ECC configuration register is an 8-bit control register for specifying which types of memory errors are reported. In all systems with ECC, the occurrence of a non-correctable error causes the current access to be terminated with an error condition. In many cases, this error termination is reported directly by the initiating bus master. However, there are certain situations where the occurrence of this type of non-correctable error is not reported by the master. Examples include speculative instruction fetches that are discarded due to a change-of-flow operation and buffered operand writes. The ECC reporting logic in the ECSM provides an optional error interrupt mechanism to signal all non-correctable memory errors. In addition to the interrupt generation, the ECSM captures specific information (memory address, attributes and data, bus master number, etc.) that may be useful for subsequent failure analysis.

See Figure 22-8 and Table 22-9 for the ECC configuration register definition.

Offset: ECSM_BASE_ADDR + 0x0043 Access: User read/write

	0	1	2	3	4	5	6	7
R	0	0	ER1BR	EF1BR	0	0	ERNCR	EFNCR
W								
Reset	0	0	0	0	0	0	0	0

Figure 22-8. ECC Configuration (ECSM_ECR) Register

Table 22-9. ECSM_ECR Field Descriptions

Field	Description
0–1	Reserved
2 ER1BR	Enable RAM 1-bit reporting. 0 Reporting of single-bit RAM corrections is disabled. 1 Reporting of single-bit RAM corrections is enabled. The occurrence of a single-bit RAM correction generates an ECSM ECC interrupt request as signalled by the assertion of ECSM_ESR[R1BC]. The address, attributes and data are also captured in the ECSM_REAR, ECSM_RESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers.
3 EF1BR	Enable flash 1-bit reporting. 0 Reporting of single-bit flash corrections is disabled. 1 Reporting of single-bit flash corrections is enabled. The occurrence of a single-bit flash correction generates a MCM ECC interrupt request as signalled by the assertion of ECSM_ESR[F1BC]. The address, attributes and data are also captured in the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT and ECSM_FEDR registers.
4–5	Reserved
6 ERNCR	Enable RAM Non-Correctable Reporting. The occurrence of a non-correctable multi-bit RAM error generates an ECSM ECC interrupt request as signaled by the assertion of ECSM_ESR[RNCE]. The faulting address, attributes, and data in either the 512 KB or 80 KB array are also captured in the ECSM_REAR, ECSM_RESR, ECSM_REMR, ECSM_REAT, and ECSM_REDR registers. 0 Reporting of non-correctable RAM errors is disabled. 1 Reporting of non-correctable RAM errors is enabled.
7 EFNCR	Enable Flash Non-Correctable Reporting. The occurrence of a non-correctable multi-bit flash error generates an ECSM ECC interrupt request as signaled by the assertion of ECSM_ESR[FNCE]. The faulting address, attributes, and data are also captured in the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT, and ECSM_FEDR registers. 0 Reporting of non-correctable flash errors is disabled. 1 Reporting of non-correctable flash errors is enabled.

22.2.2.9 ECC Status Register (ECSM_ESR)

The ECC status register is an 8-bit control register for signaling which types of properly-enabled ECC events have been detected. The ECSM_ESR signals the last properly-enabled memory event to be detected. An ECC interrupt request is asserted if any flag bit is asserted and its corresponding enable bit is asserted.

The ECSM allows a maximum of one bit of the ECSM_ESR to be asserted at any given time. This preserves the association between the ECSM_ESR and the corresponding address and attribute registers, which are loaded on each occurrence of an properly-enabled ECC event. If there is a pending ECC interrupt and another properly-enabled ECC event occurs, the ECSM hardware automatically handles the ECSM_ESR reporting, clearing the previous data and loading the new state and thus guaranteeing that only a single flag is asserted.

To maintain the coherent software view of the reported event, the following sequence in the ECSM error interrupt service routine is suggested:

1. Read the ECSM_ESR and save it.
2. Read and save all the address and attribute reporting registers.
3. Re-read the ECSM_ESR and verify the current contents matches the original contents. If the two values are different, repeat from step one.
4. When the values are identical, write a 1 to the asserted ECSM_ESR flag to negate the interrupt request.

See Figure 22-9 and Table 22-10 for the ECC status register definition.

Offset: ECSM_BASE_ADDR + 0x0047 Access: User read/write

	0	1	2	3	4	5	6	7
R	0	0	R1BC	F1BC	0	0	RNCE	FNCE
W			w1c	w1c			w1c	w1c
Reset	0	0	0	0	0	0	0	0

Figure 22-9. ECC Status (ECSM_ESR) Register

Table 22-10. ECSM_ESR Field Descriptions

Field	Description
0–1	Reserved
2 R1BC	RAM 1-bit Correction. This bit can only be set if ECSM_ECR[EPR1BR] is asserted. The occurrence of a properly-enabled single-bit RAM correction generates an ECSM ECC interrupt request. The address, attributes and data are also captured in the ECSM_REAR, ECSM_RESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect 0 No reportable single-bit RAM correction has been detected. 1 A reportable single-bit RAM correction has been detected.
3 F1BC	Flash 1-bit Correction. This bit can only be set if ECSM_ECR[EPF1BR] is asserted. The occurrence of a properly-enabled single-bit flash correction generates an ECSM ECC interrupt request. The address, attributes and data are also captured in the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT and ECSM_FEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. 0 No reportable single-bit flash correction has been detected. 1 A reportable single-bit flash correction has been detected.
4–5	Reserved

Table 22-10. ECSM_ESR Field Descriptions (continued)

Field	Description
6 RNCE	RAM Non-Correctable Error. The occurrence of a properly-enabled non-correctable RAM error generates an ECSM ECC interrupt request. The faulting address, attributes, and data in either the 512K or 80K array are also captured in the ECSM_REAR, ECSM_RESR, ECSM_REMR, ECSM_REAT, and ECSM_REDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. 0 No reportable non-correctable RAM error has been detected. 1 A reportable non-correctable RAM error has been detected.
7 FNCE	Flash Non-Correctable Error. The occurrence of a properly-enabled non-correctable flash error generates an ECSM ECC interrupt request. The faulting address, attributes and data are also captured in the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT, and ECSM_FEDR registers. To clear this interrupt flag, write a 1 to this bit. Writing a 0 has no effect. 0 No reportable non-correctable flash error has been detected. 1 A reportable non-correctable flash error has been detected.

If both a flash and RAM non-correctable error occur at the same time, the ECSM records the event with the highest priority, RNCE, and finally FNCE. If both a 512K and 80K RAM non-correctable error occur at the same time, the ECSM records the event with the 512K array.

22.2.2.10 ECC Error Generation Register (ECSM_EEGR)

The ECC error generation register is a 16-bit control register used to force the generation of single- and double-bit data inversions in the memories with ECC, most notably the RAM. This capability is provided for two purposes:

- It provides a software-controlled mechanism for injecting errors into the memories during data writes to verify the integrity of the ECC logic.
- It provides a mechanism to allow testing of the software service routines associated with memory error logging.

It should be noted that while the EEGR is associated with the RAM, similar capabilities exist for the flash,i.e., the ability to program the non-volatile memory with single- or double-bit errors is supported for the same two reasons previously identified. For both types of memories (RAM and flash), the intent is to generate errors during data write cycles, such that subsequent reads of the corrupted address locations generate ECC events, either single-bit corrections or double-bit non-correctable errors that are terminated with an error response. Single-bit errors are corrected but not reported. Double-bit errors are reported, but by definition are non-correctable.

See Figure 22-10 and Table 22-11 for the ECC error generation register definition.

Offset: ECSM_BASE_ADDR + 0x004A

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FRCAP	0	FRC1BI	FR11BI	0	0	FRCNCI	FR1NCI	0	ERRBIT[0:6]						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-10. ECC Error Generation (ECSM_EEGR) Register

Table 22-11. ECSM_EEGR Field Descriptions

Field	Description
0 FRCAP	Force Platform RAM Error Injection Access Protection 0 All masters are able to generate RAM ECC errors via the EEGR register. 1 Only the master defined with as having hmaster = 0 (usually the core) can generate RAM ECC errors via the EEGR register. The assertion of this bit ensures that RAM data inversions can only occur from the master module with the master ID of 0. Since this is usually the core, this protects the RAM from errant or multiple simultaneous attempted data inversions from other master modules, and in the case of a multi-core system, ensures that only one core can issue a RAM data inversion. In LSM, the system is seen as having only one core. In DPM, only the core in lake_A, which is core_0, can perform inversions by setting this bit. The reset value of the bit is 0 and as a result, RAM data inversions can be requested from any master module. Software must ensure the proper setting of this bit.
1	Reserved

Table 22-11. ECSM_EEGR Field Descriptions (continued)

Field	Description
2 FRC1BI	Force RAM Continuous 1-Bit Data Inversions. The assertion of this bit forces the RAM controller to create 1-bit data inversions, as defined by the bit position specified in ERRBIT, continuously on every write operation. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the RAM. After this bit has been enabled to generate another continuous 1-bit data inversion, it must be cleared before being set again to properly re-enable the error generation logic. 0 No RAM continuous 1-bit data inversions are generated. 1 1-bit data inversions in the RAM are continuously generated.
3 FR11BI	Force RAM One 1-bit Data Inversion. The assertion of this bit forces the RAM controller to create one 1-bit data inversion, as defined by the bit position specified in ERRBIT, on the first write operation after this bit is set. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT is inverted to introduce a 1-bit ECC event in the RAM. After this bit has been enabled to generate a single 1-bit data inversion, it must be cleared before being set again to properly re-enable the error generation logic. 0 No RAM single 1-bit data inversion is generated. 1 One 1-bit data inversion in the RAM is generated.
4–5	Reserved
6 FRCNCI	Force RAM Continuous Non-correctable Data Inversions. The assertion of this bit forces the RAM controller to create 2-bit data inversions, as defined by the bit position specified in ERRBIT and the overall odd parity bit, continuously on every write operation. After this bit has been enabled to generate another continuous non-correctable data inversion, it must be cleared before being set again to properly re-enable the error generation logic. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the RAM. 0 No RAM continuous 2-bit data inversions are generated. 1 2-bit data inversions in the RAM are continuously generated.
7 FR1NC	Force RAM One Non-correctable Data Inversions. The assertion of this bit forces the RAM controller to create one 2-bit data inversion, as defined by the bit position specified in ERRBIT and the overall odd parity bit, on the first write operation after this bit is set. The normal ECC generation takes place in the RAM controller, but then the polarity of the bit position defined by ERRBIT and the overall odd parity bit are inverted to introduce a 2-bit ECC error in the RAM. After this bit has been enabled to generate a single 2-bit error, it must be cleared before being set again to properly re-enable the error generation logic. 0 No RAM single 2-bit data inversions are generated. 1 One 2-bit data inversion in the RAM is generated.

Table 22-11. ECSM_EEGR Field Descriptions (continued)

Field	Description
8	Reserved

9–15
ERRBIT
	Error Bit Position. The vector defines the bit position which is complemented to create the data inversion on the write operation. For the creation of 2-bit data inversions, the bit specified by this field plus the odd parity bit of the ECC code are inverted. The platform RAM controller follows a vector bit ordering scheme where LSB = 0. Errors in the ECC syndrome bits can be generated by setting this field to a value greater than the RAM width.
	For example, consider a 64-bit RAM implementation.
	The following association between the ERRBIT field and the corrupted memory bit is defined:
	`if ERRBIT = 0, then RAM[0] is inverted.`
	`if ERRBIT = 1, then RAM[1] is inverted.`
	`. . .`
	`if ERRBIT = 63, then RAM[63] is inverted.`
	`if ERRBIT = 64,then ECC Parity[0] is inverted.`
	`if ERRBIT = 65,then ECC Parity[1] is inverted.`
	`. . .`
	`if ERRBIT = 70,then ECC Parity[6] is inverted.`
	`if ERRBIT = 71,then ECC Parity[7] of the even bank is inverted.`
	`For ERRBIT values between 72 and 98, no bit position is inverted. To accommodate address bus`
	`inversions, the ERRBIT values start at 99 as defined:`
	`if ERRBIT = 99, then ADDR[3] is inverted.`
	`if ERRBIT = 100, then ADDR[4] is inverted.`
	`. . .`
	`if ERRBIT = 126, then ADDR[30] is inverted.`
	`if ERRBIT = 127, then ADDR[31] is inverted.`

NOTE

If an attempt to force a non-correctable inversion by asserting ECSM_EEGR[FRCNCI] or ECSM_EEGR[FRC1NCI], and ECSM_EEGR[ERRBIT] equals 64, no data inversion is generated.

The only allowable values for the 4 control bit enables {FR11BI, FRC1BI, FRCNCI, FR1NCI} are {0,0,0,0}, {1,0,0,0}, {0,1,0,0}, {0,0,1,0} and {0,0,0,1}. All other values result in undefined behavior.

Inversions of the address bus must be configured as non-correctable for the inversion to properly function. Address bus inversions defined as 1-bit inversions are ignored.

22.2.2.11 Flash ECC Address Register (ECSM_FEAR)

The ECSM_FEAR is a 32-bit register for capturing the address of the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC configuration register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT, and ECSM_FEDR registers and also the appropriate flag (F1BC or FNCE) in the ECC status register to be asserted.

This register is read-only; any attempted write is ignored. See Figure 22-11 and Table 22-12 for the flash ECC address register definition.

Offset: ECSM_BASE_ADDR + 0x0050 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FEAR															
W																
Reset	U[1]	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FEAR															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 22-11. Flash ECC Address (ECSM_FEAR) Register

[1] U = undefined at reset

Table 22-12. ECSM_FEAR Field Descriptions

Field	Description
0–15 FEAR	Flash ECC Address Register. Contains the faulting access address of the last, properly enabled flash ECC event.

22.2.2.12 Flash ECC Master Number Register (ECSM_FEMR)

The ECSM_FEMR is an 8-bit register for capturing the XBAR bus master number of the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT, and ECSM_FEDR registers and also the appropriate flag (FNCE) in the ECC status register to be asserted.

This register is read-only; any attempted write is ignored. See Figure 22-12 and Table 22-13 for the flash ECC master number register definition.

Offset: ECSM_BASE_ADDR + 0x0056 Access: User read-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	FEMR			
W								
Reset	0	0	0	0	U[1]	U	U	U

Figure 22-12. Flash ECC Master Number (ECSM_FEMR) Register

[1] U = undefined at reset

Table 22-13. ECSM_FEMR Field Descriptions

Field	Description
0–7 FEMR	Flash CC Master Number Register. Contains the XBAR bus master number of the faulting access of the last, properly enabled flash ECC event.

22.2.2.13 Flash ECC Attributes Register (ECSM_FEAT)

The ECSM_FEAT is an 8-bit register for capturing the XBAR bus master attributes of the last, properly-enabled ECC event in the flash memory. Depending on the state of the ECC Configuration Register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT, and ECSM_FEDR registers and also the appropriate flag (FNCE) in the ECC status register to be asserted.

This register is read-only; any attempted write is ignored. See Figure 22-13 and Table 22-14 for the flash ECC attributes register definition.

Offset: ECSM_BASE_ADDR + 0x0057 Access: User read-only

	0	1	2	3	4	5	6	7
R	WRITE	SIZE			PROTECTION			
W								
Reset	U[1]	U	U	U	U	U	U	U

Figure 22-13. Flash ECC Attributes (ECSM_FEAT) Register

[1] U = undefined at reset

Table 22-14. ECSM_FEAT Field Descriptions

Field	Description
0 WRITE	0 Read access. 1 Write access.
1–3 SIZE	000 8-bit access 001 16-bit access 010 32-bit access 011 64-bit access 1xx Reserved
4–7 PROTECTION	Cache: 0xxx Non-cacheable 1xxx Cacheable Buffer: x0xx Non-bufferable x1xx Bufferable Mode: xx0x User mode xx1x Supervisor mode Type: xxx0 I-Fetch xxx1 Data

22.2.2.14 Flash ECC Data Register (ECSM_FEDR)

The ECSM_FEDR is a 64-bit register for capturing the data associated with the last properly enabled ECC event in the flash memory. Depending on the state of the ECC configuration register, an ECC event in the flash causes the address, attributes and data associated with the access to be loaded into the ECSM_FEAR, ECSM_FEMR, ECSM_FEAT, and ECSM_FEDR registers and also the appropriate flag (F1BC or FNCE) in the ECC status register to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined.

This register is read-only; any attempted write is ignored. See Figure 22-15 and Table 22-15 for the flash ECC data register definition.

Offset: ECSM_BASE_ADDR + 0x0058 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FEDR[0:15]															
W																
Reset	U[1]	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FEDR[16:31]															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 22-14. Flash ECC Data High (ECSM_FEDRH) Register

[1] U = undefined at reset

Offset: ECSM_BASE_ADDR + 0x005C

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FEDR[32:47]															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	FEDR[48:63]															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 22-15. Flash ECC Data Low (ECSM_FEDRL) Register

Table 22-15. PECSM_FEDR Field Descriptions

Field	Description
0–63 FEDR	Flash ECC Data Register. Contains the data associated with the faulting access of the last properly enabled flash ECC event. The register contains the data value taken directly from the platform data bus.

22.2.2.15 RAM ECC Address Register (ECSM_REAR)

The ECSM_REAR is a 32-bit register for capturing the address of the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC configuration register, an ECC event in the RAM causes the address, attributes and data associated with the access to be loaded into the ECSM_REAR, ECSM_RESR, ECSM_REMR, ECSM_REAT, and ECSM_REDR registers and also the appropriate flag (RNCE) in the ECC status register to be asserted.

This register is read-only; any attempted write is ignored. See Figure 22-16 and Table 22-16 for the RAM ECC address register definition.

Offset: ECSM_BASE_ADDR + 0x0060

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	REAR															
W																
Reset	U[1]	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	REAR															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 22-16. RAM ECC Address (ECSM_REAR) Register

[1] U = undefined at reset

Table 22-16. ECSM_REAR Field Descriptions

Field	Description
0–31 REAR	RAM ECC Address Register. Contains the faulting access address of the last, properly-enabled RAM ECC event.

22.2.2.16 RAM ECC Syndrome Register (ECSM_RESR)

The ECSM_RESR is an 8-bit register for capturing the error syndrome of the last properly enabled ECC event in the RAM memory. Depending on the state of the ECC configuration register, an ECC event in the RAM causes the address, attributes, and data associated with the access to be loaded into the ECSM_REAR, ECSM_RESR, ECSM_REMR, ECSM_REAT and ECSM_REDR registers, and the appropriate flag (R1BC or RNCE) in the ECC status register to be asserted.

This register is read-only; any attempted write is ignored. See Figure 22-17 and Table 22-17 for the RAM ECC syndrome register definition.

Offset: ECSM_BASE_ADDR + 0x0065 Access: User read-only

	0	1	2	3	4	5	6	7
R	RESR[0:7]							
W								
Reset	U[1]	U	U	U	U	U	U	U

Figure 22-17. RAM ECC Syndrome (ECSM_RESR) Register

[1] U = undefined at reset

Table 22-17. PECSM_RESR Field Descriptions

Field	Description
0–7 RESR	RAM ECC Syndrome Register. This 8-bit syndrome field includes 7 bits of Hamming decoded parity plus an odd-parity bit for the entire 72-bit (64-bit data + 8 ECC) code word. The upper 7 bits of the syndrome specify the exact bit position in error for single-bit correctable codewords, and the combination of a non-zero 7-bit syndrome plus overall incorrect parity bit signal a multi-bit, non-correctable error. For correctable single-bit errors, the mapping shown inTable 22-18 associates the upper 7 bits of the syndrome with the data bit in error.

Table 22-18. RAM Syndrome Mapping for Single-Bit Correctable Errors

PRESR[7:0]	Data Bit in Error	PRESR[7:0]	Data Bit in Error	PRESR[7:0]	Data Bit in Error
0x00	No Error	0x34	DATA[41]	0x8A	DATA[14]
0x01	ECC[0]	0x37	DATA[24]	0x8C	DATA[7]
0x02	ECC[1]	0x38	DATA[29]	0x91	DATA[33]
0x04	ECC[2]	0x40	ECC[6]	0x92	DATA[44]
0x07	DATA[42]	0x43	DATA[1]	0x94	DATA[21]
0x08	ECC[3]	0x45	DATA[49]	0x98	DATA[37]
0x0B	DATA[13]	0x46	DATA[32]	0x9E	DATA[2]

Table 22-18. RAM Syndrome Mapping for Single-Bit Correctable Errors (continued)

PRESR[7:0]	Data Bit in Error	PRESR[7:0]	Data Bit in Error	PRESR[7:0]	Data Bit in Error
0x0D	DATA[26]	0x49	DATA[36]	0xA1	DATA[8]
0x0E	DATA[12]	0x4A	DATA[6]	0xA2	DATA[58]
0x10	ECC[4]	0x4C	DATA[5]	0xA4	DATA[25]
0x13	DATA[50]	0x51	DATA[60]	0xA7	DATA[46]
0x15	DATA[4]	0x52	DATA[48]	0xA	DATA[45]
0x16	DATA[22]	0x54	DATA[47]	0xB0	DATA[55]
0x19	DATA[16]	0x58	DATA[31]	0xC1	DATA[0]
0x1A	DATA[17]	0x61	DATA[35]	0xC2	DATA[62]
0x1C	DATA[57]	0x62	DATA[52]	0xC4	DATA[27]
0x20	ECC[5]	0x64	DATA[23]	0xC8	DATA[43]
0x23	DATA[54]	0x68	DATA[39]	0xD0	DATA[63]
0x25	DATA[10]	0x70	DATA[53]	0xD9	DATA[9]
0x26	DATA[18]	0x75	DATA[28]	0xE0	DATA[59]
0x29	DATA[20]	0x80	ECC[7]	0xEA	DATA[19]
0x2A	DATA[15]	0x83	DATA[3]	0xF8	DATA[11]
0x2C	DATA[61]	0x85	DATA[51]	Others	Multiple bit error
0x31	DATA[56]	0x86	DATA[34]		
0x32	DATA[40]	0x89	DATA[38]		

22.2.2.17 RAM ECC Master Number Register (ECSM_REMR)

The ECSM_REMR is an 8-bit register for capturing the XBAR bus master number of the last, properly-enabled ECC event in the RAM memory. Depending on the state of the ECC configuration register, an ECC event in the RAM causes the address, attributes, and data associated with the access to be loaded into the ECSM_REAR, ECSM_RESR, ECSM_REMR, ECSM_REAT, and ECSM_REDR registers and also the appropriate flag (RNCE) in the ECC status register to be asserted.

This register is read-only; any attempted write is ignored. See Figure 22-18 and Table 22-19 for the RAM ECC master number register definition.

Offset: ECSM_BASE_ADDR + 0x0066

Access: User read-only

	0	1	2	3	4	5	6	7
R	0	0	0	0	REMR			
W								
Reset	0	0	0	0	U[1]	U	U	U

Figure 22-18. RAM ECC Master Number (ECSM_REMR) Register

[1] U = undefined at reset

Table 22-19. ECSM_REMR Field Descriptions

Field	Description
0–7 REMR	RAM ECC Master Number Register. Contains the XBAR bus master number of the faulting access of the last, properly-enabled RAM ECC event.

22.2.2.18 RAM ECC Attributes Register (ECSM_REAT)

The ECSM_REAT is an 8-bit register for capturing the XBAR bus master attributes of the last properly enabled ECC event in the RAM memory. Depending on the state of the ECC configuration register, an ECC event in the RAM causes the address, attributes, and data associated with the access to be loaded into the ECSM_REAR, ECSM_RESR, ECSM_REMR, ECSM_REAT, and ECSM_REDR registers and also the appropriate flag (RNCE) in the ECC status register to be asserted.

This register is read-only; any attempted write is ignored. See Figure 22-19 and Table 22-20 for the RAM ECC attributes register definition.

Offset: ECSM_BASE_ADDR + 0x0067 Access: User read-only

	0	1	2	3	4	5	6	7
R	WRITE	SIZE			PROTECTION			
W								
Reset	U[1]	U	U	U	U	U	U	U

Figure 22-19. RAM ECC Attributes (ECSM_REAT) Register

[1] U = undefined at reset

Table 22-20. ECSM_REAT Field Descriptions

Field	Description
0 WRITE	0 Read access. 1 Write access.
1–3 SIZE	000 8-bit access. 001 16-bit access. 010 32-bit access. 011 64-bit access. 1xx Reserved.
4–7 PROTECTION	Cache: 0xxx Non-cacheable. 1xxx Cacheable. Buffer: x0xx Non-bufferable. x1xx Bufferable. Mode: xx0x User mode. xx1x Supervisor mode. Type: xxx0 I-Fetch. xxx1 Data.

22.2.2.19 RAM ECC Data Register (ECSM_REDR)

The ECSM_REDR is a 64-bit register for capturing the data associated with the last properly enabled ECC event in the RAM memory. Depending on the state of the ECC configuration register, an ECC event in the RAM causes the address, attributes, and data associated with the access to be loaded into the ECSM_REAR, ECSM_RESR, ECSM_REMR, ECSM_REAT, and ECSM_REDR registers and also the appropriate flag (R1BC or RNCE) in the ECC status register to be asserted.

The data captured on a multi-bit non-correctable ECC error is undefined. This register is read-only; any attempted write is ignored. See Figure 22-21 and Table 22-21 for the RAM ECC data register definition.

Offset: ECSM_BASE_ADDR + 0x0068 Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	REDR[0:15]															
W																
Reset	U[1]	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	REDR[16:31]															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 22-20. RAM ECC Data High (ECSM_REDRH) Register

[1] U = undefined at reset

Offset: ECSM_BASE_ADDR + 0x006C Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	REDR[32:47]															
W																
Reset	U[1]	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	REDR[48:63]															
W																
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Figure 22-21. RAM ECC Data Low (ECSM_REDRL) Register

[1] U = undefined at reset

Table 22-21. REDR Field Descriptions

Field	Description
0–63 REDR	RAM ECC Data Register. Contains the data associated with the faulting access of the last properly enabled platform RAM ECC event. The register contains the data value taken directly from the platform data bus.

Chapter 23
External Bus Interface (EBI)

23.1 Introduction

23.1.1 Overview

The External Bus Interface (EBI) handles the transfer of information between the internal busses and the memories or peripherals in the external address space. The EBI includes a memory controller that generates interface signals to support a variety of external memories. This includes Single Data Rate (SDR) burst mode flash, SRAM, and asynchronous memories. It supports up to 4 regions (via chip selects), each with its own programmed attributes.

NOTE

The External Bus Interface is implemented differently on the MPC5676R in comparison to the MPC55xx family.

On the MPC5676R, references to the External Bus Interface (EBI) or Calibration Bus Interface (CBI) refer to the same single bus interface, as described in this chapter.

23.1.2 Features

- 22-Bit Address bus with transfer size indication
- 16-Bit Data bus (32-bit Data Bus in muxed mode)
- Multiplexed Address on Data pins
- Memory controller with support for various memory types:
  - synchronous burst SDR flash and SRAM
  - asynchronous/legacy flash and SRAM
- Burst support (wrapped only)
- Bus monitor
- Port size configuration per chip select (16 or 32 bits)
- Configurable wait states
- Configurable internal or external transfer acknowledge ($\overline{\text{D\_TA}}$) per chip select
- Support for Dynamic Calibration with up to 4 chip-selects
- Four Write/Byte Enable ($\overline{\text{D\_WE}}$[0:3]) signals
- Slower-speed clock modes
- Stop and Module Disable Modes for power savings

- Optional automatic D_CLKOUT gating to save power and reduce EMI
- Misaligned access support (for chip-select accesses only)

23.1.3 Signal Naming

Generic signal names like ADDR and DATA have been replaced with signal names according to the table below. Refer to the Signals chapter for the full signal names and muxing information (Table 23-1 lists the function partial-signal name).

Table 23-1. Device-Specific Signal Naming

Generic Signal Name	Equivalent Chip-level Signal Name[1]	I/O Type	Function
ADDR	D_ADD[9:30]	I/O	Address bus
$\overline{\text{BDIP}}$	$\overline{\text{D\_BDIP}}$	Output	Burst Data in Progress
CLKOUT	D_CLKOUT[2]	Output	Clockout
$\overline{\text{CS}}$	$\overline{\text{D\_CS}}$[0:3]	—	—
$\overline{\text{CAL\_CS}}$		Output	Chip Selects
DATA	D_ADD_DAT[0:15] (through muxing D_ADD_DAT[16:31] are available)	I/O	Data bus[3]
$\overline{\text{OE}}$	$\overline{\text{D\_OE}}$	Output	Output Enable
RD_$\overline{\text{WR}}$	D_RD_$\overline{\text{WR}}$	I/O	Read_Write
$\overline{\text{TA}}$	$\overline{\text{D\_TA}}$	I/O	Transfer Acknowledge
$\overline{\text{TEA}}$	$\overline{\text{D\_TEA}}$	I/O	Transfer Error Acknowledge
$\overline{\text{TS}}$	$\overline{\text{D\_TS}}$	I/O	Transfer Start
$\overline{\text{WE/BE}}$	$\overline{\text{D\_WE}}$[0:3]	Output	Write/Byte Enables
ALE	D_ALE	Output	Address Latch Enable

NOTES:

[1] This pin function may not be the pins primary function, Refer to the Signals chapter for muxing information.

[2] The D_CLKOUT signal is driven by the System Clock Block outside the EBI.

[3] In Address/Data multiplexing modes, Data will also show the address during the address phase.

23.1.4 Modes of Operation

The mode of the EBI is determined by the MDIS, and AD_MUX bits in the EBI_MCR. See Section 23.3.1.1, "EBI Module Configuration Register (EBI_MCR) for details. Slower-speed modes, Debug Mode, Stop Mode, and Factory Test Mode are modes that the MCU may enter, in parallel to the EBI being configured in one of its block-specific modes.

23.1.4.1 Single Master Mode

In Single Master Mode, the EBI responds to internal requests matching one of its regions, but ignores all externally-initiated bus requests. The MCU is the only master allowed to initiate transactions on the external bus in this mode; therefore, it acts as a parked master and does not have to arbitrate for the bus before starting each cycle. Single Master Mode is entered when MDIS=0 in the EBI_MCR.

23.1.4.2 Module Disable Mode

The Module Disable Mode is used for MCU power management. The clock to the non-memory mapped logic in the EBI can be stopped while in Module Disable Mode. Logic on the MCU external to the EBI is needed to fully implement the Module Disable Mode (to shut off the clock). Internal master requests made to the external bus in Module Disable Mode are terminated with transfer error (internally, no external D_TEA assertion). Module Disable Mode is entered when MDIS=1 in the EBI_MCR.

23.1.4.3 Stop Mode

The EBI supports the SIU Halt Mode mechanism used for MCU power management. When a request is made to enter Stop Mode (controlled by the EBI bit in the SIU_HLT register), the EBI block completes any pending bus transactions and acknowledges the stop request. After the acknowledgement, the MCU shuts off the system clock input, and sets the EBI bit in the SIU_HLTACK register. While the clocks are shut off, the EBI is not accessible. While in stop mode, accesses to the EBI from the internal master will terminate with transfer error (internally, no external D_TEA assertion).

23.1.4.4 Slower-Speed Modes

In slower-speed modes, the external D_CLKOUT frequency is divided (by 2, 3, etc.) compared with that of the internal system bus. The EBI behavior remains dictated by the mode of the EBI, except that it drives and samples signals at the D_CLKOUT frequency rather than the internal system frequency. This mode is selected by writing to the SIU_ECCR[EBDF] field.

23.1.4.5 16-Bit Data Bus Mode

For MCUs that have only 16 data bus signals pinned out, or for systems where the use of a different multiplexed function (e.g. GPIO) is desired on 16 of the 32 data pins, the EBI supports a 16-bit Data Bus Mode. In this mode, only 16 data signals are used by the EBI. The user can select which 16 data signals are used (D_ADD_DAT[0:15] or D_ADD_DAT[16:31]) by writing the D16_31 bit in the EBI_MCR.

For EBI-mastered accesses, the operation in 16-bit Data Bus Mode (DBM=1, PS=x) is similar to a chip-select access to a 16-bit port in 32-bit Data Bus Mode (DBM=0, PS=1), except for the case of a non-chip-select access of exactly 32-bit size.

EBI-mastered non-chip-select accesses of exactly 32-bit size are supported via a two (16-bit) beat burst for both reads and writes. See Section 23.4.2.9, “Non-Chip-Select Burst in 16-bit Data Bus Mode. Non-chip-select transfers of non-32-bit size are supported in standard non-burst fashion.

16-bit Data Bus Mode is entered when DBM=1 in the EBI_MCR.

23.1.4.6 Multiplexed Address on Data Bus Mode

This mode covers several cases aimed at reducing pin count on MCU and external components. In this mode, the D_ADD_DAT pins will drive the address value on the first clock of the cycle (while $\overline{\text{D\_TS}}$ is asserted).

The memory controller supports per-chip-select selection of multiplexing address/data through the EBI_CAL_BRx[AD_MUX] bit.

Address on Data bus multiplexing also supports the 16-bit data bus mode (MCR[DBM]=1) and 16-bit memories (EBI_CAL_ORx[PS]=1). The user can select which 16 device balls are used (D_ADD_DAT[0:15] or D_ADD_DAT[16:31]) by writing the D16_31 bit in the EBI_MCR. For either setting of D16_31, the 16 LSBs of external address are driven onto the selected 16 D_ADD_DAT pins. If additional address lines are required to interface to the memory, then non-muxed address pins (e.g. D_ADD[9:15]) or the unused 16 D_ADD_DAT signals may be used to complete the address space.

The EBI drives the unused 16 D_ADD_DAT signals with the MSBs of the external address, padded with 8 zero bits. This allows the device to optionally use these signals for the upper 8 external address lines instead of the separate non-muxed address signals.

For more details (e.g. timing diagrams), see Section 23.4.2.11, “Address Data Multiplexing.”

The Address Latch Enable (ALE) indicates when the address is present during a multiplexed bus access using a D_ADD_DAT signal. This can be used in conjunction with an external latch to hold the state of the address access if connecting the MCU to a non-multiplexed bus compatible memory. ALE signal timing is shown in Figure 23-1.

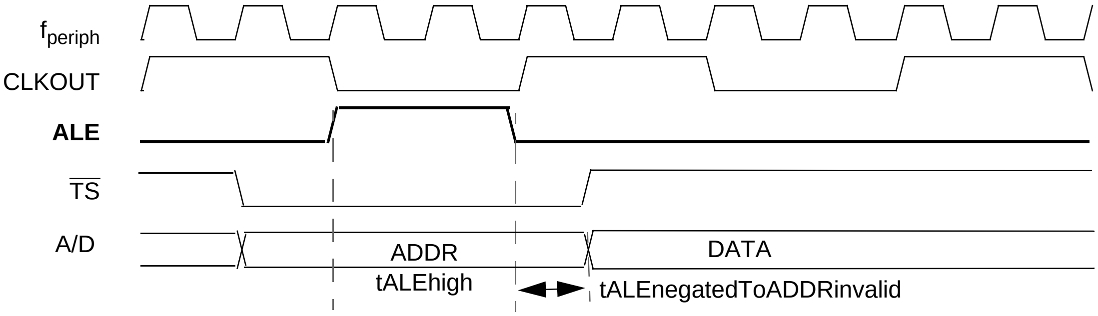


Figure 23-1. ALE signal timing

There are at least two timing requirements that relate to external components for multiplexed addr/data bus in systems using ALE. These are not EBI specs. The two are

- ALE - minimum high time
- ALE negated to ADDR invalid

These values depend on maximum clkout frequency and the actual clock tree insertion on the ALE clock gate.

Typical values should be calculated (and specified) as:

tALEhigh = f(Clkout) / 2 -1nS (rise/fall uncertainty) tALEToADDRInvalid = 1-3nS (depending on specified external part).

23.1.4.7 Debug Mode

When the MCU is in Debug Mode, the EBI behavior is unaffected and remains dictated by the mode of the EBI.

23.2 External Signal Description

23.2.1 Overview

Table 23-2 lists the external pins used by the EBI.

Table 23-2. Signal Properties

Name	I/O Type	Function
D_ADD[9:30]	Output	Address bus
D_BDIP	Output	Burst Data in Progress
D_CLKOUT[1]	Output	Clockout
CAL_CS[0:3]	Output	Chip Selects
D_ADD_DAT[0:31]	I/O	Address/Data bus[2]
D_OE	Output	Output Enable
D_RD_WR	Output	Read_Write
D_TA	I/O	Transfer Acknowledge
D_TEA	I/O	Transfer Error Acknowledge
D_TS	Output	Transfer Start
D_WE[0:3]	Output	Write/Byte Enables
D_ALE	Output	Address Latch Enable

NOTES:

[1] The D_CLKOUT signal is driven by the System Clock Block outside the EBI.

[2] In non-multiplexed mode, D_ADD_DAT[0:15] provide data only.

23.2.2 Address/Data Bus Configurations

Table 23-3 shows the function of the external pins in each of the possible muxed/non-muxed usage configurations allowed for this device.

Table 23-3. Function of EBI Pins for All Possible Configurations

Mode	D_WE[0:3]	D_ADD[9:15]	D_ADD[16:30][1] D_ADD_DAT[16:30][2]	D_ADD_DAT [0:15]	D_CS2[3]
Non-muxed 16-bit mode	write/byte enable [0:1]	Address 9:15	Address 16:30	Data 0:15	Address 31
Muxed 32-bit mode	write/byte enable [0:3]	Not used	Address 16:30 / Data 16:30	Address 0:15 / Data 0:15	Address 31 / Data 31
Muxed 16-bit mode (EBI_MCR[D16_31]=1)	write/byte enable [0:1]	Address 9:15	Address 16:30 / Data 0:14	Not used	Address 31 / Data 15
Muxed 16-bit mode (EBI_MCR[D16_31]=0)	write/byte enable [0:1]	Address 9:15	Not used	Address 16:31 / Data 0:15	Not used

NOTES:

[1] D_ADD[16:30] SIU PCR functionality must be selected in non-multiplexed mode (AD_MUX = 0)

[2] D_ADD_DAT[16:30] SIU PCR functionality must be selected in multiplexed mode (AD_MUX = 1)

[3] D_CS2 is the primary function. Secondary function is EBI data only in non-mux mode and address/data in mux mode.

23.2.3 Detailed Signal Descriptions

23.2.3.1 D_ADD [9:30] — Address Lines 9-30

The D_ADD[9:30] signals specify the physical address of the bus transaction.

The 22 address lines correspond to bits 9-30 of the EBI’s 32-bit internal address bus. Table 23-3 lists the different available configurations of these signals.

23.2.3.2 D_ALE - Address Latch Enable

The D_ALE signal is asserted to indicate that an address is valid during a multiplexed addr/data access. See Section 23.1.4.6, “Multiplexed Address on Data Bus Mode” and Figure 23-1 for details.

23.2.3.3 D_BDIP — Burst Data in Progress

D_BDIP is asserted to indicate that the master is requesting another data beat following the current one.

This signal is driven by the EBI on all EBI-mastered external burst cycles, but is only sampled by burst mode memories that have a corresponding pin. See Section 23.4.2.5, “Burst Transfer.

23.2.3.4 D_CLKOUT — Clockout

D_CLKOUT is a general-purpose clock output signal to connect to the clock input of SDR external memories and in some cases to the input clock of another MCU in multi-master configurations.

23.2.3.5 CAL_CS [0:3] — Calibration Chip Selects 0-3

CAL_CSx is asserted by the EBI to indicate that this transaction is targeted for a particular memory bank on the external bus.

See Section 23.4.1.4, “Memory Controller with Support for Various Memory Types for details on chip-select operation.

23.2.3.6 D_ADD_DAT [0:31] — Data Lines 0-31

The D_ADD_DAT[0:31] signals contain the data to be transferred for the current transaction.

D_ADD_DAT[0:31] is driven by the EBI when it initiates a write transaction to an external device.

D_ADD_DAT[0:31] is driven by an external device during a read transaction from the EBI.

For 8-bit and 16-bit transactions, the byte lanes not selected for the transfer do not supply valid data.

D_ADD_DAT[0:31] is driven by the EBI in the address phase with the address value if the Address on Data multiplexing mode is enabled. See Section 23.1.4.6, “Multiplexed Address on Data Bus Mode,” for details.

In 16-bit Data Bus Mode, (or for chip-select accesses to a 16-bit port), only D_ADD_DAT[0:15] or D_ADD_DAT[16:31] are used by the EBI, depending on the setting of the D16_31 bit in the EBI_MCR. See Section 23.1.4.5, “16-Bit Data Bus Mode.

Table 23-3 lists the different available configurations of these signals.

23.2.3.7 D_OE — Output Enable

D_OE is used to indicate when an external memory is permitted to drive back read data. External memories must have their data output buffers off when D_OE is negated. D_OE is only asserted for chip-select accesses.

For read cycles, D_OE is asserted one clock after D_TS assertion and held until the termination of the transfer. For write cycles, D_OE is negated throughout the cycle.

23.2.3.8 D_RD_WR — Read / Write

D_RD_WR indicates whether the current transaction is a read access or a write access.

D_RD_WR is driven in the same clock as the assertion of D_TS and valid address, and is kept valid until the cycle is terminated.

23.2.3.9 D_TA — Transfer Acknowledge

D_TA is asserted to indicate that the slave has received the data (and completed the access) for a write cycle, or returned data for a read cycle. If the transaction is a burst read, D_TA is asserted for each one of the transaction beats. For write transactions, D_TA is only asserted once at access completion, even if more than one write data beat is transferred.

D_TA is driven by the EBI when the access is controlled by the chip selects (and SETA=0). Otherwise, D_TA is driven by the slave device to which the current transaction was addressed.

See Section 23.4.2.8, “Termination Signals Protocol for more details.

23.2.3.10 D_TEA — Transfer Error Acknowledge

D_TEA is asserted by either the EBI or an external device to indicate that an error condition has occurred during the bus cycle.

D_TEA is asserted by the EBI when the internal bus monitor detected a timeout error.

See Section 23.4.2.8, “Termination Signals Protocol for more details.

23.2.3.11 D_TS — Transfer Start

D_TS is asserted by the EBI to indicate the start of a transaction on the external bus.

D_TS is only asserted for the first clock cycle of the transaction, and is negated in the successive clock cycles until the end of the transaction.

23.2.3.12 D_WE [0:3] — Write/Byte Enables 0-3

Write enables are used to enable program operations to a particular memory. These signals can also be used as byte enables for read and write operation by setting the WEBS bit in the appropriate Base Register. D_WE[0:3] are only asserted for chip-select accesses.

For chip-select accesses to a 16-bit port, only D_WE[0:1] are used by the EBI, regardless of which half of the D_ADD_DAT bus is selected via the D16_31 bit in the EBI_MCR.

See Section 23.4.1.10, “Four Write/Byte Enable (D_WE) Signals for more details on D_WE[0:3] functionality.

23.2.4 Signal Output Buffer Enable

If the EBI is disabled (MDIS=1), the EBI does not drive any of the signals at the balls. When the EBI is enabled and an EBI function is selected with the PCR PA field, the EBI signal at the ball is driven, subject to the constraints described in Table 23-4, and regardless of the state of the OBE bit in the PCR, which remains unchanged. The drive strength of the signal is determined by the signal's PCR setting.

Table 23-4. Signal Output Buffer Enable

Signal	Output
D_ADD[9:30]	Enabled
D_BDIP	Enabled
CAL_CS[0:3]	Enabled
D_ADD_DAT[0:31]	Only enabled during write access or on Address phase when Addr/Data muxing is enabled.
D_OE	Enabled
D_RD_WR	Enabled
D_TA	Only enabled during chip-select SETA=0 access
D_TEA	Only enabled for 2 cycles when timeout occurs
D_TS	Enabled
D_WE[0:3]	Enabled
D_ALE	Enabled

23.3 Memory Map/Register Definition

Table 23-5 shows the EBI registers.

Table 23-5. EBI Address Map

Address	Use
EBI_BASE	EBI Module Configuration Register (EBI_MCR)
EBI_BASE+0x4	Reserved
EBI_BASE+0x8	EBI Transfer Error Status Register (EBI_TESR)
EBI_BASE+0xC	EBI Bus Monitor Control Register (EBI_BMCR)
EBI_BASE+0x10 – EBI_BASE+0x3C	Reserved
EBI_BASE+0x40	EBI Calibration Base Register Bank 0 (EBI_CAL_BR0)
EBI_BASE+0x44	EBI Calibration Option Register Bank 0 (EBI_CAL_OR0)
EBI_BASE+0x48	EBI Calibration Base Register Bank 1 (EBI_CAL_BR1)
EBI_BASE+0x4C	EBI Calibration Option Register Bank 1 (EBI_CAL_OR1)
EBI_BASE+0x50	EBI Calibration Base Register Bank 2 (EBI_CAL_BR2)
EBI_BASE+0x54	EBI Calibration Option Register Bank 2 (EBI_CAL_OR2)
EBI_BASE+0x58	EBI Calibration Base Register Bank 3 (EBI_CAL_BR3)
EBI_BASE+0x5C	EBI Calibration Option Register Bank 3 (EBI_CAL_OR3)

23.3.1 Register Descriptions

NOTE

> Other than the exceptions noted below, EBI registers must not be written while a transaction to the EBI is in progress (or within 2 D_CLKOUT cycles after a transaction has just completed, to allow internal state machines to go IDLE). In those cases, the behavior is undefined.
>
> Exceptions that can be written while an EBI transaction is in progress:
>
> - All bits in EBI_TESR
>
> See Section 23.5.1, “Booting from External Memory for related application information.

23.3.1.1 EBI Module Configuration Register (EBI_MCR)

EBI_BASE+0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ACGE	0	0	0	1	0	0	0	0	MDIS	0	0	0	D16_31	AD_MUX	DBM
W		▨	▨	▨	▨	▨	▨	▨	▨		▨	▨	▨			
RESET:	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 23-2. EBI Module Configuration Register (EBI_MCR)

The EBI Module Configuration Register contains bits which configure various attributes associated with EBI operation.

ACGE - Automatic D_CLKOUT Gating Enable

The ACGE bit enables the EBI feature of turning off D_CLKOUT (holding it high) during idle periods in-between external bus accesses.
1 = Automatic D_CLKOUT Gating is enabled
0 = Automatic D_CLKOUT Gating is disabled

MDIS — Module Disable Mode

The MDIS bit controls an internal EBI “enable clk” signal which can be used to control the clocks to the EBI. The MDIS bit allows the clock to be stopped to the non-memory mapped logic in the EBI, effectively putting the EBI in a software controlled power-saving state. See Section 23.1.4.2, “Module Disable Mode for more information. No external bus accesses can be performed when the EBI is in Module Disable Mode (MDIS=1).

1 = Module Disable Mode is active (negate “enable clk” signal)
0 = Module Disable Mode is inactive (assert “enable clk” signal)

D16_31 — Data Bus 16_31 Select

The D16_31 bit controls whether the EBI uses the D_ADD_DAT[0:15] or D_ADD_DAT[16:31] signals, when in 16-bit Data Bus Mode (DBM=1) or for chip-select accesses to a 16-bit port (PS=1). For systems using A/D muxing with a 16-bit port, it is recommended to set D16_31 to 1.
1 = D_ADD_DAT[16:31] signals are used for 16-bit port accesses
0 = D_ADD_DAT[0:15] signals are used for 16-bit port accesses

AD_MUX — Address on Data Bus Multiplexing Mode

The AD_MUX bit controls whether non-chip-select accesses have the address driven on the data bus in the address phase of a cycle.
1 = Address on Data Multiplexing Mode is used for non-CS accesses.
0 = Only Data on data pins for non-CS accesses.

DBM — Data Bus Mode

The DBM bit controls whether the EBI is in 32-bit or 16-bit Data Bus Mode.
1 = 16-bit Data Bus Mode is used
0 = 32-bit Data Bus Mode is used

23.3.1.2 EBI Transfer Error Status Register (EBI_TESR)

EBI_BASE+0x8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TEAF	BMTF
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 23-3. EBI Transfer Error Status Register (EBI_TESR)

The EBI Transfer Error Status Register contains a bit for each type of transfer error on the external bus. A bit set to logic 1 indicates what type of transfer error occurred since the last time the bits were cleared. Each bit can be cleared by reset or by writing a 1 to it. Writing a 0 has no effect. This register cannot be written when the MDIS bit is set in the EBI_MCR.

TEAF — Transfer Error Acknowledge Flag

This bit is set if the cycle was terminated by an externally generated $\overline{\text{D\_TEA}}$ signal.
1 = External $\overline{\text{D\_TEA}}$ occurred
0 = No error

BMTF — Bus Monitor Timeout Flag

This bit is set if the cycle was terminated by a bus monitor timeout.

1 = Bus monitor timeout occurred
0 = No error

23.3.1.3 EBI Bus Monitor Control Register (EBI_BMCR)

EBI_BASE+0xC

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BMT								BME	0	0	0	0	0	0	0
W										▨	▨	▨	▨	▨	▨	▨
RESET:	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 23-4. EBI Bus Monitor Control Register (EBI_BMCR)

The EBI Bus Monitor Control Register controls the timeout period of the bus monitor and whether it is enabled or disabled.

BMT —Bus Monitor Timing

This field defines the timeout period, in 8 external bus clock resolution, for the Bus Monitor. See Section 23.4.1.6, “Bus Monitor for more details on bus monitor operation.

Timeout Period = (2 + (8 * BMT)) / external bus clock frequency.

BME —Bus Monitor Enable

This bit controls whether the bus monitor is enabled for internal to external bus cycles. The BME bit is ignored (treated as 0) for chip-select accesses with internal $\overline{\text{D\_TA}}$ (SETA=0).

1 = Enable bus monitor (for external $\overline{\text{D\_TA}}$ accesses only)
0 = Disable bus monitor

23.3.1.4 EBI Base Registers (EBI_CAL_BR0-3)

EBI_BASE+0x40, EBI_BASE+0x48, EBI_BASE+0x50, EBI_BASE+0x58

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BA[1]															
RESET:	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BA	0	0	LWRN	PS	EOE		SBL	AD_ MUX	BL	WEBS	TBDIP	GCSN	SETA	BI	V
W		▨	▨													
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

▨ = Unimplemented or Reserved

NOTES:

[1] BA[0:2] is fixed as 0b001.

Figure 23-5. EBI Base Registers (EBI_CAL_BR0-3)

The EBI Base Registers are used to define the base address and other attributes for the corresponding chip select.

BA — Base Address

These bits are compared to the corresponding unmasked address signals among D_ADD[0:16] of the internal address bus to determine if a memory bank controlled by the memory controller is being accessed by an internal bus master.

LWRN — Late RD_$\overline{\text{WR}}$ Negation

The LWRN bit determines the timing of RD_$\overline{\text{WR}}$ signal negation for a write transfer (except for SETA=1 transfers, where RD_$\overline{\text{WR}}$ negates with $\overline{\text{CS}}$ by default, so LWRN has no effect).

1 = Negate RD_$\overline{\text{WR}}$ the same cycle as $\overline{\text{CS}}$ negation. See Figure 23-20.
0 = Negate RD_$\overline{\text{WR}}$ one cycle earlier than $\overline{\text{CS}}$ negation. See Figure 23-16.

PS — Port Size

The PS bit determines the data bus width of transactions to this chip-select bank.

NOTE

In the case where the DBM bit in EBI_MCR is set for 16-bit Data Bus Mode, the PS bit value is ignored and is always treated as a '1' (16-bit port).

1 = 16-bit port
0 = 32-bit port

EOE — Early $\overline{\text{OE}}$

The EOE field determines the timing of $\overline{\text{OE}}$ signal assertion for a read transfer. When EBI_BR[ADMUX]=1, the EOE value is ignored and treated as 0b00 (in order to avoid contention on shared address/data bus for muxed transfers).

Table 23-6. EOE Values

Value	Timing of $\overline{\text{OE}}$ Assertion
0b00[1]	Assert $\overline{\text{OE}}$ 1 CLKOUT cycle after $\overline{\text{TS}}$ assertion (see Figure 23-10)
0b01	Assert $\overline{\text{OE}}$ the same CLKOUT cycle as $\overline{\text{TS}}$ assertion (see Figure 23-14)
0b10	Assert $\overline{\text{OE}}$ one internal clock cycle after $\overline{\text{TS}}$ assertion (see Figure 23-14)[2]
0b11	Reserved

NOTES:

[1] The 0b00 timing case also applies when EBI_BR[ADMUX]=1, regardless of EOE value in Base Register.

[2] In divided bus speed modes (where the EBI runs at a slower frequency than the internal system bus), the EBI uses the internal higher-frequency clock to assert $\overline{\text{OE}}$ partway through the CLKOUT cycle where $\overline{\text{TS}}$ is asserted. In 1:1 bus speed mode (where CLKOUT and internal system bus run at same frequency), the behavior for EOE Value 0b10 is identical to that of Value 0b00.

SBL — Short Burst Length

The SBL bit provides support for a 2-word external burst. When SBL=1, the number of beats in a burst is automatically determined by the EBI to be 2 or 4 according to the Port Size (PS bit), regardless of BL bit value. When SBL=0, the number of beats in a burst is determined by the BL bit. See Table 23-7 under BL bit description for SBL and BL encodings.

AD_MUX — Address on Data Bus Multiplexing

The AD_MUX bit controls whether accesses for this chip select have the address driven on the data bus in the address phase of a cycle

1 = Address on Data Multiplexing Mode is enabled for this chip select.
0 = Address on Data Multiplexing Mode is disabled for this chip select.

BL — Burst Length

The BL bit (along with SBL bit) determines the amount of data transferred in a burst for this chip select, measured in 32-bit words. When SBL=0, the number of beats in a burst is automatically determined by the EBI to be 4, 8, or 16 according to the Port Size (PS bit) so that the burst fetches the number of words chosen by BL. When SBL=1, the BL bit value is a don’t care. See Table 23-7 for SBL and BL encodings.

Table 23-7. SBL, BL Values

Value		Burst Length[1]	PS	# Beats in Burst[2]
SBL	BL			
0	0	8-word[3]	0 (32-bit)	8
			1 (16-bit)	16
0	1	4-word	0 (32-bit)	4
			1 (16-bit)	8
1	X	2 word[3]	0 (32-bit)	2
			1 (16-bit)	4

NOTES:

[1] Total amount of data fetched in a burst transfer, measured in 32-bit words.

[2] Number of external data beats used in external burst transfer. The size of each beat is determined by PS value.

[3] A word always refers to 32-bits of data, regardless of PS.

WEBS — Write Enable / Byte Select

This bit controls the functionality of the D_WE[0:3] signals.

1 = The D_WE[0:3] signals function as BE[0:3]
0 = The D_WE[0:3] signals function as WE[0:3]

TBDIP — Toggle Burst Data in Progress

This bit determines how long the D_BDIP signal is asserted for each data beat in a burst cycle. See Section 23.4.2.5.1, “TBDIP Effect on Burst Transfer for details.

1 = Only assert D_BDIP (BSCY+1) external cycles before expecting subsequent burst data beats
0 = Assert D_BDIP throughout the burst cycle, regardless of wait state configuration

GCSN — Guarantee CS Negation

The GCSN bit allows support for guaranteeing that the EBI will negate CS between all back-to-back transfer cases (even those that are part of a set of Small Accesses). See Figure 23-25.

1 = Negate CS between all external transfers. This adds an extra dead cycle for some back-to-back cases.
0 = Default operation (CS may or may not negate between transfers, depending on the particular back-to-back case).

SETA — Select External Transfer Acknowledge

The SETA bit controls whether accesses for this chip select will terminate (end transfer without error) based on externally asserted D_TA or internally asserted D_TA. SETA should only be set when the BI bit is 1 as well, since burst accesses with SETA=1 are not supported. Setting SETA=1 causes the BI bit to be ignored (treated as 1, burst inhibited).

1 = Transfer Acknowledge (D_TA) is an input to the EBI, data phase will be terminated by an external device
0 = Transfer Acknowledge (D_TA) is an output from the EBI, data phase will be terminated by the EBI

BI — Burst Inhibit

This bit determines whether or not burst read accesses are allowed for this chip-select bank. The BI bit is ignored (treated as 1) for chip-select accesses with external $\overline{\text{D\_TA}}$ (SETA=1).

1 = Disable burst accesses for this bank. This is the default value out of reset (or when SETA=1).
0 = Enable burst accesses for this bank

V — Valid bit

The user writes this bit to indicate that the contents of this Base Register and Option Register pair are valid. The appropriate $\overline{\text{CS}}$ signal does not assert unless the corresponding V-bit is set.

1 = This bank is valid
0 = This bank is not valid

23.3.1.5 EBI Option Registers (EBI_CAL_OR0-3)

EBI_BASE+0x44, EBI_BASE+0x4C, EBI_BASE+0x54, EBI_BASE+0x5C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	AM[1]															
RESET:	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	AM	0	0	0	0	0	0	0	SCY				0	BSCY		0
W		▨	▨	▨	▨	▨	▨	▨					▨			▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

NOTES:

[1] AM[0:2] is set to a fixed value of 0b111.

Figure 23-6. EBI Option Registers (EBI_CAL_OR0-3)

The EBI Option Registers are used to define the address mask and other attributes for the corresponding chip select.

AM — Address Mask

This field allows masking of any corresponding bits in the associated Base Register. Masking the address independently allows external devices of different size address ranges to be used. Any clear bit masks the corresponding address bit. Any set bit causes the corresponding address bit to be used in comparison with the address pins. Address mask bits can be set or cleared in any order in the field, allowing a resource to reside in more than one area of the address map. This field can be read or written at any time.

SCY — Cycle length in clocks

This field represents the number of wait states (external cycles) inserted after the address phase in the single transfer case, or in the first beat of a burst, when the memory controller handles the external memory access. Values range from 0 to 15. This is the main parameter for determining the length of the cycle. These bits are ignored when SETA=1.

The total cycle length for the first beat (including the $\overline{\text{D\_TS}}$ cycle) = (2+SCY) external clock cycles.

See Section 23.5.3.1, “Example Wait State Calculation for related application information.

BSCY — Burst beats length in clocks

This field determines the number of wait states (external cycles) inserted in all burst beats except the first, when the memory controller starts handling the external memory access and thus is using SCY[0:3] to determine the length of the first beat. These bits are ignored when SETA=1.

The total memory access length for each beat is (1 + BSCY) external clock cycles.

The total cycle length (including the $\overline{\text{D\_TS}}$ cycle) = (2+SCY) + (#beats[1]-1) * (BSCY+1).

Table 23-8. BSCY Values

Value	Meaning
00	0-clock cycle wait states (1 clock per data beat)
01	1-clock cycle wait states (2 clocks per data beat)
10	2-clock cycle wait states (3 clocks per data beat)
11	3-clock cycle wait states (4 clocks per data beat)

23.4 Functional Description

23.4.1 External Bus Interface Features

23.4.1.1 Address Bus (up to 22 available on pins)

Valid transaction sizes are 8, 16 and 32 bits. Only 22 address lines are pinned out externally, but a full 32-bit decode is done internally to determine the target of the transaction and whether a chip select should be asserted.

23.4.1.2 32-Bit Data Bus (16-bit Data Bus Mode also supported)

The entire 32-bit data bus is available (through muxing) for external memory accesses. There is also a 16-bit Data Bus Mode available via the DBM bit in EBI_MCR. See Section 23.1.4.5, “16-Bit Data Bus Mode.

23.4.1.3 Multiplexed Address on Data Pins

When this mode is enabled, the address shows up on the data pins during the address phase of the cycle. This mode can be enabled separately for non-chip-select accesses and per chip-select access. See Section 23.1.4.6, “Multiplexed Address on Data Bus Mode.

23.4.1.4 Memory Controller with Support for Various Memory Types

The EBI contains a memory controller that supports a variety of memory types, including synchronous burst mode flash and SRAM, and asynchronous/legacy flash and SRAM with a compatible interface.

Each of the four chip select banks is configured via its own pair of Base and Option Registers. Each time an internal to external bus cycle access is requested, the internal address is compared with the base address of each valid Base Register (with 17 bits having mask). See Figure 23-7. If a match is found, the attributes

---

[1] #beats is the number of beats (4,8,16) determined by BL and PS bits in Base Register.

defined for this bank in its BR and OR are used to control the memory access. If a match is found in more than one bank, the lowest bank matched handles the memory access (e.g., bank 0 is selected over bank 1).

This means chip select 0 has the highest priority and chip select 3 the lowest.

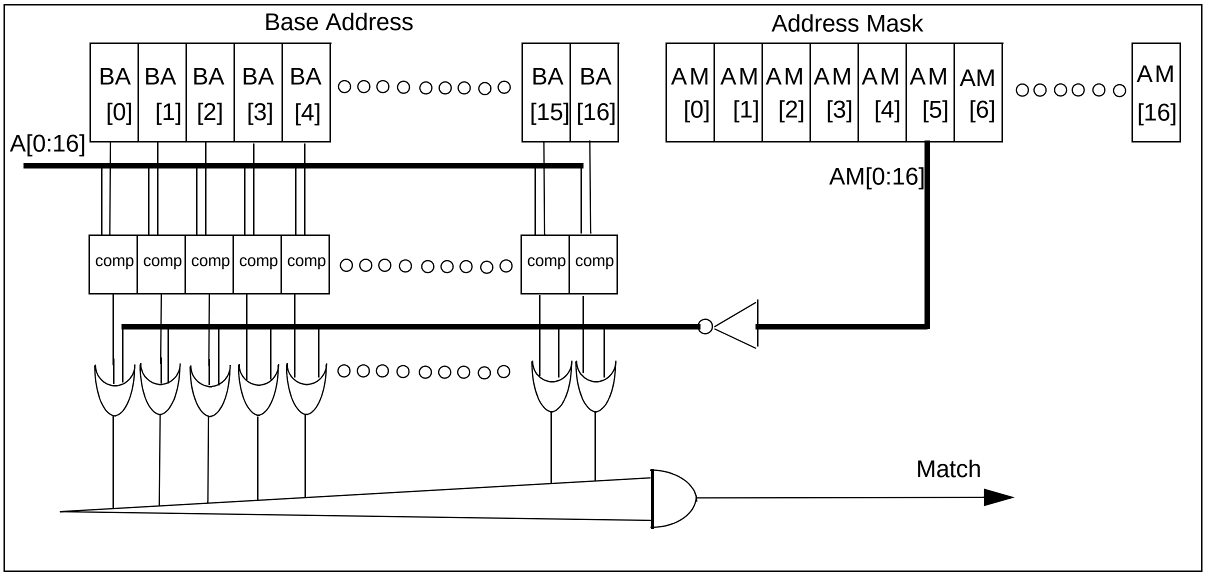


Figure 23-7. Bank Base Address & Match Structure

When a match is found on one of the chip-select banks, all its attributes (from the appropriate Base and Option Registers) are selected for the functional operation of the external memory access, such as:

- Number of wait states for a single memory access, and for any beat in a burst access
- Burst enable
- Port size for the external accessed device

See Section 23.3.1.4, "EBI Base Registers (EBI_CAL_BR0-3) and Section 23.3.1.5, "EBI Option Registers (EBI_CAL_OR0-3) for a full description of all chip-select attributes.

When no match is found on any of the chip-select banks, the default transfer attributes shown in Table 23-9 are used.

Table 23-9. Default Attributes for Non-Chip-Select Transfers

CS Attribute	Default Value	Comment
PS	0	32-bit port size
BL	0	burst length is don't care since burst is disabled
WEBS	0	write enables
TBDIP	0	don't care since burst is disabled
BI	1	burst inhibited
SCY	0	don't care since external $\overline{D\_TA}$ is used
BSCY	0	don't care since external $\overline{D\_TA}$ is used
AD_MUX	0	Address on Data multiplexing (depends on EBI_MCR[AD_MUX] value)

Table 23-9. Default Attributes for Non-Chip-Select Transfers (continued)

CS Attribute	Default Value	Comment
SETA	1	Select external D_TA to terminate access
SBL	0	don't care since burst is disabled
EOE	0	OE not used for non-CS access
GCSN	00	N/A for non-CS access
LWRN	0	default timing

23.4.1.5 Burst Support (wrapped only)

The EBI supports burst read accesses of external burstable memory. To enable bursts to a particular memory region, clear the BI (Burst Inhibit) bit in the appropriate Base Register. External burst lengths of 4 and 8 words are supported. Burst length is configured for each chip select by using the BL bit in the appropriate Base Register. See Section 23.4.2.5, “Burst Transfer for more details on burst operation.

In 16-bit data bus mode (DBM=1 in EBI_MCR), a special 2-beat burst case is supported for reads and writes for 32-bit non-chip-select accesses only. This is to allow 32-bit coherent accesses to another MCU. See Section 23.4.2.9, “Non-Chip-Select Burst in 16-bit Data Bus Mode.

Bursting of accesses that are not controlled by the chip selects is not supported for any other case besides the special case of 32-bit accesses in 16-bit data bus mode.

Burst writes are not supported for any other case besides the special case of 32-bit non-chip-select writes in 16-bit data bus mode. Internal requests to write >32 bits (such as a cache line) externally are broken up into separate 32-bit or 16-bit external transactions according to the port size. See Section 23.4.2.6, “Small Accesses (Small Port Size and Short Burst Length) for more detail on these cases.

23.4.1.6 Bus Monitor

When enabled (via the BME bit in the EBI_BMCR), the bus monitor detects when no $\overline{\text{D\_TA}}$ assertion is received within a maximum timeout period for external $\overline{\text{D\_TA}}$ accesses. The timeout for the bus monitor is specified by the BMT field in the EBI_BMCR. Each time a timeout error occurs, the BMTF bit is set in the EBI_TESR. The timeout period is measured in external bus (D_CLKOUT) cycles. Thus the effective real-time period is multiplied (by 2, 3, etc.) when a slower-speed mode is used, even though the BMT field itself is unchanged.

23.4.1.7 Port Size Configuration per Chip Select (16 or 32 bits)

The EBI supports memories with data widths of 16 or 32 bits. The port size for a particular chip select is configured by writing the PS bit in the corresponding Base Register.

23.4.1.8 Configurable Wait States

From 0 to 15 wait states can be programmed for any cycle that the memory controller generates, via the SCY bits in the appropriate Option Register. From 0 to 3 wait states between burst beats can be programmed using the BSCY bits in the appropriate Option Register.

23.4.1.9 Configurable internal or external D_TA per chip select

Each chip select can be configured (via the SETA bit) to have D_TA driven internally (by the EBI), or externally (by an external device). See Section 23.3.1.4, “EBI Base Registers (EBI_CAL_BR0-3)” for more details on SETA bit usage.

23.4.1.10 Four Write/Byte Enable (D_WE) Signals

The functionality of the D_WE[0:3] signals depends on the value of the WEBS bit in the corresponding Base Register. Setting WEBS to 1 configures these pins as BE[0:3], while resetting it to 0 configures them as WE[0:3]. WE[0:3] are asserted only during write accesses, while BE[0:3] is asserted for both read and write accesses. The timing of the D_WE[0:3] signals remains the same in either case.

The upper Write/Byte Enable (D_WE0) indicates that the upper eight bits of the data bus (D_ADD_DAT[0:7]) contain valid data during a write/read cycle. The upper middle Write/Byte Enable (D_WE1) indicates that the upper middle eight bits of the data bus (D_ADD_DAT[8:15]) contain valid data during a write/read cycle. The lower middle Write/Byte Enable (D_WE2) indicates that the lower middle eight bits of the data bus (D_ADD_DAT[16:23]) contain valid data during a write/read cycle. The lower Write/Byte Enable (D_WE3) indicates that the lower eight bits of the data bus (D_ADD_DAT[24:31]) contain valid data during a write/read cycle.

NOTE

The exception to the preceding D_WE description is that for 16-bit port transfers (DBM=1 or PS=1), only the D_WE[0:1] signals are used, regardless of whether D_ADD_DAT[0:15] or D_ADD_DAT[16:31] are selected (via the D16_31 bit in the EBI_MCR). This means for the case where D_ADD_DAT[16:31] are selected, that WE0 indicates that D_ADD_DAT[16:23] contains valid data, and WE1 indicates that D_ADD_DAT[24:31] contains valid data.

The Write/Byte Enable lines affected in a transaction for a 32-bit port (PS = 0) and a 16-bit port (PS=1) are shown in Table 23-10. Only Big Endian byte ordering is supported by the EBI.

Table 23-10. Write/Byte Enable Signals Function [1]

Transfer Size	Address		32-Bit Port Size				16-Bit Port Size[2]			
	A30	A31	$\overline{\text{D\_WE}}$ 0	$\overline{\text{D\_WE}}$ 1	$\overline{\text{D\_WE}}$ 2	$\overline{\text{D\_WE}}$ 3	$\overline{\text{D\_WE}}$ 0	$\overline{\text{D\_WE}}$ 1	$\overline{\text{D\_WE}}$ 2	$\overline{\text{D\_WE}}$ 3
Byte	0	0	X				X			
	0	1		X				X		
	1	0			X		X			
	1	1				X		X		
16-bit	0	0	X	X			X	X		
	1	0			X	X	X	X		
32-bit	0	0	X	X	X	X	X[3]	X[3]		
Burst	0	0	X	X	X	X	X	X		

NOTES:

[1] This table applies to aligned internal master transfers only. In the case of a misaligned internal master transfer that is split into multiple aligned external transfers, not all of the write enables X'd in the table will necessarily assert. See Section 23.4.2.10, "Misaligned Access Support.

[2] Also applies when DBM=1 for 16-bit data bus mode.

[3] This case consists of two 16-bit external transactions, but for both transactions the $\overline{\text{D\_WE}}$[0:1] signals are the only $\overline{\text{D\_WE}}$ signals affected.

23.4.1.11 Slower-Speed Clock Modes

For memories that cannot run with a full-speed external bus, the EBI supports slower-speed clock modes. Refer to Section 23.1.4.4, "Slower-Speed Modes for more details on this feature. The timing diagrams for slower-speed modes are identical to those for full-speed mode, except that the frequency of D_CLKOUT is reduced.

23.4.1.12 Stop and Module Disable Modes for Power Savings

See Section 23.1.4, "Modes of Operation for a description of the power saving modes.

23.4.1.13 Optional Automatic D_CLKOUT Gating

The EBI has the ability to hold the external D_CLKOUT pin high when the EBI's internal master state machine is idle and no requests are pending. The EBI outputs a signal to the pads logic in the MCU to disable D_CLKOUT. This feature is disabled out of reset, and can be enabled or disabled by the ACGE bit in the EBI_MCR.

NOTE

This feature must be disabled for multi-master systems. In those cases, one master is getting its clock source from the other master and needs it to stay valid continuously.

23.4.1.14 Misaligned Access Support

The EBI has limited misaligned access support. Misaligned non-burst chip-select transfers from internal masters are supported. The EBI aligns the accesses when it sends them out to the external bus (splitting them into multiple aligned accesses if necessary), so that external devices are not required to support misaligned accesses. Burst accesses (internal master) must match the internal bus size (64-bit aligned). See Section 23.4.2.10, “Misaligned Access Support” for more details.

23.4.2 External Bus Operations

The following sections provide a functional description of the external bus, the bus cycles provided for data transfer operations, and error conditions.

23.4.2.1 External Clocking

Possible division factors for D_CLKOUT: 1, 2, and 4.

The D_CLKOUT signal sets the frequency of operation for the bus interface directly. Internally, the MCU uses a phase-locked loop (PLL) circuit to generate a master clock for all of the MCU circuitry (including the EBI) which is phase-locked to the D_CLKOUT signal. In general, all signals for the EBI are specified with respect to the rising-edge of the D_CLKOUT signal, and they are guaranteed to be sampled as inputs or changed as outputs with respect to that edge.

23.4.2.2 Reset

Upon detection of internal reset assertion, the EBI immediately ends all transactions (abruptly, not through normal termination protocol), and ignores any transaction requests that take place while reset is asserted.

23.4.2.3 Basic Transfer Protocol

The basic transfer protocol defines the sequence of actions that must occur on the external bus to perform a complete bus transaction. A simplified scheme of the basic transfer protocol is shown in Figure 23-8.

Figure 23-8. Basic Transfer Protocol

The arbitration phase is where bus ownership is requested and granted. This phase is not needed in Single Master Mode because the EBI is the permanent bus owner in this mode.

The address transfer phase specifies the address for the transaction and the transfer attributes that describe the transaction. The signals related to the address transfer phase are $\overline{\text{D\_TS}}$, D_ADD (or D_ADD_DAT if Address/Data multiplexing is used), $\overline{\text{CS}}$[0:3], D_RD_$\overline{\text{WR}}$, and $\overline{\text{D\_BDIP}}$. The address and its related signals (with the exception of $\overline{\text{D\_TS}}$, $\overline{\text{D\_BDIP}}$) are driven on the bus with the assertion of the $\overline{\text{D\_TS}}$ signal, and kept valid until the bus master receives $\overline{\text{D\_TA}}$ asserted (the EBI holds them one cycle beyond $\overline{\text{D\_TA}}$ for writes and external D_TA accesses). Note that for writes with internal D_TA, D_RD_WR is not held one cycle past D_TA.

The data transfer phase performs the transfer of data, from master to slave (in write cycles) or from slave to master (on read cycles), if any is to be transferred. The data phase may transfer a single beat of data (1-4 bytes) for non-burst operations or a 2-beat (special DBM=1 case only), 4-beat, 8-beat, or 16-beat burst of data (2 or 4 bytes per beat depending on Port Size) when burst is enabled. On a write cycle, the master must not drive write data until after the address transfer phase is complete. This is to avoid electrical contentions when switching between drivers. The master must start driving write data one cycle after the address transfer cycle. The master can stop driving the data bus as soon as it samples the D_TA line asserted on the rising edge of D_CLKOUT. To facilitate asynchronous write support, the EBI keeps driving valid write data on the data bus until 1 clock after the rising edge where D_RD_WR and WE are negated (for chip-select accesses only). See Figure 23-16 for an example of write timing. On a read cycle, the master accepts the data bus contents as valid on the rising edge of the D_CLKOUT in which the D_TA signal is sampled asserted. See Figure 23-10 for an example of read timing.

The termination phase is where the cycle is terminated by the assertion of either D_TA (normal termination) or D_TEA (termination with error). Termination is discussed in detail in Section 23.4.2.8, “Termination Signals Protocol.

23.4.2.4 Single Beat Transfer

The flow and timing diagrams in this section assume that the EBI is configured in Single Master Mode. Therefore, arbitration is not needed and is not shown in these diagrams.

23.4.2.4.1 Single Beat Read Flow

The handshakes for a single beat read cycle are illustrated in the following flow and timing diagrams.

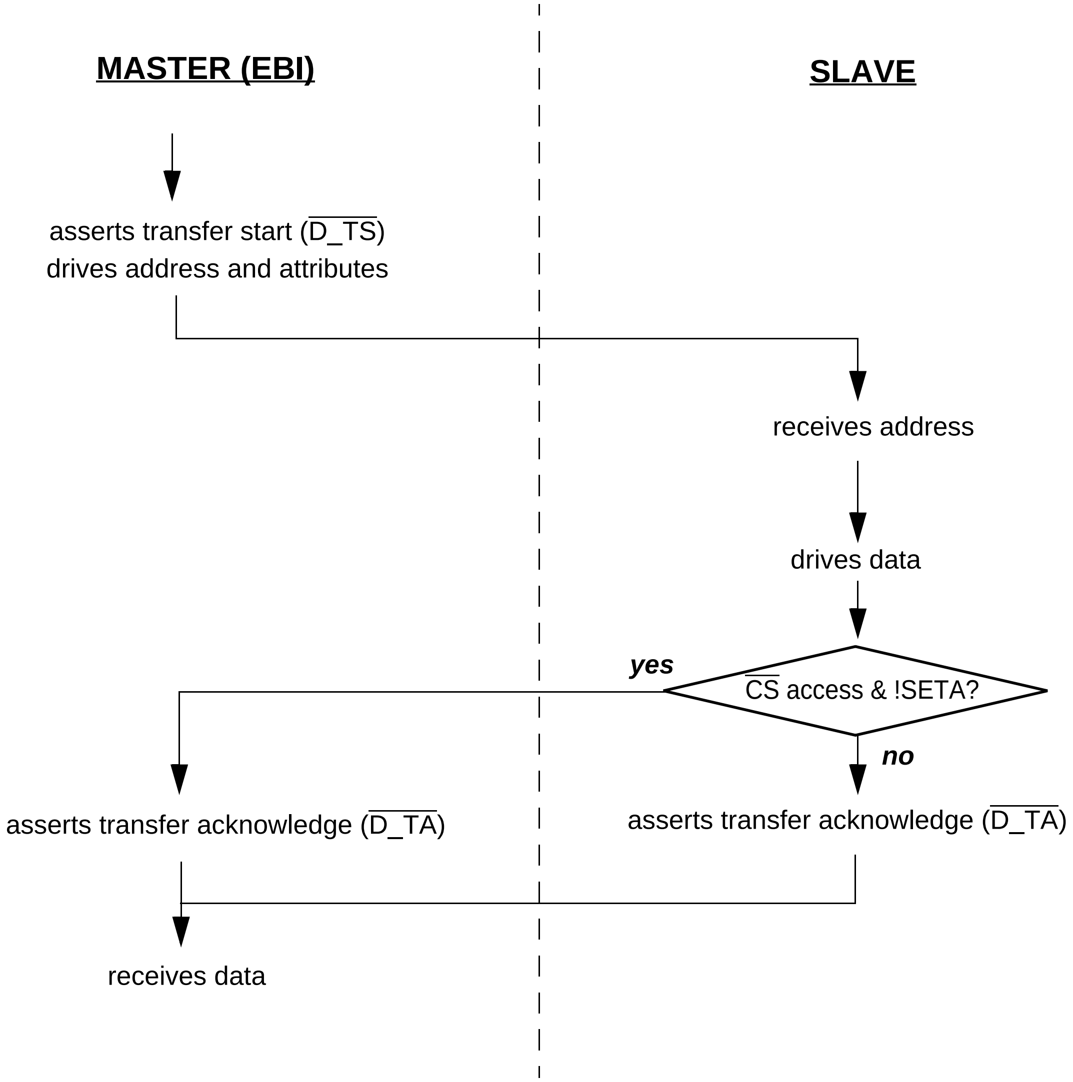


Figure 23-9. Basic Flow Diagram of a Single Beat Read Cycle

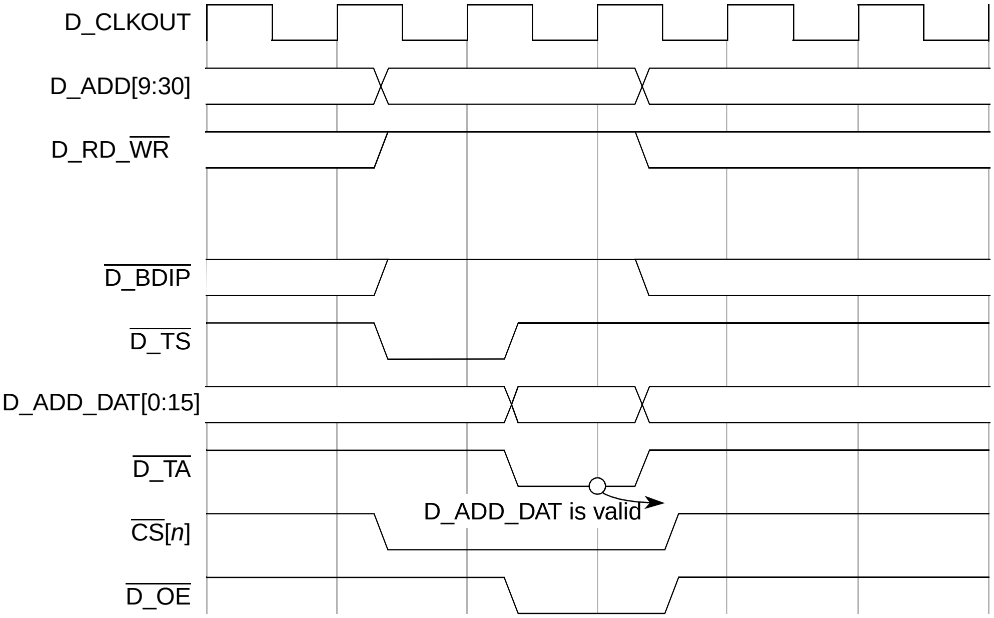


Figure 23-10. Single Beat 16-bit Read Cycle, CS Access, Zero Wait States

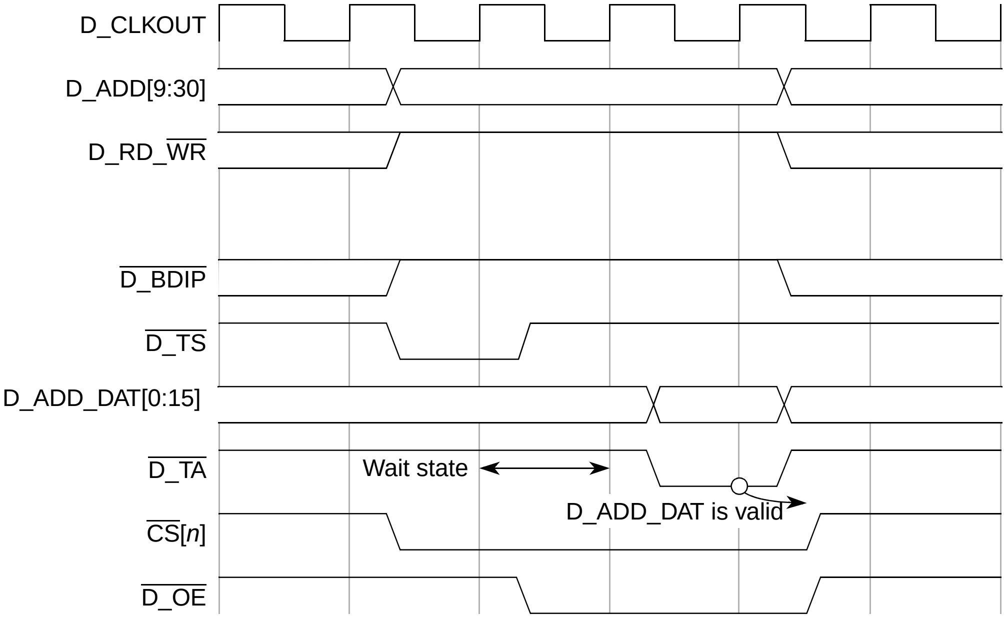
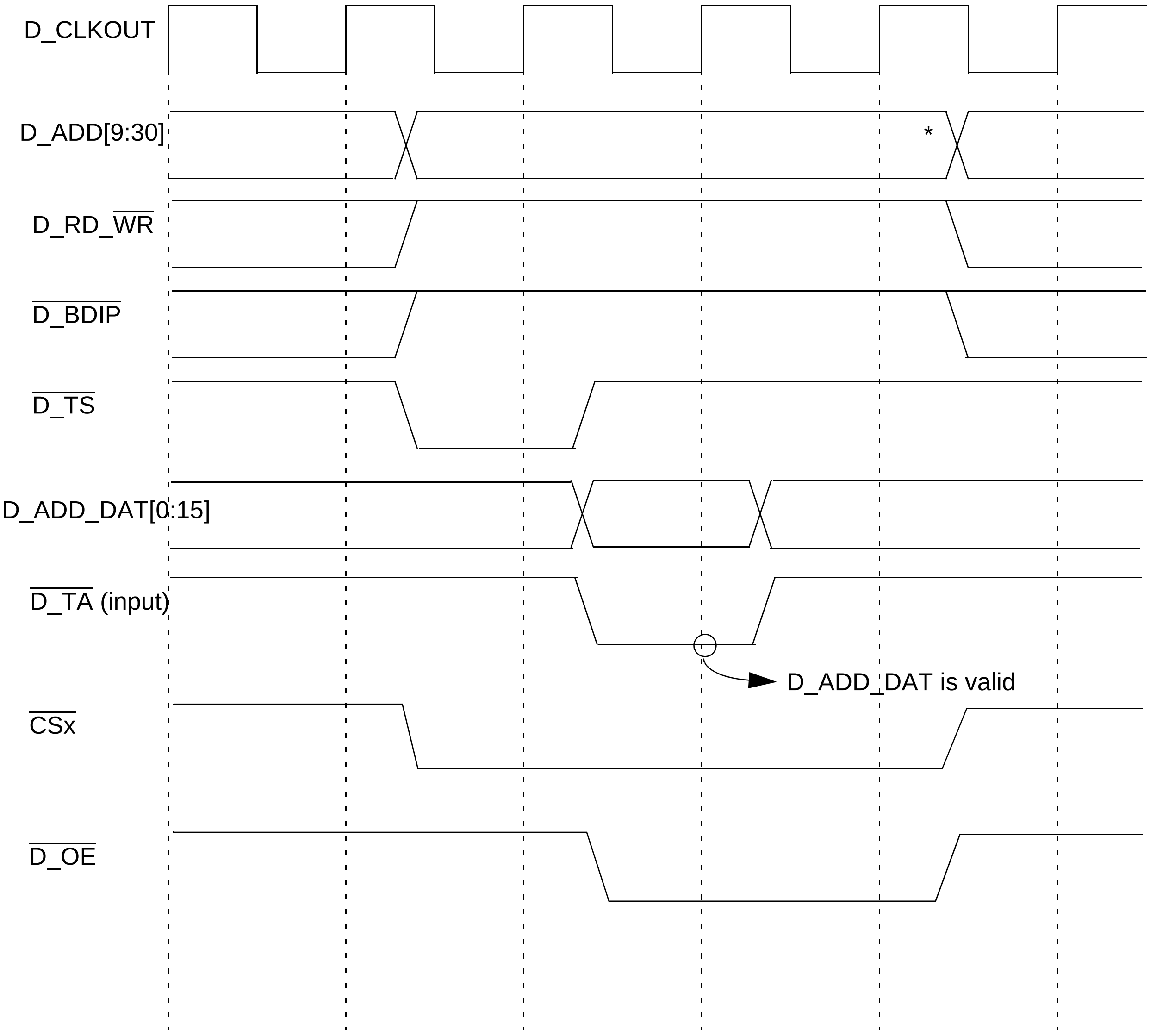
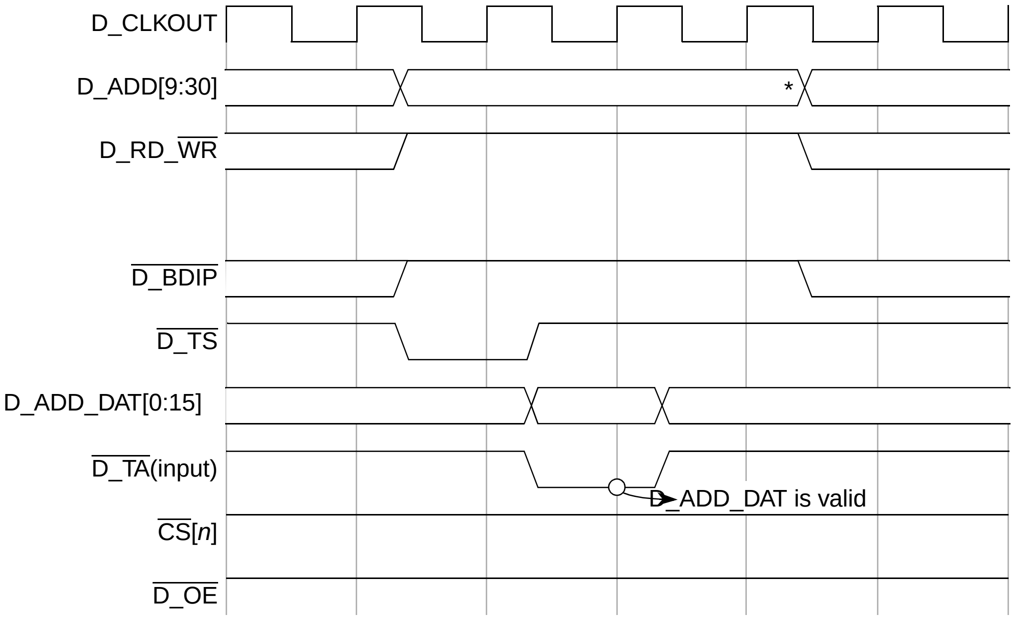


Figure 23-11. Single Beat 32-bit Read Cycle, CS Access, One Wait State



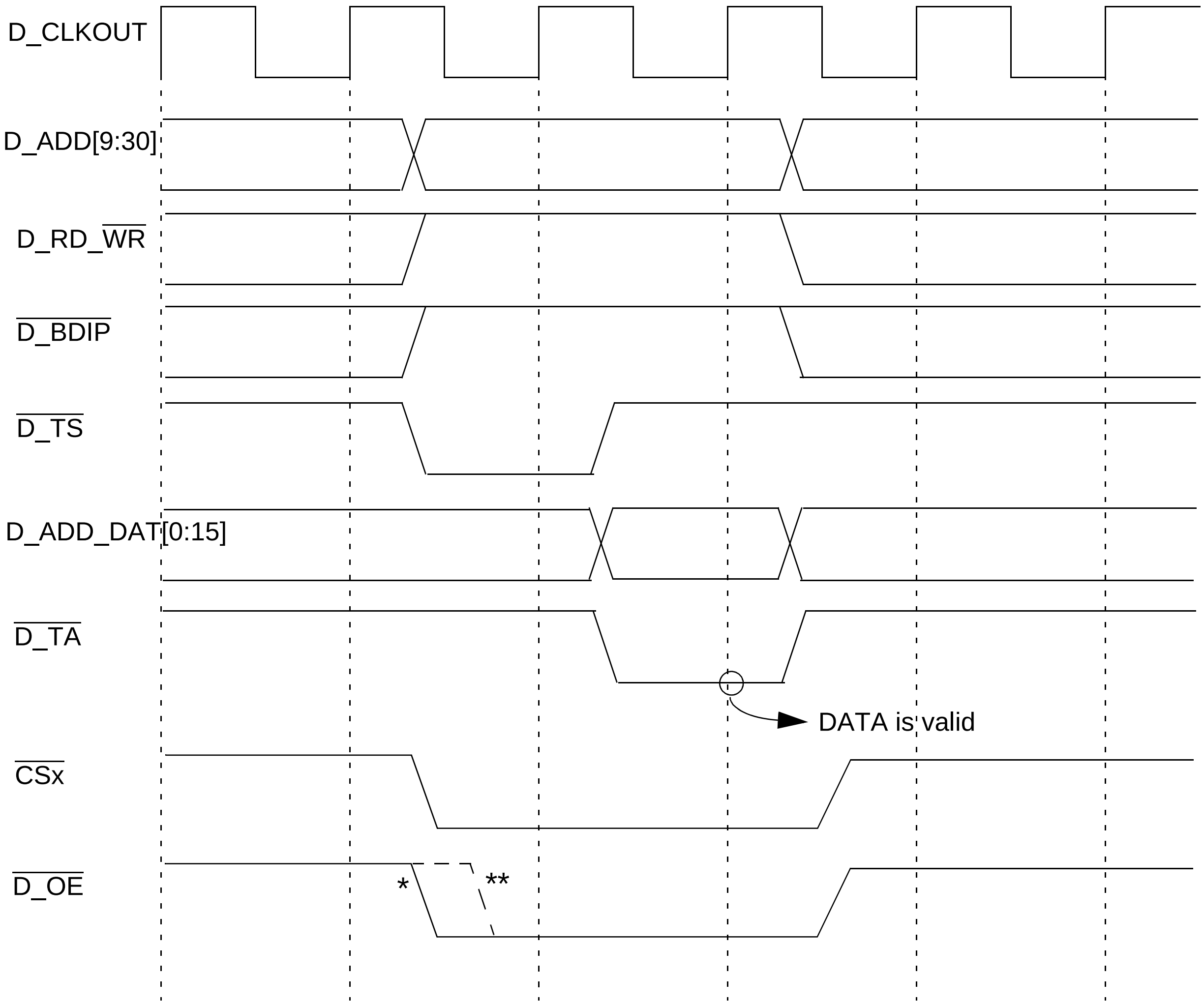
* The EBI drives address and control signals an extra cycle because it uses a latched version of the external D_TA (1 cycle delayed) to terminate the cycle.

Figure 23-12. Single Beat 16-bit Read Cycle, CS Access, SETA=1, Zero Wait States



* The EBI drives address and control signals an extra cycle because it uses a latched version of the external D_TA (1 cycle delayed) to terminate the cycle.

Figure 23-13. Single Beat 16-bit Read Cycle, Non-CS Access, Zero Wait States



* When EBI_CAL_BRn[EOE]=0b01 (or 0b10 in 1:1 bus speed mode), D_OE asserts around the same tir as TS (could be slightly before or after D_TS, order of D_TS/D_OE assertion is not guaranteed).

** When EBI_CAL_BRn[EOE]=0b10 (and not in 1:1 bus speed mode), D_OE asserts 1 internal system clc (partial CLKOUT cycle) later as compared to D_TS.

Figure 23-14. Single Beat 16-bit Read Cycle, CS Access, Zero Wait States (EOE=0b01, 0b10)

23.4.2.4.2 Single Beat Write Flow

The handshakes for a single beat write cycle are illustrated in the following flow and timing diagrams.

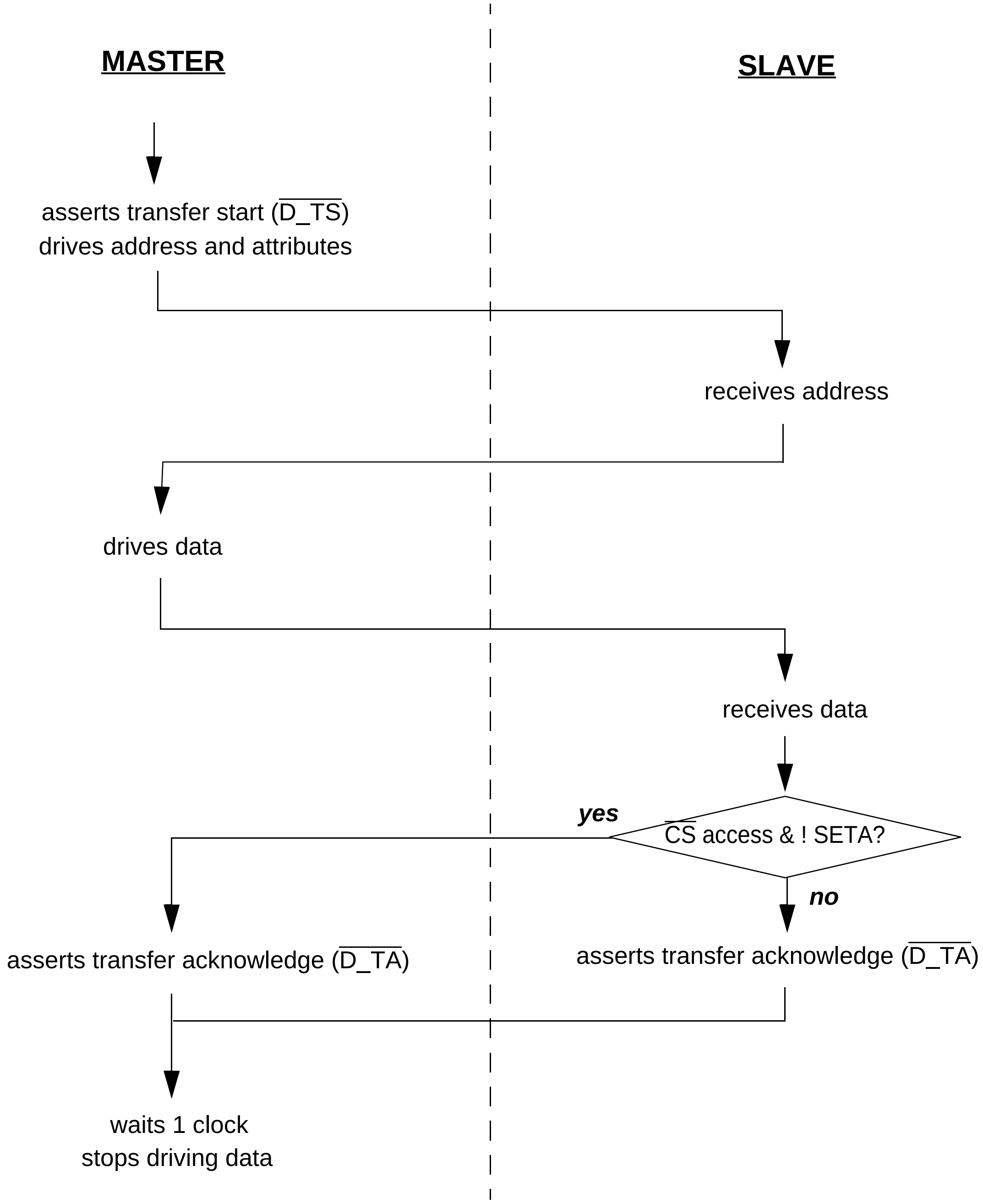


Figure 23-15. Basic Flow Diagram of a Single Beat Write Cycle

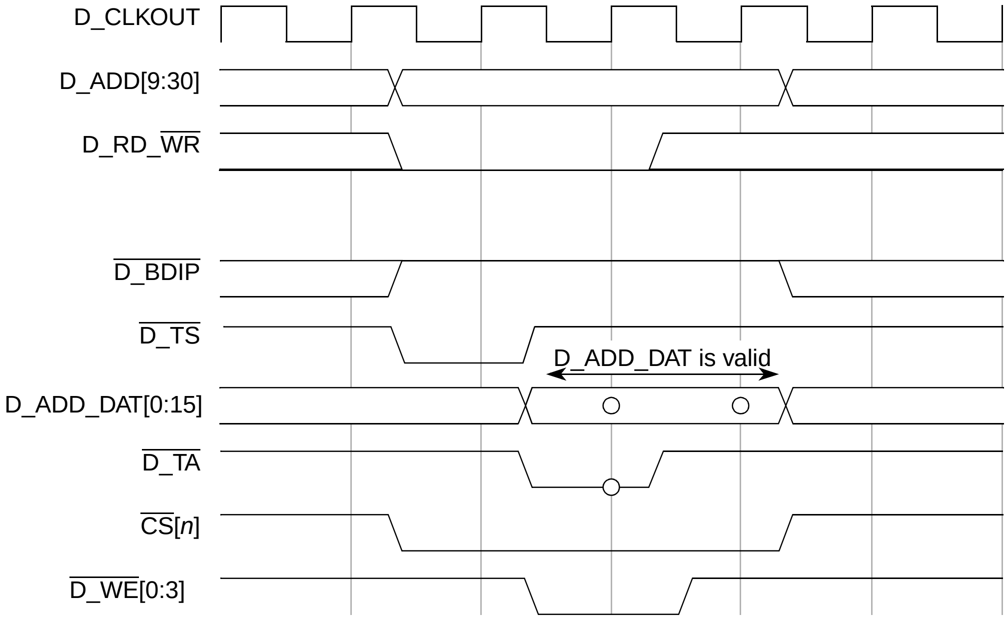


Figure 23-16. Single Beat 16-bit Write Cycle, CS Access, Zero Wait States

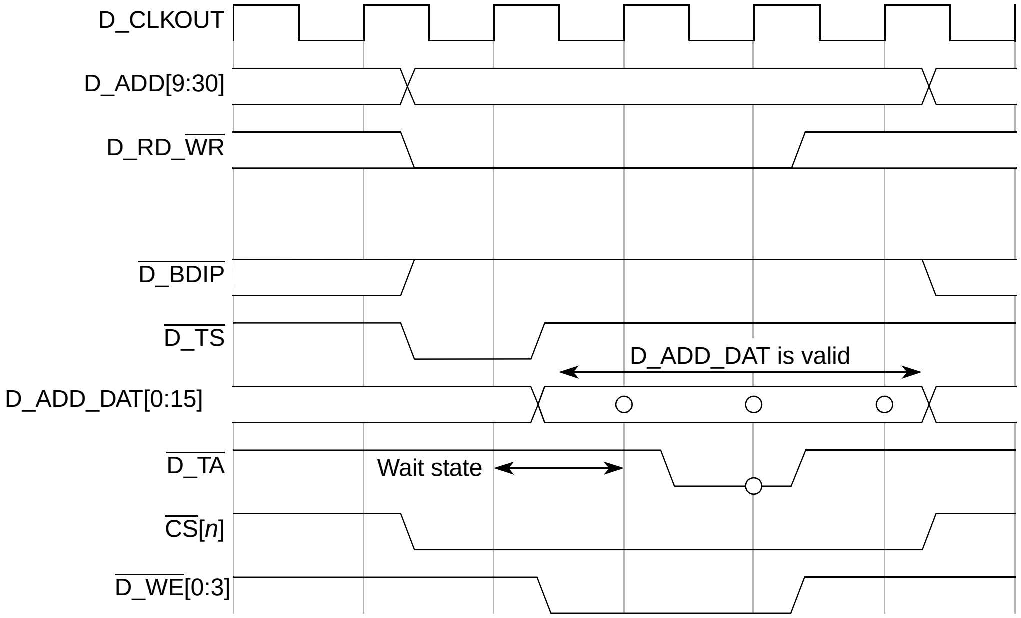
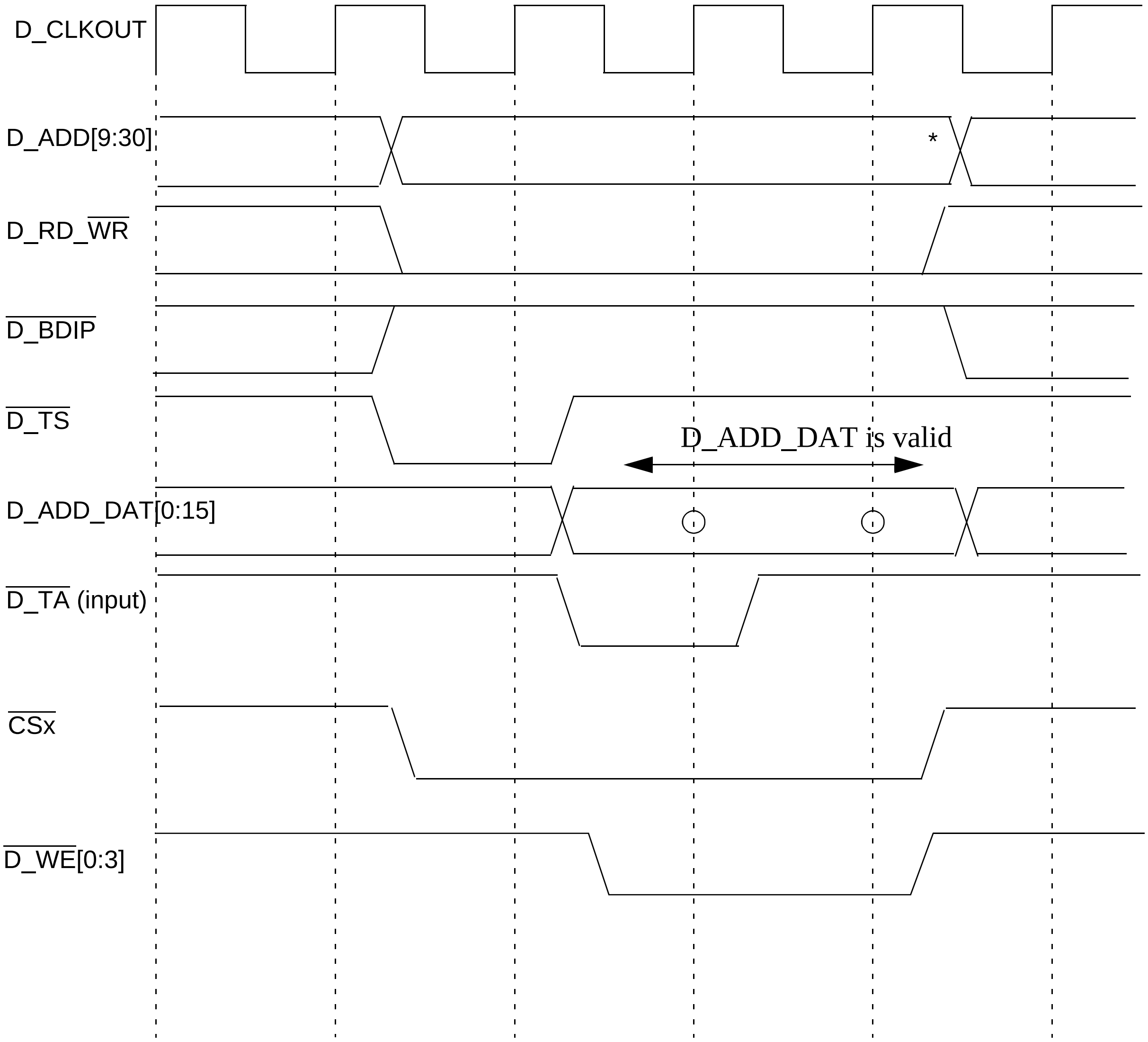
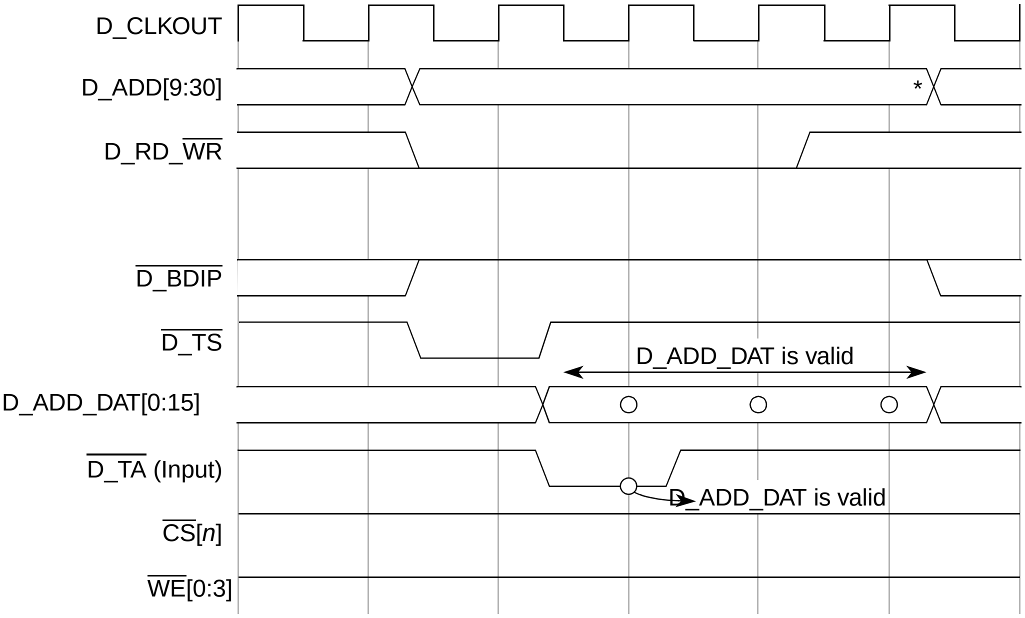


Figure 23-17. Single Beat 32-bit Write Cycle, CS Access, One Wait State



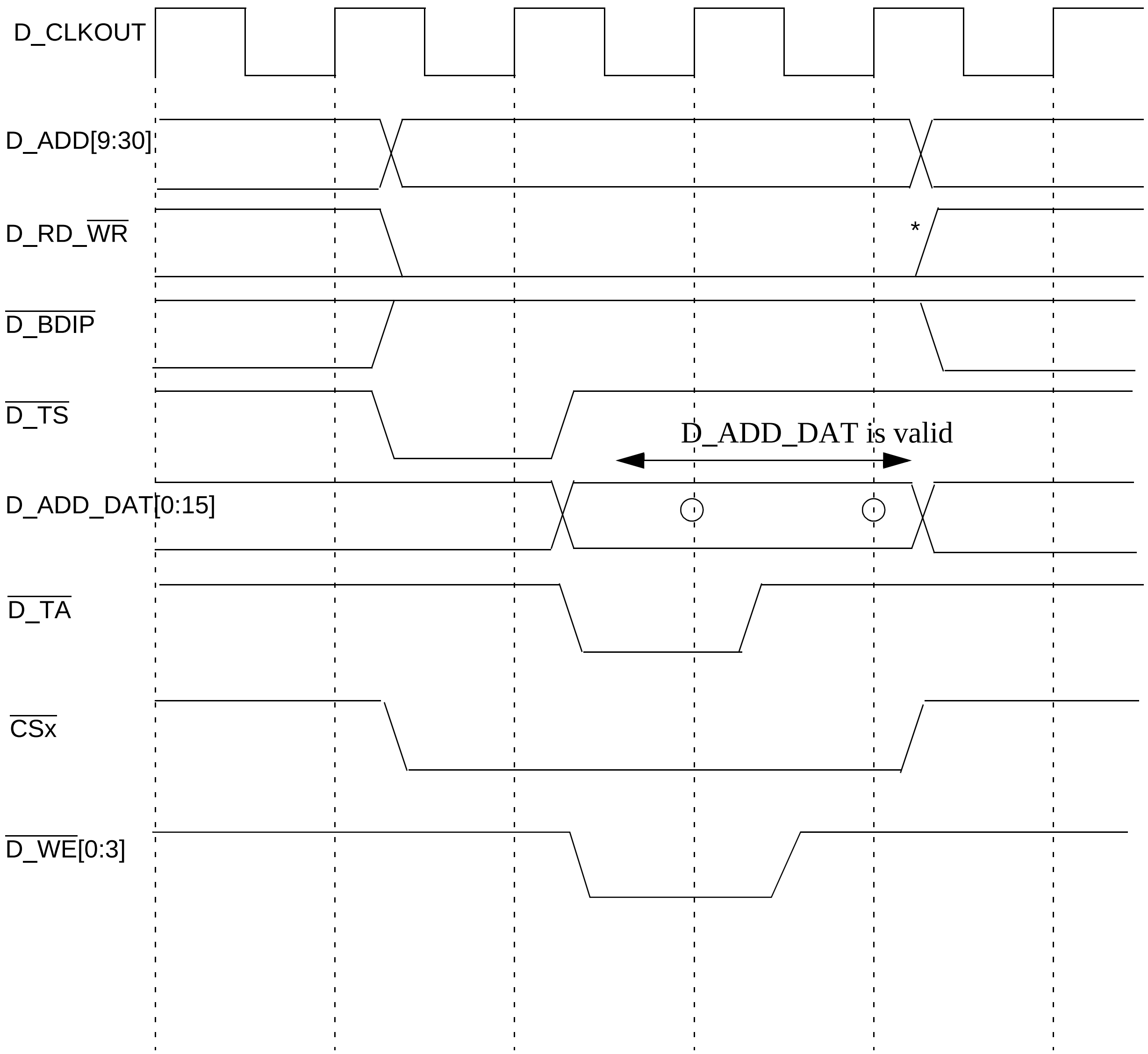
* The EBI drives address and control signals an extra cycle because it uses a latched version of the external D_TA (1 cycle delayed) to terminate the cycle.

Figure 23-18. Single Beat 16-bit Write Cycle, CS Access, SETA=1, Zero Wait States



* The EBI drives address and control signals an extra cycle because it uses a latched version of the external D_TA (1 cycle delayed) to terminate the cycle.

Figure 23-19. Single Beat 16-bit Write Cycle, Non-CS Access, Zero Wait States



* Negation of D_RD_WR signal is delayed by 1 cycle (as shown) when EBI_BR[LWRN]=1.

Figure 23-20. Single Beat 16-bit Write Cycle, CS Access, Zero Wait States, LWRN=1

23.4.2.4.3 Back-to-Back Accesses

Due to internal bus protocol, one dead cycle is necessary between back-to-back external bus accesses that are not part of a set of small accesses (see Section 23.4.2.6, "Small Accesses (Small Port Size and Short Burst Length) for small access timing). A dead cycle refers to a cycle between the D_TA of a previous transfer and the D_TS of the next transfer.

NOTE

> In some cases, CS remains asserted during this dead cycle, such as the cases of back-to-back writes or read-after-write to the same chip-select (with EBI_BR[GCSN]=0). See Figure 23-31 and Figure 23-32. However, if EBI_BR[GCSN]=1 for the first access (see Figure 23-25 and Figure 23-27), then the EBI inserts an extra dead cycle between the accesses for these cases in order to guarantee CS negation between accesses.

Besides this dead cycle, in most cases, back-to-back accesses on the external bus do not cause any change in the timing from that shown in the previous diagrams, and the two transactions are independent of each other. The only exceptions to this are listed below:

- Back-to-back accesses where the first access ends with an externally-driven D_TA or D_TEA. In these cases, an extra cycle is required between the end of the first access and the D_TS assertion of the second access. See Section 23.4.2.8, "Termination Signals Protocol for more details.
- 4-word burst transfers whose starting address is not 0x20 aligned. In these cases, an extra cycle is required between the end of the 2nd 4-word burst access and the TS assertion of a subsequent access. See Figure 23-36.

The following diagrams show a few examples of back-to-back accesses on the external bus.

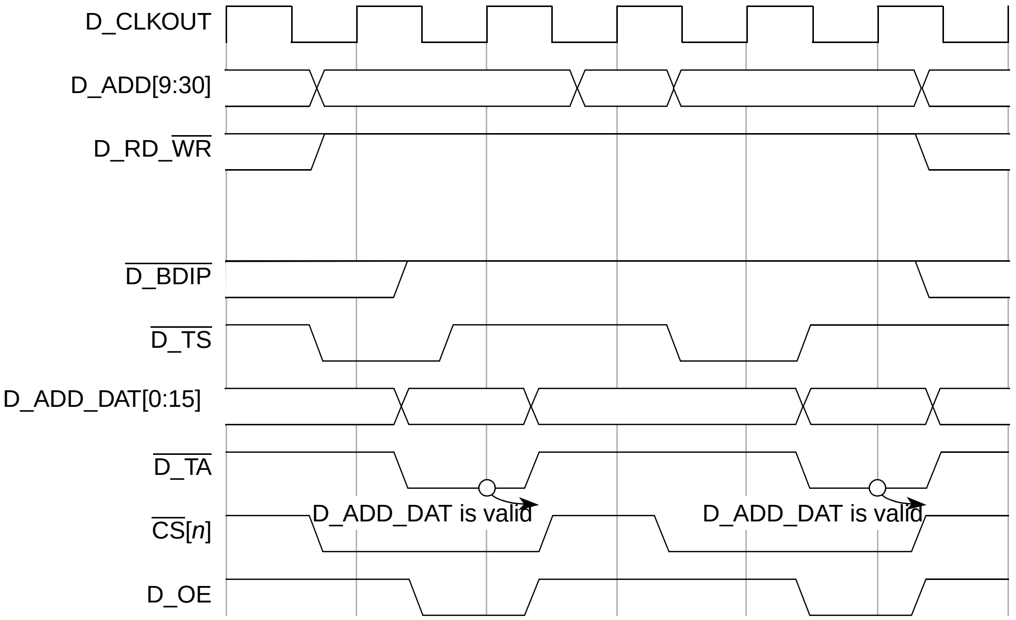


Figure 23-21. Back-to-Back 16-bit Reads to the Same CS Bank

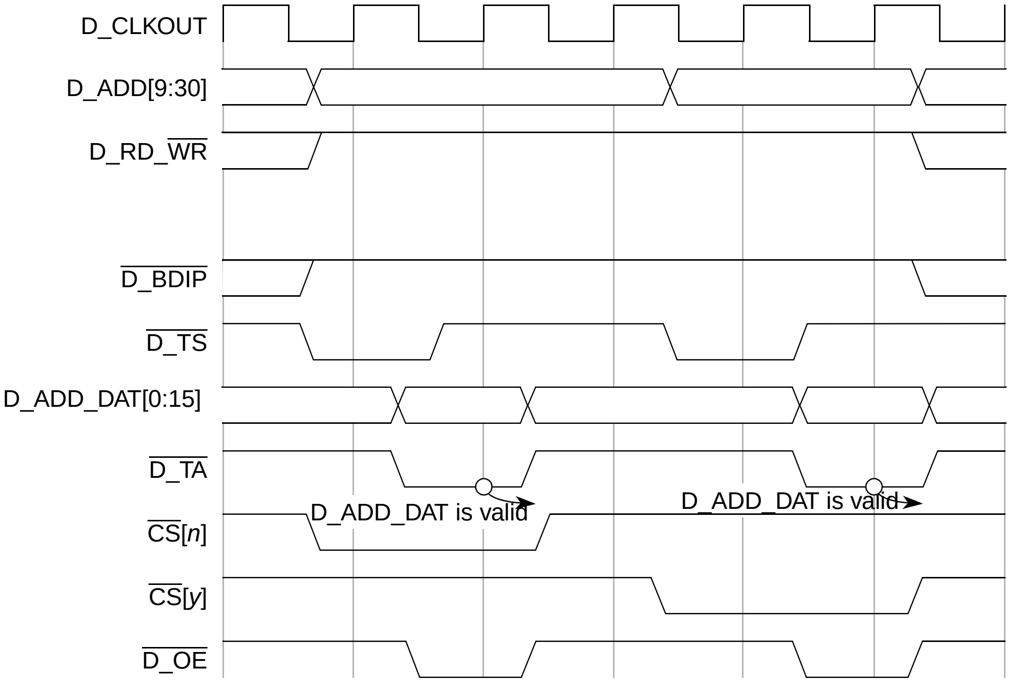


Figure 23-22. Back-to-Back 16-bit Reads to Different CS Banks

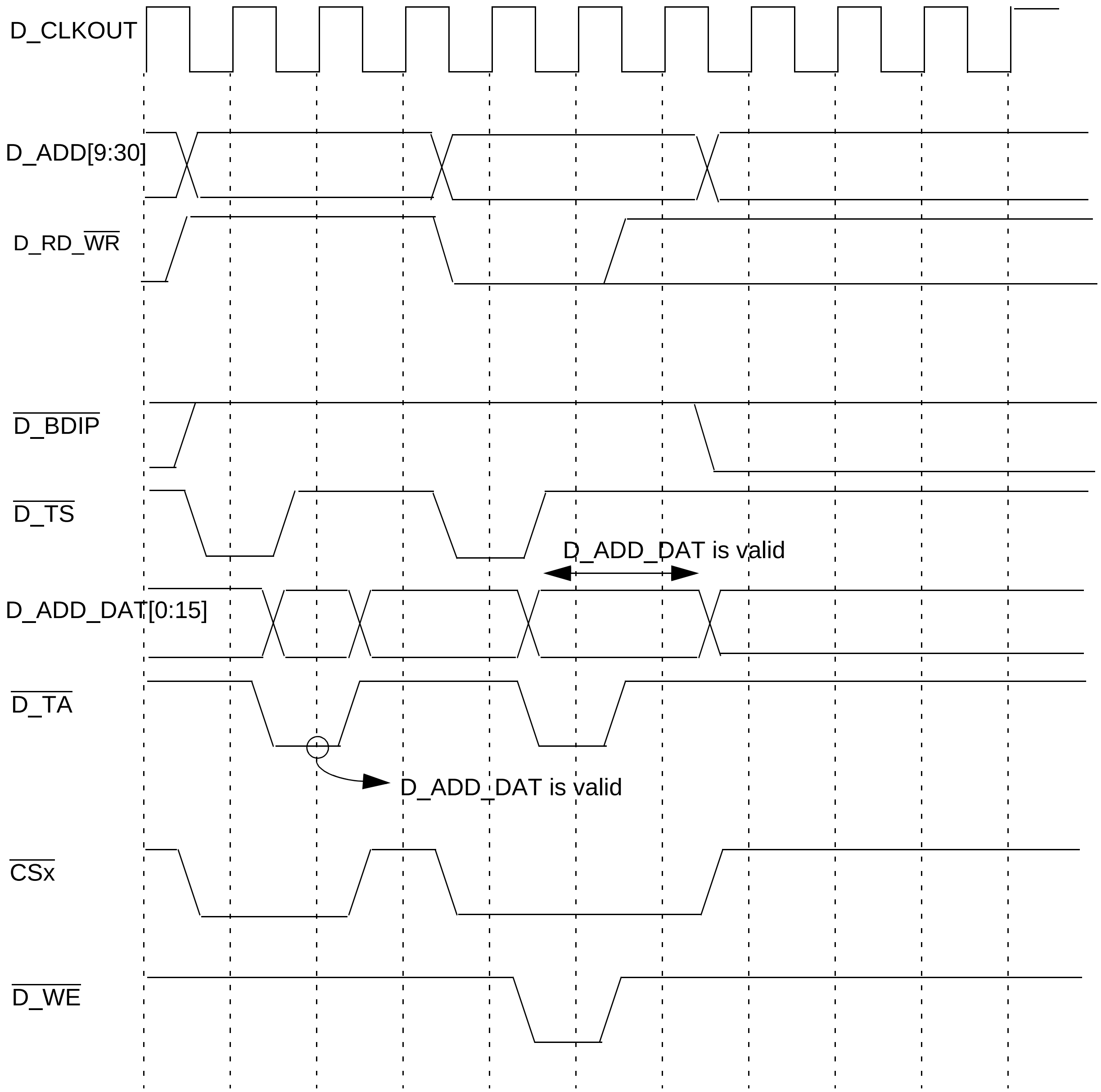
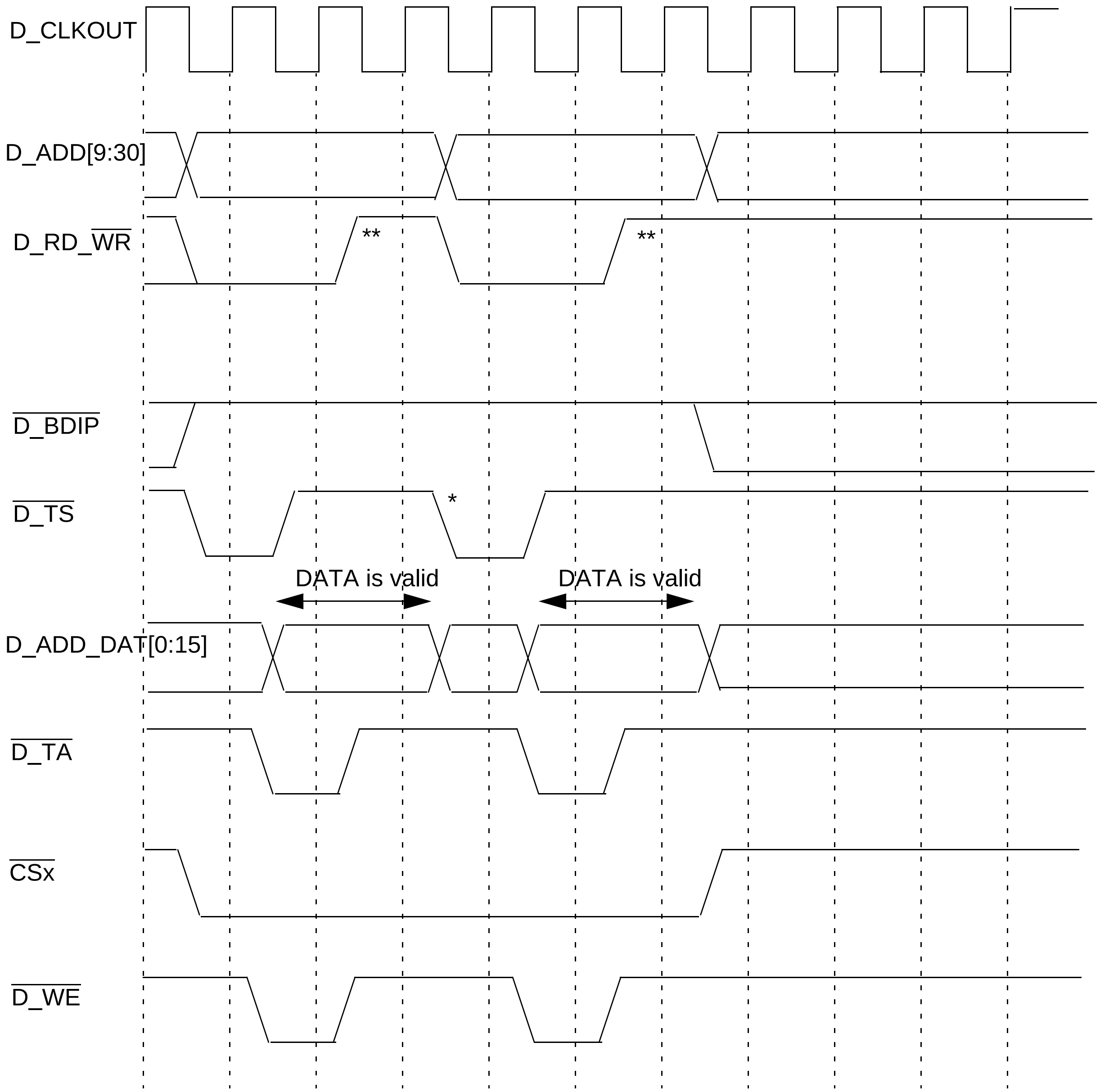


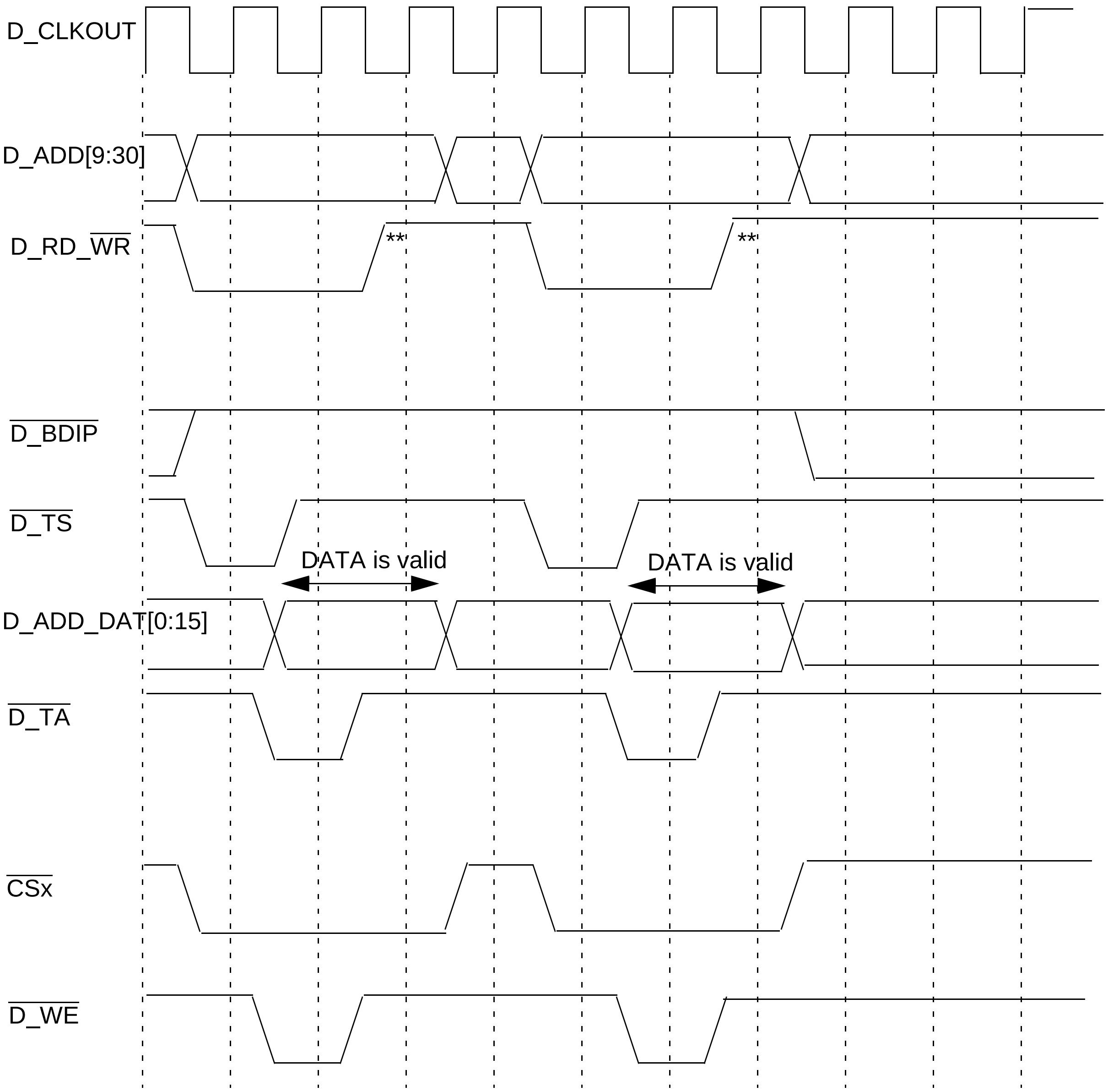
Figure 23-23. Write After Read to the Same $\overline{\text{CS}}$ Bank



* Timing shown applies when EBI_BR[GCSN]=0. When EBI_BR[GCSN]=1, the 2nd TS assertion is delayed by 1 cycle and CS negates between the two transfers.

** Timing shown applies when EBI_BR[LWRN]=0. When EBI_BR[LWRN]=1, RD_WR negation is delayed by 1 cycle, such that it does not negate between these two b-t-b transfers.

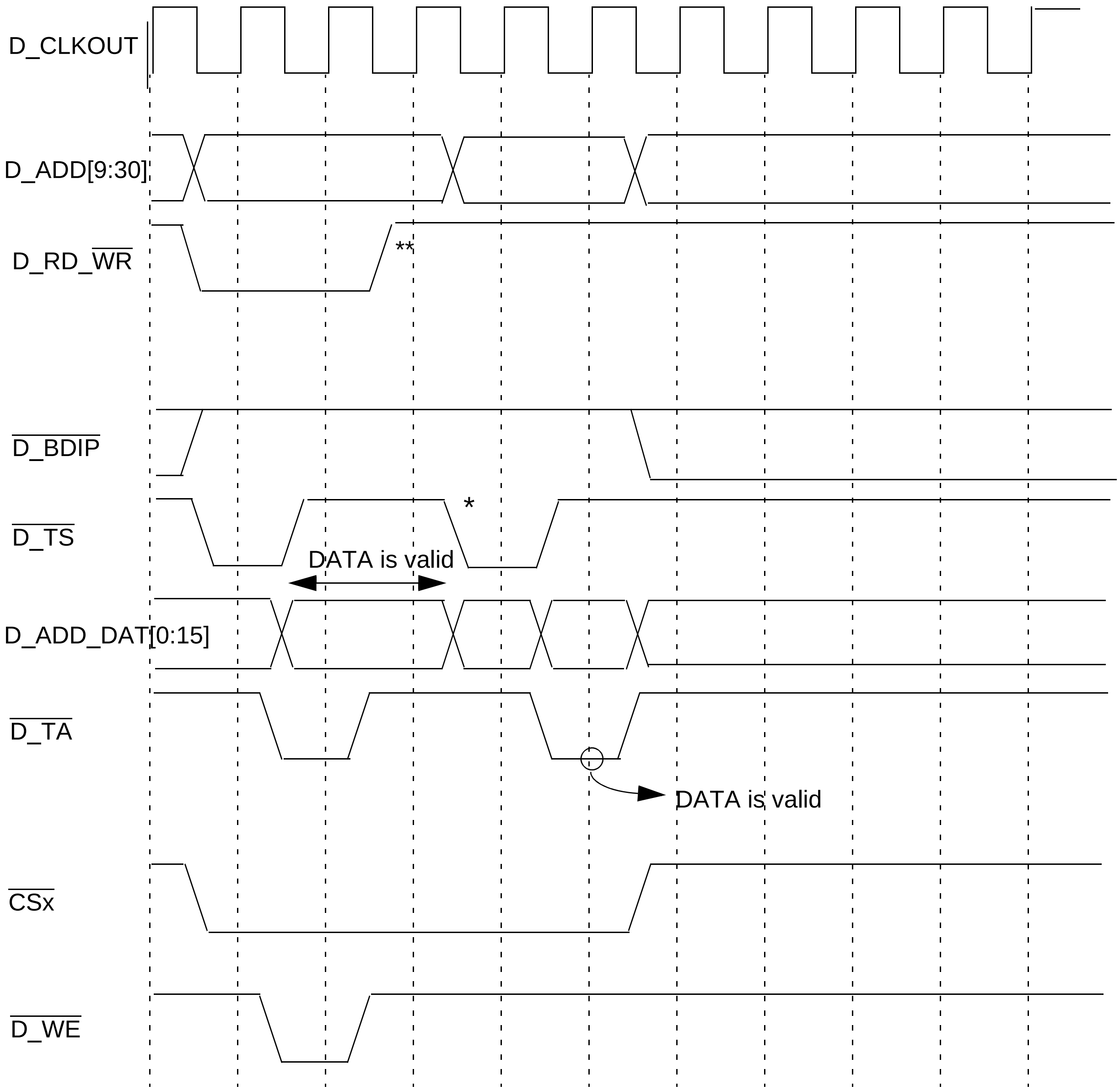
Figure 23-24. Back-to-Back 16-bit Writes to the Same CS Bank, GCSN=0



* Timing shown applies when EBI_BR[GCSN]=1.

** Timing shown applies when EBI_BR[LWRN]=0. When EBI_BR[LWRN]=1, RD_WR negation is delayed by 1 cycle.

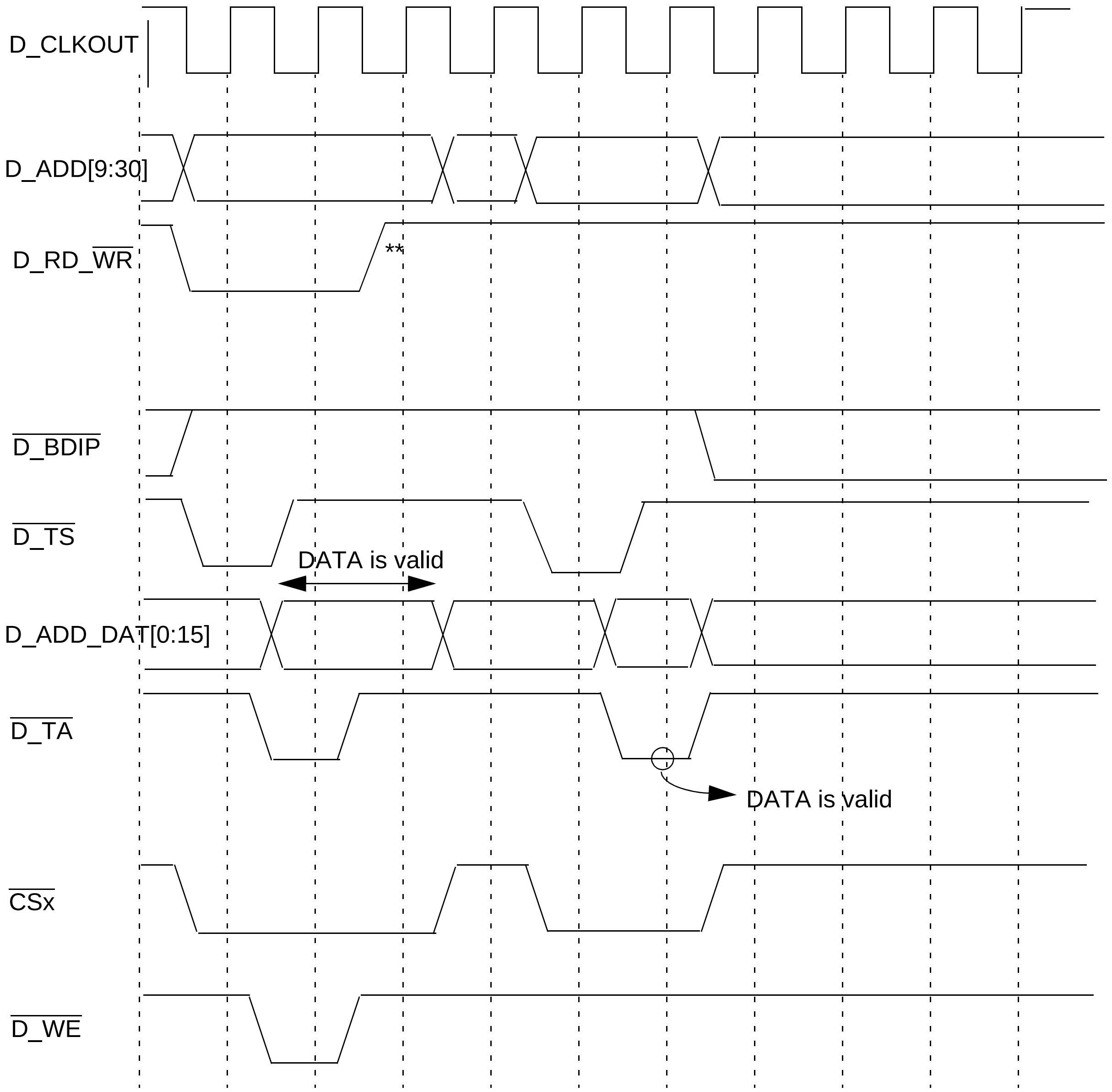
Figure 23-25. Back-to-Back 16-bit Writes to the Same CS Bank, GCSN=1



* Timing shown applies when EBI_BR[GCSN]=0. When EBI_BR[GCSN]=1, the 2nd TS assertion is delayed by 1 cycle and CS negates between the two transfers.

** Timing shown applies when EBI_BR[LWRN]=0. When EBI_BR[LWRN]=1, RD_WR negation is delayed by 1 cycle.

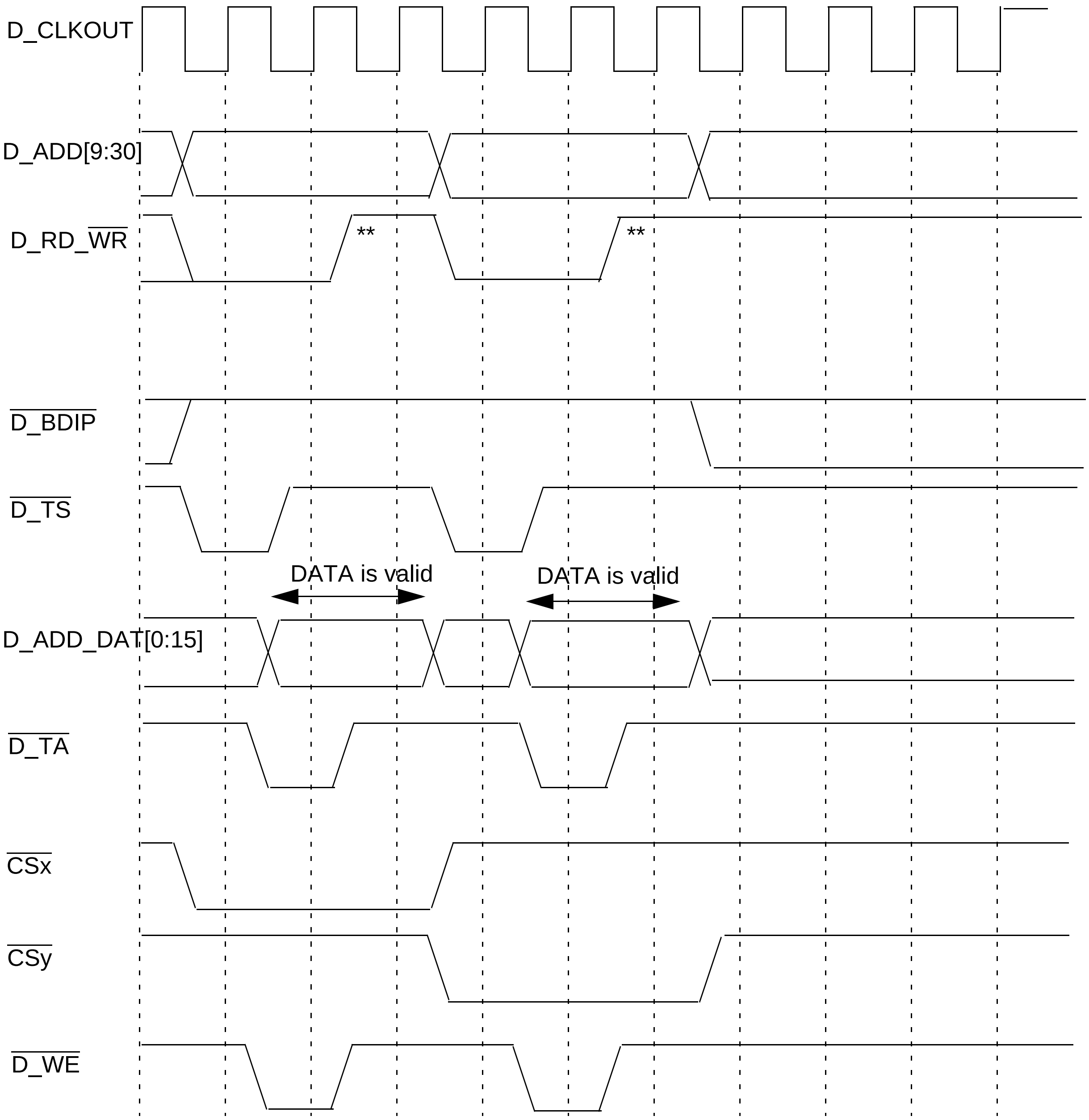
Figure 23-26. Read After Write to the Same CS Bank, GCSN=0



* Timing shown applies when EBI_BR[GCSN]=1.

** Timing shown applies when EBI_BR[LWRN]=0. When EBI_BR[LWRN]=1, RD_WR negation is delayed by 1 cycle.

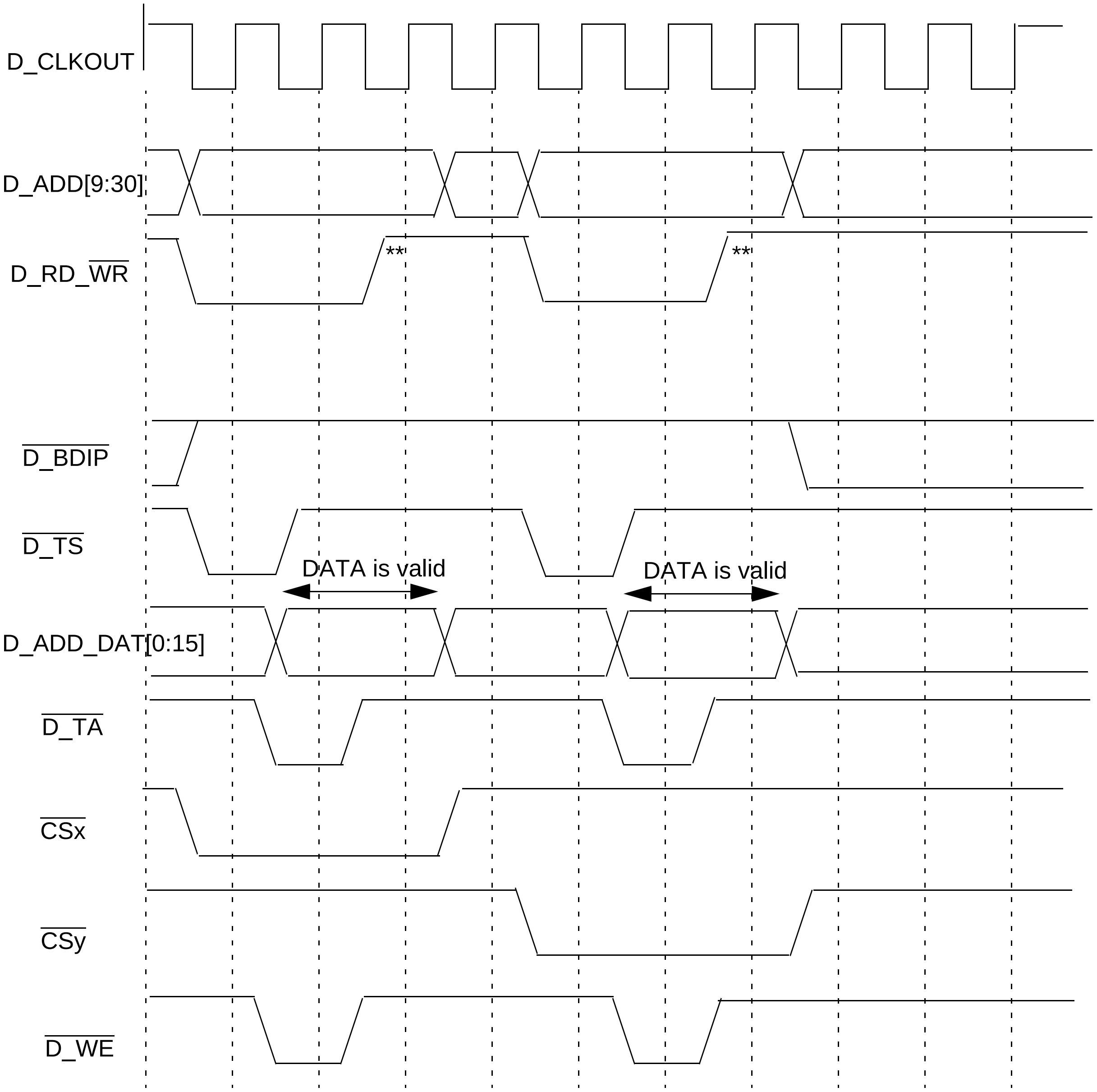
Figure 23-27. Read After Write to the Same CS Bank, GCSN=1



* This timing diagram applies when EBI_BR[GCSN]=0.

** Timing shown applies when EBI_BR[LWRN]=0. When EBI_BR[LWRN]=1, RD_WR negation is delayed by 1 cycle, such that it does not negate between these two b-t-b transfers.

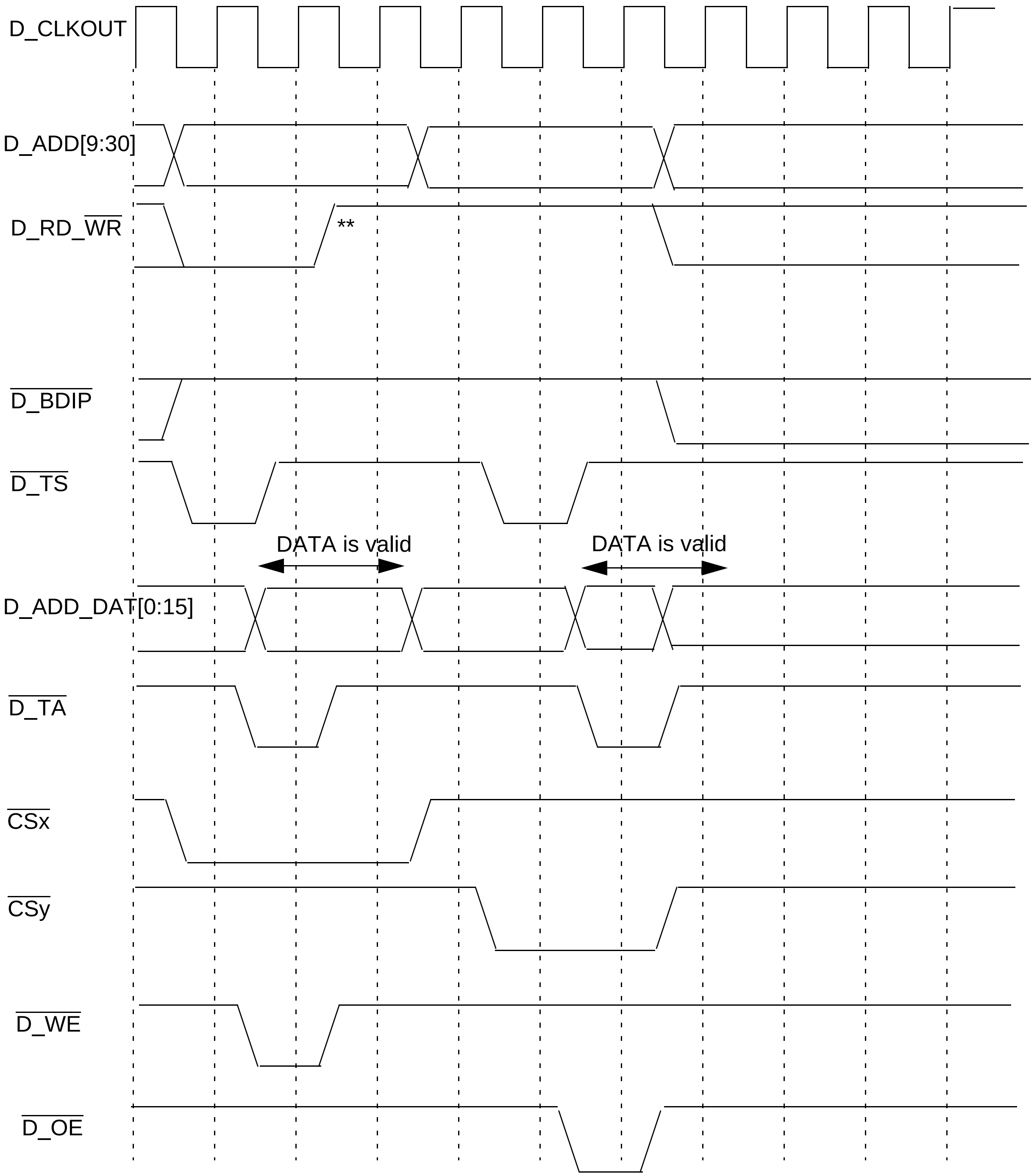
Figure 23-28. Back-to-Back 16-bit Writes to Different CS Banks (GCSN=0)



* Timing shown applies when EBI_BR[GCSN]=1.

** Timing shown applies when EBI_BR[LWRN]=0. When EBI_BR[LWRN]=1, RD_WR negation is delayed by 1 cycle.

Figure 23-29. Back-to-Back 16-bit Writes to the Same CS Bank, GCSN=1



* This timing diagram is unaffected by EBI_BR[GCSN].

** Timing shown applies when EBI_BR[LWRN]=0. When EBI_BR[LWRN]=1, RD_WR negation is delayed by 1 cycle.

Figure 23-30. Read After Write to Different CS Banks

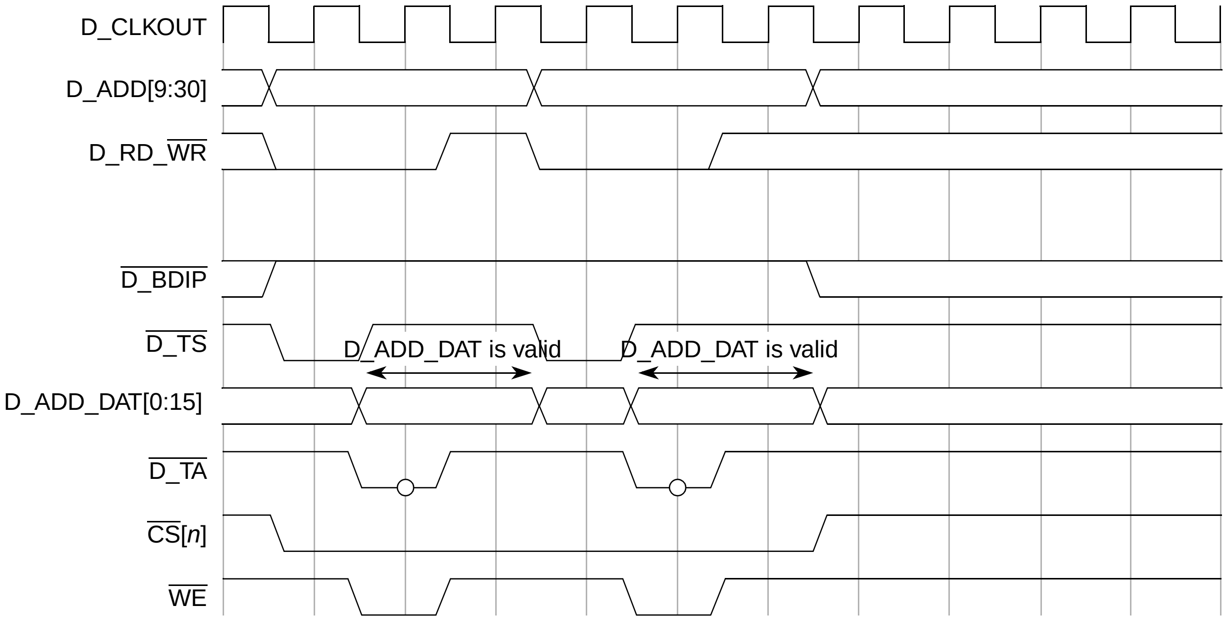


Figure 23-31. Back-to-Back 16-bit Writes to the Same CS Bank

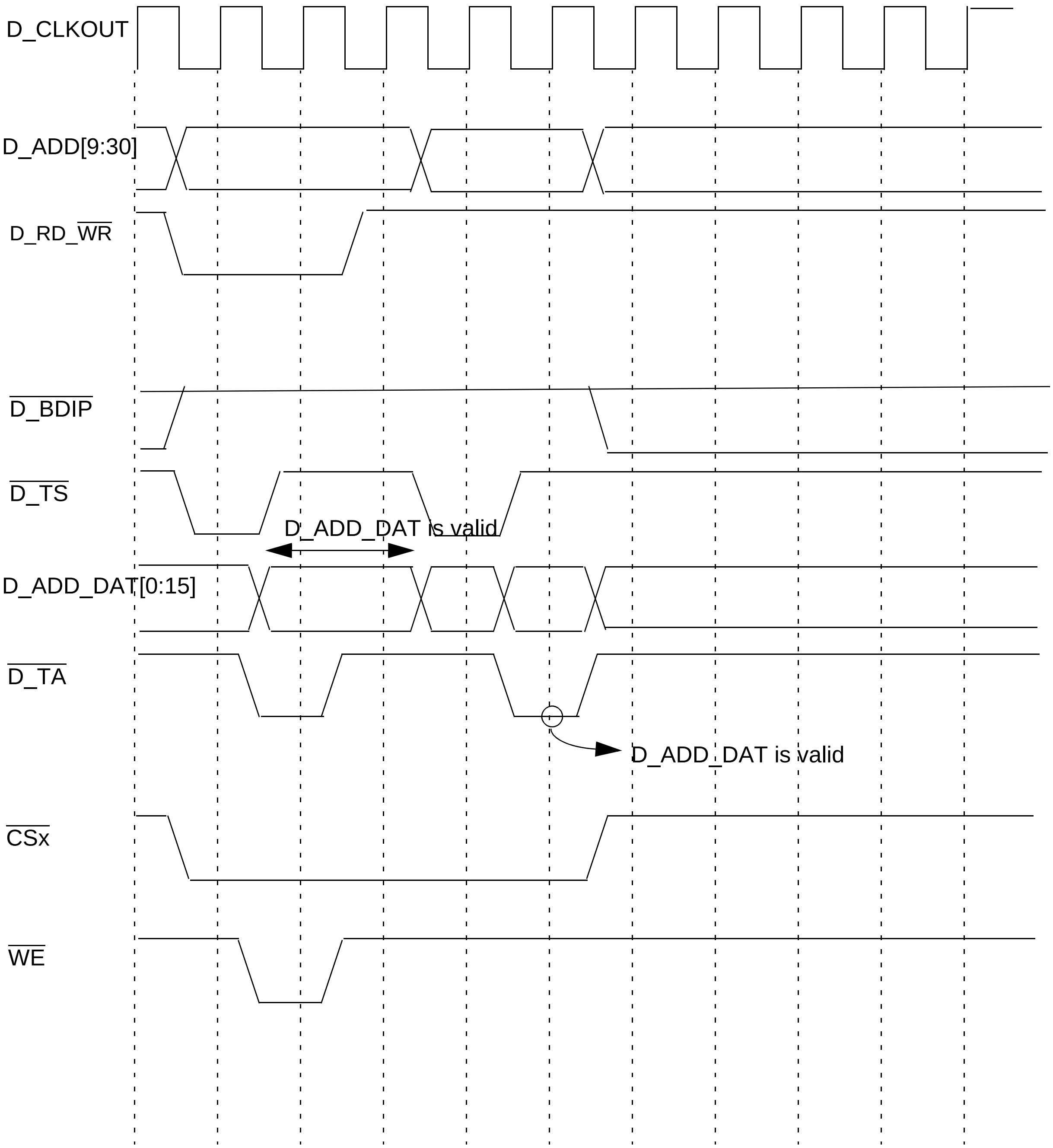


Figure 23-32. Read After Write to the Same CS Bank

23.4.2.5 Burst Transfer

The EBI supports wrapping 32-byte critical-doubleword-first burst transfers. Bursting is supported only for internally-requested cache-line size (32-byte) read accesses to external devices that use the chip selects[1].

Accesses to devices operating without a chip select are always single beat. If an internal request to the EBI indicates a size of less than 32 bytes, the request is fulfilled by running one or more single-beat external transfers, not by an external burst transfer.

An 8-word wrapping burst reads eight 32-bit words by supplying a starting address that points to one of the words (doubleword aligned) and requiring the memory device to sequentially drive each word on the data bus. The selected slave device must internally increment D_ADD[27:29] (also D_ADD30 in the case of a 16-bit port size device) of the supplied address for each transfer, until the address reaches an 8-word boundary, and then wrap the address to the beginning of the 8-word boundary. The address and transfer attributes supplied by the EBI remain stable during the transfers. Termination of each beat transfer occurs by the EBI asserting $\overline{\text{D\_TA}}$ (SETA=1 is not supported for burst transfers). The EBI requires that addresses be aligned to a doubleword boundary on all burst cycles.

Table 23-11 shows the burst order of beats returned for an 8-word burst to a 32-bit port.

Table 23-11. Wrap Bursts Order

Burst Starting Address D_ADD[27:28]	Burst Order (Assuming 32-bit Port Size)
00	word0 -> word1 -> word2 -> word3 -> word4 -> word5 -> word6 -> word7
01	word2 -> word3 -> word4 -> word5 -> word6 -> word7 -> word0 -> word1
10	word4 -> word5 -> word6 -> word7 -> word0 -> word1 -> word2 -> word3
11	word6 -> word7 -> word0 -> word1 -> word2 -> word3 -> word4 -> word5

The general case of burst transfers assumes that the external memory has 32-bit port size and 8-word burst length. The EBI can also burst from 16-bit port size memories, taking twice as many external beats to fetch the data as compared to a 32-bit port with the same burst length. The EBI can also burst from 16-bit or 32-bit memories that have a 4-word burst length (BL=1 in the appropriate Base Register). In this case, two external 4-word burst transfers (wrapping on 4-word boundary) are performed to fulfill the internal 8-word request[2]. This operation is considered atomic by the EBI, so the EBI does not allow other unrelated master accesses or bus arbitration to intervene between the transfers.

During burst cycles, the $\overline{\text{D\_BDIP}}$ (Burst Data In Progress) signal is used to indicate the duration of the burst data. During the data phase of a burst read cycle, the EBI receives data from the addressed slave. If the EBI needs more than one data, it asserts the $\overline{\text{D\_BDIP}}$ signal. Upon receiving the data prior to the last data, the EBI negates $\overline{\text{D\_BDIP}}$. Thus, the slave stops driving new data after it receives the negation of $\overline{\text{D\_BDIP}}$ on the rising edge of the clock. Some slave devices have their burst length and timing configurable internally and thus may not support connecting to a $\overline{\text{D\_BDIP}}$ pin. In this case, $\overline{\text{D\_BDIP}}$ is driven by the EBI normally, but the output is ignored by the memory and the burst data behavior is

---

[1] Except for the special case of a 32-bit non-chip-select access in 16-bit data bus mode. See Section 23.4.2.9, “Non-Chip-Select Burst in 16-bit Data Bus Mode.”

[2] This case (of 2 external burst transfers being required) applies only to AMBA data bus width of 64 bits.

determined by the internal configuration of the EBI and slave device. When the TBDIP bit is set in the appropriate Base Register, the timing for $\overline{\text{D\_BDIP}}$ is altered. See Section 23.4.2.5.1, “TBDIP Effect on Burst Transfer for this timing.

Since burst writes are not supported by the EBI[1], the EBI negates $\overline{\text{D\_BDIP}}$ during write cycles.

---

[1] Except for the special case of a 32-bit non-chip-select access in 16-bit data bus mode. See Section 23.4.2.9, “Non-Chip-Select Burst in 16-bit Data Bus Mode.

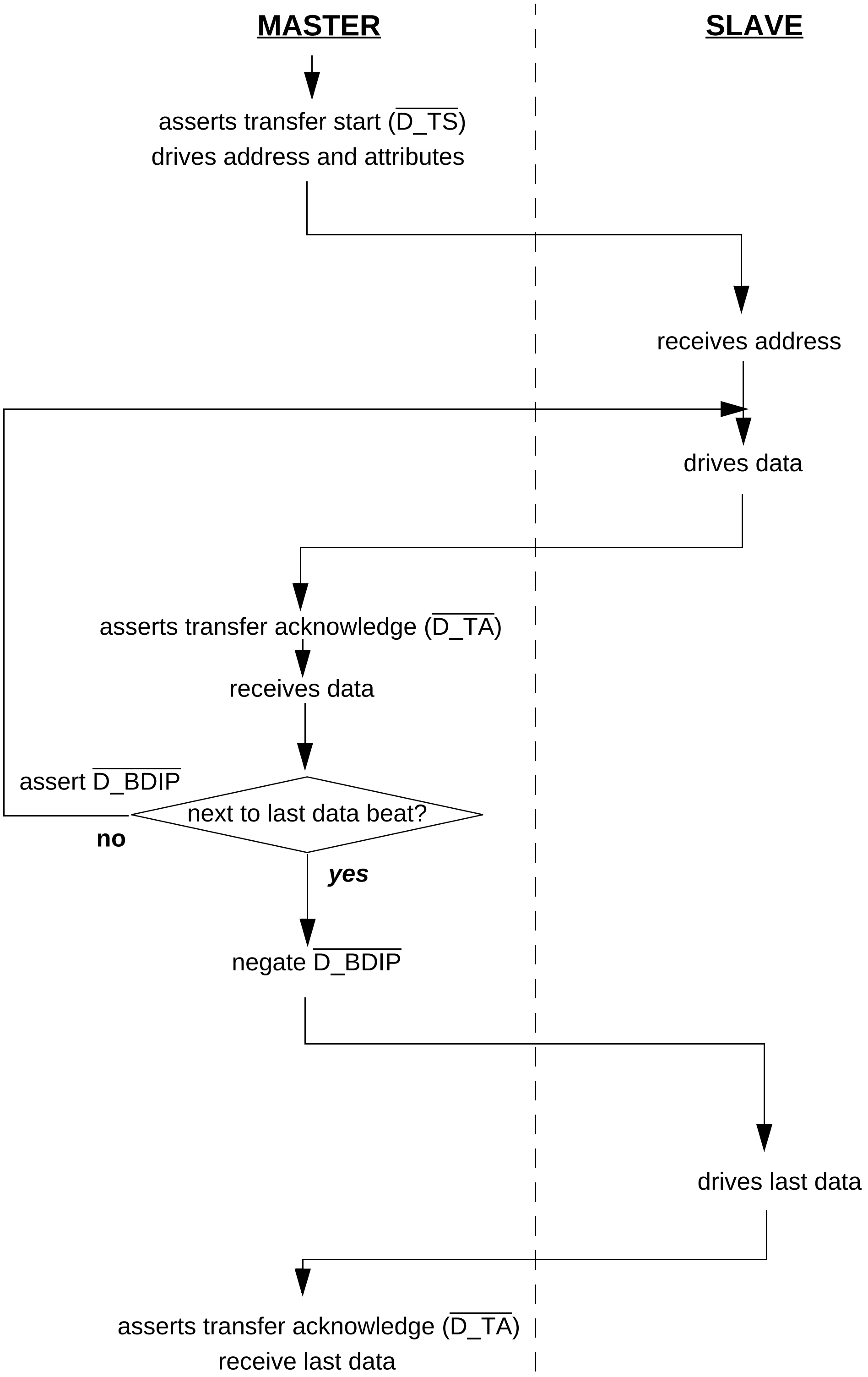


Figure 23-33. Basic Flow Diagram of a Burst Read Cycle

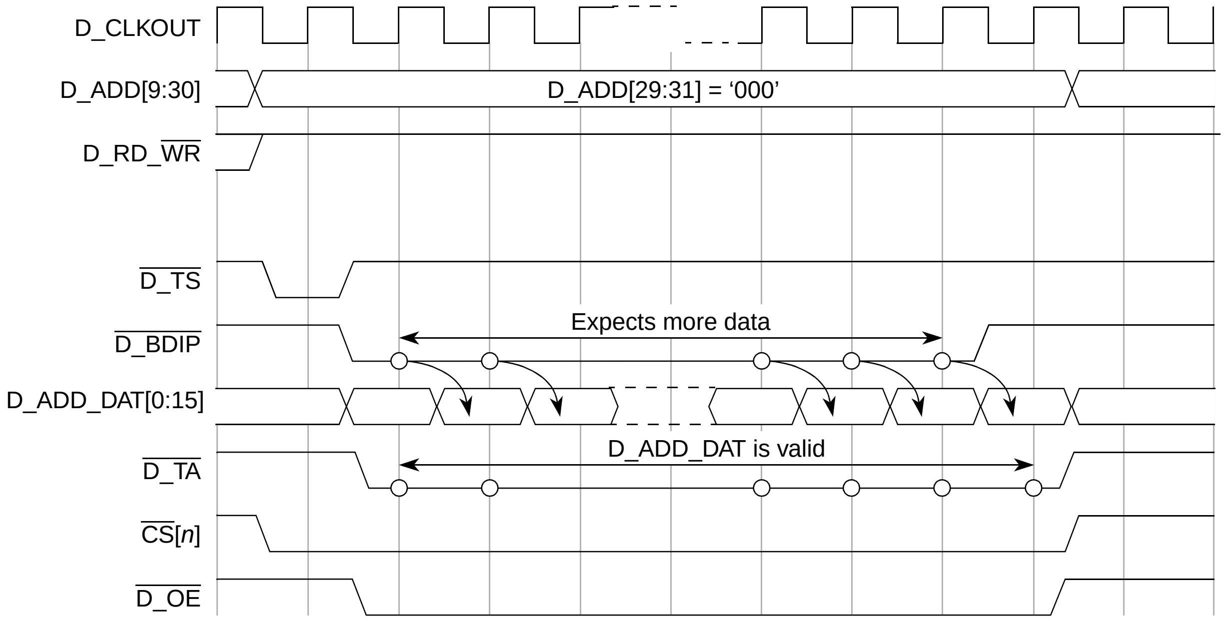


Figure 23-34. Burst 16-bit Read Cycle, Zero Wait States

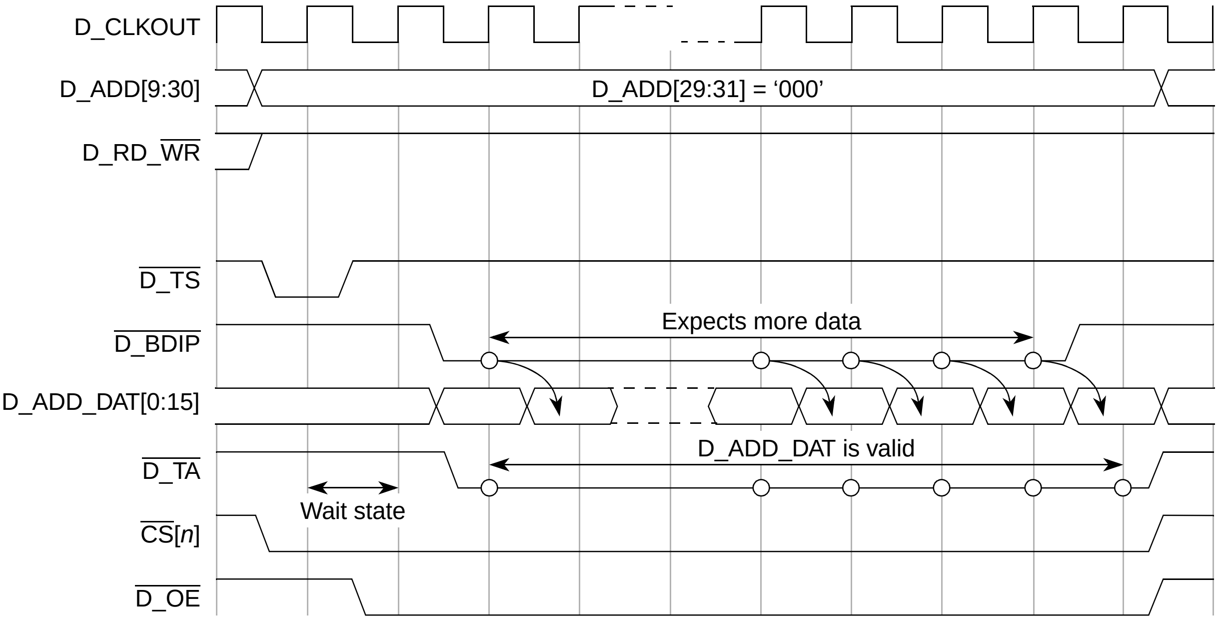


Figure 23-35. Burst 16-bit Read Cycle, One Initial Wait State

23.4.2.5.1 TBDIP Effect on Burst Transfer

Some memories require different timing on the D_BDIP signal than the default to run burst cycles. Using the default value of TBDIP=0 in the appropriate EBI Base Register results in D_BDIP being asserted (SCY+1) cycles after the address transfer phase, and being held asserted throughout the cycle regardless of the wait states between beats (BSCY). Figure 23-36 shows an example of the TBDIP=0 timing for a 4-beat burst with BSCY=1.

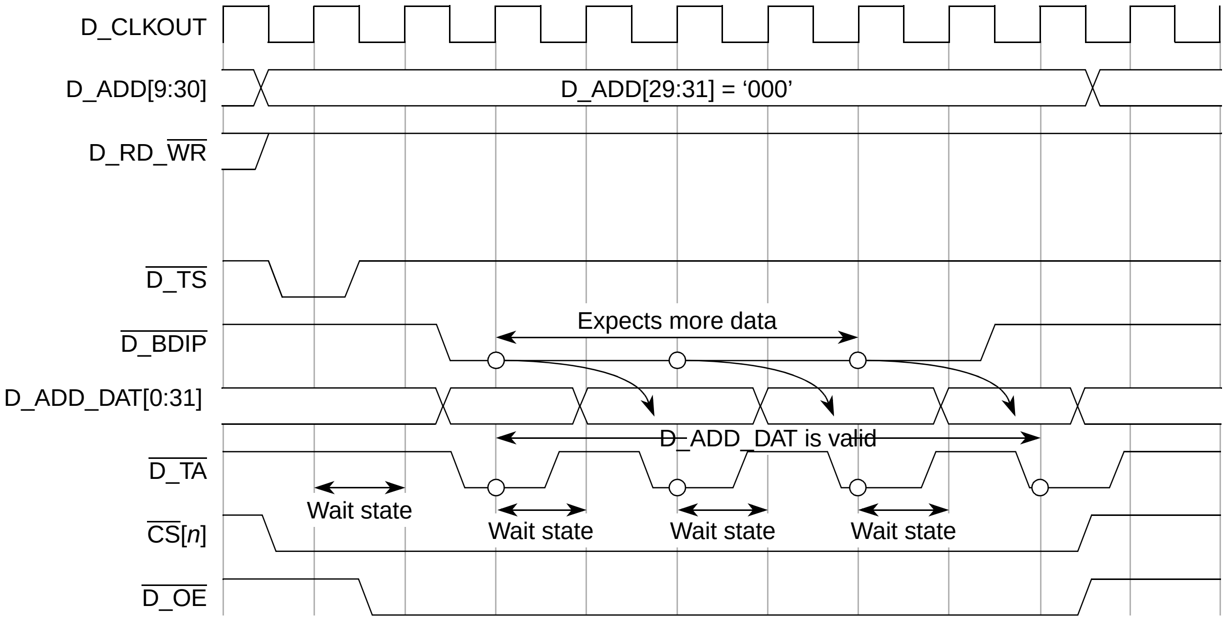


Figure 23-36. Burst 32-bit Read Cycle, One Wait State between Beats, TBDIP=0

When using TBDIP=1, the D_BDIP behavior changes to toggle between every beat when BSCY is a non-zero value. Figure 23-37 shows an example of the TBDIP=1 timing for the same 4-beat burst shown in Figure 23-36.

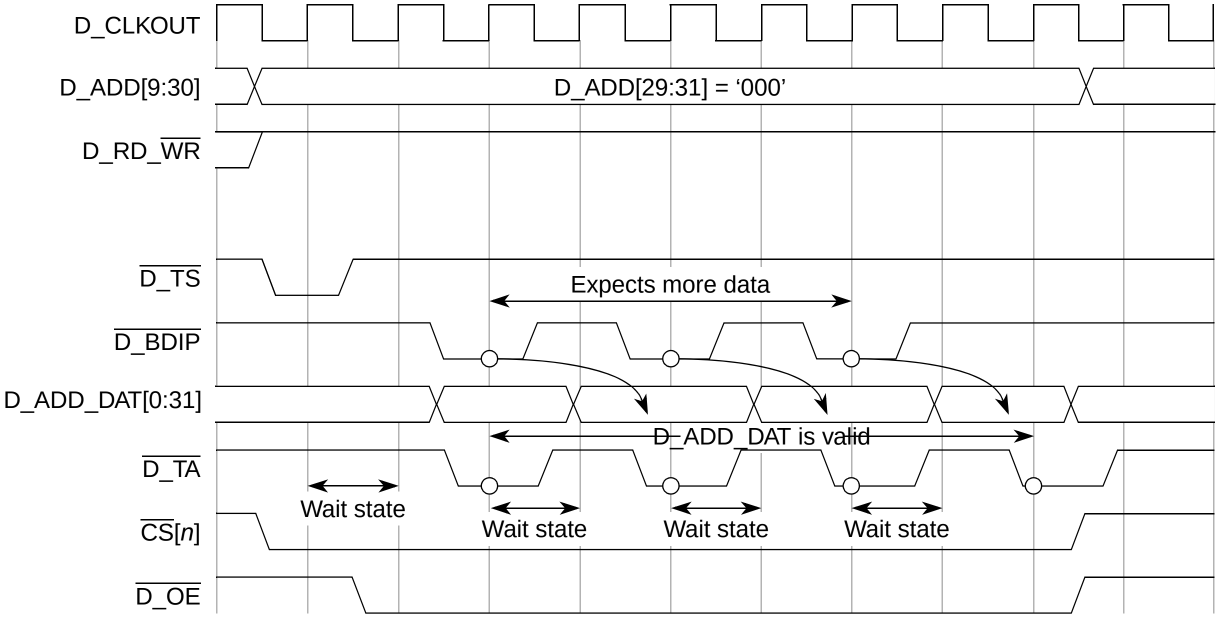


Figure 23-37. Burst 32-bit Read Cycle, One Wait State between Beats, TBDIP=1

23.4.2.6 Small Accesses (Small Port Size and Short Burst Length)

In this context, a *small access* refers to an access whose burst length and port size (BL, PS bits in Base Register for chip-select access or default burst disabled, 32-bit port for non-chip-select access) are such that the number of bytes requested by the internal master cannot all be fetched (or written) in one external transaction. If this is the case, the EBI initiates multiple transactions until all the requested data is transferred. It should be noted that all the transactions initiated to complete the data transfer are considered as an atomic transaction, so the EBI does not allow other unrelated master accesses to intervene between the transfers.

Table 23-12 shows all the combinations of burst length, port size, and requested byte count that cause the EBI to run multiple external transactions to fulfill the request.

Table 23-12. Small Access Cases

Byte Count Requested by internal master	Burst Length	Port Size	# External Accesses to Fulfill Request
Non-Burstable Chip-Select Banks (BI=1) or Non-Chip-Select Access			
4	1 beat	16-bit	2/1[1]
8	1 beat	32-bit	2
8	1 beat	16-bit	4
32	1 beat	32-bit	8
32	1 beat	16-bit	16
Burstable Chip-Select Banks (BI=0)			
32	4 words	16-bit (8 beats), 32-bit (4 beats)	2

NOTES:
[1] In 32-bit data bus mode (DBM=0 in EBI_MCR), two accesses are performed. In 16-bit data bus mode (DBM=1), one 2-beat burst access is performed and this is not considered a “small access” case. See Section 23.4.2.9, “Non-Chip-Select Burst in 16-bit Data Bus Mode for this special DBM=1 case.

In most cases, the timing for small accesses is the same as for normal single-beat and burst accesses, except that multiple back-to-back external transfers are executed for each internal request. These transfers have no additional dead cycles in-between that are not present for back-to-back stand-alone transfers except for the case of writes with an internal request size of >64 bits.

The following sections show a few examples of small accesses. The timing for the remaining cases in Table 23-12 can be extrapolated from these and the other timing diagrams in this document.

23.4.2.6.1 Small Access Example #1: 32-bit Write to 16-bit Port

Figure 23-38 shows an example of a 32-bit write to a 16-bit port, requiring two 16-bit external transactions.

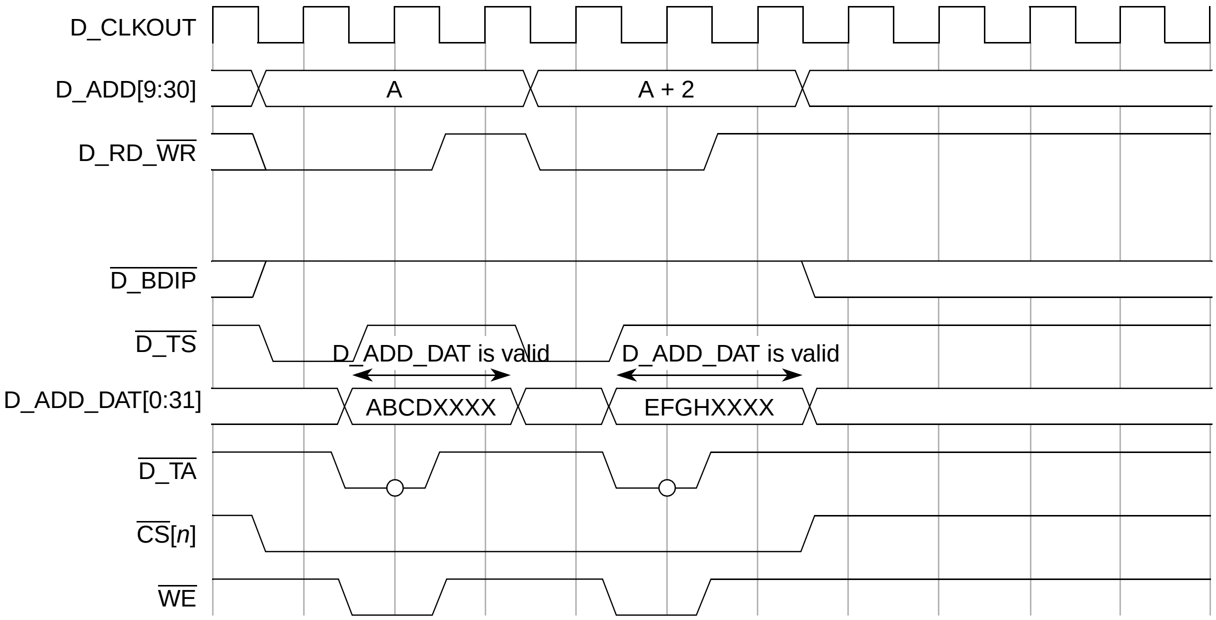
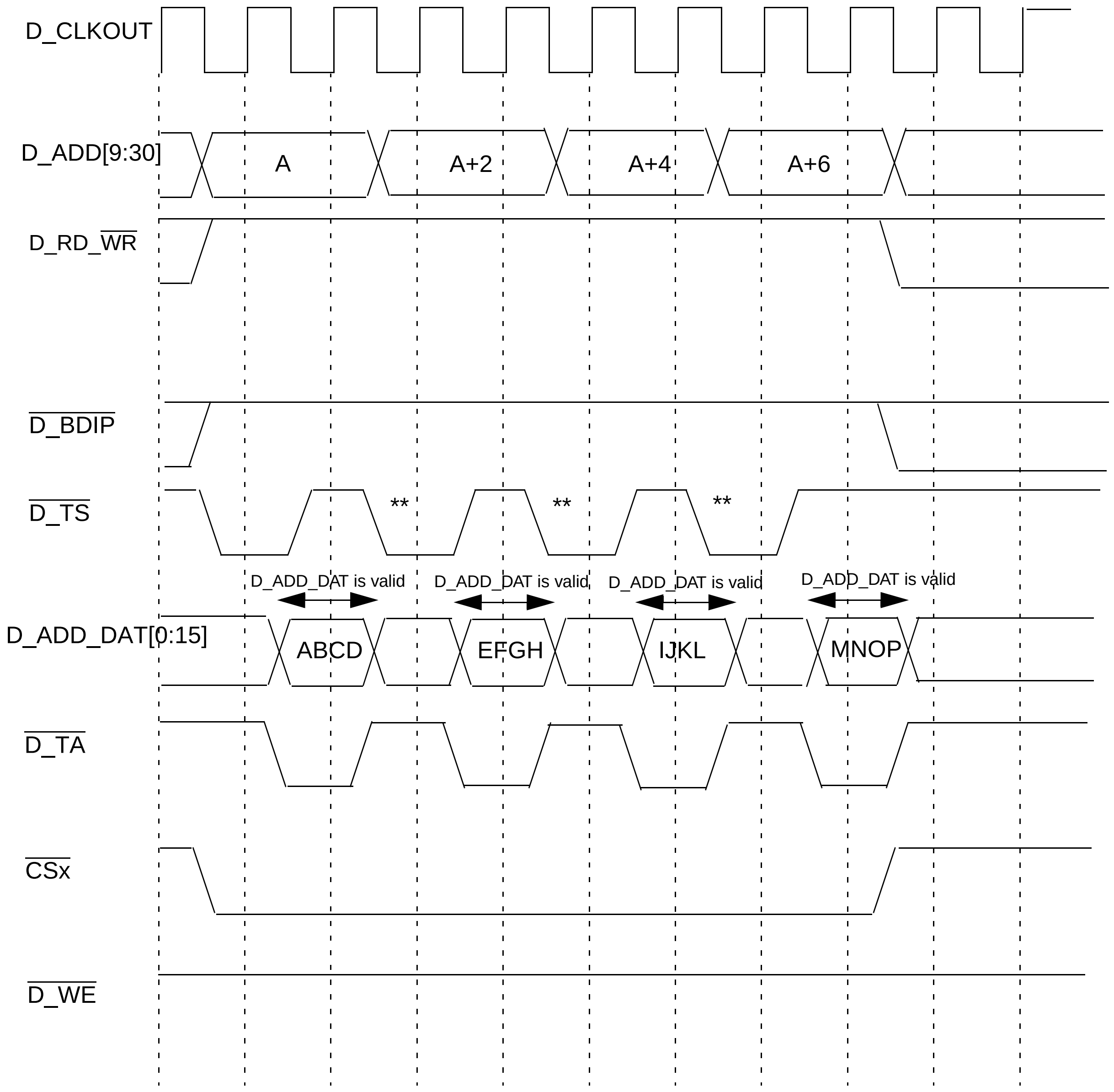


Figure 23-38. Single Beat 32-bit Write Cycle, 16-bit Port Size, Basic Timing

23.4.2.6.2 Small Access Example #4: 64-bit Read to 16-bit Port

Figure 23-39 shows an example of a 64-bit read to a 16-bit port, requiring four 16-bit external transactions.



** Timing shown applies when EBI_BR[GCSN]=0. When EBI_BR[GCSN]=1, the 2nd-4th D_TS assertions are each delayed by 1 cycle and CS negates between each transfer.

Figure 23-39. Single Beat 64-bit Read Cycle, 16-bit Port Size, Basic Timing

23.4.2.7 Size, Alignment and Packaging on Transfers

Table 23-15 shows the allowed sizes that an internal master can request from the EBI. The behavior of the EBI for request sizes not shown below is undefined. No error signal is asserted for these erroneous cases.

Table 23-15. Transaction Sizes Supported by EBI

Bytes (internal master)
1
2
4
3[1]
8
32

NOTES:

[1] Some misaligned access cases may result in 3-byte writes. These cases are treated as power-of-2 sized requests by the EBI, using $\overline{\text{D\_WE}}$[0:3] to make sure only the appropriate 3 bytes get written.

Even though misaligned non-burst transfers from internal masters are supported, the EBI naturally aligns the accesses when it sends them out to the external bus, splitting them into multiple aligned accesses if necessary. See Section 23.4.2.10, “Misaligned Access Support for these cases.

Natural alignment for the EBI means:

- Byte access can have any address
- 16-bit access, address bit 31 must be 0
- 32-bit access, address bits 30-31 must be 0
- For burst accesses of any size, address bits 29-31 must be 0

The EBI requires that the portion of the data bus used for a transfer to/from a particular port size be fixed. A 32-bit port must reside on data bus bits 0-31,and a 16-bit port must reside on bits 0-15.

In the following figures and tables the following convention is adopted:

- The most significant byte of a 32-bit operand is OP0, and OP3 is the least significant byte.
- The two bytes of a 16-bit operand are OP0 (most significant) and OP1, or OP2 (most significant) and OP3, depending on the address of the access.
- The single byte of a byte-length operand is OP0, OP1, OP2, or OP3, depending on the address of the access.

This can be seen in Figure 23-40.

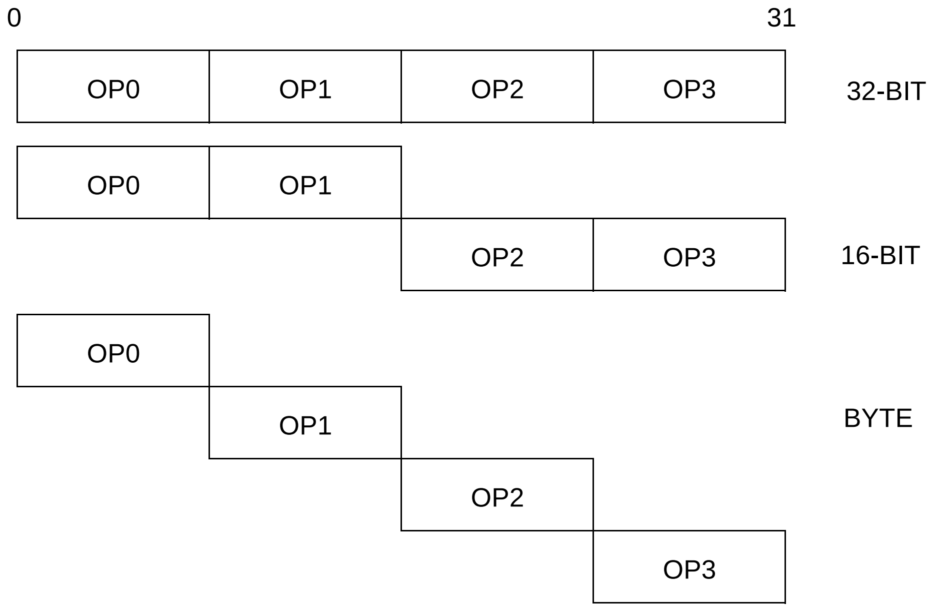


Figure 23-40. Internal Operand Representation

Figure 23-41 shows the device connections on the D_ADD_DAT[0:31] bus.

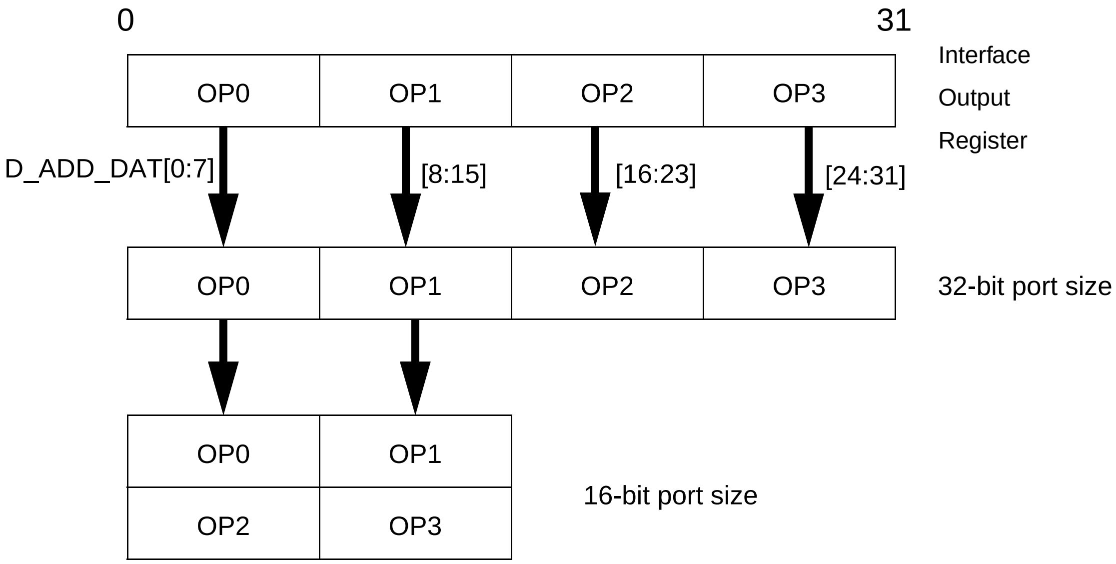


Figure 23-41. Interface to Different Port Size Devices

Table 23-16 lists the bytes required on the data bus for read cycles. The bytes indicated as '-' are not required during that read cycle.

Table 23-16. Data Bus Requirements for Read Cycles

Transfer Size	Address		32-Bit Port Size				16-Bit Port Size[1]	
	A30	A31	D0:D7	D8:D15	D16:D23	D24:D31	D0:D7[2]	D8:D15[3]
Byte	0	0	OP0	—	—	—	OP0	—
	0	1	—	OP1	—	—	—	OP1
	1	0	—	—	OP2	—	OP2	—
	1	1	—	—	—	OP3	—	OP3
16-bit	0	0	OP0	OP1	—	—	OP0	OP1
	1	0	—	—	OP2	OP3	OP2	OP3
32-bit	0	0	OP0	OP1	OP2	OP3	OP0 or OP2 [4]	OP1 or OP3

NOTES:

[1] Also applies when DBM=1 for 16-bit data bus mode.

[2] For address/data muxed transfers, D_ADD_DAT[16:23] are used externally, not D_ADD_DAT[0:7].

[3] For address/data muxed transfers, D_ADD_DAT[24:31] are used externally, not D_ADD_DAT[8:15].

[4] This case consists of two 16-bit external transactions, the first fetching OP0 and OP1, the second fetching OP2 and OP3.

Table 23-17 lists the patterns of the data transfer for write cycles when accesses are initiated by the MCU. The bytes indicated as ‘-’ are not driven during that write cycle.

Table 23-17. Data Bus Contents for Write Cycles

Transfer Size	Address		32-Bit Port Size				16-Bit Port Size[1]	
	A30	A31	D0:D7	D8:D15	D16:D23	D24:D31	D0:D7[2]	D8:D15[3]
Byte	0	0	OP0	—	—	—	OP0	—
	0	1	OP1	OP1	—	—	—	OP1
	1	0	OP2	—	OP2	—	OP2	—
	1	1	OP3	OP3	—	OP3	—	OP3
16-bit	0	0	OP0	OP1	—	—	OP0	OP1
	1	0	OP2	OP3	OP2	OP3	OP2	OP3
32-bit	0	0	OP0	OP1	OP2	OP3	OP0 or OP2 [4]	OP1 or OP3

NOTES:

[1] Also applies when DBM=1 for 16-bit data bus mode.

[2] For address/data muxed transfers, D_ADD_DAT[16:23] are used externally, not D_ADD_DAT[0:7].

[3] For address/data muxed transfers, D_ADD_DAT[24:31] are used externally, not D_ADD_DAT[8:15].

[4] This case consists of two 16-bit external transactions, the first writing OP0 and OP1, the second writing OP2 and OP3.

23.4.2.8 Termination Signals Protocol

The termination signals protocol was defined in order to avoid electrical contention on lines that can be driven by various sources. In order to do that, a slave must not drive signals associated with the data transfer until the address phase is completed and it recognizes the address as its own. The slave must disconnect from signals immediately after it acknowledges the cycle and not later than the termination of the next address phase cycle.

For EBI-mastered non-chip-select accesses, the EBI requires assertion of D_TA from an external device to signal that the bus cycle is complete. The EBI uses a latched version of D_TA (1 cycle delayed) for these accesses to help make timing at high frequencies. This results in the EBI driving the address and control signals 1 cycle longer than required, as seen in Figure 23-42. However, the D_ADD_DAT does not need to be held 1 cycle longer by the slave, because the EBI latches D_ADD_DAT every cycle during non-chip-select accesses. During these accesses, the EBI does not drive the D_TA signal, leaving it up to an external device (or weak internal pullup) to drive D_TA.

For EBI-mastered chip-select accesses, when the SETA bit is 0, the EBI drives D_TA the entire cycle, asserting according to internal wait state counters to terminate the cycle. When the SETA bit is 1, the EBI samples the D_TA for the entire cycle. During idle periods on the external bus, the EBI drives D_TA negated as long as it is granted the bus; when it no longer owns the bus, it lets go of D_TA.

If no device responds by asserting D_TA within the programmed timeout period (BMT in EBI_BMCR) after the EBI initiates the bus cycle, the internal Bus Monitor (if enabled) asserts D_TEA to terminate the cycle. An external device may also drive D_TEA when it detects an error on an external transaction. D_TEA assertion causes the cycle to terminate and the processor to enter exception processing for the error condition. To properly control termination of a bus cycle for a bus error with external circuitry, D_TEA must be asserted at the same time or before (external) D_TA is asserted. D_TEA must be negated before the second rising edge after it was sampled asserted in order to avoid the detection of an error for the following bus cycle initiated. D_TEA is only driven by the EBI during the cycle where the EBI is asserting D_TEA and the cycle immediately following this assertion (for fast negation). During all other cycles, the EBI relies on a weak internal pullup to hold D_TEA negated. This allows an external device to assert D_TEA when it needs to indicate an error. External devices must follow the same protocol as the EBI, only driving D_TEA during the assertion cycle and 1 cycle afterwards for negation.

When D_TEA is asserted from an external source, the EBI uses a latched version of D_TEA (1 cycle delayed) to help make timing at high frequencies. This means that for any accesses where the EBI drives D_TA (chip-select accesses with SETA=0), a D_TEA assertion that occurs 1 cycle before or during the last D_TA of the access could be ignored by the EBI, since it will have completed the access internally before it detects the latched D_TEA assertion. This means that non-burst chip-select accesses with no wait states (SCY=0) cannot be reliably terminated by external D_TEA. If external error termination is required for such a device, the EBI must be configured for SCY>=1.

NOTE

> For the cases discussed above where D_TEA “could be ignored”, this is not gauranteed. For some small access cases (which always use chip-select and internally-driven D_TA), a D_TEA that occurs 1 cycle before or during the D_TA cycle or for SCY=0 may in fact lead to terminating the cycle with error. However, proper error termination is not guaranteed for these cases, so D_TEA must always be asserted at least 2 cycles before an internally-driven D_TA cycle for proper error termination.

External D_TEA assertion that occurs during the same cycle that D_TS is asserted by the EBI is always treated as an error (terminating the access) regardless of SCY.

Table 23-18 summarizes how the EBI recognizes the termination signals provided from an external device.

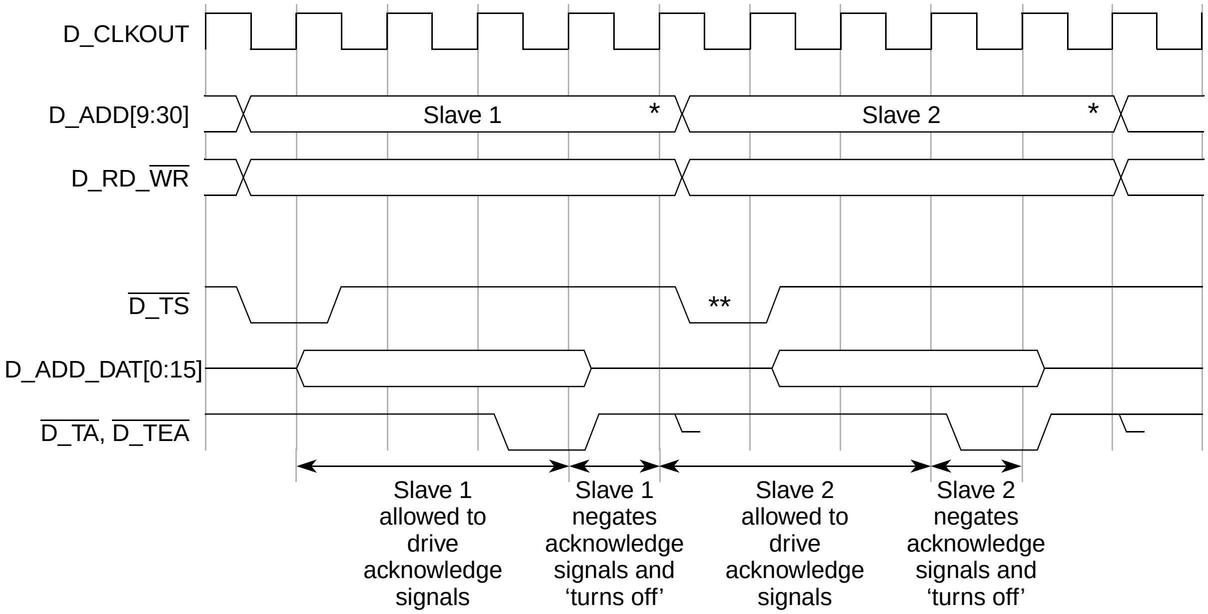
Table 23-18. Termination Signals Protocol

D_TEA[1]	D_TA[1]	Action
Negated	Negated	No Termination
Asserted	X	Transfer Error Termination
Negated	Asserted	Normal Transfer Termination

NOTES:

[1] Latched version (1 cycle delayed) used for externally driven D_TEA and D_TA.

Figure 23-42 shows an example of the termination signals protocol for back-to-back reads to two different slave devices who properly “take turns” driving the termination signals. This assumes a system using slave devices that drive termination signals.



* The EBI drives address and control signals an extra cycle because it uses a latched version of D_TA (1 cycle delayed) to terminate the cycle. An external master is not required to do this.

** This is the earliest that the EBI can start another transfer, in the case of continuing a set of small accesses. For all other cases, an extra cycle is needed before the EBI can start another D_TS.

Figure 23-42. Termination Signals Protocol Timing Diagram

23.4.2.9 Non-Chip-Select Burst in 16-bit Data Bus Mode

The timing diagrams in this section apply only to the special case of a non-chip-select 32-bit access in 16-bit data bus mode (DBM=1 in EBI_MCR).

For this case, a special 2-beat burst protocol is used for reads and writes, so that a slave device (using the same EBI) can internally generate one 32-bit read or write access (thus 32-bit coherent), as opposed to two separate 16-bit accesses.

NOTE

> Since this device does not support multi-master systems, the original intent of this protocol does not apply. However, this 2-beat burst protocol can also occur in a single-master system, if a non-chip-select 32-bit access to a 16-bit port is performed.

Figure 23-43 shows a 32-bit non-chip-select read from an external master in 16-bit data bus mode.

Figure 23-44 shows a 32-bit non-chip-select write from an external master in 16-bit data bus mode.

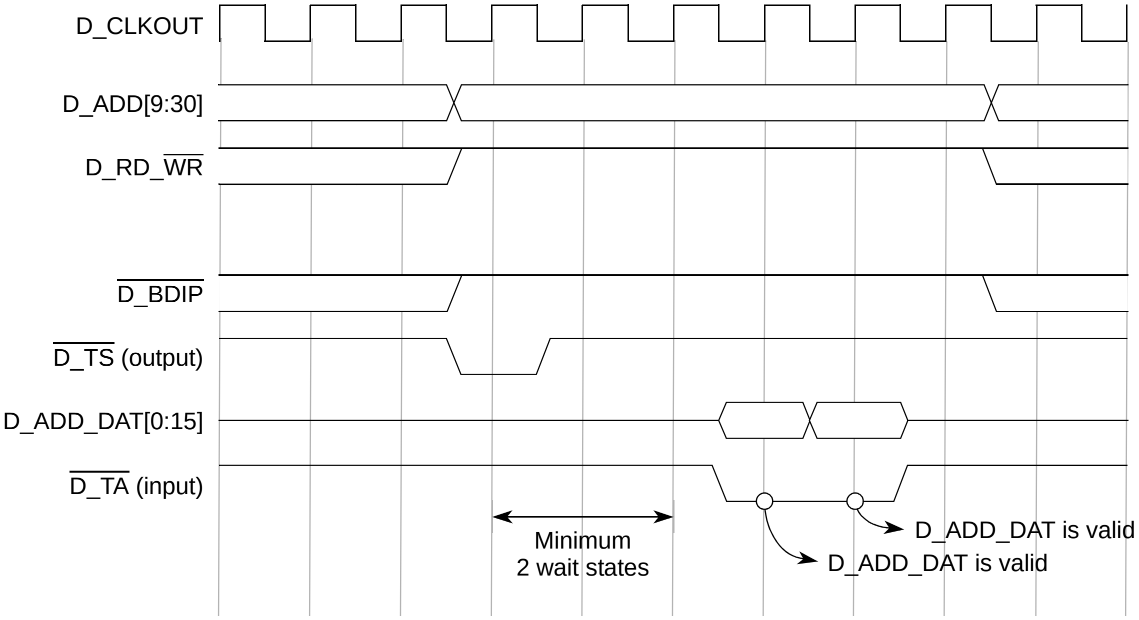


Figure 23-43. 32-bit non-Chip-Select Read from MCU with DBM=1

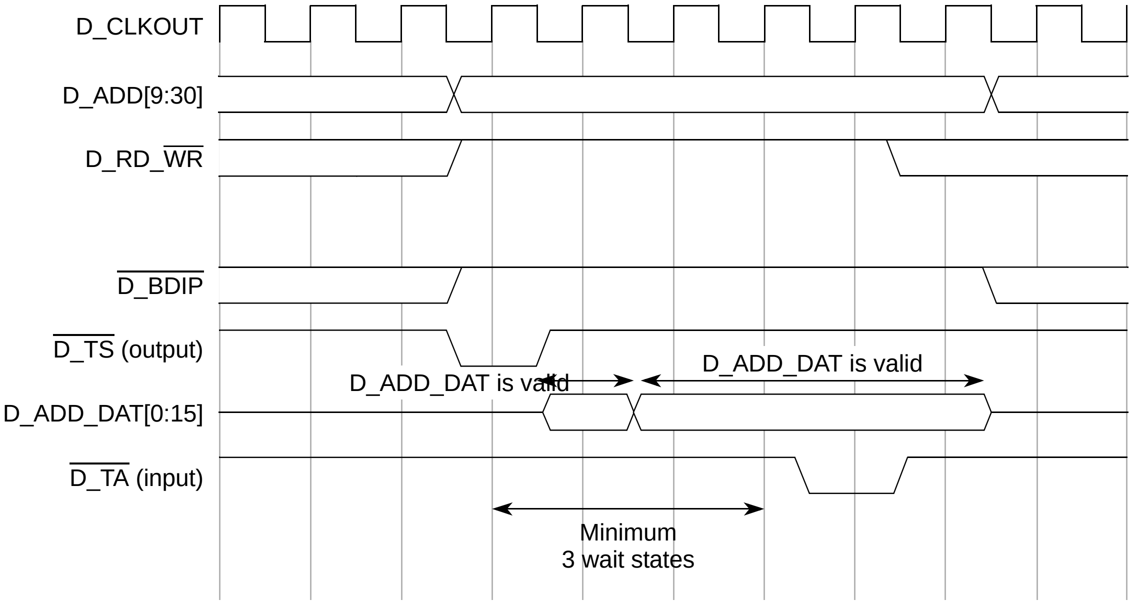


Figure 23-44. 32-bit non-Chip-Select Write to MCU with DBM=1

23.4.2.10 Misaligned Access Support

This section describes all the misaligned cases supported by the EBI. The EBI works under the assumption that all internal masters on the device do not produce any misaligned access cases (to the EBI) other than the ones below.

23.4.2.10.1 Misaligned Access Support (64 bit AMBA)

Table 23-19 shows all the misaligned access cases supported by the EBI (using a 64-bit AMBA implementation), as seen on the internal master AMBA bus. All other misaligned cases are not supported. If an unsupported misaligned access to the EBI is attempted (such as non-chip-select or burst misaligned access), the EBI errors the access on the internal bus and does not start the access (nor assert D_TEA) externally.

Table 23-19. Misalignment Cases Supported by a 64 bit AMBA EBI (internal bus)

No.[1]	Program Size and byte offset	Address [29:31][2]	Data Bus Byte Strobes[3]	HSIZE[4]	HUNALIGN[5]
1	Half @0x1,0x9	001	0110_0000	10	1
2	Half @0x3,0xB	011	0001_1000	11	1
3	Half @0x5,0xD	101	0000_0110	10	1
4 -	Half @0x7, 0xF (2 AHB transfers)	111 000	0000_0001 1000_0000	01[6] 00	1 0
5	Word @0x1,0x9	001	0111_1000	11	1
6	Word @0x2,0xA	010	0011_1100	11	1
7	Word @0x3,0xB	011	0001_1110	11	1
8 -	Word @0x5,0xD (2 AHB transfers)	101 000	0000_0111 1000_0000	10 00	1 0
9 -	Word @0x6, 0xE (2 AHB transfers)	110 000	0000_0011 1100_0000	10[7] 01	1 0

Table 23-19. Misalignment Cases Supported by a 64 bit AMBA EBI (internal bus) (continued)

No.[1]	Program Size and byte offset	Address [29:31][2]	Data Bus Byte Strobes[3]	HSIZE[4]	HUNALIGN[5]
10 11	Word @0x7,0xF (2 AHB transfers)	111 000	0000_0001 1110_0000	10[6] 10	1 1
12 -	Doubleword @0x4,0x8 (2 AHB transfers)	100 000	0000_1111 1111_0000	11[8] 10	1 0
13 -	Doubleword @0x2,0xA (2 AHB transfers)	010 000	0011_1111 1100_0000	11 01	1 0
14 15	Doubleword 0x6,0xE (2 AHB transfers)	110 000	0000_0011 1111_1100	11[7] 11	1 1

NOTES:

[1] Misaligned case number. Only transfers where HUNALIGN=1 are numbered as misaligned cases.

[2] Address on internal master AHB bus, not necessarily address on external D_ADD pins.

[3] Internal byte strobe signals on AHB bus. Shown with Big-Endian byte ordering in this table, even though internal master AHB bus uses Little-Endian byte-ordering (EBI flips order internally).

[4] Internal signal on AHB bus; 00=8-bits, 01=16 bits, 10=32 bits, 11=64-bits. HSIZE is driven according to the smallest aligned container that contains all the requested bytes. This results in extra EBI external transfers in some cases.

[5] Internal signal on AHB bus that indicates that this transfer is misaligned (when 1).

[6] For this case, the EBI internally treats HSIZE as 00 (1-byte access).

[7] For this case, the EBI internally treats HSIZE as 01 (2-byte access).

[8] For this case, the EBI internally treats HSIZE as 10 (4-byte access).

Table 23-20 shows which external transfers are generated by the EBI for the misaligned access cases in Table 23-19, for each port size.

The number of external transfers for each internal AHB master request is determined by the HSIZE value for that request relative to the port size. For example, a half-word write to @011 (misaligned case #2) with 16-bit port size results in 4 external 16-bit transfers because the HSIZE is 64-bits. For cases where two or more external transfers are required for one internal transfer request, these external accesses are considered part of a “small access” set, as described in Section 23.4.2.6, “Small Accesses (Small Port Size and Short Burst Length).

Since all transfers are aligned on the external bus, normal timing diagrams and protocol apply.

Table 23-20. Misalignment Cases Supported by a 64 bit AMBA EBI (external bus)

No.[1]	PS[2]	Program Size and byte offset	D_ADD[29:31][3]	$\overline{\text{D\_WE}}$[0:3][4]
1	0	Half @0x1,0x9	000	1001
	1		000 010	1011 0111
2	0	Half @0x3,0xB	000 100	1110 0111
	1		010 100	1011 0111
3	0	Half @0x5,0xD	100	1001
	1		100 110	1011 0111

Table 23-20. Misalignment Cases Supported by a 64 bit AMBA EBI (external bus) (continued)

No.[1]	PS[2]	Program Size and byte offset	D_ADD[29:31][3]	$\overline{\text{D\_WE}}$[0:3][4]
4	0	Half @0x7,0xF (2 AHB transfers)	111[5]	1110
-			000	0111
4	1		110	1011
-			000	0111
5	0	Word @0x1,0x9	000 100	1000 0111
	1		000 010 100	1011 0011 0111
6	0	Word @0x2,0xA	000 100	1100 0011
	1		010 100	0011 0011
7	0	Word @0x3,0xB	000 100	1110 0001
	1		010 100 110	1011 0011 0111
8	0	Word @0x5,0xD (2 AHB transfers)	100	1000
-			000	0111
8	1		100 110	1011 0011
-			000	0111
9	0	Word @0x6,0xE (2 AHB transfers)	110[6]	1100
-			000	0011
9	1		110[6]	0011
-			000	0011
10	0	Word @0x7,0xF (2 AHB transfers)	111[5]	1110
11			000	0001
10	1		111[5]	1011
11			000 010	0011 0111
12	0	Doubleword @0x4,0xC (2 AHB transfers)	100[7]	0000
-			000	0000
12	1		100[7] 110	0011 0011
-			000 010	0011 0011
13	0	Doubleword @0x2,0xA (2 AHB transfers)	000 100	1100 0000
-			000	0011
13	1		010 100 110	0011 0011 0011
-			000	0011

Table 23-20. Misalignment Cases Supported by a 64 bit AMBA EBI (external bus) (continued)

No.[1]	PS[2]	Program Size and byte offset	D_ADD[29:31][3]	$\overline{\text{D\_WE}}$[0:3][4]
14	0	Doubleword @0x6,0xE (2 AHB transfers)	110[6]	1100
15			000 100	0000 0011
14	1		110[6]	0011
15			000 010 100	0011 0011 0011

NOTES:

[1] Misaligned case number, from Table 23-19.

[2] Port size; 0=32 bits, 1=16 bits.

[3] External D_ADD pins, not necessarily the address on internal master AHB bus.

[4] External $\overline{\text{D\_WE}}$ pins. Note that these pins have negative polarity, opposite of the internal byte strobes in Table 23-19.

[5] Treated as 1-byte access.

[6] Treated as 2-byte access.

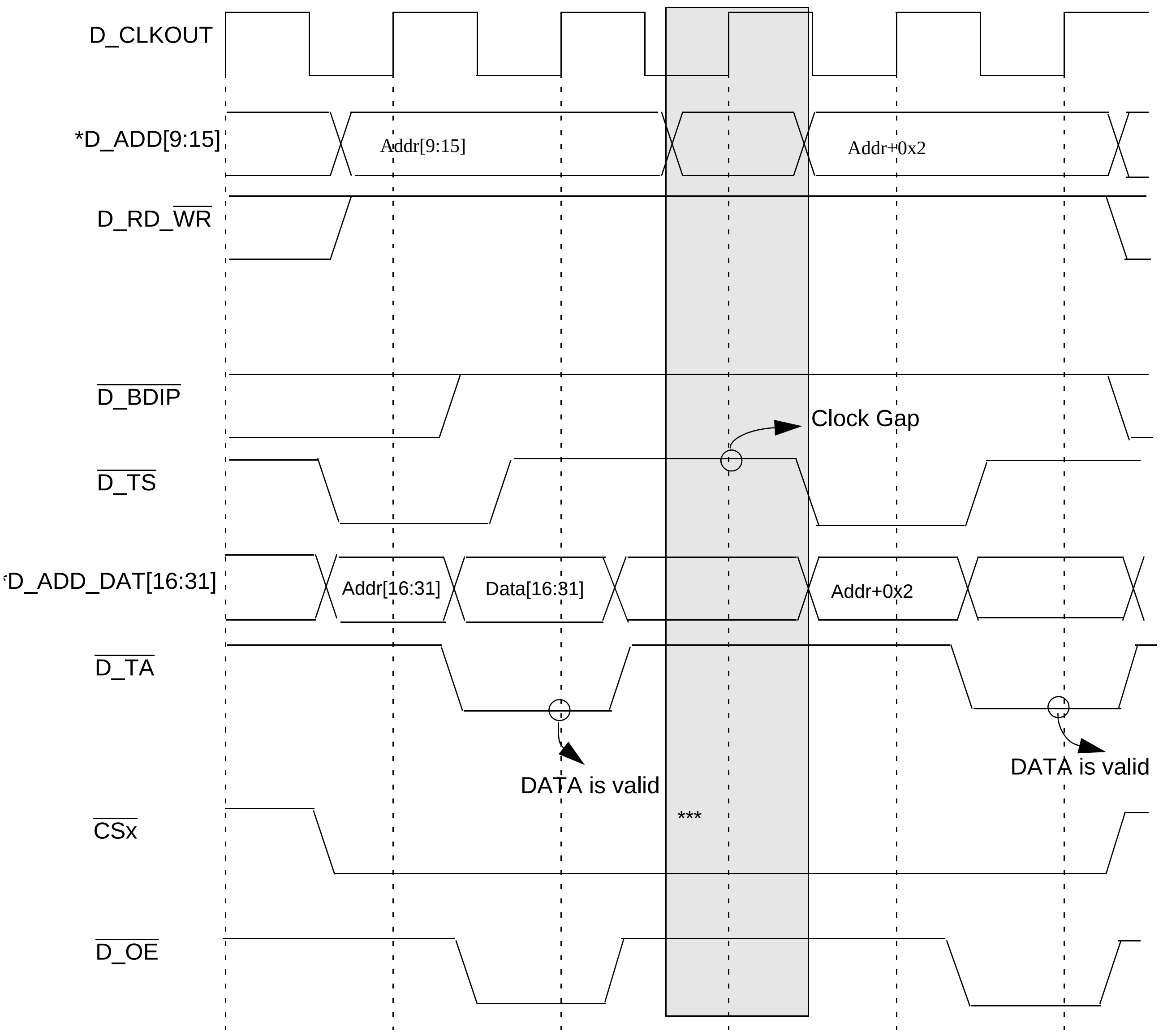
[7] Treated as 4-byte access.

23.4.2.11 Address Data Multiplexing

Address/Data multiplexing enables the design of a system with reduced pin count. In such a system, multiplexed address/data functions (on D_ADD_DAT pins) are used, instead of having separate address and data pins. Compared to the normal EBI specification (e.g. 24 address pins+32 data pins), only 32 data pins are required (e.g. D_ADD_DAT[0:15] and D_ADD_DAT[16:31]). Compared to a 16-bit bus implementation, only 24 pins are required (e.g. D_ADD[8:15] + D_ADD_DAT[16:31]).

When performing a small access read, as described in Section 23.4.2.6, “Small Accesses (Small Port Size and Short Burst Length), with A/D multiplexing enabled for this access, the EBI inserts an idle clock cycle with $\overline{\text{D\_OE}}$ negated and $\overline{\text{CS}}$ asserted, to allow for the memory to three-state the bus prior to the EBI driving the address on the next clock. This clock gap already exists (for other reasons) for non-small-access transfers, so no additional clock gap is inserted for those cases. See Figure 23-45 for an example of a small access read with A/D multiplexing enabled.

In general, timing diagrams in A/D multiplexing mode are very similar to other diagrams in this document, except for the behavior of the D_ADD_DAT busses, which can be seen in Figure 23-45.



* While the EBI drives all of ADDR[3:31] to valid address, typically only ADDR[3:15] (or less) are used in the system, as D_ADD_DATA[16:31] (or D_ADD_DATA[0:15]) would be used for address and data on an external mu

** Or D_ADD_DATA[0:15], based on D16_31 bit in EBI_MCR.

*** If EBI_CAL_BRn[GCSN]=1, then CSx is negated during this idle cycle.

Figure 23-45. Small access (32-bit read to 16-bit port) on Address/Data multiplexed bus

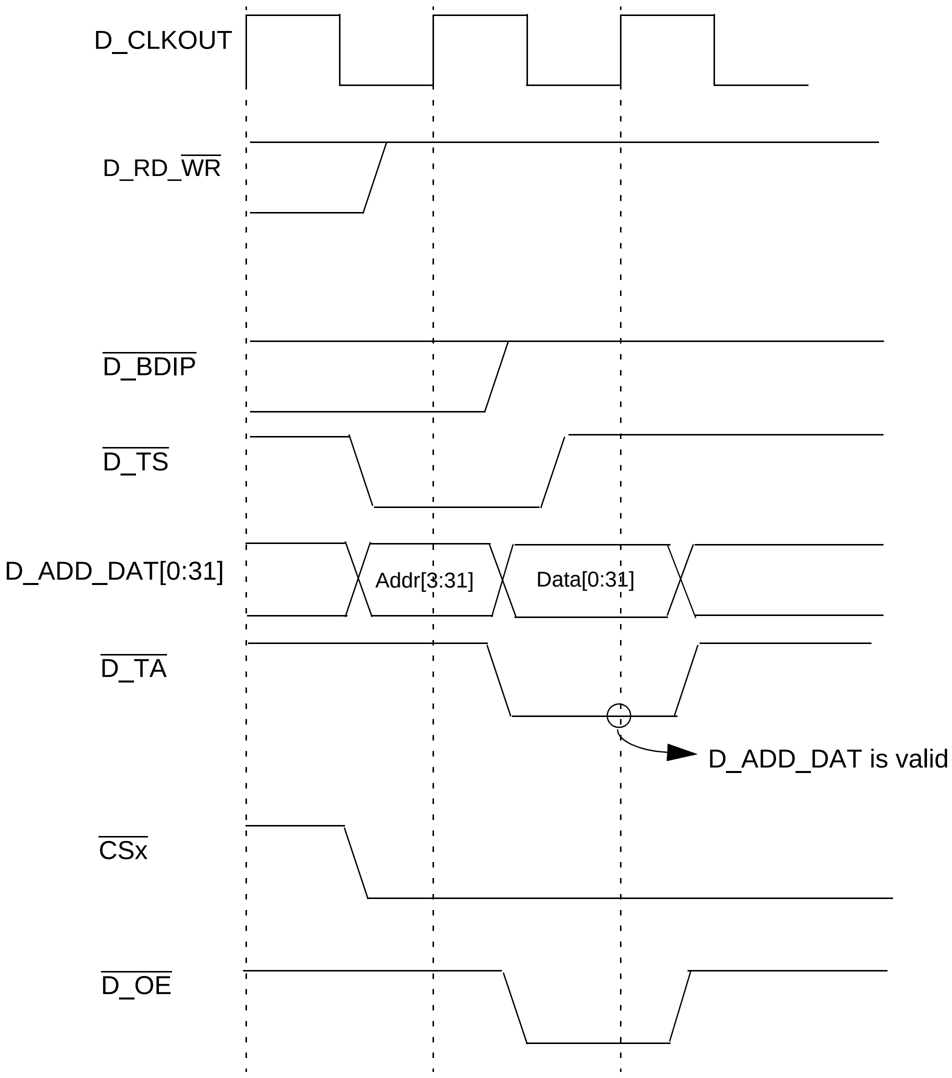


Figure 23-46. 32-bit read to 32-bit port on Address/Data multiplexed bus

23.5 Initialization/Application Information

23.5.1 Booting from External Memory

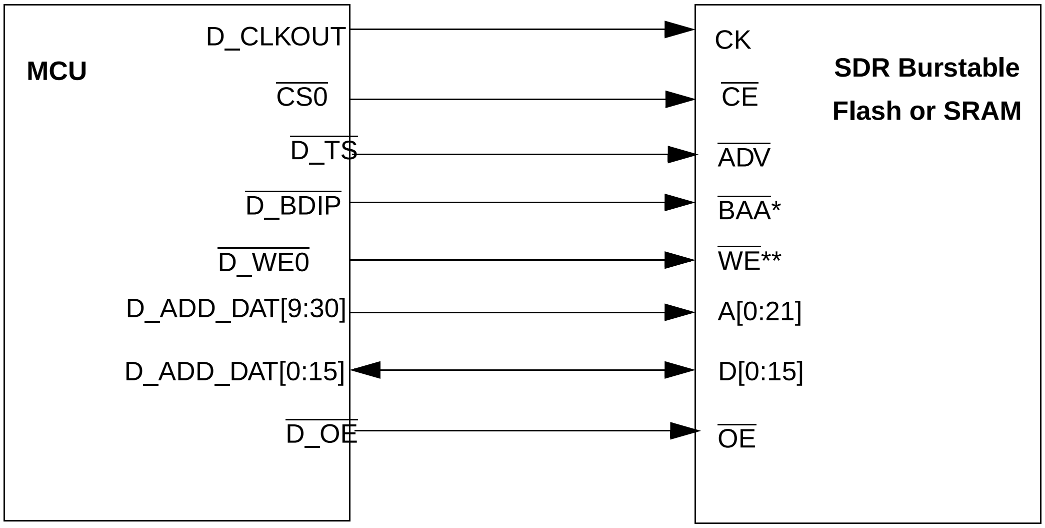
The EBI block does not support booting directly to external memory (i.e. fetching the first instruction after reset externally). One common method for an MCU to resemble an external boot with this EBI is to use an internal Boot Assist Module on the MCU, which fetches the first instruction internally and configures EBI registers before branching to an external address to “boot” externally.

If code in external memory needs to write EBI registers, this must be done in a way that avoids modifying EBI registers while external accesses are being performed, such as the following method:

- Copy the code that is doing the register writes (plus a return branch) to internal SRAM
- Branch to internal SRAM to run this code, ending with a branch back to external flash

23.5.2 Running with SDR (Single Data Rate) Burst Memories

This includes FLASH and SRAM memories with a compatible burst interface. $\overline{\text{D\_BDIP}}$ is required only for some SDR memories. Figure 23-47 shows a block diagram of an MCU connected to a 16-bit SDR burst memory.



* May or may not be connected, depending on the memory used.

** Flash memories typically use one $\overline{\text{WE}}$ signal as shown, RAMs use 2

Figure 23-47. MCU Connected to SDR Burst Memory

NOTE

An external latch may be required to hold Address bus signals during the bus access if the memory does not support Address changing on the Data phase of the bus access. In this case, the ALE signal should be used to signal when the address bus should be latched.

Refer to Figure 23-34 for an example of the timing of a typical Burst Read operation to an SDR burst memory. Refer to Figure 23-16 for an example of the timing of a typical Single Write operation to SDR memory.

23.5.3 Running with Asynchronous Memories

The EBI also supports asychronous memories. In this case, the D_CLKOUT, $\overline{\text{D\_TS}}$, and $\overline{\text{D\_BDIP}}$ pins are not used by the memory and bursting is not supported. However, the EBI still drives these outputs, and always drives and latches all signals at posedge D_CLKOUT (i.e., there is no “asynchronous mode” for the EBI). The data timing is controlled by setting the SCY bits in the appropriate Option Register to the proper number of wait states to work with the access time of the asynchronous memory, just as done for a synchronous memory.

23.5.3.1 Example Wait State Calculation

This example applies to any chip-select memory, synchronous or asynchronous.

As an example, say we have a memory with 50ns access time, and we are running the external bus @ 66 MHz (D_CLKOUT period: 15.2ns). Assume the input data spec for the MCU is 4ns.

number of wait states = (access time) / (D_CLKOUT period) + (0 or 1) (depending on setup time)

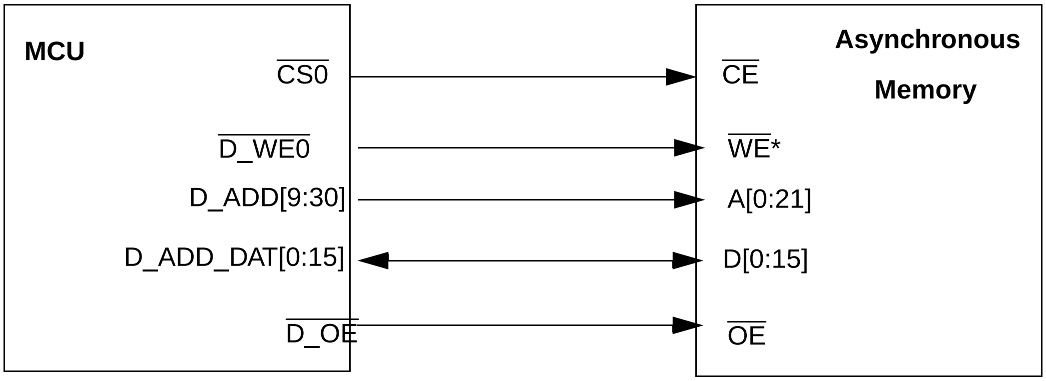
50/15.2 = 3 with 4.4ns remaining (so we need at least 3 wait states, now check setup time)

15.2-4.4=10.8ns (this is the achieved input data setup time)

Since actual input setup (10.8ns) is greater than the input setup spec (4.0ns), 3 wait states is sufficient. If the actual input setup was less than 4.0ns, we would have to use 4 wait states instead.

23.5.3.2 Timing and Connections for Asynchronous Memories

The connections to an asynchronous memory are the same as for a synchronous memory, except that the D_CLKOUT, $\overline{\text{D\_TS}}$, and $\overline{\text{D\_BDIP}}$ signals are not used. Figure 23-48 shows a block diagram of an MCU connected to an asynchronous memory.



* Flash memories typically use one $\overline{\text{WE}}$ signal as shown, RAMs use 2 or 4 (16-bit or 32-bit)

Figure 23-48. MCU Connected to Asynchronous Memory

Figure 23-49 shows a timing diagram of a read operation to a 16-bit asynchronous memory using 3 wait states.

Figure 23-50 shows a timing diagram of a write operation to a 16-bit asynchronous memory using 3 wait states.

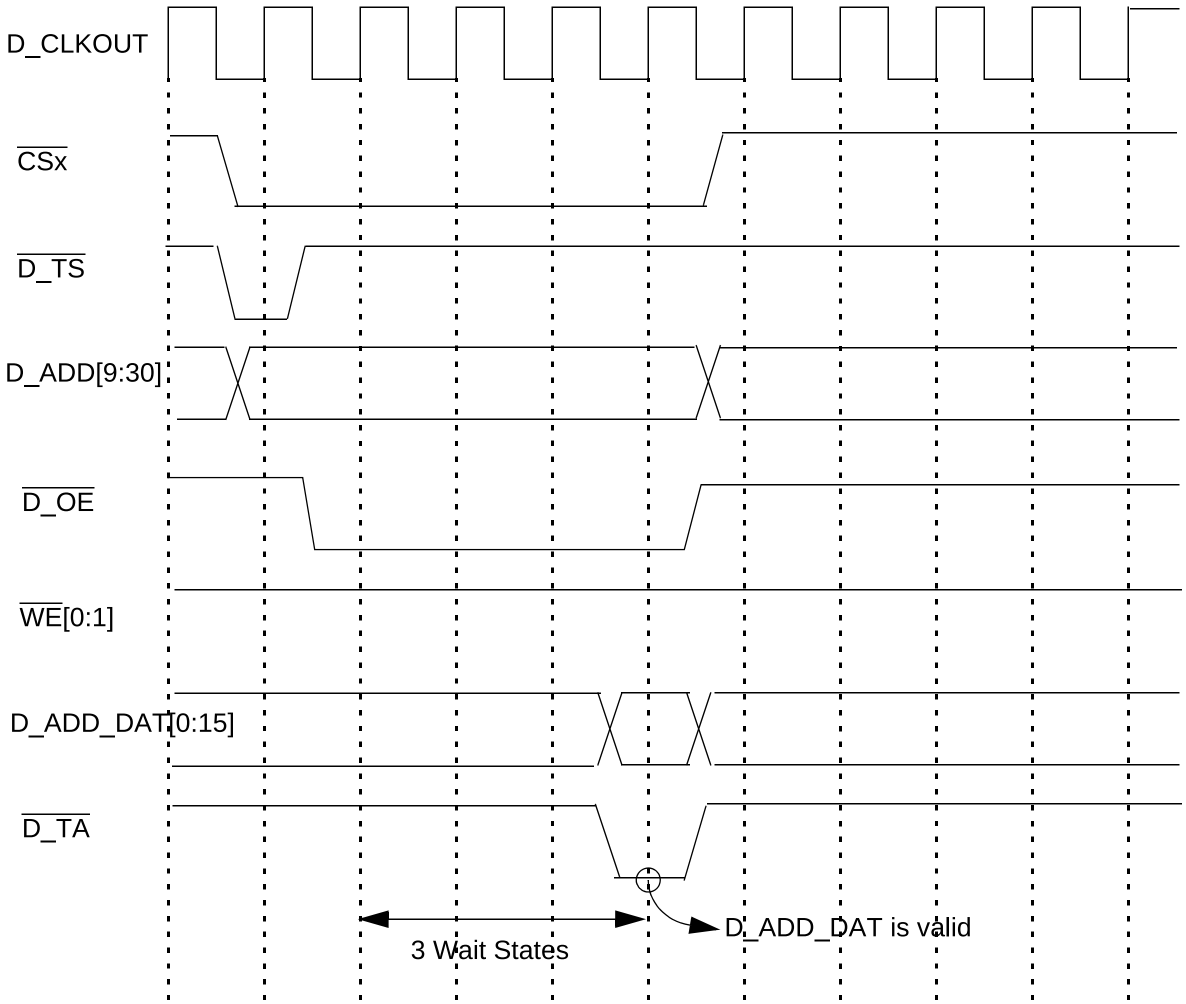


Figure 23-49. Read Operation to Asynchronous Memory, Three Initial Wait States

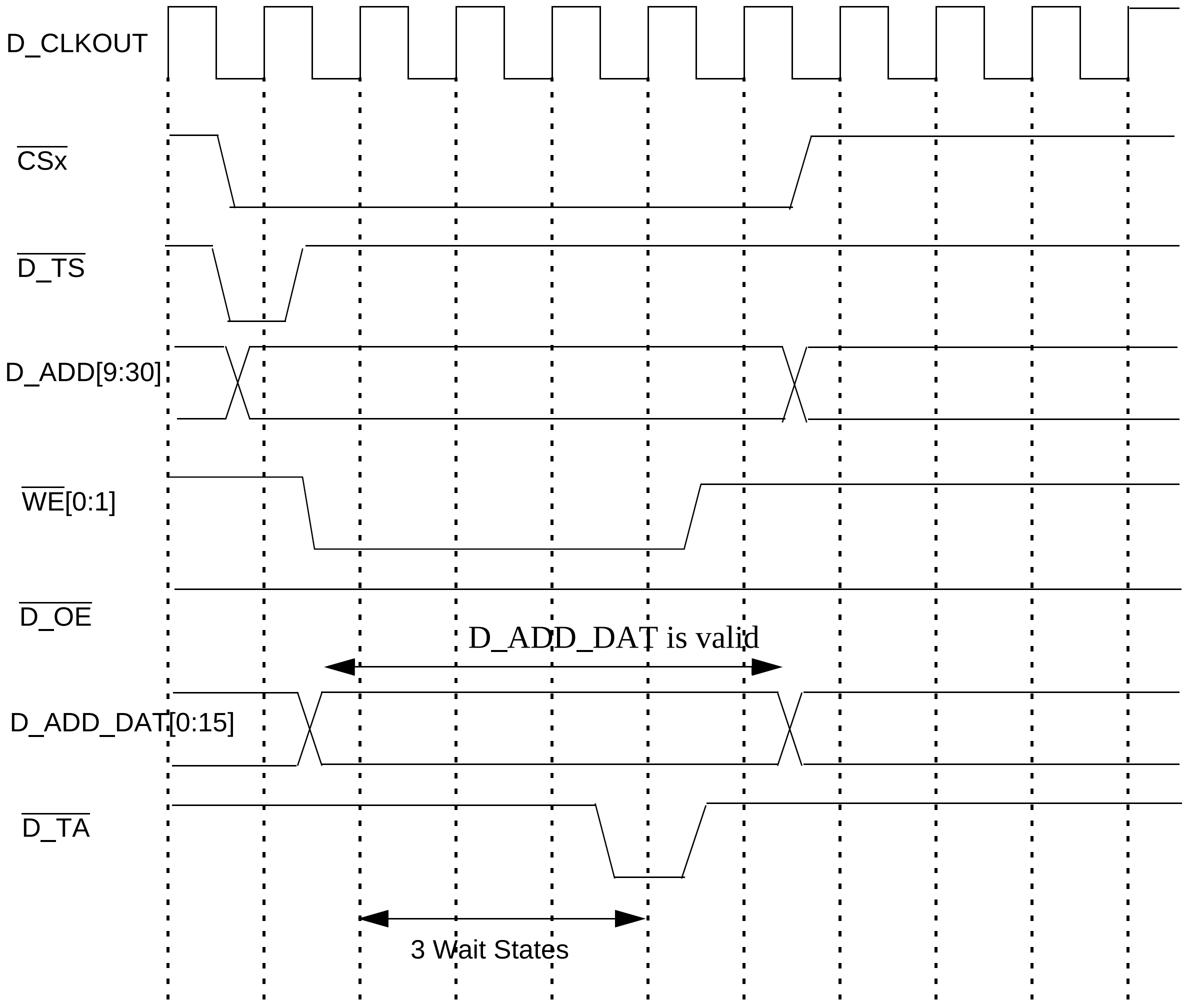
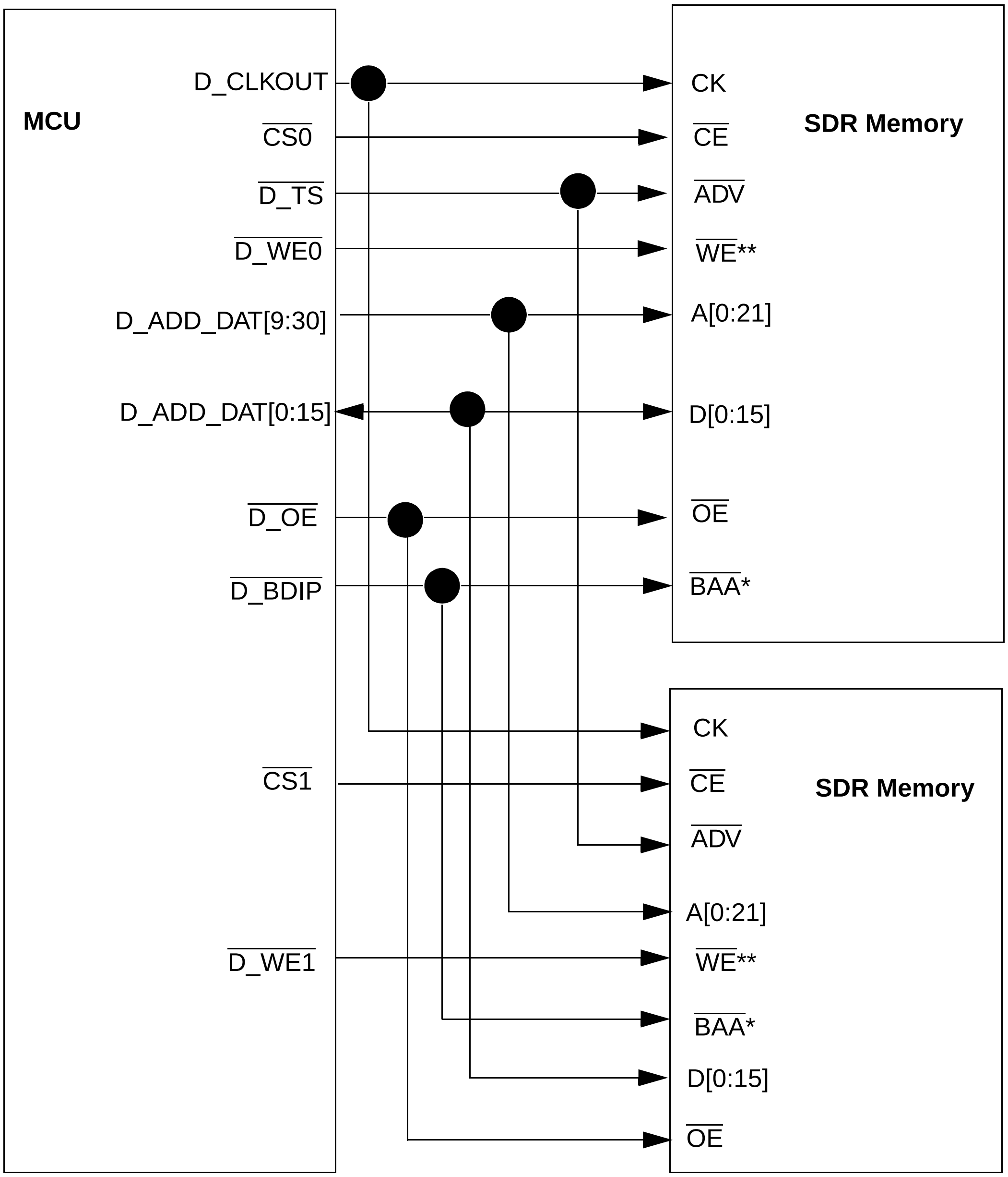


Figure 23-50. Write Operation to Asynchronous Memory, Three Initial Wait States

23.5.4 Connecting an MCU to Multiple Memories

The MCU can be connected to more than one memory at a time.

Figure 23-51 shows an example of how two memories could be connected to one MCU.



* May or may not be connected, depending on the memory used.

** Flash memories typically use one WE signal as shown, RAMs use 2

Figure 23-51. MCU Connected to Multiple Memories

Chapter 24
Flash Memory Array and Control

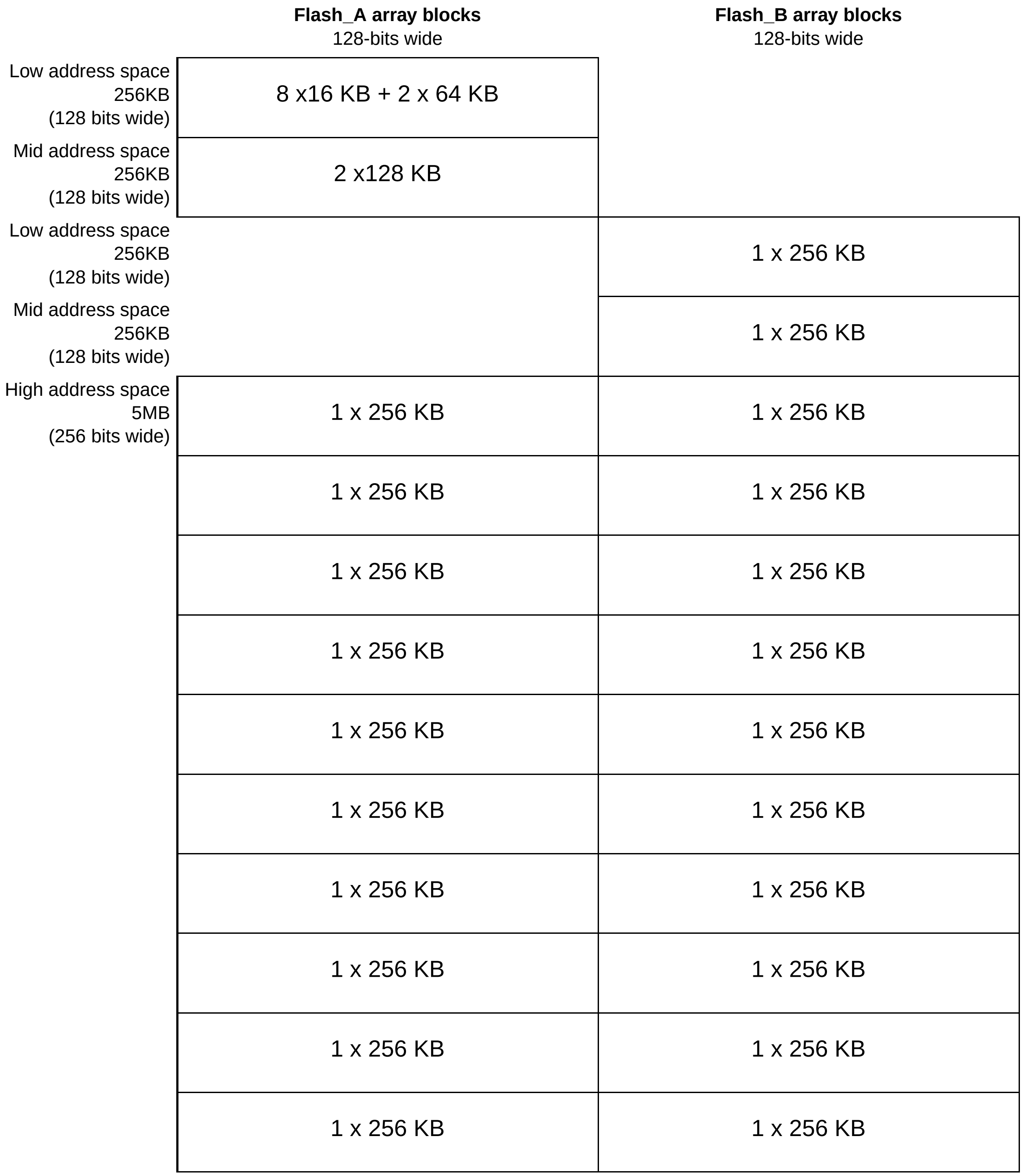
24.1 Introduction

This section presents information about the following components on this device:

- The flash memory blocks
- The flash memory controller

The primary function of the flash memory module is to serve as electrically programmable and erasable non-volatile memory. The NVM memory can be used for instruction and data storage.

The flash memory module contains two physical arrays (Flash_A and Flash_B) that are organized to provide a contiguous address range for program code and data. Each flash array has three address spaces: low-address space, mid-address space, and high-address space that are subdivided into blocks as shown in Figure 24-1.



Note: The High Address space blocks are interleaved on every 16 byte address boundary; the first 16 bytes of every 32 bytes are located in Flash_A array, and the second 16 bytes are located in Flash_B array.

Figure 24-1. Flash Segmentation

24.1.1 Block Diagram

24.1.1.1 Flash Memory Module

Figure 24-4 shows a block diagram of the flash memory module. Program execution and data fetches from the Flash module are performed through the FBIU. For this purpose, each core has a separate physical port to the FBIU connected to separate slave ports on the XBAR, as shown in Figure 24-2. Other bus masters may also access the Flash memory contents through one of the ports. Refer to Chapter 11, “AMBA Crossbar Switch (XBAR)”, Table 11-1 for details of the bus master mapping to Flash slave ports. Control and status registers, used for flash memory reprogramming purposes are accessed through PBRIDGE_A.

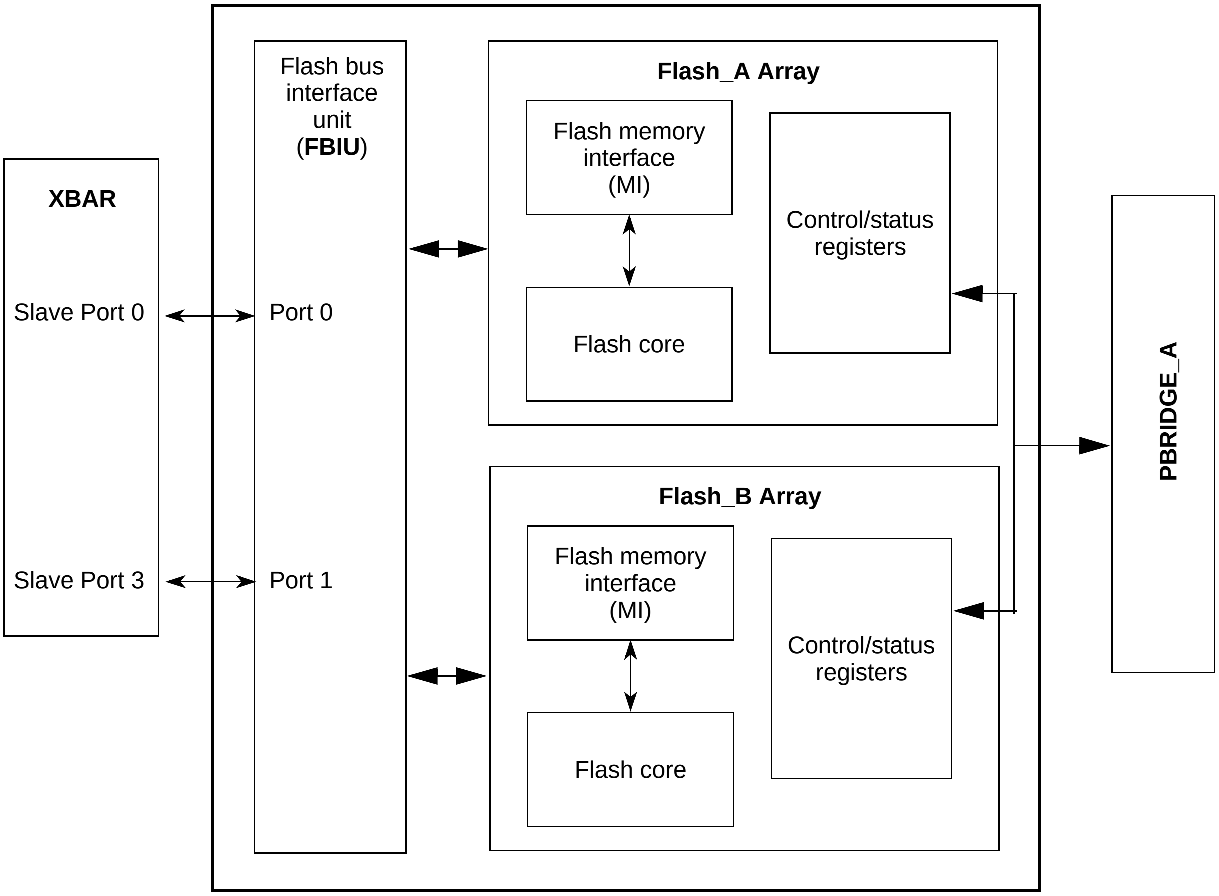


Figure 24-2. Flash Memory Module Block Diagram

24.1.1.2 Dual Ported Flash Bus Interface

The dual ported FBIU provides separate read buffers for each port, and separate read buffers for 128 bit and 256 bit fetches on each port. Several prefetch control algorithms are available for controlling line read buffer fills. Prefetch triggering may be restricted to instruction accesses only, data accesses only, or may be unrestricted. Prefetch triggering may also be controlled on a per-master basis. Buffers may also be selectively enabled or disabled for allocation by instruction and data prefetch. Access protections may be applied on a per-master basis for both reads and writes to support security and privilege mechanisms.

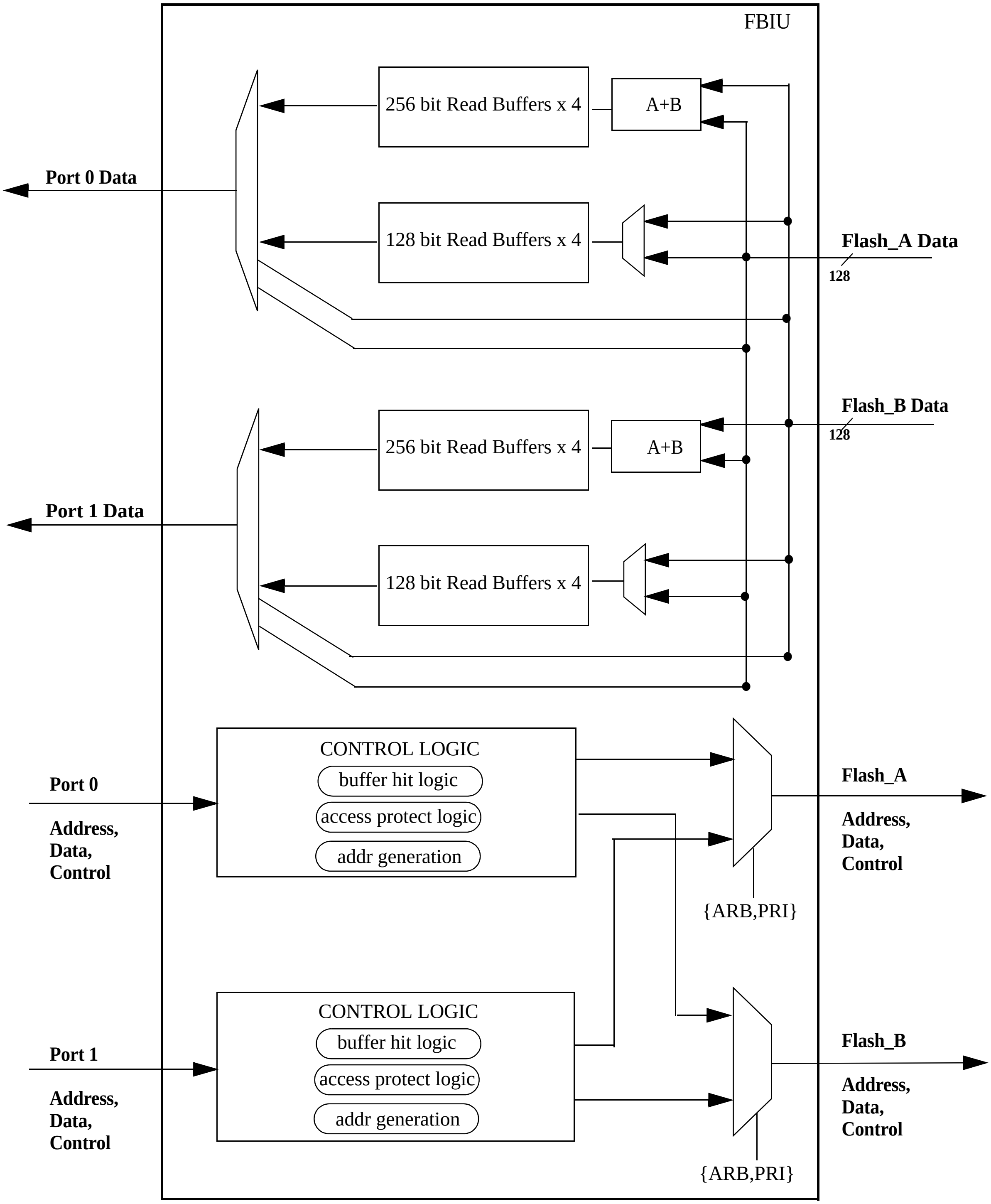


Figure 24-3. Dual Port Flash Controller Block Diagram

24.1.2 Features

The flash memory module has these major features:

- Support for a 64-bit data bus for instruction fetch.
- Two 64 bit slave ports to accelerate concurrent access by two bus masters.
- Support for a 32-bit data bus for CPU loads and DMA access. Byte, halfword, word and doubleword reads are supported. Only aligned word and doubleword writes are supported.
- Configurable read buffering and line prefetch support. Two sets of buffers (one for 128-bit accesses and one for 256-bit accesses) per port and a prefetch controller are used to support single-cycle read responses for hits in the buffers.
- Hardware and software configurable read and write access protections on a per-master basis.
- Interface to the flash array controller is pipelined with a depth of 1, allowing overlapped accesses to proceed in parallel.
- Configurable access timing allowing use in a wide range of system frequencies.
- Multiple-mapping support and mapping-based block access timing (0-31 additional cycles) allowing use for emulation of other memory types.
- Software programmable block program/erase restriction control for low, mid and high address spaces.
- Erase of selected block(s).
- Read page size of 128 bits (low/mid-address space) and 256 bits (for high-address space).
- Support for concurrent dual port access to low/mid-address space of Flash_A and Flash_B.
- ECC with single-bit correction, single-bit detection, and double-bit detection.
- Minimum program size is 2 consecutive 32 bit words, aligned on a 0-modulo-8 byte address, due to ECC.
- Embedded hardware program and erase algorithm.
- Read while Write with multiple partitions.
- Erase suspend, program suspend and erase-suspended program.
- Automotive flash which meets automotive endurance and reliability requirements.
- Shadow information stored in non-volatile shadow block.
- Independent program/erase of the shadow block.

24.1.3 Modes of Operation

24.1.3.1 Flash User Mode

User mode is the default operating mode of the flash module. In this mode, it is possible to read and write, program and erase the flash module.

24.1.3.2 User Test Mode

User Test (UTest) mode is where a portion of the Freescale test modes are made available to end users. This mode is protected, but accessible.

24.2 Memory Map and Registers

This section provides a detailed description of all flash memory registers.

24.2.1 Module Memory Map

The flash memory map is shown below. The addresses are given as an offset to the flash memory base address.

There are no program-visible registers that physically reside in the flash. The flash controller contains the registers to control and configure the flash (see Table 24-3). Only reference these registers with 32-bit accesses.

Table 24-1. Memory Map

Offset from FLASH_BASE (0x0000_0000)	Use	Flash A Block	Flash B Block	Partition	Flash A Block Size	Flash B Block Size	Data Width
0x0000_0000	Low-address space (Flash A)	L0	—	1	16K	—	128
0x0000_4000		L1	—		16K	—	128
0x0000_8000		L2	—		16K	—	128
0x0000_C000		L3	—		16K	—	128
0x0001_0000		L4	—	2	16K	—	128
0x0001_4000		L5	—		16K	—	128
0x0001_8000		L6	—		16K	—	128
0x0001_C000		L7	—		16K	—	128
0x0002_0000		L8	—	3	64K	—	128
0x0003_0000		L9	—		64K	—	128
0x0004_0000	Mid-address space (Flash A)	M0	—	4	128K	—	128
0x0006_0000		M1	—		128K	—	128
0x0008_0000	Low-address space (Flash B)	—	L0	5	—	256K	128
0x000C_0000	Mid-address space (Flash B)	—	M0		—	256K	128

Table 24-1. Memory Map

Offset from FLASH_BASE (0x0000_0000)	Use	Flash A Block	Flash B Block	Partition	Flash A Block Size	Flash B Block Size	Data Width
0x0010_0000	High-address space[1]	H0	H0	6	256K	256K	256
0x0018_0000		H1	H1		256K	256K	256
0x0020_0000		H2	H2	7	256K	256K	256
0x0028_0000		H3	H3		256K	256K	256
0x0030_0000		H4	H4	8	256K	256K	256
0x0038_0000		H5	H5		256K	256K	256
0x0040_0000		H6	H6	9	256K	256K	256
0x0048_0000		H7	H7		256K	256K	256
0x0050_0000		H8	H8	10	256K	256K	256
0x0058_0000		H9	H9		256K	256K	256
0x00EF_C000	Shadow Block (Flash B space) (see Table 24-2)	—	S0	All[2]	—	16K	128
0x00FF_C000	Shadow Block (Flash A space) (see Table 24-2)	S0	—		16K	—	128
0x0100_C000	Reserved						

[1] See Figure 24-1 to see how Flash A and Flash B, together, make up the high address space.

[2] For read while write operations, the shadow row behaves as if it is in all partitions.

Table 24-2 shows the shadow block space. The 16K region of the Shadow Block for Flash B space mirrors from 0x00E0_0000 to 0x00EF_FFFF and Shadow A mirrors every 16K from 0x00F0_0000 through 0x00FF_FFFF. Mirrored operation is not guaranteed

Flash_A and Flash_B arrays have separate configuration and control registers for programming and erase operations. Flash bus configuration registers are common to both arrays.

Table 24-2. Shadow Block Memory Map

Offset from FLASH_BASE (0x0000_0000)	Shadow Block	Use
0x00EF_C000	Flash B	General use
0x00EF_DDE8		FLASH_B_LMLR reset configuration
0x00EF_DDF0		FLASH_B_HLR reset configuration
0x00EF_DDE8		FLASH_B_SLMLR reset configuration

Table 24-2. Shadow Block Memory Map (continued)

Offset from FLASH_BASE (0x0000_0000)	Shadow Block	Use
0x00FF_C000–0x00FF_FDD7	Flash A	General use
0x00FF_FDD8		Serial passcode (0xFEED_FACE_CAFE_BEEF)
0x00FF_FDE0		Censorship control word (0x55AA_55AA)
0x00FF_FDE4		General use
0x00FF_FDE8		FLASH_A_LMLR reset configuration
0x00FF_FDEC		General use
0x00FF_FDF0		FLASH_A_HLR reset configuration
0x00FF_FDF4		General use
0x00FF_FDF8		FLASH_A_SLMLR reset configuration
0x00FF_FE00		FLASH_BIUCR2 reset configuration
0x00FF_FE04		General use
0x00FF_FE08		FLASH_BIUCR3 reset configuration
0x00FF_FFFF–0xFFFF_FFFF		General use

Table 24-3. Flash Configuration Register Memory Map

Offset from FLASH_REGS_ BASE (0xC3F8_8000)	Register	Bits	Access	Reset Value[1]	Section/Page
0x0000	FLASH_A_MCR—Module configuration register	32	R/W	0x0000_0400	24.2.2.1/24-10
0x0004	FLASH_A_LMLR—Low-/Mid-address space block locking register	32	R/W	—	24.2.2.2/24-14
0x0008	FLASH_A_HLR—High-address space block locking register	32	R/W	—	24.2.2.3/24-16
0x000C	FLASH_A_SLMLR—Secondary low-/mid-address space block locking register	32	R/W	—	24.2.2.4/24-18
0x0010	FLASH_A_LMSR—Low-/mid-address space block select register	32	R/W	0x0000_0000	24.2.2.5/24-19
0x0014	FLASH_A_HSR—High-address space block select register	32	R/W	0x0000_0000	24.2.2.6/24-20
0x0018	FLASH_A_AR—Address register	32	R/W	0x0000_0000	24.2.2.7/24-20
0x001C	FLASH_BIUCR—Flash Bus Interface configuration register	32	R/W	0x0000_FF00	24.2.2.8/24-22
0x0020	FLASH_BIUAPR—Flash Bus Interface access protection register	32	R/W	0xFFFF_FFFF	24.2.2.9/24-25
0x0024	FLASH_BIUCR2—Flash Bus Interface configuration register 2	32	R/W	—	24.2.2.10/24-26

Table 24-3. Flash Configuration Register Memory Map

Offset from FLASH_REGS_ BASE (0xC3F8_8000)	Register	Bits	Access	Reset Value[1]	Section/Page
0x0028	FLASH_BIUCR3—Flash Bus Interface configuration register 3	32	R/W	—	
0x002C – 0x0038	Reserved				
0x003C	FLASH_A_UT0—UTest 0 Register	32	R/W	0x0000_0001	24.2.2.12/24-29
0x0040	FLASH_A_UT1—UTest 1 Register	32	R/W	0x0000_0001	24.2.2.13/24-31
0x0044	FLASH_A_UT2—UTest 2 Register	32	R/W	0x0000_0001	24.2.2.14/24-31
0x0048 – 0x3FFF	Reserved				
0x4000	FLASH_B_MCR—Module configuration register	32	R/W	0x0000_0400	24.2.2.1/24-10
0x4004	FLASH_B_LMLR—Low-/Mid-address space block locking register	32	R/W	—	24.2.2.2/24-14
0x4008	FLASH_B_HLR—High-address space block locking register	32	R/W	—	24.2.2.3/24-16
0x400C	FLASH_B_SLMLR—Secondary low-/mid-address space block locking register	32	R/W	—	24.2.2.4/24-18
0x4010	FLASH_B_LMSR—Low-/mid-address space block select register	32	R/W	0x0000_0000	24.2.2.5/24-19
0x4014	FLASH_B_HSR—High-address space block select register	32	R/W	0x0000_0000	24.2.2.6/24-20
0x4018	FLASH_B_AR—Address register	32	R/W	0x0000_0000	24.2.2.7/24-20
0x401c – 0x4038	Reserved				
0x403C	FLASH_B_UT0—UTest 0 Register	32	R/W	0x0000_0001	24.2.2.12/24-29
0x4040	FLASH_B_UT1—UTest 1 Register	32	R/W	0x0000_0001	24.2.2.13/24-31
0x4044	FLASH_B_UT2—UTest 2 Register	32	R/W	0x0000_0001	24.2.2.14/24-31
0x4048 – 0x7FFF	Reserved				

[1] A value of "—" means that the register is loaded from the shadow block described in Figure 24-2.

24.2.2 Register Descriptions

This section lists the flash memory registers in address order and describes the registers and their bit fields.

NOTE

Each of the two flash arrays (Flash_A and Flash_B) have separate configuration registers and control registers for programming and erase operations. In the following register descriptions, the letter "x" represents "A" for Flash_A registers, and "B" for Flash_B registers. Registers without the "x" designator are common to both arrays and are used to control the flash bus interface.

24.2.2.1 Module Configuration Register (FLASH_*x*_MCR)

The FLASH_x_MCR register is defined Figure 24-4 and Table 24-4.

Offset 0x0000 / 0x4000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	SIZE			0	LAS			0	0	0	MAS
W																
Reset	0	0	0	0	0	1	1	1	0	—[1]	—[1]	—[1]	0	0	0	—[1]

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EER	RWE	SBC	0	PEAS	DONE	PEG	0	0	0	0	PGM	PSUS	ERS	ESUS	EHV
W	**w1c**	**w1c**	**w1c**													
Reset	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

[1] See register description below.

Figure 24-4. Module Configuration Register (FLASH_x_MCR)

Table 24-4. FLASH_x_MCR Field Descriptions

Field	Description
0–4	Reserved
5–7 SIZE	Array Space Size. The value of the SIZE field is dependent upon the size of the flash array. SIZE is read only. 111 3.0 MB. (256 KB of LAS, 256 KB of MAS, and 2.5MB of HAS) (Both Flash_A and Flash_B) All others are Reserved.
8	Reserved
9–11 LAS[2:0]	Low Address Space. The value of the LAS field corresponds to the configuration of the Low Address Space. LAS is read only. 000 One 256 KB Blocks. (Flash_B) 100 Eight 16 KB, two 64 KB Blocks. (Flash_A) All others are Reserved.
12–14	Reserved
15 MAS	Mid Address Space. The value of the MAS field corresponds to the configuration of the Mid Address Space. MAS is read only. 0 Two 128 KB Blocks. (Flash_A) 1 One 256 KB Blocks. (Flash_B)
16 EER	ECC Event Error. EER provides information on previous reads. If a double bit detection occurred, the EER bit is set to a 1. This bit must then be cleared, or a reset must occur before this bit returns to a 0 state. This bit may not be set by the user. In the event of a single bit detection and correction, this bit is not be set. If EER is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of EER) are correct. Since this bit is an error flag, it must be cleared to a 0 by writing a 1 to the register location. A write of 0 has no effect. 0 Reads are occurring normally. 1 An ECC Error occurred during a previous read.

Table 24-4. FLASH_x_MCR Field Descriptions (continued)

Field	Description
17 RWE	Read While Write Event Error. RWE provides information on previous RWW reads. If a Read While Write error occurs, this bit is set to 1. This bit must then be cleared, or a reset must occur before this bit returns to a 0 state. This bit may not be written to a 1 by the user. If RWE is not set, or remains 0, this indicates that all previous RWW reads (from the last reset, or clearing of RWE) are correct. Since this bit is an error flag, it must be cleared to a 0 by writing a 1 to the register location. A write of 0 has no effect. 0 Reads are occurring normally. 1 A Read While Write Error occurred during a previous read.
18 SBC	Single Bit Correction. SBC provides information on previous reads provided the FLASH_x_UT0[SPCE] is set. If a single bit correction occurred, the SBC bit is set to a 1. This bit must then be cleared, or a reset must occur before this bit returns to a 0 state. If SBC is not set, or remains 0, this indicates that all previous reads (from the last reset, or clearing of SBC) did not require a correction. Since this bit is an error flag, it must be cleared to a 0 by writing a 1 to the register location. A write of 0 has no effect. 0 Reads are occurring without corrections. 1 A Single Bit Correction occurred during a previous read.
19	Reserved
20 PEAS	Program/Erase Access Space. PEAS is used to indicate which space is valid for program and erase operations, either main array space or shadow space. PEAS = 0 indicates that the main address space is active for all FC program and erase operations. PEAS = 1 indicates the shadow address space is active for program/erase. The value in PEAS is captured and held when the shadow block is enabled with the first interlock write done for program or erase operations. The value of PEAS is retained between sampling events (i.e. subsequent first interlock writes). The value in PEAS may be changed during erase-suspended program, and reverts back to its' original state once the erase-suspended program is completed. PEAS is read only. 0 Shadow address space is disabled for program/erase and main address space enabled. 1 Shadow address space is enabled for program/erase and main address space disabled.
21 DONE	State Machine Status. DONE indicates if the flash module is performing a high voltage operation. DONE is set to a 1 on termination of the flash module reset. DONE is read only. DONE is cleared within Tdone of a 0 to 1 transition of EHV which initiates a high voltage operation. DONE is cleared within Tres of resuming a suspended operation. DONE is set to a 1 at the end of program and erase high voltage sequences. DONE is set to a 1 within Tdones of a 1 to 0 transition of EHV which aborts a high voltage operation. 0 Flash is executing a high voltage operation. 1 Flash is not executing a high voltage operation.
22 PEG	Program/Erase Good. The PEG bit indicates the completion status of the last flash program or erase sequence for which high voltage operations were initiated. The value of PEG is updated automatically during the program and erase high voltage operations. Aborting a program/erase high voltage operation causes PEG to be cleared, indicating the sequence failed. PEG is set to a 1 when the module is reset. PEG is read only. The value of PEG is valid only when PGM = 1 and/or ERS = 1 and after DONE transitions from 0 to 1 due to an abort or the completion of a program/erase operation. PEG is valid until PGM/ERS makes a 1 to 0 transition or EHV makes a 0 to 1 transition. The value in PEG is not valid after a 0 to 1 transition of DONE caused by PSUS or ESUS being set to logic 1. If PGM and ERS are both 1 when DONE makes a qualifying 0 to 1 transition the value of PEG indicates the completion status of the PGM sequence. This happens in an erase-suspended program operation. 0 Program or erase operation failed. 1 Program or erase operation successful. **Note:** If program or erases are attempted on blocks that are locked, the response from flash is PEG = 1, indicating that the operation was successful, and the contents of the block are properly protected from the program or erase operation.

Table 24-4. FLASH_x_MCR Field Descriptions (continued)

Field	Description
23–26	Reserved
27 PGM	Program. PGM is used to set up flash for a program operation. A 0 to 1 transition of PGM initiates a program sequence. A 1 to 0 transition of PGM ends the program sequence. PGM can be set only under one of the following conditions: • User mode read (ERS is low and UTE is low). • Erase suspend (ERS and ESUS are 1) with EHV low. PGM can be cleared by the user only when PSUS and EHV are low and DONE is high. PGM is cleared on reset. 0 Flash is not executing a program sequence. 1 Flash is executing a program sequence. **Note:** In an erase-suspended program, programming Flash locations in blocks which were being operated on in the erase may corrupt FC data. This should be avoided due to reliability implications.
28 PSUS	Program Suspend. PSUS is used to indicate the flash module is in program suspend or in the process of entering a suspend state. The module is in program suspend when PSUS = 1 and DONE = 1. PSUS can be set high only when PGM and EHV are high. A 0 to 1 transition of PSUS starts the sequence which sets DONE and places the flash module in program suspend. The module enters suspend within Tpsus of this transition. PSUS can be cleared only when DONE and EHV are high. A 1 to 0 transition of PSUS with EHV = 1 starts the sequence which clears DONE and returns the flash module to program. The module cannot exit program suspend and clear DONE while EHV is low. PSUS is cleared on reset. 0 Program sequence is not suspended. 1 Program sequence is suspended.
29 ERS	Erase. ERS is used to set up flash for an erase operation. A 0 to 1 transition of ERS initiates an erase sequence. A 1 to 0 transition of ERS ends the erase sequence. ERS can only be set only in user mode read (PGM is low and UTE is low). ERS can be cleared by the user only when ESUS and EHV are low and DONE is high. ERS is cleared on reset. 0 Flash is not executing an erase sequence. 1 Flash is executing an erase sequence.

Table 24-4. FLASH_x_MCR Field Descriptions (continued)

Field	Description
30 ESUS	Erase Suspend. ESUS is used to indicate that the flash module is in erase suspend or in the process of entering a suspend state. The module is in erase suspend when ESUS = 1 and DONE = 1. ESUS can be set high only when ERS and EHV are high and PGM is low. A 0 to 1 transition of ESUS starts the sequence which sets DONE and places the flash in erase suspend. The flash module enters suspend within Tesus of this transition. ESUS can be cleared only when DONE and EHV are high and PGM is low. A 1 to 0 transition of ESUS with EHV = 1 starts the sequence which clears DONE and returns the module to erase. The flash module cannot exit erase suspend and clear DONE while EHV is low. ESUS is cleared on reset. 0 Erase sequence is not suspended. 1 Erase sequence is suspended.
31 EHV	Enable High Voltage. The EHV bit enables the flash module for a high voltage program/erase operation. EHV is cleared on reset. EHV must be set after an interlock write to start a program/erase sequence. EHV may be set, initiating a program/erase, after an interlock under one of the following conditions: • Erase (ERS = 1, ESUS = 0). • Program (ERS = 0, ESUS = 0, PGM = 1, PSUS = 0). • Erase-suspended program (ERS = 1, ESUS = 1, PGM = 1, PSUS = 0). If a program operation is to be initiated while an erase is suspended the user must clear EHV while in erase suspend before setting PGM. In normal operation, a 1 to 0 transition of EHV with DONE high, PSUS and ESUS low terminates the current program/erase high voltage operation. When an operation is aborted, there is a 1 to 0 transition of EHV with DONE low and the suspend bit for the current program/erase sequence low. An abort causes the value of PEG to be cleared, indicating a failed program/erase; address locations being operated on by the aborted operation contain indeterminate data after an abort. A suspended operation cannot be aborted. EHV may be written during suspend. EHV must be high for the flash module to exit suspend. EHV may not be written after a suspend bit is set high and before DONE transitions high. EHV may not be set low after the current suspend bit is set low and before DONE transitions low. 0 Flash is not enabled to perform a high voltage operation. 1 Flash is enabled to perform a high voltage operation. **Note:** Aborting a high voltage operation leaves FC addresses in an indeterminate data state. This may be recovered by executing an erase on the affected blocks.

24.2.2.1.1 FLASH_x_MCR Simultaneous Bit Write Priority

A number of FLASH_x_MCR bits are protected against write when another bit, or set of bits, is in a specific state. These write locks are covered on a bit by bit basis in the preceding section. The write locks detailed in the previous section do not consider the effects of trying to write two or more bits simultaneously. The effects of writing bits simultaneously which put the module in an illegal state are detailed here.

The flash module does not allow the user to write bits simultaneously which put the device into an illegal state. This is implemented through a priority mechanism among the bits. The bit changing priorities are detailed in Table 24-5.

Table 24-5. FLASH_x_MCR Bit Set/Clear Priority Levels

Priority Level	FLASH_x_MCR Bit(s)
1	ERS

Table 24-5. FLASH_x_MCR Bit Set/Clear Priority Levels

Priority Level	FLASH_x_MCR Bit(s)
2	PGM
3	EHV
4	ESUS, PSUS

If the user attempts to write two or more FLASH_x_MCR bits simultaneously then only the bit with the lowest priority level is written. Setting two bits with the same priority level is prevented by existing write locks or do not put the flash in an illegal state.

For example, setting ERS and PGM simultaneously results in only ERS being set. Attempting to clear EHV while setting PSUS results in EHV being cleared, while PSUS is unaffected.

24.2.2.2 Low/Mid Address Space Block Locking Register (FLASH_x_LMLR)

The Low/Mid Address Block Locking Register (FLASH_x_LMLR) provides a means to protect blocks from being modified. These bits, along with bits in the Secondary LLOCK (FLASH_x_SLMLR), determine if the block is locked from program or erase. An “OR” of FLASH_x_LMLR and FLASH_x_SLMLR determine the final lock status.

NOTE

A reset value of 1* in Figure 24-5 indicates that the reset value of these registers is determined by Flash values in the shadow block. An erased shadow block causes the reset value to be 1.

The following field and bit descriptions fully define the FLASH_x_LMLR register (Figure 24-5).

Offset 0x0004 / 0x4004

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	LME	0	0	0	0	0	0	0	0	0	0	SLOCK	0	0	MLOCK		0	0	0	0	0	0	LLOCK									
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	1*	0	0	1*	1*	0	0	0	0	0	0	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*

Figure 24-5. FLASH_x_LMLR Register

FLASH_x_LMLR register functions, as shown in Table 24-6.

Table 24-6. FLASH_x_LMLR Field Descriptions

Field	Description
0 LME	Low/Mid Address Lock Enable. This bit is used to enable the Lock registers (SLOCK, MLOCK and LLOCK) to be set or cleared by register writes. This bit is a status bit only, and may not be written or cleared, and the reset value is 0. The method to set this bit is to write a password, and if the password matches, the LME bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For LME, the password 0xA1A1_1111 must be written to the FLASH_x_LMLR register. 0 Low/Mid Address Locks are disabled, and can not be modified. 1 Low/Mid Address Locks are enabled to be written.
1–10	Reserved
11 SLOCK	Shadow Lock. This bit is used to lock the shadow block from programs and erases. A value of 1 in the SLOCK register signifies that the shadow block is locked for program and erase. A value of 0 in the SLOCK register signifies that the shadow block is available to receive program and erase pulses. The SLOCK register is not writable once an interlock write is completed until FLASH_x_MCR[DONE] is set at the completion of the requested operation. Likewise, SLOCK register is not writable if a high voltage operation is suspended. SLOCK is also not writeable during UTest operations, when AIE is high. Upon reset, information from the shadow block is loaded into the SLOCK register. The SLOCK bit may be written as a register. Reset causes the bits to go back to their shadow block value. The default value of the SLOCK bits (assuming erased shadow location) is locked. SLOCK is not writable unless LME is high.
12–13	Reserved
14–15 MLOCK[1:0]	Mid Address Space Block Lock. A value of 1 in a bit of the lock register signifies that the corresponding block is locked for program and erase. A value of 0 in the lock register signifies that the corresponding block is available to receive program and erase pulses. The block numbering for Mid Address Space is given in the table below, and is different for Flash_A and Flash_B. <table><tr><th>MLOCK Bit</th><th>Flash_A Block</th><th>Flash_B Block</th></tr><tr><td>0</td><td>M0</td><td>M0</td></tr><tr><td>1</td><td>M1</td><td>none</td></tr></table> The lock register is not writable once an interlock write is completed until FLASH_x_MCR[DONE] is set at the completion of the requested operation. Likewise, the lock register is not writable if a high voltage operation is suspended. MLOCK is also not writeable during UTest operations, when AIE is high. Upon reset, information from the shadow block is loaded into the block registers. The LOCK bits may be written as a register. Reset causes the bits to go back to their shadow block value. The default value of the LOCK bits (assuming erased shadow location) is locked. In the event that blocks are not present (due to configuration or total memory size), the LOCK bits default to be locked, and are not writable. The reset value is always 1 (independent of the shadow block), and register writes have no effect. MLOCK is not writable unless LME is high.

Table 24-6. FLASH_x_LMLR Field Descriptions (continued)

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>16–21</td><td>Reserved</td></tr>
<tr><td>22–31
LLOCK</td><td>Low Address Space Block Lock. A value of 1 in a bit of the lock register signifies that the corresponding block is locked for program and erase. A value of 0 in the lock register signifies that the corresponding block is available to receive program and erase pulses. The block numbering for Low Address Space is given in the table below, and is different for Flash_A and Flash_B.
<table>
<tr><th>LLOCK Bit</th><th>Flash_A Block</th><th>Flash_B Block</th></tr>
<tr><td>0</td><td>L0</td><td>L0</td></tr>
<tr><td>1</td><td>L1</td><td>none</td></tr>
<tr><td>2</td><td>L2</td><td>none</td></tr>
<tr><td>3</td><td>L3</td><td>none</td></tr>
<tr><td>4</td><td>L4</td><td>none</td></tr>
<tr><td>5</td><td>L5</td><td>none</td></tr>
<tr><td>6</td><td>L6</td><td>none</td></tr>
<tr><td>7</td><td>L7</td><td>none</td></tr>
<tr><td>8</td><td>L8</td><td>none</td></tr>
<tr><td>9</td><td>L9</td><td>none</td></tr>
</table>
For more details on LLOCK, please see MLOCK bit description.
LLOCK is not writable unless LME is high.</td></tr>
</table>

24.2.2.3 High Address Space Block Locking Register (FLASH_x_HLR)

The High Address Space Block Locking Register (FLASH_x_HLR) provides a means to protect blocks from being modified.

NOTE

> A reset value of 1* in Figure 24-6 indicates that the reset value of these registers is determined by Flash values in the shadow block. An erased shadow block causes the reset value to be 1.

The following field and bit descriptions fully define the FLASH_x_HLR register (Figure 24-6).

Offset 0x0008 / 0x4008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	HBE	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	HLOCK									
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*

Figure 24-6. FLASH_x_HLR Register

FLASH_x_HLR register functions, as shown in Table 24-7.

Table 24-7. FLASH_x_HLR Field Descriptions

Field	Description
0 HBE	High Address Lock Enable This bit is used to enable the Lock registers (HLOCK) to be set or cleared by register writes. This bit is a status bit only, and may not be written or cleared, and the reset value is 0. The method to set this bit is to provide a password, and if the password matches, the HBE bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For HBE, the password 0xB2B2_2222 must be written to the FLASH_x_HLR register. 0 High Address Locks are disabled, and can not be modified. 1 High Address Locks are enabled to be written.
1–21	Reserved
22–31 HLOCK	High Address Space Block Lock. HLOCK has the same characteristics as LLOCK. Please see this description for more information. The block numbering for High Address Space is given in the table below, and is the same for Flash_A and Flash_B. HLOCK is not writable unless HBE is high.

HLOCK Bit	Flash_A Block	Flash_B Block
0	H0	H0
1	H1	H1
2	H2	H2
3	H3	H3
4	H4	H4
5	H5	H5
6	H6	H6
7	H7	H7
8	H8	H8
9	H9	H9

24.2.2.4 Secondary Low/Mid Address Space Block Locking Register (FLASH_x_SLMLR)

The Secondary Low/Mid Address Block Locking Register (FLASH_x_SLMLR) provides an alternative means to protect blocks from being modified. This has the effect of creating a “tiered” locking scheme to enable different flash users to provide different default locking on blocks. These bits, along with bits in the LLOCK (FLASH_x_LMLR), determine if the block is locked from program or erase. An “OR” of FLASH_x_LMLR and FLASH_x_SLMLR determine the final lock status.

NOTE

A reset value of 1* in Figure 24-7 indicates that the reset value of these registers is determined by Flash values in the shadow block. An erased shadow block causes the reset value to be 1.

The following field and bit descriptions fully define the FLASH_x_SLMLR register (Figure 24-7).

Offset 0x000C / 0x400C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SLE	0	0	0	0	0	0	0	0	0	0	SSLOCK	0	0	SMLOCK		0	0	0	0	0	0	SLLOCK									
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	1*	0	0	1*	1*	0	0	0	0	0	0	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*

Figure 24-7. FLASH_x_SLMLR Register

NOTE

Writing the password 0xC3C3_3333 to this register does not modify the SSLOCK, SMLOCK, or SLLOCK fields. Only the SLE bit changes.

FLASH_x_SLMLR register functions, as shown in Table 24-8.

Table 24-8. FLASH_x_SLMLR Field Descriptions

Field	Description
0 SLE	Secondary Low/Mid Address Lock Enable. This bit is used to enable the Lock registers (SSLOCK, SMLOCK, and SLLOCK) to be set or cleared by register writes. This bit is a status bit only, and may not be written or cleared, and the reset value is 0. The method to set this bit is to provide a password, and if the password matches, the SLE bit is set to reflect the status of enabled, and is enabled until a reset operation occurs. For SLE, the password 0xC3C3_3333 must be written to the FLASH_x_SLMLR register 0 Secondary Low/Mid Address Locks are disabled, and can not be modified. 1 Secondary Low/Mid Address Locks are enabled to be written.
1–10	Reserved
11 SSLOCK	Secondary Shadow Lock. This bit is an alternative method that may be used to lock the shadow block from programs and erases. SSLOCK has the same description as SLOCK in the FLASH_x_LMLR register (see Figure 24-5). SSLOCK is not writable unless SLE is high.
12–13	Reserved

Table 24-8. FLASH_x_SLMLR Field Descriptions (continued)

Field	Description
14–15 SMLOCK	Secondary Mid Address Block Lock. This bit is an alternative method that may be used to lock the Mid Address Space blocks from programs and erases. SMLOCK has the same description as MLOCK in the FLASH_x_LMLR register. SMLOCK is not writable unless SLE is high.
16–21	Reserved
22–31 SLLOCK	Secondary Low Address Block Lock. This bit is an alternative method that may be used to lock the Low Address Space blocks from programs and erases. SLLOCK has the same description as LLOCK in the FLASH_x_LMLR register. SLLOCK is not writable unless SLE is high.

24.2.2.5 Low/Mid Address Space Block Select Register (FLASH_x_LMSR)

The Low/Mid Address Space Block Select Register (FLASH_x_LMSR) provides a means to select blocks to be operated on during erase.

The following field and bit descriptions fully define the FLASH_x_LMSR register (Figure 24-8).

Offset 0x0010 / 0x4010

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	MSEL		0	0	0	0	0	0	LSEL									
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-8. FLASH_x_LMSR Register

FLASH_x_LMSR register functions, as shown in Table 24-9.

Table 24-9. FLASH_x_LMSR Field Descriptions

Field	Description
0–13	Reserved
14–15 MSEL	Mid Address Space Block Select. A value of 1 in the select register signifies that the block is selected for erase. A value of 0 in the select register signifies that the block is not selected. The reset value for the select registers is 0, or un-selected. The blocks must be selected (or un-selected) before doing an erase interlock write as part of the erase sequence. The select register is not writable once an interlock write is completed until FLASH_x_MCR[DONE] is set at the completion of the requested operation, or if a high voltage operation is suspended. MSEL is also not writeable during UTest operations, when AIE is high. In the event that blocks are not present (due to configuration or total memory size), the corresponding select bits default to un-selected, and are not writable. The reset value is always 0, and register writes have no effect. MSEL bits are mapped to block numbers in the same way as described for the MLOCK bits of the FLASH_x_LMLR register.

Table 24-9. FLASH_x_LMSR Field Descriptions (continued)

Field	Description
16–21	Reserved
22–31 LSEL[9:0]	Low Address Space Block Select. A value of 1 in the select register signifies that the block is selected for erase. A value of 0 in the select register signifies that the block is not selected. The reset value for the select registers is 0, or un-selected. The blocks must be selected (or un-selected) before doing an erase interlock write as part of the erase sequence. The select register is not writable once an interlock write is completed until FLASH_x_MCR[DONE] is set at the completion of the requested operation, or if a high voltage operation is suspended. LSEL is also not writeable during UTest operations, when AIE is high. In the event that blocks are not present (due to configuration or total memory size), the corresponding select bits default to un-selected, and are not writable. The reset value is always 0, and register writes have no effect. LSEL bits are mapped to block numbers in the same way as described for the LLOCK bits of the FLASH_x_LMLR register.

24.2.2.6 High Address Space Block Select Register (FLASH_x_HSR)

The High Address Space Block Select Register (FLASH_x_HSR) provides a means to select blocks to be operated on during erase.

The following field and bit descriptions fully define the FLASH_x_HSR register (Figure 24-9).

Offset 0x0014 / 0x4014

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	HSEL									
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-9. FLASH_x_HSR Register

FLASH_x_HSR register functions, as shown in Table 24-10.

Table 24-10. FLASH_x_HSR Field Descriptions

Field	Description
0–21	Reserved
22–31 HSEL[5:0]	High Address Space Block Select. High Address Block Select has the same characteristics as LSEL.

24.2.2.7 Address Register (FLASH_x_AR)

The Address register (FLASH_x_AR) provides the first failing address in the event module failures (ECC or PGM/Erase state machine). This address is the internal array address. To convert to a logical system address, a formula must be applied:

Array	Address Range	Operation
A	0x00_0000 - 0x07_FFFF	No change
A	0x08_0000 - 0x1F_FFFF	Logical address[17:3] = Addr[16:4],0,ADDR[3]}
B	0x00_0000 - 0x07_FFFF	Add 0x08_0000
B	0x08_0000 - 0x1F_FFFF	Logical address[17:3] = Addr[16:4],1,ADDR[3]}

The following field and bit descriptions fully define the FLASH_x_AR (Figure 24-10).

Offset 0x0018 / 0x4018 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SAD	0	0	0	0	0	0	0	0	0	0	0	0	0	ADDR															0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-10. FLASH_x_AR Register

FLASH_x_AR functions, as shown in Table 24-11.

Table 24-11. FLASH_x_AR Field Descriptions

Field	Description
0 SAD	Shadow Address. The SAD bit qualifies the address captured during an ECC Event Error, Single Bit Correction, or State Machine operation. The SAD register is not writable. 0 Address Captured is from Main Array Space. 1 Address Captured is from Shadow Array Space.
1–13	Reserved
14–28 ADDR[17:3]	Address. The FLASH_x_AR provides the first failing address in the event of ECC event error (FLASH_x_MCR[EER] set), single bit correction (FLASH_x_MCR[SBC] set), as well as providing the address of a failure that may have occurred in a state machine operation (FLASH_x_MCR[PEG] cleared). ECC event errors take priority over single bit corrections, which take priority over state machine errors. This is especially valuable in the event of a RWW operation, where the read senses an ECC error or single bit correction, and the state machine fails simultaneously. This address is always a Double Word address that selects 64 bits. The FLASH_x_AR is writable, and can be used in the UTEST ECC Logic Check. If the ECC logic check is enabled (FLASH_x_UT0[EIE] = 1) then the FLASH_x_AR will not update for ECC event error, single bit correction or state machine errors. If FLASH_x_MCR[EER] or FLASH_x_MCR[SBC] are set, the FLASH_x_AR is locked from writing. FLASH_x_MCR[PEG] does not affect the writability of the FLASH_x_AR.
29–31	Reserved

24.2.2.8 Flash Bus Interface Configuration Register (FLASH_BIUCR)

FLASH_BIUCR is used to configure the prefetch controls for bus master accesses via crossbar slave port 0, plus arbitration and bus timing controls for bus master access via both slave port 0 and slave port 3. This register must not be written while executing from flash.

Offset: 0x001C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	M8PFE	0	0	0	0	0	0	0	M0PFE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	APC			WWSC		RWSC			0	DPFEN	ARB	IFPFEN	PRI	PFLIM		BFEN
W																
Reset	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

Figure 24-11. Flash Bus Interface Configuration Register (FLASH_BIUCR)

Table 24-12. FLASH_BIUCR Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>0–15
M<i>n</i>PFE</td><td>Master n prefetch enable, where n represents the master ID number in the table below. These bits are used to control whether prefetching may be triggered based on the master ID of a requesting master. These bits are cleared by hardware reset.
0 No prefetching may be triggered by this master
1 Prefetching may be triggered by this master
Note: These bits refer to the master ID, not the master port number, as shown in the following:
<table>
<tr><th>Master ID</th><th>Module</th></tr>
<tr><td>0</td><td>Z7 Core</td></tr>
<tr><td>1</td><td>– reserved –</td></tr>
<tr><td>2</td><td>– reserved –</td></tr>
<tr><td>3</td><td>– reserved –</td></tr>
<tr><td>4</td><td>eDMA_A</td></tr>
<tr><td>5</td><td>eDMA_B</td></tr>
<tr><td>6</td><td>FlexRay</td></tr>
</table>
</td></tr>
</table>

Table 24-12. FLASH_BIUCR Field Descriptions

16–18 APC	Address Pipelining Control. This field is used to control the number of cycles between pipelined access requests. It must be set to a value corresponding to the operating frequency of the flash.Higher operating frequencies require non-zero settings for this field for proper flash operation. This field is set to 0b111 by hardware reset. 000 Accesses may be pipelined back-to-back 001 Access requests require one additional hold cycle 010 Access requests require two additional hold cycles ... 110 Access requests require six additional hold cycles 111 No address pipelining **Note:** The settings for APC and RWSC must be the same. **Note:** Valid settings are specified in product Data Sheet.
19–20 WWSC	Write Wait State Control - This field is used to control the number of wait-states to be added to the best-case flash array access time for writes. The best-case flash array access time for writes is two cycles. This field must be set to a value corresponding to the operating frequency of the flash. Higher operating frequencies require non-zero settings for this field for proper flash operation. This field is set to 0b11 by hardware reset. 00 No additional wait-states are added 01 One additional wait-state is added 10 Two additional wait-states are added 11 Three additional wait-states are added
21–23 RWSC	Read Wait State Control - This field is used to control the number of wait-states to be added to the best-case flash array access time for reads. The best-case flash array access time for reads is one cycle. This field must be set to a value corresponding to the operating frequency of the flash and the actual read access time of the flash. Higher operating frequencies require non-zero settings for this field for proper flash operation. This field is set to 0b111 by hardware reset. 000 No additional wait-states are added 001 One additional wait-state is added ... 111 Seven additional wait-states are added **Note:** The settings for APC and RWSC must be the same. **Note:** Valid settings are specified in the product Data Sheet.
24	Reserved
25 DPFEN	Data Prefetch Enable - This field enables or disables prefetching initiated by a data read access. This field is cleared by hardware reset. 0 No prefetching is triggered by a data read access 1 Prefetching may be triggered by any data read access
26 ARB	Arbitration Mode - This field controls which arbitration mode is used. In both fixed priority or round-robin modes, write requests are prioritized higher than read requests, and read requests are prioritized higher than speculative prefetch requests whenever both ports issue concurrent requests. This bit is cleared by hardware reset. 0 Fixed priority arbitration is used. The port specified in PRI has highest fixed priority. 1 Round-robin arbitration is used.
27 IPFEN	Instruction Prefetch Enable - This bit enables or disables prefetching initiated by an instruction read access. This field is cleared by hardware reset. 0 No prefetching is triggered by an instruction read access 1 Prefetching may be triggered by any instruction read access

Table 24-12. FLASH_BIUCR Field Descriptions

28 PRI	Highest Priority Port - Defines which port gains access first when both issue concurrent requests. Effective only in fixed priority arbitration mode. 0 Port 0 gains access first 1 Port 1 gains access first
29–30 PFLIM	Flash Prefetch Limit - This field controls the prefetch algorithm used by the flash prefetch controller. This field defines a limit on the maximum number of sequential prefetches which will be attempted between buffer misses. In all situations when enabled, only a single prefetch is initiated on each buffer miss or hit. This field is cleared by hardware reset. 00 No prefetching or buffering is performed. 01 The referenced line is prefetched on a buffer miss, i.e., prefetch on miss. 1x The referenced line is prefetched on a buffer miss, or the next sequential line is prefetched on a buffer hit (if not already present), i.e., prefetch on miss or hit.
31 BFEN	Flash Line Read Buffers Enable - This bit enables or disables line read buffer hits. It is also used to invalidate the buffers. This bit is cleared by hardware reset. 0 The line read buffers are disabled from satisfying read requests, and all buffer valid bits are cleared. 1 The line read buffers are enabled to satisfy read requests on hits. Buffer valid bits may be set when the buffers are successfully filled.

24.2.2.9 Flash Bus Interface Access Protection Register (FLASH_BIUAPR)

This register is used to control bus master read and write access attributes to the flash array.

Offset: 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	1	1	1	1	1	1	1	1	1	1	1	1	M9AP		M8AP	
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	1	1	M6AP		M5AP		M4AP		1	1	1	1	M1AP		M0AP	
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 24-12. Flash Bus Interface Access Protection Register (FLASH_BIUAPR)

Table 24-13. FLASH_BIUAPR Bit Field Descriptions

Field	Description
0–11 16–17 24–27	Reserved
12–15 18–23 28–31 M*n*AP	Master *n* Access Protection, where *n* represents the master ID number in the table below. These fields are used to control whether read and write accesses to the flash are allowed based on the master ID of a requesting master. 00 No accesses may be performed by this master 01 Only read accesses may be performed by this master 10 Only write accesses may be performed by this master 11 Both read and write accesses may be performed by this master **Note:** These bits refer to the master ID, not the master port number, as shown in the following:

Master ID	Module
0	Core 0
1	Core 1
2	– reserved –
3	– reserved –
4	eDMA_A
5	eDMA_B
6	FlexRay
7	Reserved for EBI test (not for customer use)
8	Core 0 Nexus
9	Core 1 Nexus

24.2.2.10 Flash Bus Interface Configuration Register 2 (FLASH_BIUCR2)

This register configures the line buffers of the flash bus interfaces attached to slave port 0 and slave port 1.

NOTE

> A reset value of 1* in Figure 24-14 indicates that the reset value of these registers is determined by Flash values in the shadow block. An erased shadow block causes the reset value to be 1.

Offset: 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LBCFG_P0		LBCFG_P1		0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*

Figure 24-13. Flash Bus Interface Configuration Register 2 (FLASH_BIUCR2)

Table 24-14. FLASH_BIUCR2 Bit Field Descriptions

Field	Description
0–1 LBCFG_P0	Line Buffer Configuration. This field controls the configuration of all the line buffers in the flash bus interface unit attached to crossbar slave port 0. The buffers can be organized as a “pool” of available resources, or with a fixed partition between instruction and data buffers. In all cases, when a buffer miss occurs, it is allocated to the least-recently-used buffer within the group and the just-fetched entry then marked as most-recently-used. If the flash access is for the next-sequential line, the buffer is not marked as most-recently-used until the given address produces a buffer hit. This field is initialized by hardware reset to the value contained in address0x00FF_FE00 of the shadow block of the flash array. The initial value is given in Table 24-2. This field controls the configuration of both the 4 x 128 and 4 x 256 line buffers. 00 All four buffers are available for any flash access, i.e., there is no partitioning of the buffers based on the access type. 01 Reserved 10 The buffers are partitioned into two groups with buffers 0 and 1 allocated for instruction fetches and buffers 2 and 3 for data accesses. 11 The buffers are partitioned into two groups with buffers 0,1,2 allocated for instruction fetches and buffer 3 for data accesses.

Table 24-14. FLASH_BIUCR2 Bit Field Descriptions

Field	Description
2–3 LBCFG_P1[1:0]	Line Buffer Configuration. This field controls the configuration of all the line buffers in the flash bus interface unit attached to crossbar slave port 3. The buffers can be organized as a “pool” of available resources, or with a fixed partition between instruction and data buffers. In all cases, when a buffer miss occurs, it is allocated to the least-recently-used buffer within the group and the just-fetched entry then marked as most-recently-used. If the flash access is for the next-sequential line, the buffer is not marked as most-recently-used until the given address produces a buffer hit. This field is initialized by hardware reset to the value contained in address 0x00FF_FE00 of the shadow block of the flash array. The initial value is given in Table 24-2. This field controls the configuration of both the 4 x 128 and 4 x 256 line buffers. 00 All four buffers are available for any flash access, i.e., there is no partitioning of the buffers based on the access type. 01 Reserved 10 The buffers are partitioned into two groups with buffers 0 and 1 allocated for instruction fetches and buffers 2 and 3 for data accesses. 11 The buffers are partitioned into two groups with buffers 0,1,2 allocated for instruction fetches and buffer 3 for data accesses.
4–31	Reserved

24.2.2.11 Flash Bus Interface Configuration Register 3 (FLASH_BIUCR3)

NOTE

> A reset value of 1* in Figure 24-14 indicates that the reset value of these registers is determined by Flash values in the shadow block. An erased shadow block causes the reset value to be 1.

FLASH_BIUCR3 is used to configure the prefetch controls for bus master accesses via crossbar slave port 3.

Offset: 0x0028 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	M9PFE	0	0	M6PFE	M5PFE	M4PFE	0	0	M1PFE	0
W																
Reset	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0			0		0			0	DPFEN	0	IFPFEN	0	PFLIM		BFEN
W																
Reset	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*	1*

Figure 24-14. Flash Bus Interface Configuration Register 3 (FLASH_BIUCR3)

Table 24-15. FLASH_BIUCR Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>M<i>n</i>PFE</td><td>Master n prefetch enable, where n represents the master ID number in the table below. These bits are used to control whether prefetching may be triggered based on the master ID of a requesting master. These bits are cleared by hardware reset.
0 No prefetching may be triggered by this master
1 Prefetching may be triggered by this master
Note: These bits refer to the master ID, not the master port number, as shown in the following:
<table>
<tr><th>Master ID</th><th>Module</th></tr>
<tr><td>0</td><td>– reserved –</td></tr>
<tr><td>1</td><td>Core 1</td></tr>
<tr><td>2</td><td>– reserved –</td></tr>
<tr><td>3</td><td>– reserved –</td></tr>
<tr><td>4</td><td>eDMA_A</td></tr>
<tr><td>5</td><td>eDMA_B</td></tr>
<tr><td>6</td><td>FlexRay</td></tr>
<tr><td>7</td><td>Reserved for EBI test
(not for customer use)</td></tr>
<tr><td>8</td><td>– reserved –</td></tr>
<tr><td>9</td><td>Core 1Nexus</td></tr>
</table>
</td></tr>
<tr><td>DPFEN</td><td>Data Prefetch Enable - This field enables or disables prefetching initiated by a data read access. This field is cleared by hardware reset.
0 No prefetching is triggered by a data read access
1 Prefetching may be triggered by any data read access</td></tr>
<tr><td>IPFEN</td><td>Instruction Prefetch Enable - This bit enables or disables prefetching initiated by an instruction read access. This field is cleared by hardware reset.
0 No prefetching is triggered by an instruction read access
1 Prefetching may be triggered by any instruction read access</td></tr>
<tr><td>PFLIM[1:0]</td><td>Flash Prefetch Limit - This field controls the prefetch algorithm used by the flash prefetch controller. This field defines a limit on the maximum number of sequential prefetches which will be attempted between buffer misses. In all situations when enabled, only a single prefetch is initiated on each buffer miss or hit. This field is cleared by hardware reset.
00 No prefetching or buffering is performed.
01 The referenced line is prefetched on a buffer miss, i.e., prefetch on miss.
1x The referenced line is prefetched on a buffer miss, or the next sequential line is prefetched on a buffer hit (if not already present), i.e., prefetch on miss or hit.</td></tr>
<tr><td>BFEN</td><td>Flash Line Read Buffers Enable - This bit enables or disables line read buffer hits. It is also used to invalidate the buffers. This bit is cleared by hardware reset.
0 The line read buffers are disabled from satisfying read requests, and all buffer valid bits are cleared.
1 The line read buffers are enabled to satisfy read requests on hits. Buffer valid bits may be set when the buffers are successfully filled.</td></tr>
</table>

24.2.2.12 User Test Register 0 (FLASH_x_UT0)

The User Test Register 0 (FLASH_x_UT0) provides a means to control UTest. The UTest mode gives the users of the flash module the ability to perform test features on the flash. This register is only writable when the flash is put into UTest mode by writing a passcode.

Offset: 0x003C / 0x403C

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	UTE	SCBE	0	0	0	0	0	0	DSI							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	EA	0	MRE	MRV	EIE	AIS	AIE	AID
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

Figure 24-15. User Test Register 0 (FLASH_x_UT0)

Table 24-16. FLASH_x_UT0 Field Descriptions

Field	Description
0 UTE	UTest Enable. This status bit gives indication when UTest is enabled. All bits in FLASH_x_UT0, FLASH_x_UT1, FLASH_x_UT2 are locked when this bit is 0. This bit is not writeable to a 1, but may be cleared. The reset value is 0. The method to set this bit is to provide a password, and if the password matches, the UTE bit is set to reflect the status of enabled, and is enabled until it is cleared by a register write. The UTE password will only be accepted if FLASH_x_MCR[PGM] = 0 and FLASH_x_MCR [ERS] = 0 (program and erase are not being requested). UTE can only be cleared if FLASH_x_UT0[AID] = 1, FLASH_x_UT0[AIE] and FLASH_x_UT0[EIE] = 0. While clearing UTE, writes to set AIE or set EIE will be ignored. For UTE, the password 0xF9F9_9999 must be written to the FLASH_x_UT0 register.
1 SCBE	Single Bit Correction Enable. SBC enables Single Bit Correction results to be observed in FLASH_x_MCR[SBC]. Also is used as an enable for interrupt signals created by the c90fl module (see c90fl Integration Guide). ECC corrections that occur when SBCE is cleared will not be logged. 0 Single Bit Corrections observation is disabled. 1 Single Bit Correction observation is enabled.
2–7	Reserved
8–15 DSI	Data Syndrome Input. These bits enable checks of ECC logic by allowing check bits to be input into the ECC logic and then read out by doing array reads or array integrity checks. The DSI[7:0] correspond to the 8 ECC check bits on a double word.
16–23	Reserved
24 EA	ECC Algorithm. EA is a status bit that provides information about the ECC algorithm used within the Flash. Either a modified Hamming code is used, or a modified Hsiao code is used. 0 ECC is implemented with a modified Hamming algorithm. 1 ECC is implemented with a modified Hsiao algorithm.
25	Reserved
26 MRE	Margin Read Enable. MRE combined with MRV enables Factory Margin Reads to be done. Margin reads are only active during Array Integrity Checks. Normal user reads are not affected by MRE. MRE is not writable if AID is low. 0 Margin reads are not enabled. 1 Margin reads are enabled during Array Integrity Checks.

Table 24-16. FLASH_x_UT0 Field Descriptions (continued)

Field	Description
27 MRV	Margin Read Value. MRV selects the margin level that is being checked. Margin can be checked to an erased level (MRV = 1) or to a programmed level (MRV = 0). In order for this value to be valid, MRE must also be set. MRV is not writable if AID is low. 0 Zero's margin reads are requested. 1 One's margin reads are requested.
28 EIE	ECC Data Input Enable. EIE enables the input registers (DSI and DAI) to be the source of data for the array. This is useful in the ECC logic check. If this bit is set, data read through a BIU read request will be from the DSI and DAI registers when an address match is achieved to the FLASH_x_AR. EIE is not simultaneously writable to a 1 as UTI is being cleared to a 0. 0 Data read is from the flash array. 1 Data read is from the DSI and DAI registers.
29 AIS	Array Integrity Sequence. AIS determines the address sequence to be used during array integrity checks. The default sequence (AIS = 0) is meant to replicate sequences normal "user" code follows, and thoroughly checks the read propagation paths. This sequence is proprietary. The alternative sequence (AIS = 1) is just logically sequential. It should be noted that the time to run a sequential sequence is significantly shorter than the time to run the proprietary sequence. If MRE is set, AIS has no effect. 0 Array integrity sequence is proprietary sequence. 1 Array integrity sequence is sequential.
30 AIE	Array Integrity Enable. AIE set to one starts the array integrity check done on all selected and unlocked blocks. The address sequence selected is determined by AIS, and the MISR (UM0 through UM4) can be checked after the operation is complete, to determine if a correct signature is obtained. Once an Array Integrity operation is requested (AIE = 1), it may be terminated by clearing AIE if the operation has finished (AID = 1) or aborted by clearing AIE if the operation is ongoing (AID = 0). AIE is not simultaneously writable to a 1 as UTI is being cleared to a 0. 0 Array integrity checks are not enabled. 1 Array integrity checks are enabled.
31 AID	Array Integrity Done. AID is cleared upon an Array integrity check being enabled (to signify the operation is ongoing). Once completed, AID is set to indicate that the array integrity check is complete. At this time the MISR (UMR registers) can be checked. AID can not be written, and is status only. 0 Array integrity check is ongoing. 1 Array integrity check is done.

24.2.2.13 User Test Register 1 (FLASH_x_UT1)

The User Test Register 1 (FLASH_x_UT1) provides added controllability to UTest.

Offset: 0x0040 / 0x4040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DAI															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DAI															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-16. User Test Register 1 (FLASH_x_UT1)

Table 24-17. FLASH_x_UT1 Field Descriptions

Field	Description
0–31 DAI	Data Array Input. These bits enable checks of ECC logic by allowing data bits to be input into the ECC logic and then read out by doing array reads or array integrity checks. The DAI[31:0] correspond to the 32 Array bits representing Word 0 of the double word selected in the FLASH_x_AR.

24.2.2.14 User Test Register 2 (FLASH_x_UT2)

Offset: 0x0044 / 0x4044 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DAI															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	DAI															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-17. User Test Register 2 (FLASH_x_UT2)

Table 24-18. FLASH_x_UT2 Field Descriptions

Field	Description
0–31 DAI	Data Array Input. These bits enable checks of ECC logic by allowing data bits to be input into the ECC logic and then read out by doing array reads or array integrity checks. The DAI[63:32] correspond to the 32 Array bits representing Word 1of the double word selected in the FLASH_x_AR.

24.3 Functional Description

24.3.1 Flash User Mode

In user mode the flash module can be read and written (register writes and interlock writes), programmed or erased. The following sub-sections define all actions that can be performed in user mode.

24.3.2 Flash Read and Write

The default state of the flash module is read. The main and shadow address space can be read only in the read state. The module configuration register (FLASH_x_MCR) is always available for read. The flash module enters the read state on reset. The flash module is in the read state under these conditions:

- The read state is active when PGM = 1 or ERS = 1 in the FLASH_x_MCR and high-voltage operation is ongoing (read while write).

NOTE

> Reads done to the partition(s) being operated on (either erased or programmed) will result in an error and the RWE bit in the FLASH_x_MCR will be set.

- The read state is active when PGM = 1 and PSUS = 1 in the FLASH_x_MCR (program suspend).
- The read state is active when ERS = 1 and ESUS = 1 and PGM = 0 in the FLASH_x_MCR (erase suspend).

In flash user mode, registers can be written. Array can be written to do interlock writes.

Interlock writes attempted to invalid locations (due to blocks that do not exist in non 2^n array sizes), will result in an interlock occurring, but attempts to program or erase these blocks will not occur since they are forced to be locked.

24.3.3 Read While Write (RWW)

The flash core is divided into partitions. Partitions are always comprised of two or more blocks. Partitions are used to determine read-while-write (RWW) groupings. While a write (program or erase) is being done within a given partition, a read can be simultaneously executed to any other partition. Partitions are listed in Table 24-1.

For each Flash array, the high address space of each RWW partition is physically comprised of two 256K blocks as shown in Figure 24-1. However, because the high address space blocks are interleaved every 16 bytes between Flash array A and Flash array B, the practical size of the high address space RWW partitions is effectively four 256K blocks.

The shadow block has unique RWW restrictions described in Section 24.3.6, “Flash Shadow Block”.

The FC is also divided into blocks to implement independent erase or program protection. The shadow block exists outside the normal address space and is programmed, erased, and read independently of the other blocks. The shadow block is included to support systems that require NVM for security or system initialization information.

A software mechanism is provided to independently lock or unlock each block in high-, mid-, and low-address space against program and erase. Two hardware locks are also provided to enable/disable the FC for program/erase. See Section 24.3.4.1, “Software Locking”, for more information.

24.3.4 Flash Programming

Programming changes the value stored in an array bit from logic 1 to logic 0 only. Programming cannot change a stored logic 0 to a logic 1. Addresses in locked/disabled blocks cannot be programmed. The user can program the values in any or all of four words within a page in a single program sequence. Word addresses are selected using bits 3:2 of the page-bound word.

Whenever a program operation occurs, ECC bits are programmed. ECC is handled on a 64-bit boundary. Thus, if only one word in any given 64-bit ECC segment is programmed, the adjoining word (in that segment) should not be programmed because ECC calculation has already completed for that 64-bit segment. Attempts to program the adjoining word will probably result in an operation failure. It is recommended that all programming operations be from 64 bits to 128 bits, and be 64-bit aligned. The programming operation should completely fill selected ECC segments within the page.

The program operation consists of the following sequence of events:

1. Change the value in the FLASH_x_MCR[PGM] bit from a 0 to a 1.

NOTE

Ensure the block that contains the address to be programmed is unlocked. See Section 24.2.2.2, “Low/Mid Address Space Block Locking Register (FLASH_x_LMLR)”, Section 24.2.2.3, “High Address Space Block Locking Register (FLASH_x_HLR)”, and Section 24.2.2.4, “Secondary Low/Mid Address Space Block Locking Register (FLASH_x_SLMLR)”, for more information.

2. Write the first address to be programmed in the flash module with the program data. This write is referred to as a program data interlock write. An interlock write may be either be an aligned word or doubleword.
3. If more than one word or doubleword is to be programmed, write each additional address in the page with data to be programmed. This is referred to as a program data write. All unwritten data words default to 0xFFFF_FFFF.
4. Write a logic 1 to the FLASH_x_MCR[EHV] bit to start the internal program sequence or skip to step 9 to terminate.
5. Wait until the FLASH_x_MCR[DONE] bit goes high.
6. Confirm FLASH_x_MCR[PEG] = 1.
7. Write a logic 0 to the FLASH_x_MCR[EHV] bit.
8. If more addresses are to be programmed, return to step 2.
9. Write a logic 0 to the FLASH_x_MCR[PGM] bit to terminate the program sequence.

The program sequence is presented graphically in Figure 24-18. The program suspend operation detailed in Figure 24-18 is discussed in Section 24.3.4.1.1, “Flash Program Suspend/Resume”.

The first write after a program is initiated determines the page address to be programmed. Program may be initiated with the 0 to 1 transition of the FLASH_x_MCR[PGM] bit or by clearing the FLASH_x_MCR[EHV] bit at the end of a previous program. This first write is referred to as an interlock write. If the program is not an erase-suspended program, the interlock write determines if the shadow or normal array space will be programmed and causes FLASH_x_MCR[PEAS] to be set/cleared.

NOTE

> Only the first write after a program is initiated does the interlock write and all later writes only look at bits 2 and 3 of the address so be careful that all writes after the interlock write are for that same 128 bit section.

In the case of an erase-suspended program, the value in FLASH_x_MCR[PEAS], is retained from the erase.

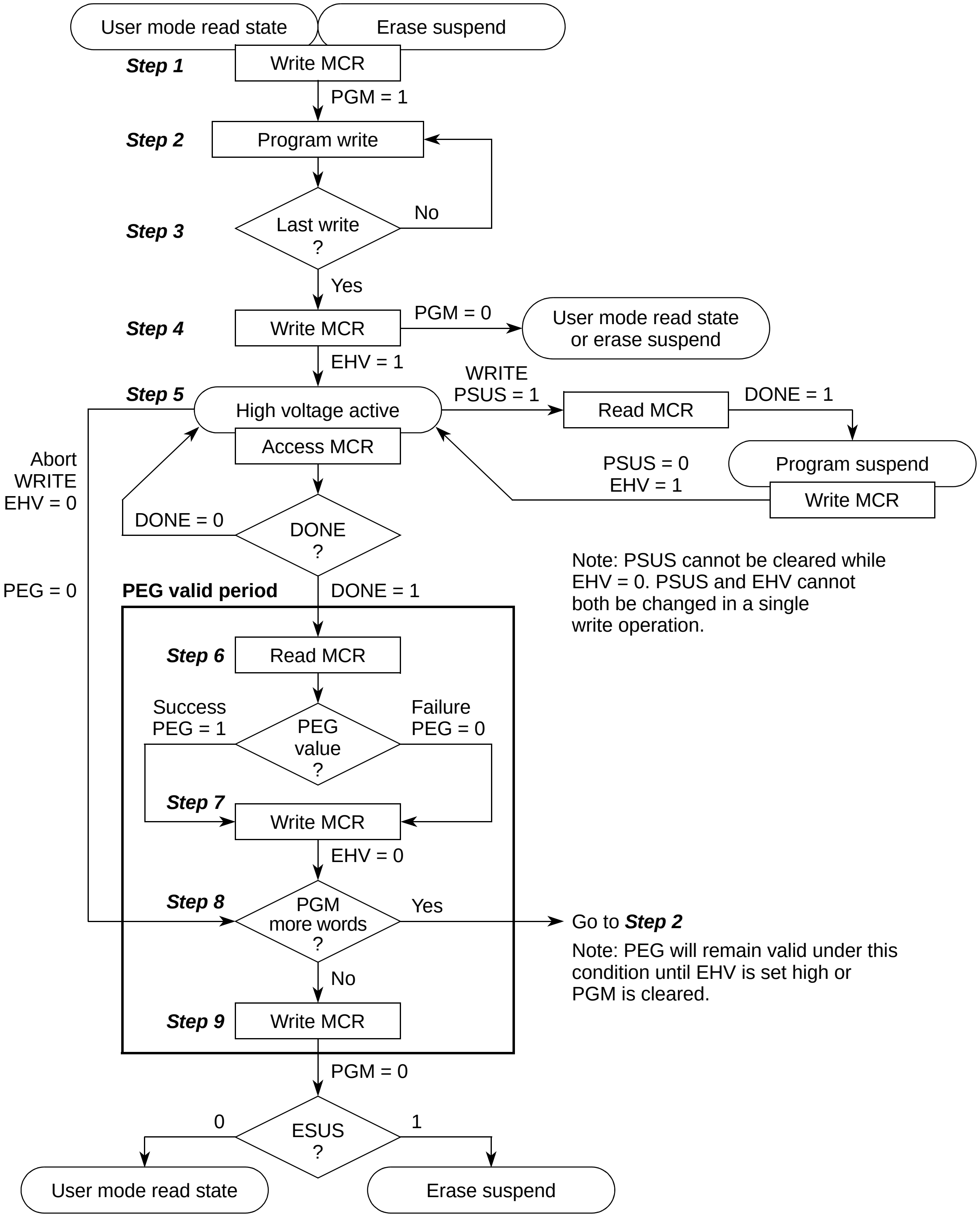
An interlock write must be performed before setting FLASH_x_MCR[EHV]. The user may terminate a program sequence by clearing FLASH_x_MCR[PGM] prior to setting FLASH_x_MCR[EHV].

If multiple writes are done to the same location the data for the last write is used in programming.

While FLASH_x_MCR[DONE] is low, FLASH_x_MCR[EHV] is high, and FLASH_x_MCR[PSUS] is low, the user may clear FLASH_x_MCR[EHV], resulting in a program abort. A program abort forces the module to step 8 of the program sequence. An aborted program will result in FLASH_x_MCR[PEG] being set low, indicating a failed operation. The data space being operated on before the abort will contain indeterminate data. The user may not abort a program sequence while in program suspend.

NOTE

> Aborting a program operation will leave the flash core addresses being programmed in an indeterminate data state. This may be recovered by executing an erase on the affected blocks.



Note: PSUS cannot be cleared while EHV = 0. PSUS and EHV cannot both be changed in a single write operation.

Note: PEG will remain valid under this condition until EHV is set high or PGM is cleared.

Figure 24-18. Program Sequence

24.3.4.1 Software Locking

A software mechanism is provided to independently lock/unlock each high-, mid-, and low-address space against program and erase.

Software locking is done through the FLASH_x_LMLR (low-/mid-address space block locking register), FLASH_x_SLMLR (secondary low-/mid-address space block locking register), or FLASH_x_HLR (high-address space block locking register). These can be written through register writes and read through register reads.

When the program/erase operations are enabled through hardware, software locks are enforced through doing register writes.

24.3.4.1.1 Flash Program Suspend/Resume

The program sequence may be suspended to allow read access to the flash core. It is not possible to erase or program during a program suspend. Interlock writes should not be attempted during program suspend.

A program suspend can be initiated by changing the value of the FLASH_x_MCR[PSUS] bit from a 0 to a 1. FLASH_x_MCR[PSUS] can be set high at any time when FLASH_x_MCR[PGM] and FLASH_x_MCR[EHV] are high. A 0 to 1 transition of FLASH_x_MCR[PSUS] causes the flash module to start the sequence to enter program suspend, which is a read state. The module is not suspended until FLASH_x_MCR[DONE] = 1. At this time flash core reads may be attempted. After it is suspended, the flash core may be read only. Reads to the blocks being programmed/erased return indeterminate data.

The program sequence is resumed by writing a logic 0 to FLASH_x_MCR[PSUS]. FLASH_x_MCR[EHV] must be set to a 1 before clearing FLASH_x_MCR[PSUS] to resume operation. When the operation resumes, the flash module continues the program sequence from one of a set of predefined points. This may extend the time required for the program operation.

24.3.5 Flash Erase

Erase changes the value stored in all bits of the selected blocks to logic 1. Locked or disabled blocks cannot be erased. If multiple blocks are selected for erase during an erase sequence, the blocks are erased sequentially starting with the lowest numbered block and terminating with the highest. Aborting an erase operation will leave the flash core blocks being erased in an indeterminate data state. This can be recovered by executing an erase on the affected blocks.

The erase sequence consists of the following sequence of events:

1. Change the value in the FLASH_x_MCR[ERS] bit from 0 to a 1.
2. Select the block, or blocks, to be erased by writing 1s to the appropriate registers in FLASH_x_LMSR or FLASH_x_HSR. If the shadow row is to be erased, this step may be skipped, and FLASH_x_LMSR and FLASH_x_HSR are ignored. For shadow row erase, see section Section 24.3.6, “Flash Shadow Block”, for more information.

NOTE

Lock and select are independent. If a block is selected and locked, no erase will occur. See Section 24.2.2.2, “Low/Mid Address Space Block Locking Register (FLASH_x_LMLR)”, Section 24.2.2.3, “High Address Space Block Locking Register (FLASH_x_HLR)”, and Section 24.2.2.4, “Secondary Low/Mid Address Space Block Locking Register (FLASH_x_SLMLR)”, for more information.

3. Write to any address in flash within the flash array A or B (i.e. if you intend to erase a B block, you need to make sure the address is in array B). This is referred to as an erase interlock write.
4. Write a logic 1 to the FLASH_x_MCR[EHV] bit to start an internal erase sequence or skip to step 9 to terminate.
5. Wait until the FLASH_x_MCR[DONE] bit goes high.
6. Confirm FLASH_x_MCR[PEG] = 1.
7. Write a logic 0 to the FLASH_x_MCR[EHV] bit.
8. If more blocks are to be erased, return to step 2.
9. Write a logic 0 to the FLASH_x_MCR[ERS] bit to terminate the erase.

The erase sequence is presented graphically in Figure 24-19. The erase suspend operation detailed in Figure 24-19 is discussed in section Section 24.3.5.1, “Flash Erase Suspend/Resume”.

After setting FLASH_x_MCR[ERS], one write (referred to as an interlock write) must be performed before FLASH_x_MCR[EHV] can be set to a 1. Data words written during erase sequence interlock writes are ignored. The user may terminate the erase sequence by clearing FLASH_x_MCR[ERS] before setting FLASH_x_MCR[EHV].

An erase operation may be aborted by clearing FLASH_x_MCR[EHV] assuming FLASH_x_MCR[DONE] is low, FLASH_x_MCR[EHV] is high, and FLASH_x_MCR[ESUS] is low. An erase abort forces the module to step 8 of the erase sequence. An aborted erase will result in FLASH_x_MCR[PEG] being set low, indicating a failed operation. The blocks being operated on before the abort contain indeterminate data. The user may not abort an erase sequence while in erase suspend.

NOTE

Aborting an erase operation will leave the flash core blocks being erased in an indeterminate data state. This may be recovered by executing an erase on the affected blocks.

24.3.5.1 Flash Erase Suspend/Resume

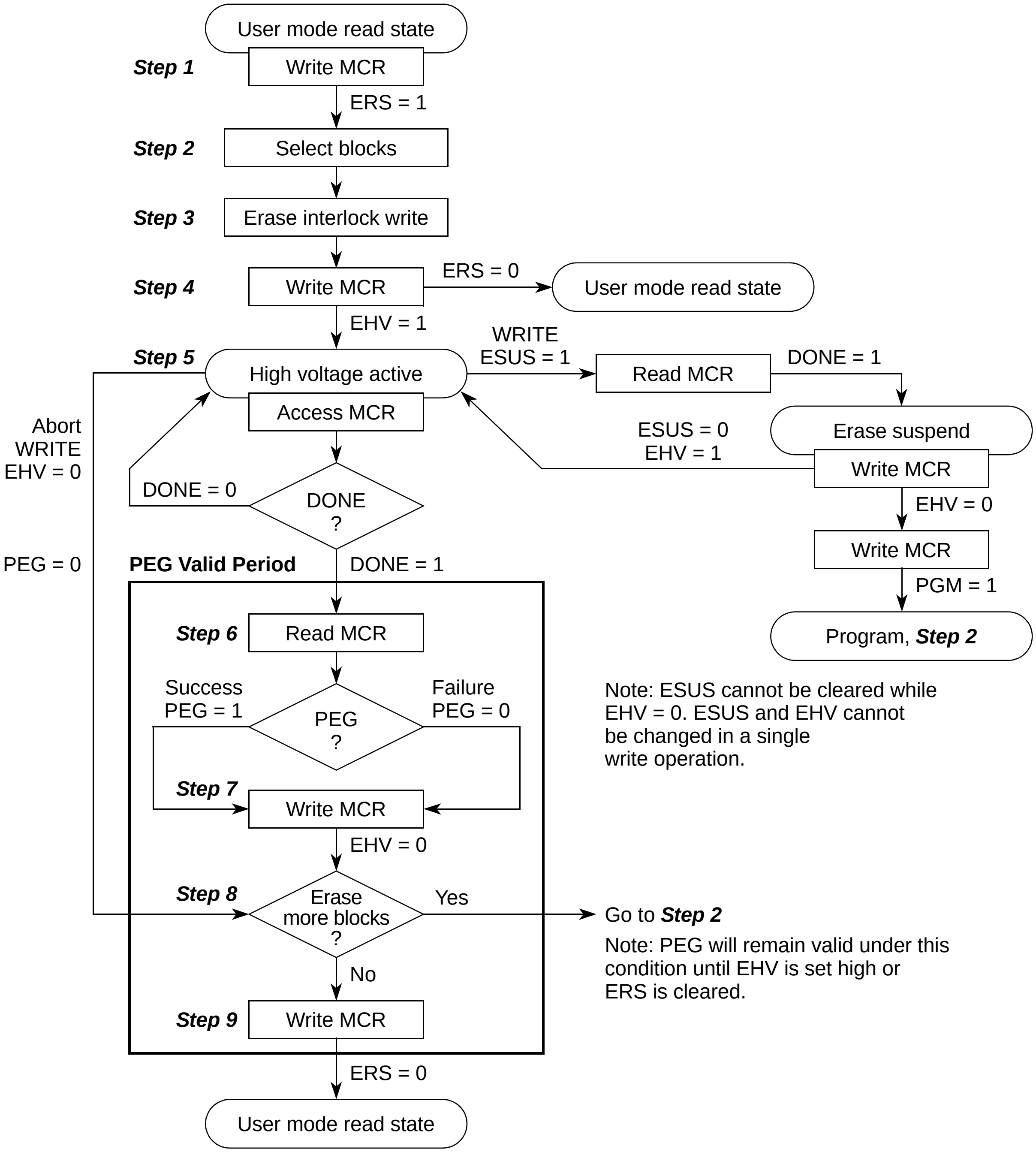
The erase sequence may be suspended to allow read access to the flash core. The erase sequence may also be suspended to program (erase-suspended program) the flash core. A program started during erase suspend can be suspended. One erase suspend and one program suspend are allowed at a time during an operation. It is not possible to erase during an erase suspend, or program during a program suspend. During suspend, all reads to flash core locations targeted for program and blocks targeted for erase return indeterminate data. Programming locations in blocks targeted for erase during erase-suspended program may result in corrupted data.

An erase suspend operation is initiated by setting the FLASH_x_MCR[ESUS] bit. FLASH_x_MCR[ESUS] can be set to a 1 at any time when FLASH_x_MCR[ERS] and FLASH_x_MCR[EHV] are high and FLASH_x_MCR[PGM] is low. A 0 to 1 transition of FLASH_x_MCR[ESUS] causes the flash module to start the sequence which places it in erase suspend. The user must wait until FLASH_x_MCR[DONE] = 1 before the module is suspended and further actions are attempted. After it is suspended, the array may be read or a program sequence may be initiated (erase-suspended program). Before initiating a program sequence the user must first clear FLASH_x_MCR[EHV]. If a program sequence is initiated, the value of the FLASH_x_MCR[PEAS] is not reset. These values are fixed at the time of the first interlock of the erase. Flash core reads from the blocks being erased while FLASH_x_MCR[ESUS] = 1 return indeterminate data.

The erase operation is resumed by clearing the FLASH_x_MCR[ESUS] bit. The flash continues the erase sequence from one of a set of predefined points. This can extend the time required for the erase operation.

CAUTION

In an erase-suspended program, programming flash locations in blocks which were being operated on in the erase may corrupt flash core data.



Note: ESUS cannot be cleared while EHV = 0. ESUS and EHV cannot be changed in a single write operation.

Note: PEG will remain valid under this condition until EHV is set high or ERS is cleared.

Figure 24-19. Erase Sequence

24.3.6 Flash Shadow Block

The flash shadow block is a memory-mapped block in the flash memory map. Program and erase of the shadow block are enabled when FLASH_x_MCR[PEAS] = 1 only. After the user has begun an erase operation on the shadow block, the operation cannot be suspended to program the main address space and vice-versa. The user must terminate the shadow erase operation to program or erase the main address space.

NOTE

If an erase of user space is requested, and a suspend is done with attempts to erase suspend program shadow space, this attempted program will be directed to user space as dictated by the state of FLASH_x_MCR[PEAS]. Likewise an attempted erase suspended program of user space, while the shadow space is being erased, will be directed to shadow space as dictated by the state of FLASH_x_MCR[PEAS].

The shadow block cannot use the RWW feature. After an operation is started in the shadow block, a read cannot be done to the shadow block, or any other block. Likewise, after an operation is started in a block in low-/mid-/high-address space, a read cannot be done in the shadow block.

The shadow block contains information about how the lock registers are reset.

The shadow block may be locked/unlocked against program or erase by using the FLASH_x_LMLR or FLASH_x_SLMLR discussed in Section 24.2.2, “Register Descriptions”.

WARNING

If the shadow flash block is erased without reprogramming a new valid password and censorship control word before a reset occurs it will contain an illegal password and the debug port will be inaccessible. Also, if code does not exist in the first bootable region of the internal flash to reprogram the shadow flash with the proper censorship control word and password, the device will be in a censored state and may no longer be usable.

Programming the shadow row has similar restrictions to programming the array in terms of how ECC is calculated. See Section 24.3.4, “Flash Programming”, for more information. Only one program is allowed per 64 bit ECC segment between erases. Erase of the shadow row is done similarly as an array erase. See section Section 24.3.5, “Flash Erase”, for more information.

24.3.7 Flash Reset

A reset is the highest priority operation for the flash and terminates all other operations.

The flash uses reset to initialize register and status bits to their default reset values. If the flash is executing a program or erase operation and a reset is issued, the operation will be aborted and the flash will disable the high voltage logic without damage to the high-voltage circuits. Reset aborts all operations and forces the flash into user mode ready to receive accesses.

After reset is negated, register accesses can be performed, although it should be noted that registers that require updating from shadow information, or other inputs, cannot read updated until flash exits reset.

Chapter 25
FlexCAN Module

25.1 Introduction

This device has four FlexCAN modules: A, B, C, and D. The remainder of this chapter describes one module.

Each FlexCAN module is a communication controller implementing the CAN protocol according to the CAN 2.0B protocol specification [Ref. 1]. A general block diagram is shown in Figure 25-1, which describes the main sub-blocks implemented in the FlexCAN module, including two embedded memories, one for storing Message Buffers (MB) and another one for storing Rx Individual Mask Registers. Support for 64 Message Buffers is provided. The functions of the sub-modules are described in subsequent sections.

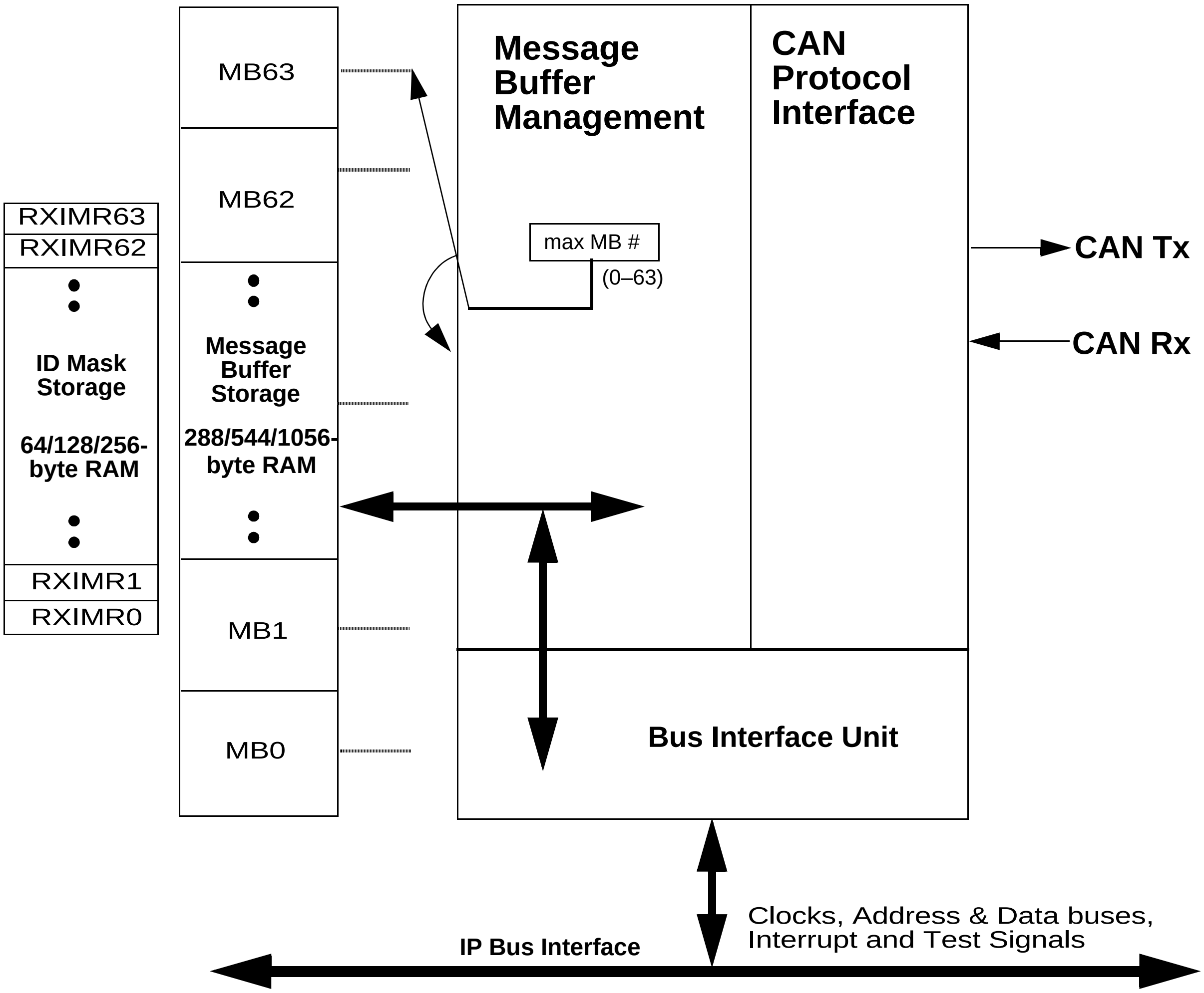


Figure 25-1. FlexCAN Block Diagram

25.1.1 Overview

The CAN protocol was primarily, but not only, designed to be used as a vehicle serial data bus, meeting the specific requirements of this field: real-time processing, reliable operation in the EMI environment of a vehicle, cost-effectiveness and required bandwidth. The FlexCAN module is a full implementation of the CAN protocol specification, Version 2.0 B [Ref. 1], which supports both standard and extended message frames. The FlexCAN modules on the MPC5676R supports 64 message buffers. The Message Buffers are stored in an embedded RAM dedicated to the FlexCAN module.

The CAN Protocol Interface (CPI) sub-module manages the serial communication on the CAN bus, requesting RAM access for receiving and transmitting message frames, validating received messages and performing error handling. The Message Buffer Management (MBM) sub-module handles Message Buffer selection for reception and transmission, taking care of arbitration and ID matching algorithms. The Bus Interface Unit (BIU) sub-module controls the access to and from the internal interface bus, in order to establish connection to the CPU and to other blocks. Clocks, address and data buses, interrupt outputs and test signals are accessed through the Bus Interface Unit.

25.1.2 FlexCAN Module Features

The FlexCAN module includes these distinctive features:

- Full Implementation of the CAN protocol specification, Version 2.0B
    - Standard data and remote frames
    - Extended data and remote frames
    - Zero to eight bytes data length
    - Programmable bit rate up to 1 Mb/sec
    - Content-related addressing
- Flexible Message Buffers (64) of zero to eight bytes data length
- Each MB configurable as Rx or Tx, all supporting standard and extended messages
- Individual Rx Mask Registers per Message Buffer
- Includes 1056 bytes (64 MBs) of RAM used for MB storage
- Includes 256 bytes (64 MBs) of RAM used for individual Rx Mask Registers
- Full featured Rx FIFO with storage capacity for 6 frames and internal pointer handling
- Powerful Rx FIFO ID filtering, capable of matching incoming IDs against either 8 extended, 16 standard or 32 partial (8 bits) IDs, with individual masking capability
- Selectable backwards compatibility with previous FlexCAN version
- Programmable clock source to the CAN Protocol Interface, either bus clock or crystal oscillator
- Unused MB and Rx Mask Register space can be used as general purpose RAM space
- Listen only mode capability
- Programmable loop-back mode supporting self-test operation
- Programmable transmission priority scheme: lowest ID, lowest buffer number or highest priority
- Time Stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message
- Maskable interrupts
- Independent of the transmission medium (an external transceiver is assumed)
- Short latency time due to an arbitration scheme for high-priority messages
- Low power modes

25.1.3 Modes of Operation

The FlexCAN module has these functional modes: Normal Mode (User and Supervisor), Freeze Mode, Listen-Only Mode and Loop-Back Mode. There are also these low power modes: Disable Mode and Stop Mode.

- Normal Mode (User or Supervisor):

  In Normal Mode, the module operates receiving and/or transmitting message frames, errors are handled normally and all the CAN Protocol functions are enabled. User and Supervisor Modes differ in the access to some restricted control registers.

- Freeze Mode:

  It is enabled when the FRZ bit in the FLEXCAN_x_MCR Register is asserted. If enabled, Freeze Mode is entered when the HALT bit in FLEXCAN_x_MCR is set or when Debug Mode is requested at MCU level. In this mode, no transmission or reception of frames is done and synchronicity to the CAN bus is lost. See Section 25.4.9.1, “Freeze Mode”, for more information.

- Listen-Only Mode:

  The module enters this mode when the LOM bit in the Control Register is asserted. In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode [Ref. 1]. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message.

- Loop-Back Mode:

  The module enters this mode when the LPB bit in the Control Register is asserted. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is internally fed back to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic ‘1’). FlexCAN behaves as it normally does when transmitting and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field to ensure proper reception of its own message. Both transmit and receive interrupts are generated.

- Module Disable Mode:

  This low power mode is entered when the MDIS bit in the FLEXCAN_x_MCR Register is asserted by the CPU. When the FlexCAN module is disabled, the module sends a request to disable the clocks to the CAN Protocol Interface and Message Buffer Management sub-modules. Exit from this mode is done by negating the MDIS bit in the FLEXCAN_x_MCR Register. See Section 25.4.9.2, “Module Disable Mode”, for more information.

- Stop Mode:

  This low power mode is entered when Stop Mode is requested at MCU level. When in Stop Mode, the module puts itself in an inactive state and then informs the CPU that the clocks can be shut down globally. Exit from this mode happens when the Stop Mode request is removed.

25.2 External Signal Description

25.2.1 Overview

The FlexCAN module has two I/O signals connected to the external MCU pins. These signals are summarized in Table 25-1 and described in more detail in the next subsections.

Table 25-1. FlexCAN Signals

Signal Name[1]	Direction	Description
CAN Rx	Input	CAN Receive Pin
CAN Tx	Output	CAN Transmit Pin

NOTES:

[1] The actual MCU pins may have different names. Please consult the Device User Guide for the actual signal names.

25.2.2 Signal Descriptions

25.2.2.1 CAN Rx

This pin is the receive pin from the CAN bus transceiver. Dominant state is represented by logic level ‘0’. Recessive state is represented by logic level ‘1’.

25.2.2.2 CAN Tx

This pin is the transmit pin to the CAN bus transceiver. Dominant state is represented by logic level ‘0’. Recessive state is represented by logic level ‘1’.

25.3 Memory Map/Register Definition

This section describes the registers and data structures in the FlexCAN module. The base address of the module depends on the particular memory map of the MCU. The addresses presented here are relative to the base address.

The address space occupied by FlexCAN has 96 bytes for registers starting at the module base address, followed by MB storage space in embedded RAM starting at address 0x0060, and an extra ID Mask storage space in a separate embedded RAM starting at address 0x0880.

25.3.1 FlexCAN Memory Mapping

The complete memory map for a FlexCAN module with 64 MBs capability is shown in Table 25-2. Each individual register is identified by its complete name and the corresponding mnemonic. The access type is always Supervisor (S).

The FLEXCAN_x_IFLAG2 and FLEXCAN_x_IMASK2 registers are considered reserved space when FlexCAN is configured with 16 or 32 MBs. The Rx Global Mask (FLEXCAN_x_RXGMASK), Rx Buffer 14 Mask (FLEXCAN_x_RX14MASK) and the Rx Buffer 15 Mask (FLEXCAN_x_RX15MASK) registers are provided for backwards compatibility, and are not used when the MBFEN bit in FLEXCAN_x_MCR is asserted.

The address ranges 0x0060–0x047F and 0x0880–0x097F are occupied by two separate embedded memories. These two ranges are completely occupied by RAM (1056 and 256 bytes, respectively) only when FlexCAN is configured with 64 MBs. When it is configured with 16 MBs, the memory sizes are 288 and 64 bytes, so the address ranges 0x0180–0x047F and 0x08C0–0x097F are considered reserved space. When it is configured with 32 MBs, the memory sizes are 544 and 128 bytes, so the address ranges 0x0280–0x047F and 0x0900–0x097F are considered reserved space. Furthermore, if the MBFEN bit in FLEXCAN_x_MCR is negated, then the whole Rx Individual Mask Registers address range (0x0880–0x097F) is considered reserved space.

Table 25-2. Module Memory Map

Address[1]	Register	Bits	Access	Reset Value	Affected by Hard Reset	Affected by Soft Reset	Section/Page
Base + 0x0000	FLEXCAN_x_MCR—Module Configuration	32	S	0x5990_000F	Yes	Yes	25.3.4.1/14
Base + 0x0004	FLEXCAN_x_CTRL—Control Register	32	S	0x0000_0000	Yes	No	25.3.4.2/18
Base + 0x0008	FLEXCAN_x_TIMER—Free Running Timer	32	S	0x0000_0000	Yes	Yes	25.3.4.3/21
Base + 0x000C	Reserved						
Base + 0x0010	FLEXCAN_x_RXGMASK—Rx Global Mask	32	S	0xFFFF_FFFF	Yes	No	25.3.4.4/22
Base + 0x0014	FLEXCAN_x_RX14MASK—Rx Buffer 14 Mask	32	S	0xFFFF_FFFF	Yes	No	25.3.4.5/22
Base + 0x0018	FLEXCAN_x_RX15MASK—Rx Buffer 15 Mask	32	S	0xFFFF_FFFF	Yes	No	25.3.4.6/23
Base + 0x001C	FLEXCAN_x_ECR—Error Counter Register	32	S	0x0000_0000	Yes	Yes	25.3.4.7/23
Base + 0x0020	FLEXCAN_x_ESR—Error and Status Register	32	S	0x0000_0000	Yes	Yes	25.3.4.8/25
Base + 0x0024	FLEXCAN_x_IMASK2—Interrupt Masks 2	32	S	0x0000_0000	Yes	Yes	25.3.4.9/28
Base + 0x0028	FLEXCAN_x_IMASK1—Interrupt Masks 1	32	S	0x0000_0000	Yes	Yes	25.3.4.10/28
Base + 0x002C	FLEXCAN_x_IFLAG2—Interrupt Flags 2	32	S	0x0000_0000	Yes	Yes	25.3.4.11/29
Base + 0x0030	FLEXCAN_x_IFLAG1—Interrupt Flags 1	32	S	0x0000_0000	Yes	Yes	25.3.4.12/30
Base + 0x0034–0x005F	Reserved						
Base + 0x0060–0x007F	Reserved						
Base + 0x0080–0x017F	Message Buffers MB0–MB15	—	S	—	No	No	—
Base + 0x0180–0x027F	Message Buffers MB16–MB31	—	S	—	No	No	—
Base + 0x0280–0x047F	Message Buffers MB32–MB63	—	S	—	No	No	—
Base + 0x0480-087F	Reserved						
Base + 0x0880-0x08BF	RXIMR0-RXIMR15—Rx Individual Mask Registers		S		No	No	25.3.4.13/32
Base + 0x08C0-0x08FF	RXIMR16-RXIMR31—Rx Individual Mask Registers		S		No	No	25.3.4.13/32
Base + 0x0900-0x097F	RXIMR32-RXIMR63—Rx Individual Mask Registers		S		No	No	25.3.4.13/32

NOTES:
[1] FLEXCAN_A = 0xFFFC_0000
FLEXCAN_B = 0xFFFC_4000
FLEXCAN_C = 0xFFFC_8000
FLEXCAN_D = 0xFFFC_C000

The FlexCAN module stores CAN messages for transmission and reception using a Message Buffer structure. Each individual MB is formed by 16 bytes mapped in memory as described in Table 25-3. Table 25-3 shows a Standard/Extended Message Buffer (MB0) memory map, using 16 bytes total (0x80–0x8F space).

Table 25-3. Message Buffer MB0 Memory Mapping

Address Offset	MB Field
0x80	Control and Status (C/S)
0x84	Identifier Field
0x88–0x8F	Data Field 0 – Data Field 7 (1 byte each)

25.3.2 Message Buffer Structure

The Message Buffer structure used by the FlexCAN module is represented in Figure 25-2. Both Extended and Standard Frames (29-bit Identifier and 11-bit Identifier, respectively) used in the CAN specification (Version 2.0 Part B) are represented.

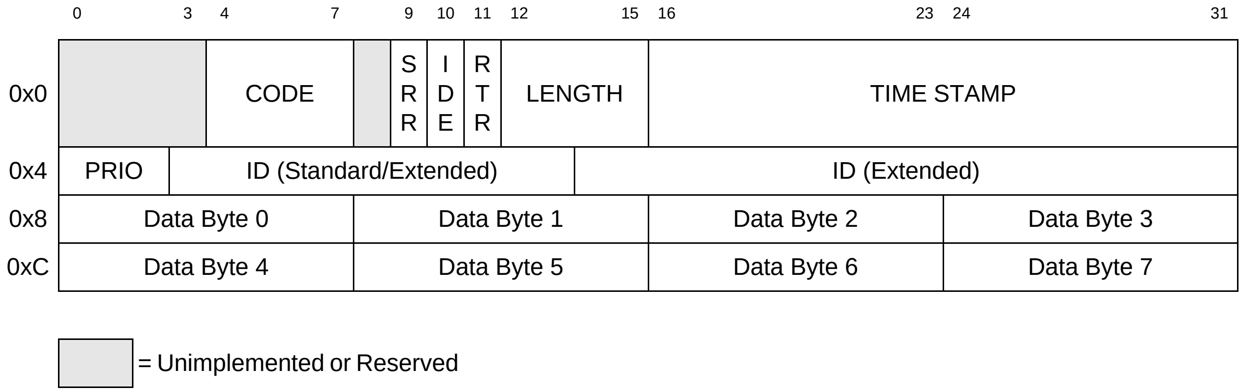


Figure 25-2. Message Buffer Structure

Table 25-4. Message Buffer Structure Field Descriptions

Field	Description
CODE	Message Buffer Code This 4-bit field can be accessed (read or write) by the CPU and by the Flexcan module itself, as part of the message buffer matching and arbitration process. The encoding is shown in Table 25-5 and Table 25-6. See Section 25.4, “Functional Description”, for additional information.
SRR	Substitute Remote Request Fixed recessive bit, used only in extended format. It must be set to ‘1’ by the user for transmission (Tx Buffers) and will be stored with the value received on the CAN bus for Rx receiving buffers. It can be received as either recessive or dominant. If FlexCAN receives this bit as dominant, then it is interpreted as arbitration loss. 0 Dominant is not a valid value for transmission in Extended Format frames 1 Recessive value is compulsory for transmission in Extended Format frames
IDE	ID Extended Bit This bit identifies whether the frame format is standard or extended. 0 Frame format is standard 1 Frame format is extended
RTR	Remote Transmission Request This bit is used for requesting transmissions of a data frame. If FlexCAN transmits this bit as ‘1’ (recessive) and receives it as ‘0’ (dominant), it is interpreted as arbitration loss. If this bit is transmitted as ‘0’ (dominant), then if it is received as ‘1’ (recessive), the FlexCAN module treats it as bit error. If the value received matches the value transmitted, it is considered as a successful bit transmission. 0 Indicates the current MB has a Data Frame to be transmitted 1 Indicates the current MB has a Remote Frame to be transmitted
LENGTH	Length of Data in Bytes This 4-bit field is the length (in bytes) of the Rx or Tx data, which is located in offset 0x8 through 0xF of the MB space (see Figure 25-2). In reception, this field is written by the FlexCAN module, copied from the DLC (Data Length Code) field of the received frame. In transmission, this field is written by the CPU and corresponds to the DLC field value of the frame to be transmitted. When RTR=1, the Frame to be transmitted is a Remote Frame and does not include the data field, regardless of the Length field.
TIME STAMP	Free-Running Counter Time Stamp This 16-bit field is a copy of the Free-Running Timer, captured for Tx and Rx frames at the time when the beginning of the Identifier field appears on the CAN bus.
PRIO	Local priority This 3-bit field is only used when LPRIO_EN bit is set in FLEXCAN_x_MCR and it only makes sense for Tx buffers. These bits are not transmitted. They are appended to the regular ID to define the transmission priority. See Section 25.4.3, “Arbitration process”.
ID	Frame Identifier In Standard Frame format, only the 11 most significant bits (3 to 13) are used for frame identification in both receive and transmit cases. The 18 least significant bits are ignored. In Extended Frame format, all bits are used for frame identification in both receive and transmit cases.
DATA	Data Field Up to eight bytes can be used for a data frame. For Rx frames, the data is stored as it is received from the CAN bus. For Tx frames, the CPU prepares the data field to be transmitted within the frame.

Table 25-5. Message Buffer Code for Rx buffers

Rx Code BEFORE Rx New Frame	Description	Rx Code AFTER Rx New Frame	Comment
0000	INACTIVE: MB is not active.	–	MB does not participate in the matching process.
0100	EMPTY: MB is active and empty.	0010	MB participates in the matching process. When a frame is received successfully, the code is automatically updated to FULL.
0010	FULL: MB is full.	0010	The act of reading the C/S word followed by unlocking the MB does not make the code return to EMPTY. It remains FULL. If a new frame is written to the MB after the C/S word was read and the MB was unlocked, the code still remains FULL.
		0110	If the MB is FULL and a new frame is overwritten to this MB before the CPU had time to read it, the code is automatically updated to OVERRUN. Refer to Section 25.4.5, “Matching Process”, for details about overrun behavior.
0110	OVERRUN: a frame was overwritten into a full buffer.	0010	If the code indicates OVERRUN but the CPU reads the C/S word and then unlocks the MB, when a new frame is written to the MB the code returns to FULL.
		0110	If the code already indicates OVERRUN, and yet another new frame must be written, the MB will be overwritten again, and the code will remain OVERRUN. Refer to Section 25.4.5, “Matching Process”, for details about overrun behavior.
0XY1[1]	BUSY: Flexcan is updating the contents of the MB. The CPU must not access the MB.	0010	An EMPTY buffer was written with a new frame (XY was 01).
		0110	A FULL/OVERRUN buffer was overwritten (XY was 11).

NOTES:

[1] Note that for Tx MBs (see Table 25-6), the BUSY bit should be ignored upon read, except when AEN bit is set in the FLEXCAN_x_MCR register.

Table 25-6. Message Buffer Code for Tx buffers

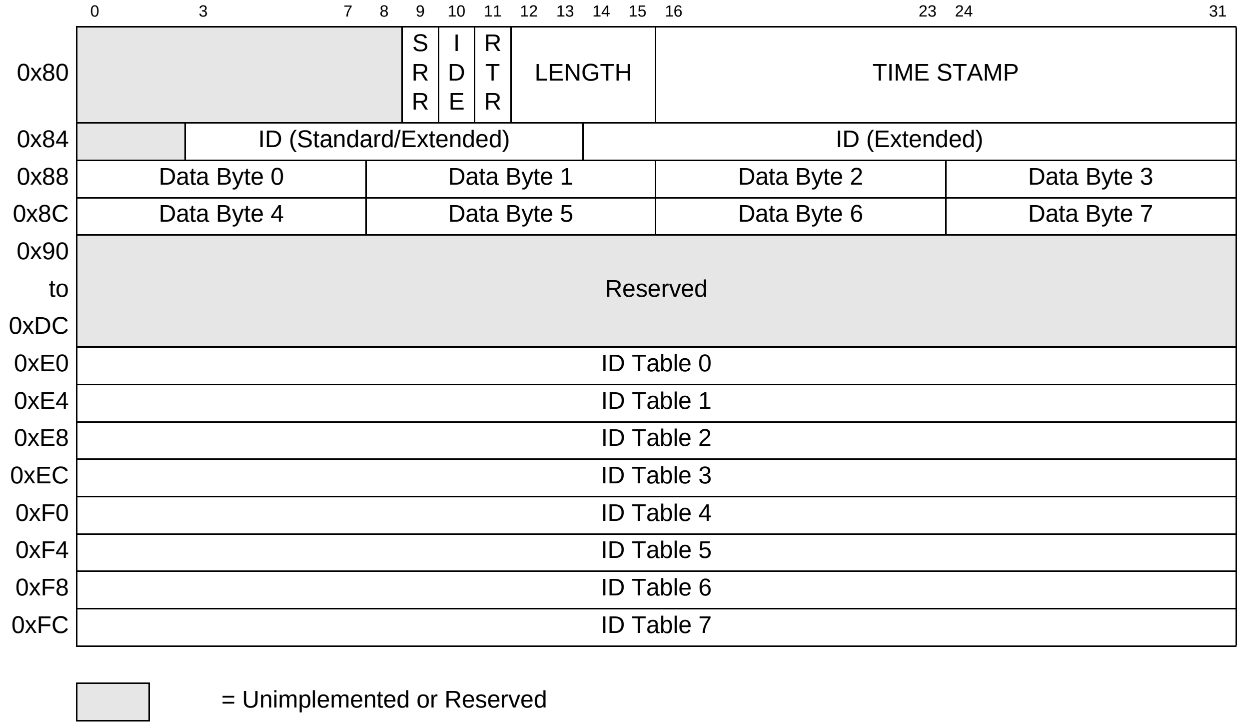
RTR	Initial Tx code	Code after successful transmission	Description
X	1000	–	INACTIVE: MB does not participate in the arbitration process.
X	1001	–	ABORT: MB was configured as Tx and CPU aborted the transmission. This code is only valid when AEN bit in FLEXCAN_x_MCR is asserted. MB does not participate in the arbitration process.
0	1100	1000	Transmit data frame unconditionally once. After transmission, the MB automatically returns to the INACTIVE state.
1	1100	0100	Transmit remote frame unconditionally once. After transmission, the MB automatically becomes an Rx MB with the same ID.

Table 25-6. Message Buffer Code for Tx buffers (continued)

RTR	Initial Tx code	Code after successful transmission	Description
0	1010	1010	Transmit a data frame whenever a remote request frame with the same ID is received. This MB participates simultaneously in both the matching and arbitration processes. The matching process compares the ID of the incoming remote request frame with the ID of the MB. If a match occurs this MB is allowed to participate in the current arbitration process and the Code field is automatically updated to ‘1110’ to allow the MB to participate in future arbitration runs. When the frame is eventually transmitted successfully, the Code automatically returns to ‘1010’ to restart the process again.
0	1110	1010	This is an intermediate code that is automatically written to the MB by the MBM as a result of match to a remote request frame. The data frame will be transmitted unconditionally once and then the code will automatically return to ‘1010’. The CPU can also write this code with the same effect.

25.3.3 Rx FIFO Structure

When the FEN bit is set in the FLEXCAN_x_MCR, the memory area from 0x80 to 0xFC (which is normally occupied by MBs 0 to 7) is used by the reception FIFO engine. Figure 25-3 shows the Rx FIFO data structure. The region 0x80-0x8C contains an MB structure which is the port through which the CPU reads data from the FIFO (the oldest frame received and not read yet). The region 0x90-0xDC is reserved for internal use of the FIFO engine. The region 0xE0-0xFC contains an 8-entry ID table that specifies filtering criteria for accepting frames into the FIFO. Figure 25-4 shows the three different formats that the elements of the ID table can assume, depending on the IDAM field of the FLEXCAN_x_MCR. Note that all elements of the table must have the same format. See Section 25.4.7, “Rx FIFO”, for more information.



<table>
<tr><th></th><th>0</th><th>3 … 7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12 13</th><th>14 15</th><th>16 … 23</th><th>24 … 31</th></tr>
<tr><td>0x80</td><td colspan="3">▨</td><td>SRR</td><td>IDE</td><td>RTR</td><td colspan="2">LENGTH</td><td colspan="2">TIME STAMP</td></tr>
<tr><td>0x84</td><td>▨</td><td colspan="6">ID (Standard/Extended)</td><td colspan="3">ID (Extended)</td></tr>
<tr><td>0x88</td><td colspan="2">Data Byte 0</td><td colspan="6">Data Byte 1</td><td>Data Byte 2</td><td>Data Byte 3</td></tr>
<tr><td>0x8C</td><td colspan="2">Data Byte 4</td><td colspan="6">Data Byte 5</td><td>Data Byte 6</td><td>Data Byte 7</td></tr>
<tr><td>0x90
to
0xDC</td><td colspan="10">Reserved</td></tr>
<tr><td>0xE0</td><td colspan="10">ID Table 0</td></tr>
<tr><td>0xE4</td><td colspan="10">ID Table 1</td></tr>
<tr><td>0xE8</td><td colspan="10">ID Table 2</td></tr>
<tr><td>0xEC</td><td colspan="10">ID Table 3</td></tr>
<tr><td>0xF0</td><td colspan="10">ID Table 4</td></tr>
<tr><td>0xF4</td><td colspan="10">ID Table 5</td></tr>
<tr><td>0xF8</td><td colspan="10">ID Table 6</td></tr>
<tr><td>0xFC</td><td colspan="10">ID Table 7</td></tr>
</table>

▨ = Unimplemented or Reserved

Figure 25-3. Rx FIFO Structure

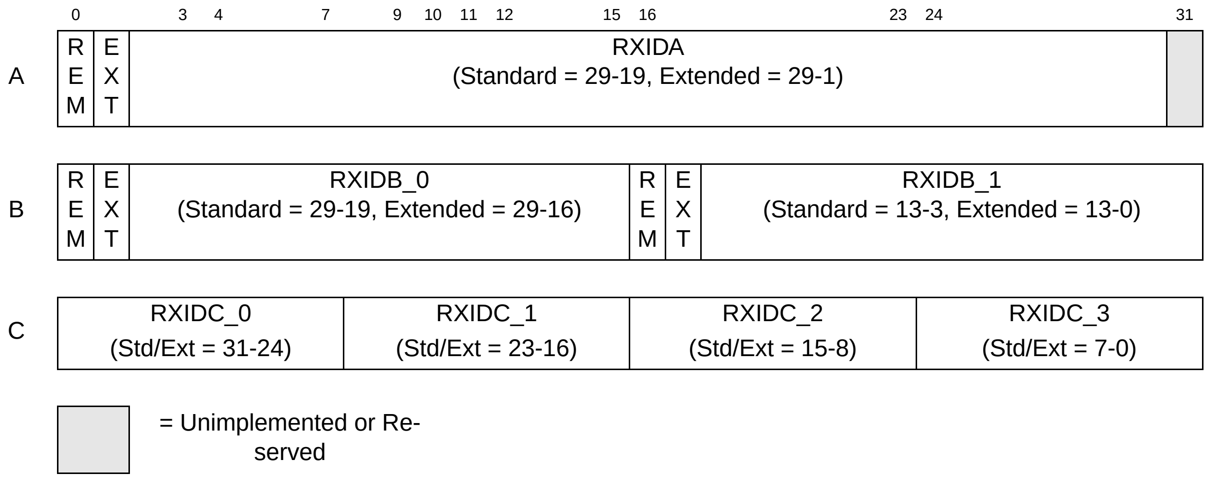


Figure 25-4. ID Table 0 - 7

Table 25-7. Rx FIFO Structure Field Descriptions

Field	Description
REM	Remote Frame This bit specifies if Remote Frames are accepted into the FIFO if they match the target ID. 0 Remote Frames are rejected and data frames can be accepted 1 Remote Frames can be accepted and data frames are rejected
EXT	Extended Frame Specifies whether extended or standard frames are accepted into the FIFO if they match the target ID. 0 Extended frames are rejected and standard frames can be accepted 1 Extended frames can be accepted and standard frames are rejected
RXIDA	Rx Frame Identifier (Format A) Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, only the 11 most significant bits (3 to 13)are used for frame identification. In the extended frame format, all bits are used.
RXIDB_0 RXIDB_1	Rx Frame Identifier (Format B) Specifies an ID to be used as acceptance criteria for the FIFO. In the standard frame format, the 11 most significant bits (a full standard ID) (3 to 13)are used for frame identification. In the extended frame format, all 14 bits of the field are compared to the 14 most significant bits of the received ID.
RXIDC_0 RXIDC_1 RXIDC_2 RXIDC_3	Rx Frame Identifier (Format C) Specifies an ID to be used as acceptance criteria for the FIFO. In both standard and extended frame formats, all 8 bits of the field are compared to the 8 most significant bits of the received ID.

25.3.4 Register Descriptions

The FlexCAN registers are described in this section in ascending address order.

25.3.4.1 Module Configuration Register (FLEXCAN_x_MCR)

This register defines global system configurations, such as the module operation mode (e.g., low power) and maximum message buffer configuration. Most of the fields in this register can be accessed at any time, except the MAXMB field, which should only be changed while the module is in Freeze Mode.

Base + 0x0000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MDIS	FRZ	FEN	HALT	NOT_RDY	0	SOFT_RST	FRZ_ACK	SUPV	0	WRN_EN	MDISACK	0	DOZE	SRX_DIS	MBFEN
W																
RESET:	0	1	0	1	1	0	0	1	1	0	0	1	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	LPRIO_EN	AEN	0	0	IDAM		0	0	MAXMB					
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 25-5. Module Configuration Register (FLEXCAN_x_MCR)

Table 25-8. FLEXCAN_x_MCR Field Descriptions

Field	Description
0 MDIS	Module Disable This bit controls whether FlexCAN is enabled or not. When disabled, FlexCAN shuts down the clocks to the CAN Protocol Interface and Message Buffer Management sub-modules. This is the only bit in FLEXCAN_x_MCR not affected by soft reset. See Section 25.4.9.2, “Module Disable Mode”, for more information. 0 Enable the FlexCAN module 1 Disable the FlexCAN module
1 FRZ	Freeze Enable The FRZ bit specifies the FlexCAN behavior when the HALT bit in the FLEXCAN_x_MCR Register is set or when Debug Mode is requested at MCU level. When FRZ is asserted, FlexCAN is enabled to enter Freeze Mode. Negation of this bit field causes FlexCAN to exit from Freeze Mode. 0 Not enabled to enter Freeze Mode 1 Enabled to enter Freeze Mode
2 FEN	FIFO Enable This bit controls whether the FIFO feature is enabled or not. When FEN is set, MBs 0 to 7 cannot be used for normal reception and transmission because the corresponding memory region (0x80-0xFF) is used by the FIFO engine. See Section 25.3.3, “Rx FIFO Structure”, and Section 25.4.7, “Rx FIFO”, for more information. 0 FIFO not enabled 1 FIFO enabled

Table 25-8. FLEXCAN_x_MCR Field Descriptions (continued)

Field	Description
3 HALT	Halt FlexCAN Assertion of this bit puts the FlexCAN module into Freeze Mode. The CPU should clear it after initializing the Message Buffers and Control Register. No reception or transmission is performed by FlexCAN before this bit is cleared. While in Freeze Mode, the CPU has write access to the Error Counter Register, that is otherwise read-only. Freeze Mode can not be entered while FlexCAN is in any of the low power modes. See Section 25.4.9.1, “Freeze Mode”, for more information. 0 No Freeze Mode request. 1 Enters Freeze Mode if the FRZ bit is asserted.
4 NOT_RDY	FlexCAN Not Ready This read-only bit indicates that FlexCAN is either in Disable Mode, Stop Mode or Freeze Mode. It is negated once FlexCAN has exited these modes. 0 FlexCAN module is either in Normal Mode, Listen-Only Mode or Loop-Back Mode 1 FlexCAN module is either in Disable Mode, Stop Mode or Freeze Mode
5	Reserved
6 SOFT_RST	Soft Reset When this bit is asserted, FlexCAN resets its internal state machines and some of the memory mapped registers. The following registers are reset: FLEXCAN_x_MCR (except the MDIS bit), FLEXCAN_x_TIMER, FLEXCAN_x_ECR, FLEXCAN_x_ESR, FLEXCAN_x_IMASK1, FLEXCAN_x_IMASK2, FLEXCAN_x_IFLAG1, FLEXCAN_x_IFLAG2. Configuration registers that control the interface to the CAN bus are not affected by soft reset. The following registers are unaffected: • FLEXCAN_x_CTRL • RXIMR0–RXIMR63 • FLEXCAN_x_RXGMASK, FLEXCAN_x_RX14MASK, FLEXCAN_x_RX15MASK • all Message Buffers The SOFT_RST bit can be asserted directly by the CPU when it writes to the FLEXCAN_x_MCR Register, but it is also asserted when global soft reset is requested at MCU level. Since soft reset is synchronous and has to follow a request/acknowledge procedure across clock domains, it may take some time to fully propagate its effect. The SOFT_RST bit remains asserted while reset is pending, and is automatically negated when reset completes. Therefore, software can poll this bit to know when the soft reset has completed. Soft reset cannot be applied while clocks are shut down in any of the low power modes. The module should be first removed from low power mode, and then soft reset can be applied. 0 No reset request 1 Resets the registers marked as “affected by soft reset” in Table 25-2.
7 FRZ_ACK	Freeze Mode Acknowledge This read-only bit indicates that FlexCAN is in Freeze Mode and its prescaler is stopped. The Freeze Mode request cannot be granted until current transmission or reception processes have finished. Therefore the software can poll the FRZ_ACK bit to know when FlexCAN has actually entered Freeze Mode. If Freeze Mode request is negated, then this bit is negated once the FlexCAN prescaler is running again. If Freeze Mode is requested while FlexCAN is in any of the low power modes, then the FRZ_ACK bit will only be set when the low power mode is exited. See Section 25.4.9.1, “Freeze Mode”, for more information. 0 FlexCAN not in Freeze Mode, prescaler running 1 FlexCAN in Freeze Mode, prescaler stopped

Table 25-8. FLEXCAN_x_MCR Field Descriptions (continued)

Field	Description
8 SUPV	Supervisor Mode Although the FlexCAN module provides a differentiation between Supervisor and User access types, all accesses will be always considered of the Supervisor type. As a consequence, the SUPV bit in the Module Configuration Register (FLEXCAN_x_MCR) has no effect on the module behavior. . 1 Affected registers are in Supervisor memory space. Any access without supervisor permission behaves as though the access was done to an unimplemented register location
9	Reserved
10 WRN_EN	Warning Interrupt Enable When asserted, this bit enables the generation of the TWRN_INT and RWRN_INT flags in the Error and Status Register. If WRN_EN is negated, the TWRN_INT and RWRN_INT flags will always be zero, independent of the values of the error counters, and no warning interrupt will ever be generated. 0 TWRN_INT and RWRN_INT bits are zero, independent of the values in the error counters. 1 TWRN_INT and RWRN_INT bits are set when the respective error counter transition from <96 to ≥ 96.
11 MDISACK	Module Disable Acknowledge This read-only bit indicates that FlexCAN is either in Disable Mode or Stop Mode. Either of these low power modes can not be entered until all current transmission or reception processes have finished, so the CPU can poll the MDISACK bit to know when FlexCAN has actually entered low power mode. See Section 25.4.9.2, “Module Disable Mode”, and Section 25.4.9.3, “Stop Mode”, for more information. 0 FlexCAN not in any of the low power modes 1 FlexCAN is either in Disable Mode or Stop mode
12	Reserved
13 DOZE	Doze Mode Enable Doze Mode is not supported on this device. Leave this bit as ‘0’. 0 FlexCAN is not enabled to enter low power mode when Doze Mode is requested
14 SRX_DIS	Self Reception Disable This bit defines whether FlexCAN is allowed to receive frames transmitted by itself. If this bit is asserted, frames transmitted by the module will not be stored in any MB, regardless if the MB is programmed with an ID that matches the transmitted frame, and no interrupt flag or interrupt signal will be generated due to the frame reception. 0 Self reception enabled 1 Self reception disabled
15 MBFEN	Message buffer filter enable. This bit provides the capability of enabling either individual masking of every message buffer, or global masking of message buffers. This bit is provided to support backwards compatibility with previous FlexCAN versions. When this bit is negated, the following configuration is applied: • Individual Rx ID masking is disabled. Instead of individual ID masking per MB, FlexCAN uses its previous masking scheme with FLEXCAN_x_RXGMASK, FLEXCAN_x_RX14MASK and FLEXCAN_x_RX15MASK. • The reception queue feature is disabled. Upon receiving a message, if the first MB with a matching ID that is found is still occupied by a previous unread message, FlexCAN will not look for another matching MB. It will override this MB with the new message and set the CODE field to ‘0110’ (overrun). Upon reset this bit is negated, allowing legacy software to work without modification. 0 Individual Rx masking and queue feature are disabled. 1 Individual Rx masking and queue feature are enabled.
16–17	Reserved

Table 25-8. FLEXCAN_x_MCR Field Descriptions (continued)

Field	Description
18 LPRIO_EN	Local Priority Enable This bit is provided for backwards compatibility reasons. It controls whether the local priority feature is enabled or not. It is used to extend the ID used during the arbitration process. With this extended ID concept, the arbitration process is done based on the full 32-bit word, but the actual transmitted ID still has 11-bit for standard frames and 29-bit for extended frames. 0 Local Priority disabled 1 Local Priority enabled
19 AEN	Abort Enable This bit is supplied for backwards compatibility reasons. When asserted, it enables the Tx abort feature. This feature guarantees a safe procedure for aborting a pending transmission, so that no frame is sent in the CAN bus without notification. 0 Abort disabled 1 Abort enabled
20–21	Reserved
22–23 IDAM	ID Acceptance Mode This 2-bit field identifies the format of the elements of the Rx FIFO filter table, as shown in Table 25-9. Note that all elements of the table are configured at the same time by this field (they are all the same format). See Section 25.3.3, “Rx FIFO Structure”.
24–25	Reserved
26–31 MAXMB	Maximum Number of Message Buffers This 6-bit field defines the maximum number of message buffers that will take part in the matching and arbitration processes. The reset value (0x0F) is equivalent to 16 MB configuration. This field should be changed only while the module is in Freeze Mode. Maximum MBs in use = MAXMB + 1. **Note:** MAXMB must be programmed with a value smaller or equal to the number of available Message Buffers, otherwise FlexCAN can transmit and receive wrong messages.

Table 25-9. IDAM Coding

IDAM	Format	Explanation
00	A	One full ID (standard or extended) per filter element.
01	B	Two full standard IDs or two partial 14-bit extended IDs per filter element.
10	C	Four partial 8-bit IDs (standard or extended) per filter element.
11	D	All frames rejected.

25.3.4.2 Control Register (FLEXCAN_x_CTRL)

This register is defined for specific FlexCAN control features related to the CAN bus, such as bit-rate, programmable sampling point within an Rx bit, Loop Back Mode, Listen Only Mode, Bus Off recovery behavior and interrupt enabling (Bus-Off, Error, Warning). It also determines the Division Factor for the clock prescaler. Most of the fields in this register should only be changed while the module is in Disable Mode or in Freeze Mode. Exceptions are the BOFF_MSK, ERR_MSK, TWRN_MSK, RWRN_MSK and BOFF_REC bits, that can be accessed at any time.

Base + 0x0004

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	PRESDIV								RJW		PSEG1			PSEG2		
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BOFF_MSK	ERR_MSK	CLK_SRC	LPB	TWRN_MSK	RWRN_MSK	0	0	SMP	BOFF_REC	TSYN	LBUF	LOM	PROPSEG		
W							▨	▨								
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 25-6. Control Register (FLEXCAN_x_CTRL)

Table 25-10. FLEXCAN_x_CTRL Field Descriptions

Field	Description
0–7 PRESDIV	Prescaler Division Factor This 8-bit field defines the ratio between the CPI clock frequency and the Serial Clock (Sclock) frequency. The Sclock period defines the time quantum of the CAN protocol. For the reset value, the Sclock frequency is equal to the CPI clock frequency. The Maximum value of this register is 0xFF, that gives a minimum Sclock frequency equal to the CPI clock frequency divided by 256. For more information refer to Section 25.4.8.4, “Protocol Timing”.” Sclock frequency = CPI clock frequency / (PRESDIV + 1)
8–9 RJW	Resync Jump Width This 2-bit field defines the maximum number of time quanta[1] that a bit time can be changed by one re-synchronization. The valid programmable values are 0–3. Resync Jump Width = RJW + 1.
10–12 PSEG1	Phase Segment 1 This 3-bit field defines the length of Phase Buffer Segment 1 in the bit time. The valid programmable values are 0–7. Phase Buffer Segment 1 = (PSEG1 + 1) x Time-Quanta.
13–15 PSEG2	Phase Segment 2 This 3-bit field defines the length of Phase Buffer Segment 2 in the bit time. The valid programmable values are 1–7. Phase Buffer Segment 2 = (PSEG2 + 1) x Time-Quanta.
16 BOFF_MSK	Bus Off Mask This bit provides a mask for the Bus Off Interrupt. 0 Bus Off interrupt disabled 1 Bus Off interrupt enabled

Table 25-10. FLEXCAN_x_CTRL Field Descriptions (continued)

Field	Description
17 ERR_MSK	Error Mask This bit provides a mask for the Error Interrupt. 0 Error interrupt disabled 1 Error interrupt enabled
18 CLK_SRC	CAN Engine Clock Source This bit selects the clock source to the CAN Protocol Interface (CPI) to be either the system clock (driven by the PLL) or the crystal oscillator clock (direct feed from the oscillator pin EXTAL). The selected clock is the one fed to the prescaler to generate the Serial Clock (Sclock). In order to guarantee reliable operation, this bit should only be changed while the module is in Disable Mode. See Section 25.4.8.4, “Protocol Timing”, for more information. 0 The CAN engine clock source is the oscillator clock 1 The CAN engine clock source is the bus clock
19 TWRN_MSK	Tx Warning Interrupt Mask This bit provides a mask for the Tx Warning Interrupt associated with the TWRN_INT flag in the Error and Status Register. This bit has no effect if the WRN_EN bit in FLEXCAN_x_MCR is negated and it is read as zero when WRN_EN is negated. 0 Tx Warning Interrupt disabled 1 Tx Warning Interrupt enabled
20 RWRN_MSK	Rx Warning Interrupt Mask This bit provides a mask for the Rx Warning Interrupt associated with the RWRN_INT flag in the Error and Status Register. This bit has no effect if the WRN_EN bit in FLEXCAN_x_MCR is negated and it is read as zero when WRN_EN is negated. 0 Rx Warning Interrupt disabled 1 Rx Warning Interrupt enabled
21 LPB	Loop Back This bit configures FlexCAN to operate in Loop-Back Mode. In this mode, FlexCAN performs an internal loop back that can be used for self test operation. The bit stream output of the transmitter is fed back internally to the receiver input. The Rx CAN input pin is ignored and the Tx CAN output goes to the recessive state (logic ‘1’). FlexCAN behaves as it normally does when transmitting, and treats its own transmitted message as a message received from a remote node. In this mode, FlexCAN ignores the bit sent during the ACK slot in the CAN frame acknowledge field, generating an internal acknowledge bit to ensure proper reception of its own message. Both transmit and receive interrupts are generated. 0 Loop Back disabled 1 Loop Back enabled
22–23	Reserved
24 SMP	Sampling Mode This bit defines the sampling mode of CAN bits at the Rx input. 0 Just one sample is used to determine the bit value 1 Three samples are used to determine the value of the received bit: the regular one (sample point) and 2 preceding samples, a majority rule is used

Table 25-10. FLEXCAN_x_CTRL Field Descriptions (continued)

Field	Description
25 BOFF_REC	Bus Off Recovery Mode This bit defines how FlexCAN recovers from Bus Off state. If this bit is negated, automatic recovering from Bus Off state occurs according to the CAN Specification 2.0B. If the bit is asserted, automatic recovering from Bus Off is disabled and the module remains in Bus Off state until the bit is negated by the user. If the negation occurs before 128 sequences of 11 recessive bits are detected on the CAN bus, then Bus Off recovery happens as if the BOFF_REC bit had never been asserted. If the negation occurs after 128 sequences of 11 recessive bits occurred, then FlexCAN will re-synchronize to the bus by waiting for 11 recessive bits before joining the bus. After negation, the BOFF_REC bit can be re-asserted again during Bus Off, but it will only be effective the next time the module enters Bus Off. If BOFF_REC was negated when the module entered Bus Off, asserting it during Bus Off will not be effective for the current Bus Off recovery. 0 Automatic recovering from Bus Off state enabled, according to CAN Spec 2.0 part B 1 Automatic recovering from Bus Off state disabled
26 TSYN	Timer Sync Mode This bit enables a mechanism that resets the free-running timer each time a message is received in Message Buffer 0. This feature provides means to synchronize multiple FlexCAN stations with a special “SYNC” message (i.e., global network time). If the FEN bit in FLEXCAN_x_MCR is set (FIFO enabled), MB8 is used for timer synchronization instead of MB0. 0 Timer Sync feature disabled 1 Timer Sync feature enabled
27 LBUF	Lowest Buffer Transmitted First This bit defines the ordering mechanism for Message Buffer transmission. When asserted, the LPRIO_EN bit does not affect the priority arbitration. 0 Buffer with highest priority is transmitted first 1 Lowest number buffer is transmitted first
28 LOM	Listen-Only Mode This bit configures FlexCAN to operate in Listen Only Mode. In this mode, transmission is disabled, all error counters are frozen and the module operates in a CAN Error Passive mode [Ref. 1]. Only messages acknowledged by another CAN station will be received. If FlexCAN detects a message that has not been acknowledged, it will flag a BIT0 error (without changing the REC), as if it was trying to acknowledge the message. 0 Listen Only Mode is deactivated 1 FlexCAN module operates in Listen Only Mode
29–31 PROPSEG	Propagation Segment This 3-bit field defines the length of the Propagation Segment in the bit time. The valid programmable values are 0–7. Propagation Segment Time = (PROPSEG + 1) * Time-Quanta. Time-Quantum = one Sclock period.

NOTES:

[1] One time quantum is equal to the Sclock period.

25.3.4.3 Free Running Timer (FLEXCAN_x_TIMER)

This register represents a 16-bit free running counter that can be read and written by the CPU. The timer starts from 0x0000 after Reset, counts linearly to 0xFFFF, and wraps around.

The timer is clocked by the FlexCAN bit-clock (which defines the baud rate on the CAN bus). During a message transmission/reception, it increments by one for each bit that is received or transmitted. When there is no message on the bus, it counts using the previously programmed baud rate. During Freeze Mode, the timer is not incremented.

The timer value is captured at the beginning of the identifier field of any frame on the CAN bus. This captured value is written into the Time Stamp entry in a message buffer after a successful reception or transmission of a message.

Writing to the timer is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

Base + 0x0008

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	TIMER															
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 25-7. Free Running Timer (FLEXCAN_x_TIMER)

25.3.4.4 Rx Global Mask (FLEXCAN_x_RXGMASK)

This register is provided for legacy support. Setting the MBFEN bit in FLEXCAN_x_MCR causes the FLEXCAN_x_RXGMASK Register to have no effect on the module operation.

FLEXCAN_x_RXGMASK is used as acceptance mask for all Rx MBs, excluding MBs 14–15, which have individual mask registers. When the FEN bit in FLEXCAN_x_MCR is set (FIFO enabled), the FLEXCAN_x_RXGMASK also applies to all elements of the ID filter table, except elements 6-7, which have individual masks.

The contents of this register must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

Base + 0x0010

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16
RESET:	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0
RESET:	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

▨ = Unimplemented or Reserved

Figure 25-8. Rx Global Mask Register (FLEXCAN_x_RXGMASK)

Table 25-11. FLEXCAN_x_RXGMASK Field Descriptions

Field	Description
0–31 MI31–MI0	Mask Bits For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 0 The corresponding bit in the filter is “don’t care” 1 The corresponding bit in the filter is checked against the one received

25.3.4.5 Rx 14 Mask (FLEXCAN_x_RX14MASK)

This register is provided for legacy support. Setting the MBFEN bit in FLEXCAN_x_MCR causes the FLEXCAN_x_RX14MASK Register to have no effect on the module operation.

FLEXCAN_x_RX14MASK is used as acceptance mask for the Identifier in Message Buffer 14. When the FEN bit in FLEXCAN_x_MCR is set (FIFO enabled), the RXG14MASK also applies to element 6 of the ID filter table. This register has the same structure as the Rx Global Mask Register. It must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

- Address Offset: 0x14
- Reset Value: 0xFFFF_FFFF

25.3.4.6 Rx 15 Mask (FLEXCAN_x_RX15MASK)

This register is provided for legacy support. Setting the MBFEN bit in FLEXCAN_x_MCR causes the FLEXCAN_x_RX15MASK Register to have no effect on the module operation.

When the MBFEN bit is negated, FLEXCAN_x_RX15MASK is used as acceptance mask for the Identifier in Message Buffer 15. When the FEN bit in FLEXCAN_x_MCR is set (FIFO enabled), the RXG14MASK also applies to element 7 of the ID filter table. This register has the same structure as the Rx Global Mask Register. It must be programmed while the module is in Freeze Mode, and must not be modified when the module is transmitting or receiving frames.

- Address Offset: 0x18
- Reset Value: 0xFFFF_FFFF

25.3.4.7 Error Counter Register (FLEXCAN_x_ECR)

This register has 2 8-bit fields reflecting the value of two FlexCAN error counters: Transmit Error Counter (Tx_Err_Counter field) and Receive Error Counter (Rx_Err_Counter field). The rules for increasing and decreasing these counters are described in the CAN protocol and are completely implemented in the FlexCAN module. Both counters are read only except in Freeze Mode, where they can be written by the CPU.

Writing to the Error Counter Register while in Freeze Mode is an indirect operation. The data is first written to an auxiliary register and then an internal request/acknowledge procedure across clock domains is executed. All this is transparent to the user, except for the fact that the data will take some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.

FlexCAN responds to any bus state as described in the protocol, e.g. transmit ‘Error Active’ or ‘Error Passive’ flag, delay its transmission start time (‘Error Passive’) and avoid any influence on the bus when in ‘Bus Off’ state. The following are the basic rules for FlexCAN bus state transitions.

- If the value of Tx_Err_Counter or Rx_Err_Counter increases to be greater than or equal to 128, the FLT_CONF field in the Error and Status Register is updated to reflect ‘Error Passive’ state.
- If the FlexCAN state is ‘Error Passive’, and either Tx_Err_Counter or Rx_Err_Counter decrements to a value less than or equal to 127 while the other already satisfies this condition, the FLT_CONF field in the Error and Status Register is updated to reflect ‘Error Active’ state.
- If the value of Tx_Err_Counter increases to be greater than 255, the FLT_CONF field in the Error and Status Register is updated to reflect ‘Bus Off’ state, and an interrupt may be issued. The value of Tx_Err_Counter is then reset to zero.
- If FlexCAN is in ‘Bus Off’ state, then Tx_Err_Counter is cascaded together with another internal counter to count the 128th occurrences of 11 consecutive recessive bits on the bus. Hence, Tx_Err_Counter is reset to zero and counts in a manner where the internal counter counts 11 such bits and then wraps around while incrementing the Tx_Err_Counter. When Tx_Err_Counter reaches the value of 128, the FLT_CONF field in the Error and Status Register is updated to be ‘Error Active’ and both error counters are reset to zero. At any instance of dominant bit following

a stream of less than 11 consecutive recessive bits, the internal counter resets itself to zero without affecting the Tx_Err_Counter value.

- If during system start-up, only one node is operating, then its Tx_Err_Counter increases in each message it is trying to transmit, as a result of acknowledge errors (indicated by the ACK_ERR bit in the Error and Status Register). After the transition to ‘Error Passive’ state, the Tx_Err_Counter does not increment anymore by acknowledge errors. Therefore the device never goes to the ‘Bus Off’ state.
- If the Rx_Err_Counter increases to a value greater than 127, it is not incremented further, even if more errors are detected while being a receiver. At the next successful message reception, the counter is set to a value between 119 and 127 to resume to ‘Error Active’ state.

Base + 0x001C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	Rx_Err_Counter								Tx_Err_Counter							
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 25-9. Error Counter Register (FLEXCAN_x_ECR)

25.3.4.8 Error and Status Register (FLEXCAN_x_ESR)

This register reflects various error conditions, some general status of the device and it is the source of four interrupts to the CPU. The reported error conditions (bits 16-21) are those that occurred since the last time the CPU read this register. The CPU read action clears bits 16-23. Bits 22-28 are status bits.

Most bits in this register are read only, except TWRN_INT, RWRN_INT, BOFF_INT and ERR_INT, that are interrupt flags that can be cleared by writing ‘1’ to them (writing ‘0’ has no effect). See Section 25.4.10, “Interrupts”, for more details.

Base + 0x0020

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TWRN_INT	RWRN_INT
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	BIT1_ERR	BIT0_ERR	ACK_ERR	CRC_ERR	FRM_ERR	STF_ERR	TX_WRN	RX_WRN	IDLE	TXRX	FLT_CONF		0	BOFF_INT	ERR_INT	0
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨		▨			▨
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 25-10. Error and Status Register

Table 25-12. FLEXCAN_x_ESR Field Descriptions

Field	Description
0–13	Reserved
14 TWRN_INT	Tx Warning Interrupt Flag If the WRN_EN bit in FLEXCAN_x_MCR is asserted, the TWRN_INT bit is set when the TX_WRN flag transition from ‘0’ to ‘1’, meaning that the Tx error counter reached 96. If the corresponding mask bit in the Control Register (TWRN_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to ‘1’. Writing ‘0’ has no effect. 0 No such occurrence 1 The Tx error counter transition from < 96 to ≥ 96
15 RWRN_INT	Rx Warning Interrupt Flag If the WRN_EN bit in FLEXCAN_x_MCR is asserted, the RWRN_INT bit is set when the RX_WRN flag transition from ‘0’ to ‘1’, meaning that the Rx error counters reached 96. If the corresponding mask bit in the Control Register (RWRN_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to ‘1’. Writing ‘0’ has no effect. 0 No such occurrence 1 The Rx error counter transition from < 96 to ≥ 96
16 BIT1_ERR	Bit1 Error This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message. 0 No such occurrence 1 At least one bit sent as recessive is received as dominant **Note:** This bit is not set by a transmitter in case of arbitration field or ACK slot, or in case of a node sending a passive error flag that detects dominant bits.

Table 25-12. FLEXCAN_x_ESR Field Descriptions (continued)

Field	Description
17 BIT0_ERR	Bit0 Error This bit indicates when an inconsistency occurs between the transmitted and the received bit in a message. 0 No such occurrence 1 At least one bit sent as dominant is received as recessive
18 ACK_ERR	Acknowledge Error This bit indicates that an Acknowledge Error has been detected by the transmitter node, i.e., a dominant bit has not been detected during the ACK SLOT. 0 No such occurrence 1 An ACK error occurred since last read of this register
19 CRC_ERR	Cyclic Redundancy Check Error This bit indicates that a CRC Error has been detected by the receiver node, i.e., the calculated CRC is different from the received. 0 No such occurrence 1 A CRC error occurred since last read of this register.
20 FRM_ERR	Form Error This bit indicates that a Form Error has been detected by the receiver node, i.e., a fixed-form bit field contains at least one illegal bit. 0 No such occurrence 1 A Form Error occurred since last read of this register
21 STF_ERR	Stuffing Error This bit indicates that a Stuffing Error has been detected. 0 No such occurrence. 1 A Stuffing Error occurred since last read of this register.
22 TX_WRN	TX Error Counter This bit indicates when repetitive errors are occurring during message transmission. 0 No such occurrence 1 TX_Err_Counter ≥ 96
23 RX_WRN	Rx Error Counter This bit indicates when repetitive errors are occurring during message reception. 0 No such occurrence 1 Rx_Err_Counter ≥ 96
24 IDLE	CAN bus IDLE state This bit indicates when CAN bus is in IDLE state. 0 No such occurrence 1 CAN bus is now IDLE
25 TXRX	Current FlexCAN status (transmitting/receiving) This bit indicates if FlexCAN is transmitting or receiving a message when the CAN bus is not in IDLE state. This bit has no meaning when IDLE is asserted. 0 FlexCAN is receiving a message (IDLE=0) 1 FlexCAN is transmitting a message (IDLE=0)
26–27 FLT_CONF	Fault Confinement State This 2-bit field indicates the Confinement State of the FlexCAN module, as shown below. If the LOM bit in the Control Register is asserted, the FLT_CONF field will indicate “Error Passive”. Since the Control Register is not affected by soft reset, the FLT_CONF field will not be affected by soft reset if the LOM bit is asserted. 00 Error Active 01 Error Passive 1X Bus Off

Table 25-12. FLEXCAN_x_ESR Field Descriptions (continued)

Field	Description
28	Reserved
29 BOFF_INT	‘Bus Off’ Interrupt This bit is set when FlexCAN enters ‘Bus Off’ state. If the corresponding mask bit in the Control Register (BOFF_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to ‘1’. Writing ‘0’ has no effect. 0 No such occurrence 1 FlexCAN module entered ‘Bus Off’ state
30 ERR_INT	Error Interrupt This bit indicates that at least one of the Error Bits (bits 16-21) is set. If the corresponding mask bit in the Control Register (ERR_MSK) is set, an interrupt is generated to the CPU. This bit is cleared by writing it to ‘1’.Writing ‘0’ has no effect. 0 No such occurrence 1 Indicates setting of any Error Bit in the Error and Status Register
31	Reserved

25.3.4.9 Interrupt Masks 2 Register (FLEXCAN_x_IMASK2)

This register allows any number of a range of 32 Message Buffer Interrupts to be enabled or disabled. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (i.e. when the corresponding FLEXCAN_x_IFLAG2 bit is set).

Base + 0x0024

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 63M	BUF 62M	BUF 61M	BUF 60M	BUF 59M	BUF 58M	BUF 57M	BUF 56M	BUF 55M	BUF 54M	BUF 53M	BUF 52M	BUF 51M	BUF 50M	BUF 49M	BUF 48M
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 47M	BUF 46M	BUF 45M	BUF 44M	BUF 43M	BUF 42M	BUF 41M	BUF 40M	BUF 39M	BUF 38M	BUF 37M	BUF 36M	BUF 35M	BUF 34M	BUF 33M	BUF 32M
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-11. Interrupt Masks 2 Register (FLEXCAN_x_IMASK2)

Table 25-13. FLEXCAN_x_IMASK2 Field Descriptions

Field	Description
0–31 BUF63M–BUF32M	Buffer MB$_i$ Mask Each bit enables or disables the respective FlexCAN Message Buffer (MB32 to MB63) Interrupt. 0 The corresponding buffer Interrupt is disabled 1 The corresponding buffer Interrupt is enabled **Note:** Setting or clearing a bit in the FLEXCAN_x_IMASK2 Register can assert or negate an interrupt request, if the corresponding FLEXCAN_x_IFLAG2 bit is set.

25.3.4.10 Interrupt Masks 1 Register (FLEXCAN_x_IMASK1)

This register allows to enable or disable any number of a range of 32 Message Buffer Interrupts. It contains one interrupt mask bit per buffer, enabling the CPU to determine which buffer generates an interrupt after a successful transmission or reception (i.e., when the corresponding FLEXCAN_x_IFLAG1 bit is set).

Base + 0x0028

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 31M	BUF 30M	BUF 29M	BUF 28M	BUF 27M	BUF 26M	BUF 25M	BUF 24M	BUF 23M	BUF 22M	BUF 21M	BUF 20M	BUF 19M	BUF 18M	BUF 17M	BUF 16M
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 15M	BUF 14M	BUF 13M	BUF 12M	BUF 11M	BUF 10M	BUF 9M	BUF 8M	BUF 7M	BUF 6M	BUF 5M	BUF 4M	BUF 3M	BUF 2M	BUF 1M	BUF 0M
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-12. Interrupt Masks 1 Register (FLEXCAN_x_IMASK1)

Table 25-14. FLEXCAN_x_IMASK1 Field Descriptions

Field	Description
0–31 BUF31M –BUF0M	Buffer MB$_i$ Mask Each bit enables or disables the respective FlexCAN Message Buffer (MB0 to MB31) Interrupt. 0 The corresponding buffer Interrupt is disabled 1 The corresponding buffer Interrupt is enabled **Note:** Setting or clearing a bit in the FLEXCAN_x_IMASK1 Register can assert or negate an interrupt request, if the corresponding FLEXCAN_x_IFLAG1 bit is set.

25.3.4.11 Interrupt Flags 2 Register (FLEXCAN_x_IFLAG2)

This register defines the flags for 32 Message Buffer interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding FLEXCAN_x_IFLAG2 bit. If the corresponding FLEXCAN_x_IMASK2 bit is set, an interrupt will be generated. The interrupt flag must be cleared by writing it to ‘1’. Writing ‘0’ has no effect.

When the AEN bit in the FLEXCAN_x_MCR is set (Abort enabled), while the FLEXCAN_x_IFLAG2 bit is set for a MB configured as Tx, the writing access done by CPU into the corresponding MB will be blocked.

Base + 0x002C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 63I	BUF 62I	BUF 61I	BUF 60I	BUF 59I	BUF 58I	BUF 57I	BUF 56I	BUF 55I	BUF 54I	BUF 53I	BUF 52I	BUF 51I	BUF 50I	BUF 49I	BUF 48I
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 47I	BUF 46I	BUF 45I	BUF 44I	BUF 43I	BUF 42I	BUF 41I	BUF 40I	BUF 39I	BUF 38I	BUF 37I	BUF 36I	BUF 35I	BUF 34I	BUF 33I	BUF 32I
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-13. Interrupt Flags 2 Register (FLEXCAN_x_IFLAG2)

Table 25-15. FLEXCAN_x_IFLAG2 Field Descriptions

Field	Description
0–31 BUF32I– BUF63I	Buffer MB$_i$ Interrupt Each bit flags the respective FlexCAN Message Buffer (MB32 to MB63) interrupt. 0 No such occurrence 1 The corresponding buffer has successfully completed transmission or reception

25.3.4.12 Interrupt Flags 1 Register (FLEXCAN_x_IFLAG1)

This register defines the flags for 32 Message Buffer interrupts and FIFO interrupts. It contains one interrupt flag bit per buffer. Each successful transmission or reception sets the corresponding FLEXCAN_x_IFLAG1 bit. If the corresponding FLEXCAN_x_IMASK1 bit is set, an interrupt will be generated. The Interrupt flag must be cleared by writing it to ‘1’. Writing ‘0’ has no effect.

When the AEN bit in the FLEXCAN_x_MCR is set (Abort enabled), while the FLEXCAN_x_IFLAG1 bit is set for a MB configured as Tx, the writing access done by CPU into the corresponding MB will be blocked.

When the FEN bit in the FLEXCAN_x_MCR is set (FIFO enabled), the function of the 8 least significant interrupt flags (BUF7I - BUF0I) is changed to support the FIFO operation. BUF7I, BUF6I and BUF5I indicate operating conditions of the FIFO, while BUF4I to BUF0I are not used.

Base + 0x0030

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	BUF 31I	BUF 30I	BUF 29I	BUF 28I	BUF 27I	BUF 26I	BUF 25I	BUF 24I	BUF 23I	BUF 22I	BUF 21I	BUF 20I	BUF 19I	BUF 18I	BUF 17I	BUF 16I
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R W	BUF 15I	BUF 14I	BUF 13I	BUF 12I	BUF 11I	BUF 10I	BUF 9I	BUF 8I	BUF 7I	BUF 6I	BUF 5I	BUF 4I	BUF 3I	BUF 2I	BUF 1I	BUF 0I
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-14. Interrupt Flags 1 Register (FLEXCAN_x_IFLAG1)

Table 25-16. FLEXCAN_x_IFLAG1 Field Descriptions

Field	Description
0–23 BUF31I–BUF8I	Buffer MB_i Interrupt Each bit flags the respective FlexCAN Message Buffer (MB8 to MB31) interrupt. 0 No such occurrence 1 The corresponding MB has successfully completed transmission or reception

Table 25-16. FLEXCAN_x_IFLAG1 Field Descriptions (continued)

Field	Description
24 BUF7I	Buffer MB7 Interrupt or “FIFO Overflow” If the FIFO is not enabled, this bit flags the interrupt for MB7. If the FIFO is enabled, this flag indicates an overflow condition in the FIFO (frame lost because FIFO is full). 0 No such occurrence 1 MB7 completed transmission/reception or FIFO overflow
25 BUF6I	Buffer MB6 Interrupt or “FIFO Warning” If the FIFO is not enabled, this bit flags the interrupt for MB6. If the FIFO is enabled, this flag indicates that 5 out of 6 buffers of the FIFO are already occupied (FIFO almost full). 0 No such occurrence 1 MB6 completed transmission/reception or FIFO almost full
26 BUF5I	Buffer MB5 Interrupt or “Frames available in FIFO” If the FIFO is not enabled, this bit flags the interrupt for MB5. If the FIFO is enabled, this flag indicates that at least one frame is available to be read from the FIFO. 0 No such occurrence 1 MB5 completed transmission/reception or frames available in the FIFO
27–31 BUF4I–BUF0I	Buffer MB_i Interrupt or “reserved” If the FIFO is not enabled, these bits flag the interrupts for MB0 to MB4. If the FIFO is enabled, these flags are not used and must be considered as reserved locations. 0 No such occurrence 1 Corresponding MB completed transmission/reception

25.3.4.13 Rx Individual Mask Registers (RXIMR0–RXIMR63)

These registers are used as acceptance masks for ID filtering in Rx MBs and the FIFO. If the FIFO is not enabled, one mask register is provided for each available Message Buffer, providing ID masking capability on a per Message Buffer basis. When the FIFO is enabled (FEN bit in FLEXCAN_x_MCR is set), the first 8 Mask Registers apply to the 8 elements of the FIFO filter table (on a one-to-one correspondence), while the rest of the registers apply to the regular MBs, starting from MB8.

> The Individual Rx Mask Registers are implemented in RAM, so they are not affected by reset and must be explicitly initialized prior to any reception. Furthermore, they can only be accessed by the CPU while the module is in Freeze Mode. Out of Freeze Mode, write accesses are blocked and read accesses will return “all zeros”. Furthermore, if the MBFEN bit in the FLEXCAN_x_MCR Register is negated, any read or write operation to these registers results in access error.

Base + 0x0880–0x097F

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MI31	MI30	MI29	MI28	MI27	MI26	MI25	MI24	MI23	MI22	MI21	MI20	MI19	MI18	MI17	MI16

16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MI15	MI14	MI13	MI12	MI11	MI10	MI9	MI8	MI7	MI6	MI5	MI4	MI3	MI2	MI1	MI0

Figure 25-15. Rx Individual Mask Registers (RXIMR0 - RXIMR63)

Table 25-17. RXIMR0 — RXIMR63 Field Descriptions

Field	Description
0–31 MI31–MI0	Mask Bits For normal Rx MBs, the mask bits affect the ID filter programmed on the MB. For the Rx FIFO, the mask bits affect all bits programmed in the filter table (ID, IDE, RTR). 0 The corresponding bit in the filter is “don’t care” 1 The corresponding bit in the filter is checked against the one received\

25.4 Functional Description

25.4.1 Overview

The FlexCAN module is a CAN protocol engine with a very flexible mailbox system for transmitting and receiving CAN frames. The mailbox system is composed by a set of 64 Message Buffers (MB) that store configuration and control data, time stamp, message ID and data (see Section 25.3.2, “Message Buffer Structure”). The memory corresponding to the first 8 MBs can be configured to support a FIFO reception scheme with a powerful ID filtering mechanism, capable of checking incoming frames against a table of IDs (up to 8 extended IDs or 16 standard IDs or 32 8-bit ID slices), each one with its own individual mask register. Simultaneous reception through FIFO and mailbox is supported. For mailbox reception, a matching algorithm makes it possible to store received frames only into MBs that have the same ID programmed on its ID field. A masking scheme makes it possible to match the ID programmed on the MB with a range of IDs on received CAN frames. For transmission, an arbitration algorithm decides the prioritization of MBs to be transmitted based on the message ID (optionally augmented by 3 local priority bits) or the MB ordering.

Before proceeding with the functional description, an important concept must be explained. A Message Buffer is said to be “active” at a given time if it can participate in the matching and arbitration algorithms that are happening at that time. An Rx MB with a ‘0000’ code is inactive (refer to Table 25-5). Similarly, a Tx MB with a ‘1000’ or ‘1001’ code is also inactive (refer to Table 25-6). An MB not programmed with ‘0000’, ‘1000’ or ‘1001’ will be temporarily deactivated (will not participate in the current arbitration or matching run) when the CPU writes to the C/S field of that MB (see Section 25.4.6.2, “Message Buffer Deactivation”).

25.4.2 Transmit Process

In order to transmit a CAN frame, the CPU must prepare a Message Buffer for transmission by executing the following procedure:

- If the MB is active (transmission pending), write ‘1000’ to the Code field to deactivate the MB. The deactivated MB can transmit without setting IFLAG and without updating the CODE field (see Section 25.4.6.2, “Message Buffer Deactivation”).
- Write the ID word.
- Write the data bytes.
- Write the Length, Control and Code fields of the Control and Status word to activate the MB.

Once the MB is activated in the fourth step, it will participate into the arbitration process and eventually be transmitted according to its priority. At the end of the successful transmission, the value of the Free Running Timer is written into the Time Stamp field, the Code field in the Control and Status word is updated, a status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding Interrupt Mask Register bit. The new Code field after transmission depends on the code that was used to activate the MB in step four (see Table 25-5 and Table 25-6 in Section 25.3.2, “Message Buffer Structure”). When the Abort feature is enabled (AEN in FLEXCAN_x_MCR is asserted), after the Interrupt Flag is asserted for a MB configured as transmit buffer, the MB is blocked, therefore the CPU is not able to update it until the Interrupt Flag be negated by CPU. It means that the CPU must clear the corresponding IFLAG before starting to prepare this MB for a new transmission or reception.

25.4.3 Arbitration process

The arbitration process is an algorithm executed by the MBM that scans the whole MB memory looking for the highest priority message to be transmitted. All MBs programmed as transmit buffers will be scanned to find the lowest ID[1] or the lowest MB number or the highest priority, depending on the LBUF and LPRIO_EN bits on the Control Register. The arbitration process is triggered in the following events:

- During the CRC field of the CAN frame

---

[1] Actually, if LBUF is negated, the arbitration considers not only the ID, but also the RTR and IDE bits placed inside the ID at the same positions they are transmitted in the CAN frame.

- During the error delimiter field of the CAN frame
- During Intermission, if the winner MB defined in a previous arbitration was deactivated, or if there was no MB to transmit, but the CPU wrote to the C/S word of any MB after the previous arbitration finished
- When MBM is in Idle or Bus Off state and the CPU writes to the C/S word of any MB
- Upon leaving Freeze Mode

When LBUF is asserted, the LPRIO_EN bit has no effect and the lowest number buffer is transmitted first. When LBUF and LPRIO_EN are both negated, the MB with the lowest ID is transmitted first but. If LBUF is negated and LPRIO_EN is asserted, the PRIO bits augment the ID used during the arbitration process. With this extended ID concept, arbitration is done based on the full 32-bit ID and the PRIO bits define which MB should be transmitted first, therefore MBs with PRIO = 000 have higher priority. If two or more MBs have the same priority, the regular ID will determine the priority of transmission. If two or more MBs have the same priority (3 extra bits) and the same regular ID, the lowest MB will be transmitted first.

Once the highest priority MB is selected, it is transferred to a temporary storage space called Serial Message Buffer (SMB), which has the same structure as a normal MB but is not user accessible. This operation is called “move-out” and after it is done, write access to the corresponding MB is blocked (if the AEN bit in FLEXCAN_x_MCR is asserted). The write access is released in the following events:

- After the MB is transmitted
- FlexCAN enters in HALT or BUS OFF
- FlexCAN loses the bus arbitration or there is an error during the transmission

At the first opportunity window on the CAN bus, the message on the SMB is transmitted according to the CAN protocol rules. FlexCAN transmits up to eight data bytes, even if the DLC (Data Length Code) value is bigger.

25.4.4 Receive Process

To be able to receive CAN frames into the mailbox MBs, the CPU must prepare one or more Message Buffers for reception by executing the following steps:

- If the MB has a pending transmission, write an ABORT code (‘1001’) to the Code field of the Control and Status word to request an abortion of the transmission, then read back the Code field and the IFLAG register to check if the transmission was aborted (see Section 25.4.6.1, “Transmission Abort Mechanism”). If backwards compatibility is desired (AEN in FLEXCAN_x_MCR negated), just write ‘1000’ to the Code field to inactivate the MB, but then the pending frame may be transmitted without notification (see Section 25.4.6.2, “Message Buffer Deactivation”). If the MB already programmed as a receiver, just write ‘0000’ to the Code field of the Control and Status word to keep the MB inactive.
- Write the ID word
- Write ‘0100’ to the Code field of the Control and Status word to activate the MB

Once the MB is activated in the third step, it will be able to receive frames that match the programmed ID. At the end of a successful reception, the MB is updated by the MBM as follows:

- The value of the Free Running Timer is written into the Time Stamp field
- The received ID, Data (8 bytes at most) and Length fields are stored
- The Code field in the Control and Status word is updated (see Table 25-5 and Table 25-6 in Section 25.3.2, “Message Buffer Structure”)
- A status flag is set in the Interrupt Flag Register and an interrupt is generated if allowed by the corresponding Interrupt Mask Register bit

Upon receiving the MB interrupt, the CPU should service the received frame using the following procedure:

- Read the Control and Status word (mandatory – activates an internal lock for this buffer)
- Read the ID field (optional – needed only if a mask was used)
- Read the Data field
- Read the Free Running Timer (optional – releases the internal lock)

Upon reading the Control and Status word, if the BUSY bit is set in the Code field, then the CPU should defer the access to the MB until this bit is negated. Reading the Free Running Timer is not mandatory. If not executed the MB remains locked, unless the CPU reads the C/S word of another MB. Note that only a single MB is locked at a time. The only mandatory CPU read operation is the one on the Control and Status word to assure data coherency (see Section 25.4.6, “Data Coherence”).

The CPU should synchronize to frame reception by the status flag bit for the specific MB in one of the IFLAG Registers and not by the Code field of that MB. Polling the Code field does not work because once a frame was received and the CPU services the MB (by reading the C/S word followed by unlocking the MB), the Code field will not return to EMPTY. It will remain FULL, as explained in Table 25-5. If the CPU tries to workaround this behavior by writing to the C/S word to force an EMPTY code after reading the MB, the MB is actually deactivated from any currently ongoing matching process. As a result, a newly received frame matching the ID of that MB may be lost. In summary: *never do polling by reading directly the C/S word of the MBs. Instead, read the IFLAG registers.*

Note that the received ID field is always stored in the matching MB, thus the contents of the ID field in an MB may change if the match was due to masking. Note also that FlexCAN does receive frames transmitted by itself if there exists an Rx matching MB, provided the SRX_DIS bit in the FLEXCAN_x_MCR is not asserted. If SRX_DIS is asserted, FlexCAN will not store frames transmitted by itself in any MB, even if it contains a matching MB, and no interrupt flag or interrupt signal will be generated due to the frame reception.

To be able to receive CAN frames through the FIFO, the CPU must enable and configure the FIFO during Freeze Mode (see Section 25.4.7, “Rx FIFO”). Upon receiving the frames available interrupt from FIFO, the CPU should service the received frame using the following procedure:

- Read the Control and Status word (optional – needed only if a mask was used for IDE and RTR bits)

- Read the ID field (optional – needed only if a mask was used)
- Read the Data field
- Clear the frames available interrupt (mandatory – release the buffer and allow the CPU to read the next FIFO entry)

25.4.5 Matching Process

The matching process is an algorithm executed by the MBM that scans the MB memory looking for Rx MBs programmed with the same ID as the one received from the CAN bus. If the FIFO is enabled, the 8-entry ID table from FIFO is scanned first and then, if a match is not found within the FIFO table, the other MBs are scanned. In the event that the FIFO is full, the matching algorithm will always look for a matching MB outside the FIFO region.

When the frame is received, it is temporarily stored in a hidden auxiliary MB called Serial Message Buffer (SMB). The matching process takes place during the CRC field of the received frame. If a matching ID is found in the FIFO table or in one of the regular MBs, the contents of the SMB will be transferred to the FIFO or to the matched MB during the 6th bit of the End-Of-Frame field of the CAN protocol. This operation is called “move-in”. If any protocol error (CRC, ACK, etc.) is detected, than the move-in operation does not happen.

For the regular mailbox MBs, an MB is said to be “free to receive” a new frame if the following conditions are satisfied:

- The MB is not locked (see Section 25.4.6.3, “Message Buffer Lock Mechanism”)
- The Code field is either EMPTY or else it is FULL or OVERRUN but the CPU has already serviced the MB (read the C/S word and then unlocked the MB)

If the first MB with a matching ID is not “free to receive” the new frame, then the matching algorithm keeps looking for another free MB until it finds one. If it can not find one that is free, then it will overwrite the last matching MB (unless it is locked) and set the Code field to OVERRUN (refer to Table 25-5 and Table 25-6). If the last matching MB is locked, then the new message remains in the SMB, waiting for the MB to be unlocked (see Section 25.4.6.3, “Message Buffer Lock Mechanism”).

Suppose, for example, that the FIFO is disabled and there are two MBs with the same ID, and FlexCAN starts receiving messages with that ID. Let us say that these MBs are the second and the fifth in the array. When the first message arrives, the matching algorithm will find the first match in MB number 2. The code of this MB is EMPTY, so the message is stored there. When the second message arrives, the matching algorithm will find MB number 2 again, but it is not “free to receive”, so it will keep looking and find MB number 5 and store the message there. If yet another message with the same ID arrives, the matching algorithm finds out that there are no matching MBs that are “free to receive”, so it decides to overwrite the last matched MB, which is number 5. In doing so, it sets the Code field of the MB to indicate OVERRUN.

The ability to match the same ID in more than one MB can be exploited to implement a reception queue (in addition to the full featured FIFO) to allow more time for the CPU to service the MBs. By programming more than one MB with the same ID, received messages will be queued into the MBs. The CPU can examine the Time Stamp field of the MBs to determine the order in which the messages arrived.

The matching algorithm described above can be changed to be the same one used in previous versions of the FlexCAN module. When the MBFEN bit in FLEXCAN_x_MCR is negated, the matching algorithm stops at the first MB with a matching ID that it founds, whether this MB is free or not. As a result, the message queueing feature does not work if the MBFEN bit is negated.

Matching to a range of IDs is possible by using ID Acceptance Masks. FlexCAN supports individual masking per MB. Please refer to Section 25.3.4.13, "Rx Individual Mask Registers (RXIMR0–RXIMR63)"." During the matching algorithm, if a mask bit is asserted, then the corresponding ID bit is compared. If the mask bit is negated, the corresponding ID bit is "don't care". Please note that the Individual Mask Registers are implemented in RAM, so they are not initialized out of reset. Also, they can only be programmed if the MBFEN bit is asserted and while the module is in Freeze Mode.

FlexCAN also supports an alternate masking scheme with only three mask registers (RGXMASK, FLEXCAN_x_RX14MASK and FLEXCAN_x_RX15MASK) for backwards compatibility. This alternate masking scheme is enabled when the MBFEN bit in the FLEXCAN_x_MCR Register is negated.

25.4.6 Data Coherence

In order to maintain data coherency and FlexCAN proper operation, the CPU must obey the rules described in Section 25.4.2, "Transmit Process" and Section 25.4.4, "Receive Process". Any form of CPU accessing an MB structure within FlexCAN other than those specified may cause FlexCAN to behave in an unpredictable way.

25.4.6.1 Transmission Abort Mechanism

The abort mechanism provides a safe way to request the abortion of a pending transmission. A feedback mechanism is provided to inform the CPU if the transmission was aborted or if the frame could not be aborted and was transmitted instead. In order to maintain backwards compatibility, the abort mechanism must be explicitly enabled by asserting the AEN bit in the FLEXCAN_x_MCR.

In order to abort a transmission, the CPU must write a specific abort code (1001) to the Code field of the Control and Status word. When the abort mechanism is enabled, the active MBs configured as transmission must be aborted first and then they may be updated. If the abort code is written to an MB that is currently being transmitted, or to an MB that was already loaded into the SMB for transmission, the write operation is blocked and the MB is not deactivated, but the abort request is captured and kept pending until one of the following conditions are satisfied:

- The module loses the bus arbitration
- There is an error during the transmission
- The module is put into Freeze Mode

If none of conditions above are reached, the MB is transmitted correctly, the interrupt flag is set in the IFLAG register and an interrupt to the CPU is generated (if enabled). The abort request is automatically cleared when the interrupt flag is set. In the other hand, if one of the above conditions is reached, the frame is not transmitted, therefore the abort code is written into the Code field, the interrupt flag is set in the IFLAG and an interrupt is (optionally) generated to the CPU.

If the CPU writes the abort code before the transmission begins internally, then the write operation is not blocked, therefore the MB is updated and no interrupt flag is set. In this way the CPU just needs to read the abort code to make sure the active MB was deactivated. Although the AEN bit is asserted and the CPU wrote the abort code, in this case the MB is deactivated and not aborted, because the transmission did not start yet. One MB is only aborted when the abort request is captured and kept pending until one of the previous conditions are satisfied.

The abort procedure can be summarized as follows:

- CPU writes 1001 into the code field of the C/S word
- CPU reads the CODE field and compares it to the value that was written
- If the CODE field that was read is different from the value that was written, the CPU must read the corresponding IFLAG to check if the frame was transmitted or it is being currently transmitted. If the corresponding IFLAG is set, the frame was transmitted. If the corresponding IFLAG is reset, the CPU must wait for it to be set, and then the CPU must read the CODE field to check if the MB was aborted (CODE=1001) or it was transmitted (CODE=1000).

NOTE

An abort request to a TxMB can block any write operation into its CODE field. Therefore, the TxMB cannot be aborted or deactivated until it completes a transmission by winning the CAN bus arbitration.

25.4.6.2 Message Buffer Deactivation

Deactivation is mechanism provided to maintain data coherence when the CPU writes to the Control and Status word of active MBs out of Freeze Mode. Any CPU write access to the Control and Status word of an MB causes that MB to be excluded from the transmit or receive processes during the current matching or arbitration round. The deactivation is temporary, affecting only for the current match/arbitration round.

The purpose of deactivation is data coherency. The match/arbitration process scans the MBs to decide which MB to transmit or receive. If the CPU updates the MB in the middle of a match or arbitration process, the data of that MB may no longer be coherent, therefore deactivation of that MB is done.

Even with the coherence mechanism described above, writing to the Control and Status word of active MBs when not in Freeze Mode may produce undesirable results. Examples are:

- Matching and arbitration are one-pass processes. If MBs are deactivated after they are scanned, no re-evaluation is done to determine a new match/winner. If an Rx MB with a matching ID is deactivated during the matching process after it was scanned, then this MB is marked as invalid to receive the frame, and FlexCAN will keep looking for another matching MB within the ones it has not scanned yet. If it can not find one, then the message will be lost. Suppose, for example, that two MBs have a matching ID to a received frame, and the user deactivated the first matching MB after FlexCAN has scanned the second. The received frame will be lost even if the second matching MB was “free to receive”.
- If a Tx MB containing the lowest ID is deactivated after FlexCAN has scanned it, then FlexCAN will look for another winner within the MBs that it has not scanned yet. Therefore, it may transmit

an MB with ID that may not be the lowest at the time because a lower ID might be present in one of the MBs that it had already scanned before the deactivation.

- There is a point in time until which the deactivation of a Tx MB causes it not to be transmitted (end of move-out). After this point, it is transmitted but no interrupt is issued and the Code field is not updated. In order to avoid this situation, the abort procedures described in Section 25.4.6.1, "Transmission Abort Mechanism", should be used.

25.4.6.3 Message Buffer Lock Mechanism

Besides MB deactivation, FlexCAN has another data coherence mechanism for the receive process. When the CPU reads the Control and Status word of an "active not empty" Rx MB, FlexCAN assumes that the CPU wants to read the whole MB in an atomic operation, and thus it sets an internal lock flag for that MB. The lock is released when the CPU reads the Free Running Timer (global unlock operation), or when it reads the Control and Status word of another MB. The MB locking is done to prevent a new frame to be written into the MB while the CPU is reading it.

NOTE

The locking mechanism only applies to Rx MBs which have a code different than INACTIVE ('0000') or EMPTY[1] ('0100'). Also, Tx MBs can not be locked.

Suppose, for example, that the FIFO is disabled and the second and the fifth MBs of the array are programmed with the same ID, and FlexCAN has already received and stored messages into these two MBs. Suppose now that the CPU decides to read MB number 5 and at the same time another message with the same ID is arriving. When the CPU reads the Control and Status word of MB number 5, this MB is locked. The new message arrives and the matching algorithm finds out that there are no "free to receive" MBs, so it decides to override MB number 5. However, this MB is locked, so the new message can not be written there. It will remain in the SMB waiting for the MB to be unlocked, and only then will be written to the MB. If the MB is not unlocked in time and yet another new message with the same ID arrives, then the new message overwrites the one on the SMB and there will be no indication of lost messages either in the Code field of the MB or in the Error and Status Register.

While the message is being moved-in from the SMB to the MB, the BUSY bit on the Code field is asserted. If the CPU reads the Control and Status word and finds out that the BUSY bit is set, it should defer accessing the MB until the BUSY bit is negated.

NOTE

If the BUSY bit is asserted or if the MB is empty, then reading the Control and Status word does not lock the MB.

Deactivation takes precedence over locking. If the CPU deactivates a locked Rx MB, then its lock status is negated and the MB is marked as invalid for the current matching round. Any pending message on the SMB will not be transferred anymore to the MB.

---

[1] In previous FlexCAN versions, reading the C/S word locked the MB even if it was EMPTY. This behavior will be honoured when the MBFEN bit is negated.

25.4.7 Rx FIFO

The receive-only FIFO is enabled by asserting the FEN bit in the FLEXCAN_x_MCR. The reset value of this bit is zero to maintain software backwards compatibility with previous versions of the module that did not have the FIFO feature. When the FIFO is enabled, the memory region normally occupied by the first 8 MBs (0x80-0xFF) is now reserved for use of the FIFO engine (see Section 25.3.3, “Rx FIFO Structure”). Management of read and write pointers is done internally by the FIFO engine. The CPU can read the received frames sequentially, in the order they were received, by repeatedly accessing a Message Buffer structure at the beginning of the memory.

The FIFO can store up to 6 frames pending service by the CPU. An interrupt is sent to the CPU when new frames are available in the FIFO. Upon receiving the interrupt, the CPU must read the frame (accessing an MB in the 0x80 address) and then clear the interrupt. The act of clearing the interrupt triggers the FIFO engine to replace the MB in 0x80 with the next frame in the queue, and then issue another interrupt to the CPU. If the FIFO is full and more frames continue to be received, an OVERFLOW interrupt is issued to the CPU and subsequent frames are not accepted until the CPU creates space in the FIFO by reading one or more frames. A warning interrupt is also generated when 5 frames are accumulated in the FIFO.

A powerful filtering scheme is provided to accept only frames intended for the target application, thus reducing the interrupt servicing work load. The filtering criteria is specified by programming a table of 8 32-bit registers that can be configured to one of the following formats (see also Section 25.3.3, “Rx FIFO Structure”):

- Format A: 8 extended or standard IDs (including IDE and RTR)
- Format B: 16 standard IDs or 16 extended 14-bit ID slices (including IDE and RTR)
- Format C: 32 standard or extended 8-bit ID slices

NOTE

A chosen format is applied to all 8 registers of the filter table. It is not possible to mix formats within the table.

The eight elements of the filter table are individually affected by the first eight Individual Mask Registers (RXIMR0 - RXIMR7), allowing very powerful filtering criteria to be defined. The rest of the RXIMR, starting from RXIM8, continue to affect the regular MBs, starting from MB8. If the MBFEN bit is negated (or if the RXIMR are not available for the particular MCU), then the FIFO filter table is affected by the legacy mask registers as follows: element 6 is affected by FLEXCAN_x_RX14MASK, element 7 is affected by FLEXCAN_x_RX15MASK and the other elements (0 to 5) are affected by FLEXCAN_x_RXGMASK.

25.4.7.1 Precautions when using Global Mask and Individual Mask registers

Mask filtering alignment is affected based on the setting of the FEN and MBFEN of MCR. Table 25-18 table shows recommended actions depending on FEN and MBFEN settings.

Table 25-18. Recommended FEN and MBFEN settings

Case	MCR[FEN] RxFIFO	MCR[MBFEN] Rx Individual Mask	Notes
Case 1	FEN = 0	MBFEN = 0	RXGMASK, RX14MASK, and RX15MASK can safely be used. This allows backwards compatibility to older devices (e.g., devices without the individual masks feature). In this case, individual masks are not used.
Case 2	FEN = 1	MBFEN = 0	1st alternative: Do not use RXGMASK, RX14MASK, and RX15MASK in this case, leave the masks in their reset state.
Case 3	FEN = 1	MBFEN = 0	2nd alternative: Do not configure any MB as Rx (i.e., let all MBs as either Tx or inactive). In this case, RXGMASK, RX14MASK, and RX15MASK can be used to affect ID Tables without affecting the filtering process for Rx MBs.
Case 4	Don't care	MBFEN = 1	If MCR[MBFEN] = 1, then the RXIMRs are enabled. Thus, RXGMASK, RX14MASK, and RX15MASK are not used. Particularly, when MCR[FEN] = 0, RxFIFO is disabled; RXGMASK, RX14MASK, and RX15MASK do not affect filtering. Individual masks are used.

25.4.8 CAN protocol Related Features

25.4.8.1 Remote Frames

A remote frame is a special kind of frame. The user can program a MB to be a Request Remote frame by writing the MB as Transmit with the RTR bit set to ‘1’. After the Remote Request frame is transmitted successfully, the MB becomes a Receive Message Buffer, with the same ID as before.

When a Remote Request frame is received by the FlexCAN module, its ID is compared to the IDs of the transmit message buffers with the Code field ‘1010’. If there is a matching ID, then this MB frame is transmitted. Note that if the matching MB has the RTR bit set, then the FlexCAN module transmits a Remote Frame as a response.

A received Remote Request frame is not stored in a receive buffer. It is only used to trigger a transmission of a frame in response. The mask registers are not used in remote frame matching, and all ID bits (except RTR) of the incoming received frame should match.

In the case that a Remote Request frame was received and matched an MB, this message buffer immediately enters the internal arbitration process, but is considered as normal Tx MB, with no higher priority. The data length of this frame is independent of the DLC field in the remote frame that initiated its transmission.

25.4.8.2 Overload Frames

FlexCAN does transmit overload frames due to detection of following conditions on CAN bus:

- Detection of a dominant bit in the first/second bit of Intermission

- Detection of a dominant bit at the 7th bit (last) of End of Frame field (Rx frames)
- Detection of a dominant bit at the 8th bit (last) of Error Frame Delimiter or Overload Frame Delimiter

25.4.8.3 Time Stamp

The value of the Free Running Timer is sampled at the beginning of the Identifier field on the CAN bus, and is stored at the end of “move-in” in the TIME STAMP field, providing network behavior with respect to time.

Note that the Free Running Timer can be reset upon a specific frame reception, enabling network time synchronization. Refer to TSYN description in Section 25.3.4.2, “Control Register (FLEXCAN_x_CTRL)”.”

25.4.8.4 Protocol Timing

Figure 25-16 shows the structure of the clock generation circuitry that feeds the CAN Protocol Interface (CPI) sub-module. The clock source bit (CLK_SRC) in the FLEXCAN_x_CTRL Register defines whether the internal clock is connected to the output of a crystal oscillator (Oscillator Clock) or to the Peripheral (system) Clock from PLL). In order to guarantee reliable operation, the clock source should be selected while the module is in Disable Mode (bit MDIS set in the Module Configuration Register).

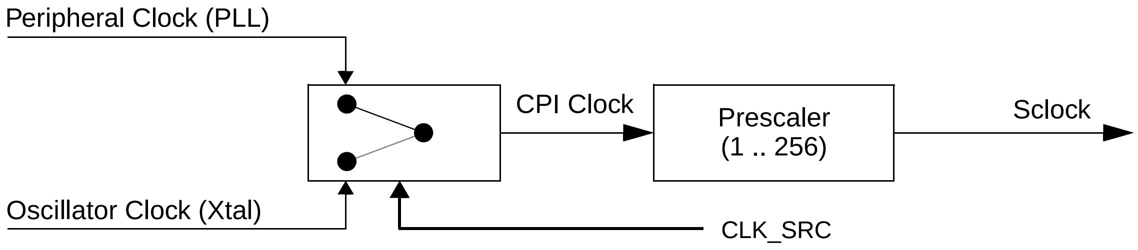


Figure 25-16. CAN Engine Clocking Scheme

The crystal oscillator clock should be selected whenever a tight tolerance (up to 0.1%) is required in the CAN bus timing. The crystal oscillator clock has better jitter performance than PLL generated clocks.

The FlexCAN module supports a variety of means to setup bit timing parameters that are required by the CAN protocol. The Control Register has various fields used to control bit timing parameters: PRESDIV, PROPSEG, PSEG1, PSEG2 and RJW. See Section 25.3.4.2, “Control Register (FLEXCAN_x_CTRL)”.”

The PRESDIV field controls a prescaler that generates the Serial Clock (Sclock), whose period defines the ‘time quantum’ used to compose the CAN waveform. A time quantum is the atomic unit of time handled by the CAN engine.

$$f_{Tq} = \frac{f_{CANCLK}}{(Prescaler \quad value)}$$

A bit time is subdivided into three segments[1] (reference Figure 25-17 and Table 25-19):

- SYNC_SEG: This segment has a fixed length of one time quantum. Signal edges are expected to happen within this section
- Time Segment 1: This segment includes the Propagation Segment and the Phase Segment 1 of the CAN standard. It can be programmed by setting the PROPSEG and the PSEG1 fields of the FLEXCAN_x_CTRL Register so that their sum (plus 2) is in the range of 4 to 16 time quanta
- Time Segment 2: This segment represents the Phase Segment 2 of the CAN standard. It can be programmed by setting the PSEG2 field of the FLEXCAN_x_CTRL Register (plus 1) to be 2 to 8 time quanta long

$$Bit \quad Rate = \frac{f_{Tq}}{(number \quad of \quad Time \quad Quanta)}$$

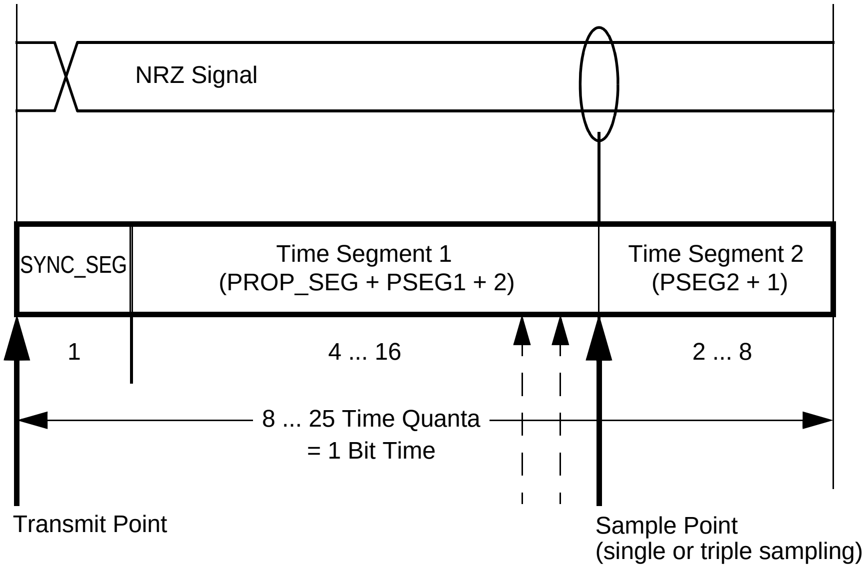


Figure 25-17. Segments within the Bit Time

Table 25-19. Time Segment Syntax

Syntax	Description
SYNC_SEG	System expects transitions to occur on the bus during this period.
Transmit Point	A node in transmit mode transfers a new value to the CAN bus at this point.
Sample Point	A node samples the bus at this point. If the three samples per bit option is selected, then this point marks the position of the third sample.

Table 25-20 gives an overview of the CAN compliant segment settings and the related parameter values.

---

[1] For further explanation of the underlying concepts please refer to ISO/DIS 11519–1, Section 10.3. Reference also the Bosch CAN 2.0A/B protocol specification dated September 1991 for bit timing.

Table 25-20. CAN Standard Compliant Bit Time Segment Settings

Time Segment 1	Time Segment 2	Re-synchronization Jump Width
5 .. 10	2	1 .. 2
4 .. 11	3	1 .. 3
5 .. 12	4	1 .. 4
6 .. 13	5	1 .. 4
7 .. 14	6	1 .. 4
8 .. 15	7	1 .. 4
9 .. 16	8	1 .. 4

NOTE

It is the user's responsibility to ensure the bit time settings are in compliance with the CAN standard. For bit time calculations, use an IPT (Information Processing Time) of 2, which is the value implemented in the FlexCAN module.

25.4.8.5 Arbitration and Matching Timing

During normal transmission or reception of frames, the arbitration, matching, move-in and move-out processes are executed during certain time windows inside the CAN frame, as shown in Figure 25-18.

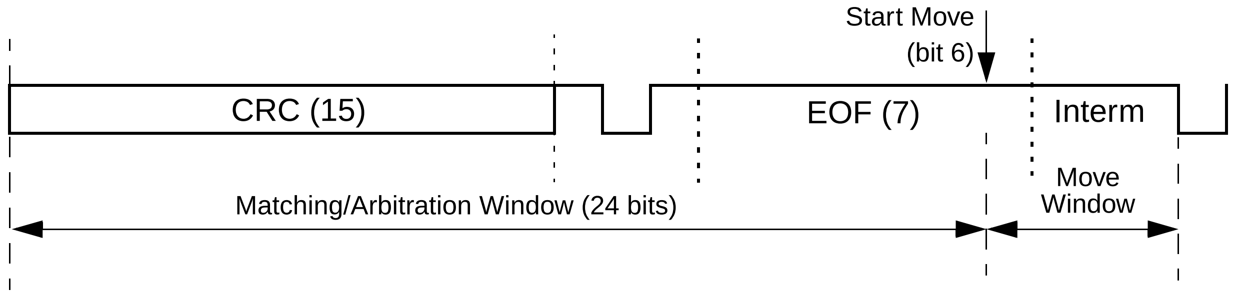


Figure 25-18. Arbitration, Match and Move Time Windows

When doing matching and arbitration, FlexCAN needs to scan the whole Message Buffer memory during the available time slot. In order to have sufficient time to do that, the following requirements must be observed:

- A valid CAN bit timing must be programmed, as indicated in Table 25-20.
- The peripheral clock frequency can not be smaller than the oscillator clock frequency, i.e. the PLL can not be programmed to divide down the oscillator clock.
- There must be a minimum ratio between the peripheral clock frequency and the CAN bit rate, as specified in Table 25-21.

Table 25-21. Minimum Ratio Between Peripheral Clock Frequency and CAN Bit Rate

Number of Message Buffers	Minimum Ratio
16	8
32	8
64	16

A direct consequence of the first requirement is that the minimum number of time quanta per CAN bit must be 8, so the oscillator clock frequency should be at least 8 times the CAN bit rate. The minimum frequency ratio specified in Table 25-21 can be achieved by choosing a high enough peripheral clock frequency when compared to the oscillator clock frequency, or by adjusting one or more of the bit timing parameters (PRESDIV, PROPSEG, PSEG1, PSEG2). As an example, taking the case of 64 MBs, if the oscillator and peripheral clock frequencies are equal and the CAN bit timing is programmed to have 8 time quanta per bit, then the prescaler factor (PRESDIV + 1) should be at least 2. For prescaler factor equal to one and CAN bit timing with 8 time quanta per bit, the ratio between peripheral and oscillator clock frequencies should be at least 2.

25.4.9 Modes of Operation Details

25.4.9.1 Freeze Mode

This mode is entered by asserting the HALT bit in the FLEXCAN_x_MCR Register or when the MCU is put into Debug Mode. In both cases it is also necessary that the FRZ bit is asserted in the FLEXCAN_x_MCR Register and the module is not in any of the low power modes (Disable, Stop). When Freeze Mode is requested during transmission or reception, FlexCAN does the following:

- Waits to be in either Intermission, Passive Error, Bus Off or Idle state
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores the Rx input pin and drives the Tx pin as recessive
- Stops the prescaler, thus halting all CAN protocol activities
- Grants write access to the Error Counters Register, which is read-only in other modes
- Sets the NOT_RDY and FRZ_ACK bits in FLEXCAN_x_MCR

After requesting Freeze Mode, the user must wait for the FRZ_ACK bit to be asserted in FLEXCAN_x_MCR before executing any other action, otherwise FlexCAN may operate in an unpredictable way. In Freeze mode, all memory mapped registers are accessible.

Exiting Freeze Mode is done in one of the following ways:

- CPU negates the FRZ bit in the FLEXCAN_x_MCR Register
- The MCU is removed from Debug Mode and/or the HALT bit is negated

Once out of Freeze Mode, FlexCAN tries to re-synchronize to the CAN bus by waiting for 11 consecutive recessive bits.

25.4.9.2 Module Disable Mode

This low power mode is entered when the MDIS bit in the FLEXCAN_x_MCR Register is asserted. If the FlexCAN module is disabled during Freeze Mode, the module sends a request to disable the clocks to the CAN Protocol Interface (CPI) and Message Buffer Management (MBM) sub-modules, sets the MDISACK bit and negates the FRZ_ACK bit. If the module is disabled during transmission or reception, FlexCAN does the following:

- Waits to be in either Idle or Bus Off state, or else waits for the third bit of Intermission and then checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Shuts down the clocks to the CPI and MBM sub-modules
- Sets the NOT_RDY and MDISACK bits in FLEXCAN_x_MCR

The Bus Interface Unit continues to operate, enabling the CPU to access memory mapped registers, except the Free Running Timer, the Error Counter Register and the Message Buffers, which cannot be accessed when the module is in Disable Mode. Exiting from this mode is done by negating the MDIS bit, which will resume the clocks and negate the MDISACK bit.

25.4.9.3 Stop Mode

This is a system low power mode in which all MCU clocks are stopped for maximum power savings. If FlexCAN receives the global Stop Mode request during Freeze Mode, it sets the MDISACK bit, negates the FRZ_ACK bit and then sends a Stop Acknowledge signal to the CPU, in order to shut down the clocks globally. If Stop Mode is requested during transmission or reception, FlexCAN does the following:

- Waits to be in either Idle or Bus Off state, or else waits for the third bit of Intermission and checks it to be recessive
- Waits for all internal activities like arbitration, matching, move-in and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive
- Sets the NOT_RDY and MDISACK bits in FLEXCAN_x_MCR
- Sends a Stop Acknowledge signal to the CPU, so that it can shut down the clocks globally

Exiting Stop Mode is done by CPU resuming the clocks and removing the Stop Mode request.

25.4.10 Interrupts

The module can generate up to 69 interrupt sources (64 interrupts due to message buffers and 5 interrupts due to ORed interrupts from MBs, Bus Off, Error, Tx Warning, and Rx Warning). The number of actual sources depends on the configured number of Message Buffers.

Each one of the message buffers can be an interrupt source, if its corresponding IMASK bit is set. There is no distinction between Tx and Rx interrupts for a particular buffer, under the assumption that the buffer is initialized for either transmission or reception. Each of the buffers has assigned a flag bit in the IFLAG

Registers. The bit is set when the corresponding buffer completes a successful transmission/reception and is cleared when the CPU writes it to ‘1’ (unless another interrupt is generated at the same time).

NOTE

It must be guaranteed that the CPU only clears the bit causing the current interrupt. For this reason, bit manipulation instructions (BSET) must not be used to clear interrupt flags. These instructions may cause accidental clearing of interrupt flags which are set after entering the current interrupt service routine.

If the Rx FIFO is enabled (bit FEN on FLEXCAN_x_MCR set), the interrupts corresponding to MBs 0 to 7 have a different behavior. Bit 7 of the FLEXCAN_x_IFLAG1 becomes the “FIFO Overflow” flag; bit 6 becomes the FIFO Warning flag, bit 5 becomes the “Frames Available in FIFO flag” and bits 4-0 are unused. See Section 25.3.4.12, “Interrupt Flags 1 Register (FLEXCAN_x_IFLAG1)”, for more information.

A combined interrupt for all MBs is also generated by an Or of all the interrupt sources from MBs. This interrupt gets generated when any of the MBs generates an interrupt. In this case the CPU must read the IFLAG Registers to determine which MB caused the interrupt.

The other 4 interrupt sources (Bus Off, Error, Tx Warning, and Rx Warning) generate interrupts like the MB ones, and can be read from the Error and Status Register. The Bus Off, Error, Tx Warning and Rx Warning interrupt mask bits are located in the Control Register.

25.4.11 Bus Interface

The CPU access to FlexCAN registers are subject to the following rules:

- Read and write access to supervisor registers in User Mode results in access error.
- Read and write access to unimplemented or reserved address space also results in access error. Any access to unimplemented MB or Rx Individual Mask Register locations results in access error. Any access to the Rx Individual Mask Register space when the MBFEN bit in FLEXCAN_x_MCR is negated results in access error.
- If MAXMB is programmed with a value smaller than the available number of MBs, then the unused memory space can be used as general purpose RAM space. Note that the Rx Individual Mask Registers can only be accessed in Freeze Mode, and this is still true for unused space within this memory. Note also that reserved words within RAM cannot be used. As an example, suppose FlexCAN is configured with 64 MBs and MAXMB is programmed with zero. The maximum number of MBs in this case becomes one. The MB memory starts at 0x0060, but the space from 0x0060 to 0x007F is reserved (for SMB usage), and the space from 0x0080 to 0x008F is used by the one MB. This leaves us with the available space from 0x0090 to 0x047F. The available memory in the Mask Registers space would be from 0x0884 to 0x097F.

NOTE

Unused MB space must not be used as general purpose RAM while FlexCAN is transmitting and receiving CAN frames.

25.5 Initialization/Application Information

This section provide instructions for initializing the FlexCAN module.

25.5.1 FlexCAN Initialization Sequence

The FlexCAN module may be reset in three ways:

- MCU level hard reset, which resets all memory mapped registers asynchronously
- MCU level soft reset, which resets some of the memory mapped registers synchronously (refer to Table 25-2 to see what registers are affected by soft reset)
- SOFT_RST bit in FLEXCAN_x_MCR, which has the same effect as the MCU level soft reset

Soft reset is synchronous and has to follow an internal request/acknowledge procedure across clock domains. Therefore, it may take some time to fully propagate its effects. The SOFT_RST bit remains asserted while soft reset is pending, so software can poll this bit to know when the reset has completed. Also, soft reset can not be applied while clocks are shut down in any of the low power modes. The low power mode should be exited and the clocks resumed before applying soft reset.

The clock source (CLK_SRC bit) should be selected while the module is in Disable Mode. After the clock source is selected and the module is enabled (MDIS bit negated), FlexCAN automatically goes to Freeze Mode. In Freeze Mode, FlexCAN is un-synchronized to the CAN bus, the HALT and FRZ bits in FLEXCAN_x_MCR Register are set, the internal state machines are disabled and the FRZ_ACK and NOT_RDY bits in the FLEXCAN_x_MCR Register are set. The Tx pin is in recessive state and FlexCAN does not initiate any transmission or reception of CAN frames. Note that the Message Buffers and the Rx Individual Mask Registers are not affected by reset, so they are not automatically initialized.

For any configuration change/initialization it is required that FlexCAN is put into Freeze Mode (see Section 25.4.9.1, “Freeze Mode”). The following is a generic initialization sequence applicable to the FlexCAN module:

- Initialize the Module Configuration Register
  - Enable the individual filtering per MB and reception queue features by setting the MBFEN bit
  - Enable the warning interrupts by setting the WRN_EN bit
  - If required, disable frame self reception by setting the SRX_DIS bit
  - Enable the FIFO by setting the FEN bit
  - Enable the abort mechanism by setting the AEN bit
  - Enable the local priority feature by setting the LPRIO_EN bit
- Initialize the Control Register
  - Determine the bit timing parameters: PROPSEG, PSEG1, PSEG2, RJW
  - Determine the bit rate by programming the PRESDIV field
  - Determine the internal arbitration mode (LBUF bit)

- Initialize the Message Buffers
  - The Control and Status word of all Message Buffers must be initialized
  - If FIFO was enabled, the 8-entry ID table must be initialized
  - Other entries in each Message Buffer should be initialized as required
- Initialize the Rx Individual Mask Registers
- Set required interrupt mask bits in the IMASK Registers (for all MB interrupts), in FLEXCAN_x_CTRL Register (for Bus Off and Error interrupts)
- Negate the HALT bit in FLEXCAN_x_MCR

Starting with the last event, FlexCAN attempts to synchronize to the CAN bus.

25.5.2 FlexCAN Addressing and RAM size configurations

There is 1 RAM configuration that can be implemented within the FlexCAN module. The possible configurations is:

- For 64 MBs: 1056 bytes for MB memory and 256 bytes for Individual Mask Registers

The user can program the maximum number of MBs that will take part in the matching and arbitration processes using the MAXMB field in the FLEXCAN_x_MCR Register. For 64 MB configuration, MAXMB can be any number between 0–63.

Chapter 26
FlexRay Module (FlexRay)

26.1 Introduction

26.1.1 Reference

The following documents are referenced.

- FlexRay Communications System Protocol Specification, Version 2.1 Rev A[1]
- FlexRay Communications System Electrical Physical Layer Specification, Version 2.1 Rev A

26.1.2 Glossary

This section provides a list of terms used in this chapter.

Table 26-1. List of Terms

Term	Definition
BCU	Buffer Control Unit. Handles message buffer access.
BMIF	Bus Master Interface. Provides master access to FlexRay memory area.
CC	Communication Controller
CDC	Clock Domain Crosser
CHI	Controller Host Interface
Cycle length in μT	The actual length of a cycle in μT for the ideal controller (±0 ppm)
EBI	External Bus Interface
FlexRay Memory Area	Memory area to store the physical message buffer payload data, frame header, frame and slot status, and synchronization frame related tables.
System Memory	Memory that is contains the FlexRay Memory Area.
System Bus	Bus that connects the controller and System Memory
FSS	Frame Start Sequence
HIF	Host Interface. Provides host access to controller.
Host	The FlexRay CC host MCU
LUT	Look Up Table. Stores message buffer header index value.
LRAM	Look Up Table RAM. Module internal memory to store message buffer configuration data and data field offsets for individual message buffers and receive shadow buffers.
MB	Message Buffer

---

[1] The FlexRay Specifications have been developed for automotive applications.The FlexRay Specifications have been neither developed nor tested for non-automotive applications.

Table 26-1. List of Terms (continued)

Term	Definition
MBIDX	Message Buffer Index: the position of a header field entry within the header area. If the header area is accessed as an array, this is the same as the array index of the entry.
MBNum	Message Buffer Number: Position of message buffer configuration registers within the register map. For example, Message Buffer Number 5 corresponds to the MBCCS5 register.
MCU	Microcontroller Unit
µT	Microtick
MT	Macrotick
MTS	Media Access Test Symbol
NIT	Network Idle Time
PE	Protocol Engine
POC	Protocol Operation Control. Each state of the POC is denoted by *POC:state*
Rx	Reception
SEQ	Sequencer Engine
TCU	Time Control Unit
Tx	Transmission
sync frame	null frame or message frame with *Sync Frame Indicator* set to 1
startup frame	null frame or message frame with both *Sync Frame Indicator* and *Startup Frame Indicator* set to 1
normal frame	null frame or message frame with both *Sync Frame Indicator* and *Startup Frame Indicator* set to 0
null frame	frame with *Null Frame Indicator* set to 0
message frame	frame with *Null Frame Indicator* set to 1

26.1.3 Color Coding

Throughout this chapter types of items are highlighted through the use of an italicized color font.

FlexRay protocol parameters, constants and variables are highlighted with *blue italics*. An example is the parameter *gdActionPointOffset*.

FlexRay protocol states are highlighted in *green italics*. An example is the state *POC:normal active*.

26.1.4 Overview

The CC is a FlexRay communication controller that implements the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*

The CC has three main components:

- Controller host interface (CHI)
- Protocol engine (PE)
- Clock domain crossing unit (CDC)

A block diagram of the CC with its surrounding modules is given in Figure 26-1.

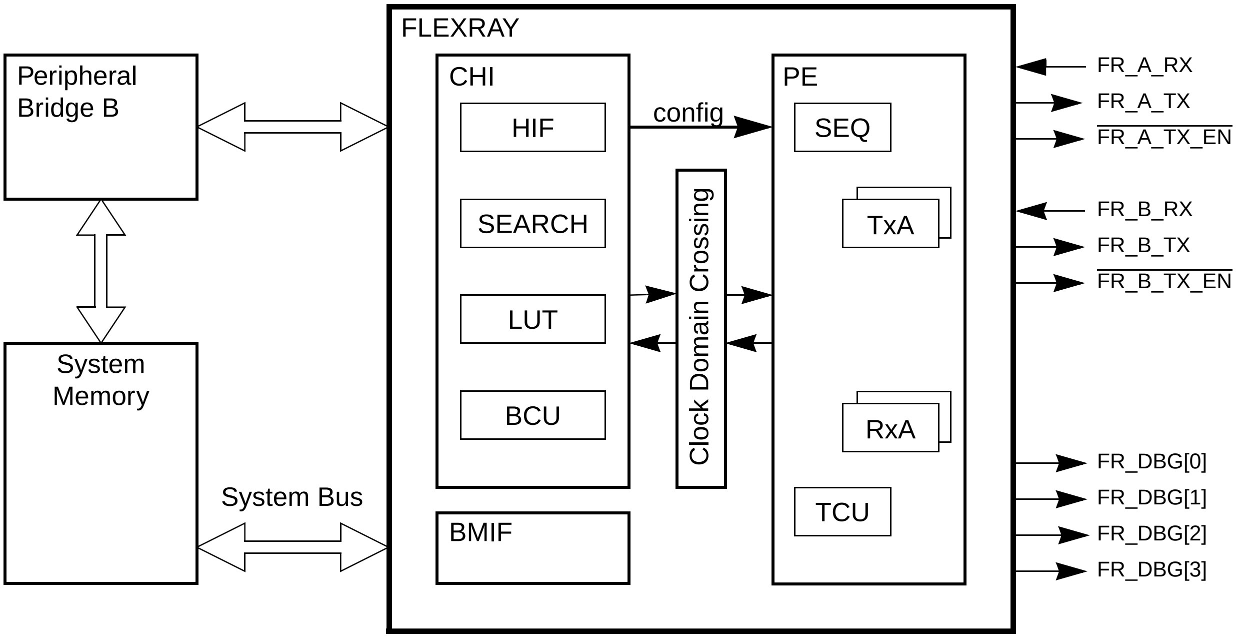


Figure 26-1. FLEXRAY Block Diagram

The protocol engine has two transmitter units TxA and TxB and two receiver units RxA and RxB for sending and receiving frames through the two FlexRay channels. The time control unit (TCU) is responsible for maintaining global clock synchronization to the FlexRay network. The overall activity of the PE is controlled by the sequencer engine (SEQ).

The controller host interface provides host access to the module's configuration, control, and status registers, as well as to the message buffer configuration, control, and status registers. The message buffers themselves, which contain the frame header and payload data received or to be transmitted, and the slot status information, are stored in the flexray memory area.

The clock domain crossing unit implements signal crossing from the CHI clock domain to the PE clock domain and vice versa, to allow for asynchronous PE and CHI clock domains.

The CC stores the frame header and payload data of frames received or of frames to be transmitted in the flexray memory area. The application accesses the flexray memory area to retrieve and provide the frames to be processed by the CC. In addition to the frame header and payload data, the CC stores the synchronization frame related tables in the flexray memory area for application processing.

The flexray memory area is located in the system memory of the MCU. The CC has access to the flexray memory area via its bus master interface (BMIF). The host provides the start address of the flexray memory area within the system memory by programming the System Memory Base Address Register (FR_SYMBADR)**.** All flexray memory area related offsets are stored in offset registers. The physical address pointer into the flexray memory area is calculated using the offset values the flexray memory base address.

NOTE

The CC does not provide a memory protection scheme for the flexray memory area.

26.1.5 Features

The CC provides the following features:

- *FlexRay Communications System Protocol Specification, Version 2.1 Rev A* compliant protocol implementation
- *FlexRay Communications System Electrical Physical Layer Specification, Version 2.1 Rev A* compliant bus driver interface
- single channel support
  - FlexRay Port A can be configured to be connected either to physical FlexRay channel A or physical FlexRay channel B.
- FlexRay bus data rates of 10 Mbit/s, 8 Mbit/s, 5 Mbit/s, and 2.5 Mbit/s supported
- 128 configurable message buffers with
  - individual frame ID filtering
  - individual channel ID filtering
  - individual cycle counter filtering
- message buffer header, status and payload data stored in dedicated flexray memory area
  - allows for flexible and efficient message buffer implementation
  - consistent data access ensured by means of buffer locking scheme
  - application can lock multiple buffers at the same time
- size of message buffer payload data section configurable from 0 up to 254 bytes
- two independent message buffer segments with configurable size of payload data section
  - each segment can contain message buffers assigned to the static segment and message buffers assigned to the dynamic segment at the same time
- zero padding for transmit message buffers in static segment
  - applied when the frame payload length exceeds the size of the message buffer data section
- transmit message buffers configurable with state/event semantics
- message buffers can be configured as
  - receive message buffer
  - transmit message buffer
- individual message buffer reconfiguration supported
  - means provided to safely disable individual message buffers
  - disabled message buffers can be reconfigured
- two independent receive FIFOs
  - one receive FIFO per channel
  - up to 255 entries for each FIFO

  - global frame ID filtering, based on both value/mask filters and range filters
  - global channel ID filtering
  - global message ID filtering for the dynamic segment
- 4 configurable slot error counters
- 4 dedicated slot status indicators
  - used to observe slots without using receive message buffers
- measured value indicators for the clock synchronization
  - internal synchronization frame ID and synchronization frame measurement tables can be copied into the flexray memory area
- fractional macroticks are supported for clock correction
- maskable interrupt sources provided via individual and combined interrupt lines
- 1 absolute timer
- 1 timer that can be configured to absolute or relative
- SECDED for protocol engine data ram
- SEDDED for chi lookup table ram

26.1.6 Modes of Operation

This section describes the basic operational power modes of the CC.

26.1.6.1 Disabled Mode

The CC enters the Disabled Mode during hard reset. The CC indicates that it is in the Disabled Mode by negating the module enable bit MEN in the Module Configuration Register (FR_MCR).

No communication is performed on the FlexRay bus.

All registers with the write access conditions *Any Time* and *Disabled Mode* can be accessed for writing as stated in Section 26.5.2, “Register Descriptions”.

The application configures the CC by accessing the configuration bits and fields in the Module Configuration Register (FR_MCR).

26.1.6.1.1 Leave Disabled Mode

The CC leaves the Disabled Mode and enters the Normal Mode, when the application writes 1 to the module enable bit MEN in the Module Configuration Register (FR_MCR)

NOTE

When the CC was enabled, it cannot be disabled the later on.

26.1.6.2 Normal Mode

In this mode the CC is fully functional. The CC indicates that it is in Normal Mode by asserting the module enable bit MEN in the Module Configuration Register (FR_MCR).

26.1.6.2.1 Enter Normal Mode

This mode is entered when the application requests the CC to leave the Disabled Mode. If the Normal Mode was entered by leaving the Disabled Mode, the application has to perform the protocol initialization described in 26.7.2.2, “Protocol Initialization” to achieve full FlexRay functionality.

Depending on the values of the SCM, CHA, and CHB bits in the Module Configuration Register (FR_MCR), the corresponding FlexRay bus driver ports are enabled and driven.

26.2 External Signal Description

This section lists and describes the CC signals, connected to external pins. These signals are summarized in Table 26-2 and described in detail in Section 26.2.1, “Detailed Signal Descriptions”.

NOTE

The off chip signals FR_A_RX, FR_A_TX, and $\overline{\mathrm{FR\_A\_TX\_EN}}$ are available on each package option. The availability of the other off chip signals depends on the package option.

Table 26-2. External Signal Properties

Name	Direction	Active	Reset	Function
FR_A_RX	Input	—	—	Receive Data Channel A
FR_A_TX	Output	—	1	Transmit Data Channel A
$\overline{\mathrm{FR\_A\_TX\_EN}}$	Output	Low	1	Transmit Enable Channel A
FR_B_RX	Input	—	—	Receive Data Channel B
FR_B_TX	Output	—	1	Transmit Data Channel B
$\overline{\mathrm{FR\_B\_TX\_EN}}$	Output	Low	1	Transmit Enable Channel B
FR_DBG[0]	Output	—	0	Debug Strobe Signal 0
FR_DBG[1]	Output	—	0	Debug Strobe Signal 1
FR_DBG[2]	Output	—	0	Debug Strobe Signal 2
FR_DBG[3]	Output	—	0	Debug Strobe Signal 3

26.2.1 Detailed Signal Descriptions

This section provides a detailed description of the CC signals, connected to external pins.

26.2.1.1 FR_A_RX — Receive Data Channel A

The FR_A_RX signal carries the receive data for channel A from the corresponding FlexRay bus driver.

26.2.1.2 FR_A_TX — Transmit Data Channel A

The FR_A_TX signal carries the transmit data for channel A to the corresponding FlexRay bus driver.

26.2.1.3 FR_A_TX_EN — Transmit Enable Channel A

The FR_A_TX_EN signal indicates to the FlexRay bus driver that the CC is attempting to transmit data on channel A.

26.2.1.4 FR_B_RX — Receive Data Channel B

The FR_B_RX signal carries the receive data for channel B from the corresponding FlexRay bus driver.

26.2.1.5 FR_B_TX — Transmit Data Channel B

The FR_B_TX signal carries the transmit data for channel B to the corresponding FlexRay bus driver

26.2.1.6 FR_B_TX_EN — Transmit Enable Channel B

The FR_B_TX_EN signal indicates to the FlexRay bus driver that the CC is attempting to transmit data on channel B.

26.2.1.7 FR_DBG[3], FR_DBG[2], FR_DBG[1], FR_DBG[0] — Strobe Signals

These signals provide the selected debug strobe signals. For details on the debug strobe signal selection refer to Section 26.6.16, “Strobe Signal Support”.

26.3 Controller Host Interface Clocking

The clock for the CHI is derived from the system bus clock and has the same phase and frequency as the system bus clock. There are two constraints for the minimum CHI clock frequency.

The first constraint corresponds to the number of utilized message buffers and is specified in Section 26.7.5, “Number of Usable Message Buffers”.

The second constraint corresponds to the value of the TIMEOUT field in the System Memory Access Time-Out Register (FR_SYMATOR) and is specified in Section 26.7.1.1, “Configure System Memory Access Time-Out Register (FR_SYMATOR)”.

26.4 Protocol Engine Clocking

The clock for the protocol engine can be generated by two sources. The first source is the internal crystal oscillator and the second source is an internal PLL. The clock source to be used is selected by the clock source select bit CLKSEL in the Module Configuration Register (FR_MCR).

26.4.1 Oscillator Clocking

If the protocol engine is clocked by the internal crystal oscillator, an 40 MHz crystal or CMOS compatible clock must be connected to the oscillator pins. The crystal or clock must fulfill the requirements given by the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.

26.4.2 PLL Clocking

If the protocol engine is clocked by the internal PLL, the frequency of the PE clock source is system clock / 4. The system clock frequency has to be 160 MHz.

26.5 Memory Map and Register Description

The CC occupies 8 KB (8192 bytes) of address space starting at the CC's base address defined by the memory map of the MCU.

26.5.1 Memory Map

The complete memory map of the CC is shown in Table 26-3. The addresses presented here are the offsets relative to the CC base address which is defined by the MCU address map.

Table 26-3. FlexRay Memory Map (Sheet 1 of 4)

Offset	Register	Access
	Module Configuration and Control	
0x0000	Module Version Register (FR_MVR)	R
0x0002	Module Configuration Register (FR_MCR)	R/W
0x0004	System Memory Base Address High Register (FR_SYMBADHR)	R/W
0x0006	System Memory Base Address Low Register (FR_SYMBADLR)	R/W
0x0008	Strobe Signal Control Register (FR_STBSCR)	R/W
0x000A	Reserved	R
0x000C	Message Buffer Data Size Register (FR_MBDSR)	R/W
0x000E	Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)	R/W
	PE Access Registers	
0x0010	PE DRAM Access Register (FR_PEDRAR)	R/W
0x0012	PE DRAM Data Register (FR_PEDRDR)	R/W
	Interrupt and Error Handling	
0x0014	Protocol Operation Control Register (FR_POCR)	R/W
0x0016	Global Interrupt Flag and Enable Register (FR_GIFER)	R/W
0x0018	Protocol Interrupt Flag Register 0 (FR_PIFR0)	R/W
0x001A	Protocol Interrupt Flag Register 1 (FR_PIFR1)	R/W
0x001C	Protocol Interrupt Enable Register 0 (FR_PIER0)	R/W
0x001E	Protocol Interrupt Enable Register 1 (FR_PIER1)	R/W
0x0020	CHI Error Flag Register (FR_CHIERFR)	R/W
0x0022	Message Buffer Interrupt Vector Register (FR_MBIVEC)	R
0x0024	Channel A Status Error Counter Register (FR_CASERCR)	R
0x0026	Channel B Status Error Counter Register (FR_CBSERCR)	R
	Protocol Status	
0x0028	Protocol Status Register 0 (FR_PSR0)	R
0x002A	Protocol Status Register 1 (FR_PSR1)	R
0x002C	Protocol Status Register 2 (FR_PSR2)	R

Table 26-3. FlexRay Memory Map (Sheet 1 of 4)

Offset	Register	Access
0x002E	Protocol Status Register 3 (FR_PSR3)	R/W
0x0030	Macrotick Counter Register (FR_MTCTR)	R
0x0032	Cycle Counter Register (FR_CYCTR)	R
0x0034	Slot Counter Channel A Register (FR_SLTCTAR)	R
0x0036	Slot Counter Channel B Register (FR_SLTCTBR)	R
0x0038	Rate Correction Value Register (FR_RTCORVR)	R
0x003A	Offset Correction Value Register (FR_OFCORVR)	R
0x003C	Combined Interrupt Flag Register (FR_CIFR)	R
0x003E	System Memory Access Time-Out Register (FR_SYMATOR)	R/W
Sync Frame Counter and Tables		
0x0040	Sync Frame Counter Register (FR_SFCNTR)	R
0x0042	Sync Frame Table Offset Register (FR_SFTOR)	R/W
0x0044	Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR)	R/W
Sync Frame Filter		
0x0046	Sync Frame ID Rejection Filter Register (FR_SFIDRFR)	R/W
0x0048	Sync Frame ID Acceptance Filter Value Register (FR_SFIDAFVR)	R/W
0x004A	Sync Frame ID Acceptance Filter Mask Register (FR_SFIDAFMR)	R/W
Network Management Vector		
0x004C	Network Management Vector Register 0 (FR_NMVR0)	R
0x004E	Network Management Vector Register 1 (FR_NMVR1)	R
0x0050	Network Management Vector Register 2 (FR_NMVR2)	R
0x0052	Network Management Vector Register 3 (FR_NMVR3)	R
0x0054	Network Management Vector Register 4 (FR_NMVR4)	R
0x0056	Network Management Vector Register 5 (FR_NMVR5)	R
0x0058	Network Management Vector Length Register (FR_NMVLR)	R/W
Timer Configuration		
0x005A	Timer Configuration and Control Register (FR_TICCR)	R/W
0x005C	Timer 1 Cycle Set Register (FR_TI1CYSR)	R/W
0x005E	Timer 1 Macrotick Offset Register (FR_TI1MTOR)	R/W
0x0060	Timer 2 Configuration Register 0 (FR_TI2CR0)	R/W
0x0062	Timer 2 Configuration Register 1 (FR_TI2CR1)	R/W
Slot Status Configuration		
0x0064	Slot Status Selection Register (FR_SSSR)	R/W
0x0066	Slot Status Counter Condition Register (FR_SSCCR)	R/W
Slot Status		
0x0068	Slot Status Register 0 (FR_SSR0)	R
0x006A	Slot Status Register 1 (FR_SSR1)	R
0x006C	Slot Status Register 2 (FR_SSR2)	R
0x006E	Slot Status Register 3 (FR_SSR3)	R
0x0070	Slot Status Register 4 (FR_SSR4)	R
0x0072	Slot Status Register 5 (FR_SSR5)	R

Table 26-3. FlexRay Memory Map (Sheet 1 of 4)

Offset	Register	Access
0x0074	Slot Status Register 6 (FR_SSR6)	R
0x0076	Slot Status Register 7 (FR_SSR7)	R
0x0078	Slot Status Counter Register 0 (FR_SSCR0)	R
0x007A	Slot Status Counter Register 1 (FR_SSCR1)	R
0x007C	Slot Status Counter Register 2 (FR_SSCR2)	R
0x007E	Slot Status Counter Register 3 (FR_SSCR3)	R
MTS Generation		
0x0080	MTS A Configuration Register (FR_MTSACFR)	R/W
0x0082	MTS B Configuration Register (MTSBCFR)	R/W
Shadow Buffer Configuration		
0x0084	Receive Shadow Buffer Index Register (FR_RSBIR)	R/W
Receive FIFO — Configuration		
0x0086	Receive FIFO Watermark and Selection Register (FR_RFWMSR)	R/W
0x0088	Receive FIFO Start Index Register (FR_RFSIR)	R/W
0x008A	Receive FIFO Depth and Size Register (RFDSR)	R/W
Receive FIFO - Control		
0x008C	Receive FIFO A Read Index Register (FR_RFARIR)	R
0x008E	Receive FIFO B Read Index Register (FR_RFBRIR)	R
Receive FIFO - Filter		
0x0090	Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)	R/W
0x0092	Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)	R/W
0x0094	Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)	R/W
0x0096	Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)	R/W
0x0098	Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)	R/W
0x009A	Receive FIFO Range Filter Control Register (FR_RFRFCTR)	R/W
Dynamic Segment Status		
0x009C	Last Dynamic Transmit Slot Channel A Register (FR_LDTXSLAR)	R
0x009E	Last Dynamic Transmit Slot Channel B Register (FR_LDTXSLBR)	R
Protocol Configuration		
0x00A0 ... 0x00DC	Protocol Configuration Register 0 (FR_PCR0) ... Protocol Configuration Register 30 (FR_PCR30)	R/W
0x00DE ... 0x00E4	Reserved[1]	R
Receive FIFO — Configuration (cont.)		
0x00E6	Receive FIFO Start Data Offset Register (FR_RFSDOR)	R/W
0x00E8	Receive FIFO System Memory Base Address High Register (FR_RFSYMBADHR)	R/W
0x00EA	Receive FIFO System Memory Base Address Low Register (FR_RFSYMBADLR)	R/W
0x00EC	Receive FIFO Periodic Timer Register (FR_RFPTR)	R/W
Receive FIFO - Control (cont.)		

Table 26-3. FlexRay Memory Map (Sheet 1 of 4)

Offset	Register	Access
0x00EE	Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)	R/W
	ECC Registers	
0x00F0	ECC Error Interrupt Flag and Enable Register (FR_EEIFER)	R/W
0x00F2	ECC Error Report and Injection Control Register (FR_EERICR)	R/W
0x00F4	ECC Error Report Address Register (FR_EERAR)	R
0x00F6	ECC Error Report Data Register (FR_EERDR)	R
0x00F8	ECC Error Report Code Register (FR_EERCR)	R
0x00FA	ECC Error Injection Address Register (FR_EEIAR)	R/W
0x00FC	ECC Error Injection Data Register (FR_EEIDR)	R/W
0x00FE	ECC Error Injection Code Register (FR_EEICR)	R/W
0x0100 ... 0x07FE	Not Implemented[1]	R
	Message Buffers Configuration, Control, Status	
0x0800	Message Buffer Configuration, Control, Status Register 0 (FR_MBCCSR0)	R/W
0x0802	Message Buffer Cycle Counter Filter Register 0 (FR_MBCCFR0)	R/W
0x0804	Message Buffer Frame ID Register 0 (FR_MBFIDR0)	R/W
0x0806	Message Buffer Index Register 0 (FR_MBIDXR0)	R/W
...	...	...
0x0BF8	Message Buffer Configuration, Control, Status Register 127 (FR_MBCCSR127)	R/W
0x0BFA	Message Buffer Cycle Counter Filter Register 127 (FR_MBCCFR127)	R/W
0x0BFC	Message Buffer Frame ID Register 127 (FR_MBFIDR127)	R/W
0x0BFE	Message Buffer Index Register 127 (FR_MBIDXR127)	R/W
0x0C00 ... 0x0FFF	Not Implemented[1]	R
0x1000 ... 0x1106	Message Buffer Data Field Offset Register 0 (FR_MBDOR0) ... Message Buffer Data Field Offset Register 131 (FR_MBDOR131)	R/W
0x1108 ... 0x1112	LRAM ECC Error Test Register 0 (FR_LEETR0) ... LRAM ECC Error Test Register 5 (FR_LEETR5)	R/W
0x1114 ... 0x1FFE	Not Implemented[1]	R

[1] Refer to Table 26-4

26.5.2 Register Descriptions

This section provides detailed descriptions of all registers in ascending address order, presented as 16-bit wide entities

Table 26-4 provides a key for the register figures and register tables.

Table 26-4. Register Access Conventions

Convention	Description
▨	Depending on its placement in the read or write row, indicates that the bit is not readable or not writeable.
R*	Reserved bit or field, will not be changed. Application must not write any value different from the reset value.
FIELDNAME	Identifies the field. Its presence in the read or write row indicates that it can be read or written.
Register Field Types	
rwm	A read/write bit that may be modified by a hardware in some fashion other than by a reset.
w1c	Write one to clear. A flag bit that can be read, is cleared by writing a one, writing 0 has no effect.
Reset Value	
0	Resets to zero.
1	Resets to one.
–	Not defined after reset and not affected by reset.

26.5.2.1 Register Reset

All registers except the Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn), Message Buffer Frame ID Registers (FR_MBFIDRn), and Message Buffer Index Registers (FR_MBIDXRn) are reset to their reset value on system reset. The registers mentioned above are located in physical memory blocks and, thus, they are not affected by reset. For some register fields, additional reset conditions exist. These additional reset conditions are mentioned in the detailed description of the register. The additional reset conditions are explained in Table 26-5.

Table 26-5. Additional Register Reset Conditions

Condition	Description
Protocol RUN Command	The register field is reset when the application writes to RUN command “0101” to the POCCMD field in the Protocol Operation Control Register (FR_POCR).
Message Buffer Disable	The register field is reset when the application has disabled the message buffer. This happens when the application writes 1 to the message buffer disable trigger bit FR_MBCCSRn[EDT] while the message buffer is enabled (FR_MBCCSRn[EDS] = 1) and the CC grants the disable to the application by clearing the FR_MBCCSRn[EDS] bit.

26.5.2.2 Register Write Access

This section describes the write access restriction terms that apply to all registers.

26.5.2.2.1 Register Write Access Restriction

For each register bit and register field, the write access conditions are specified in the detailed register description. A description of the write access conditions is given in Table 26-6. If, for a specific register bit or field, none of the given write access conditions is fulfilled, any write attempt to this register bit or field is ignored without any notification. The values of the bits or fields are not changed. The condition term [A or B] indicates that the register or field can be written to if at least one of the conditions is fulfilled.The condition term [A and B] indicates that the register or field can be written to if both conditions are fulfilled.

Table 26-6. Register Write Access Restrictions

Condition	Indication	Description
Any Time	-	No write access restriction.
Disabled Mode	FR_MCR[MEN] = 0	Write access only when CC is in Disabled Mode.
Normal Mode	FR_MCR[MEN] = 1	Write access only when CC is in Normal Mode.
POC:config	FR_PSR0[PROTSTATE] = *POC:config*	Write access only when Protocol is in the POC:config state.
MB_DIS	FR_MBCCSR[EDS] = 0	Write access only when related Message Buffer is disabled.
MB_LCK	FR_MBCCSRn[LCKS] = 1	Write access only when related Message Buffer is locked.
IDL	FR_EEIRICR[BSY] = 0	Write access only when ECC configuration is idle

26.5.2.2.2 Register Write Access Requirements

All registers can be accessed with 8-bit, 16-bit and 32-bit wide operations.

For some of the registers, at least a 16-bit wide write access is required to ensure correct operation. This write access requirement is stated in the detailed register description for each register affected. If an 8-bit wide write access is performed to any of these registers, this access is ignored without notification.

26.5.2.2.3 Internal Register Access

The following memory mapped registers are used to access multiple internal registers.

- Strobe Signal Control Register (FR_STBSCR)
- Slot Status Selection Register (FR_SSSR)
- Slot Status Counter Condition Register (FR_SSCCR)
- Receive Shadow Buffer Index Register (FR_RSBIR)

Each of these memory mapped registers provides a SEL field and a WMD bit. The SEL field is used to select the internal register. The WMD bit controls the write mode. If the WMD bit is set to 0 during the write access, all fields of the internal register are updated. If the WMD bit set to 1, only the SEL field is changed. All other fields of the internal register remain unchanged. This allows for reading back the values of the selected internal register in a subsequent read access.

26.5.2.3 Module Version Register (FR_MVR)

Base + 0x0000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CHIVER								PEVER							
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	1	0	1	0	0	1	0	0	0	1	1	0	1	0	0	0

Figure 26-2. Module Version Register (FR_MVR)

This register provides the CC version number. The module version number is derived from the CHI version number and the PE version number.

Table 26-7. FR_MVR Field Descriptions

Field	Description
CHIVER	**CHI Version Number** — This field provides the version number of the controller host interface.
PEVER	**PE Version Number** — This field provides the version number of the protocol engine.

26.5.2.4 Module Configuration Register (FR_MCR)

Base + 0x0002

Write: MEN, SBFF, SCM, CHB, CHA, ECCE, FUM, FAM, CLKSEL, BITRATE: Disabled Mode
SFFE: Disabled Mode or *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MEN	SBFF	SCM	CHB	CHA	SFFE	ECCE	R*	FUM	FAM	0	CLK SEL	BITRATE			0
W											▨					▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-3. Module Configuration Register (FR_MCR)

This register defines the global configuration of the CC.

Table 26-8. FR_MCR Field Descriptions

Field	Description
MEN	**Module Enable** — This bit indicates whether or not the CC is in the Disabled Mode. The application requests the CC to leave the Disabled Mode by writing 1 to this bit Before leaving the Disabled Mode, the application must configure the SCM, SBFF, CHB, CHA, TMODE, BITRATE values. For details see Section 26.1.6, “Modes of Operation”. 0 Write: ignored, CC disable not possible Read: CC disabled 1 Write: enable CC Read: CC enabled **Note:** If the CC is enabled it cannot be disabled.
SBFF	**System Bus Failure Freeze** — This bit controls the behavior of the CC in case of a system bus failure. 0 Continue normal operation 1 Transition to freeze mode
SCM	**Single Channel Device Mode** — This control bit defines the channel device mode of the CC as described in Section 26.6.10, “Channel Device Modes”. 0 CC works in dual channel device mode 1 CC works in single channel device mode
CHB CHA	**Channel Enable** — protocol related parameter: *pChannels* The semantic of these control bits depends on the channel device mode controlled by the SCM bit and is given Table 26-9.
SFFE	**Synchronization Frame Filter Enable** — This bit controls the filtering for received synchronization frames. For details see Section 26.6.15, “Sync Frame Filtering”. 0 Synchronization frame filtering disabled 1 Synchronization frame filtering enabled

Table 26-8. FR_MCR Field Descriptions

Field	Description
ECCE	**ECC Functionality Enable** — This bit controls the ecc memory error detection functionality. For details see Section 26.6.24, “Memory Content Error Detection”. 0 ECC functionality (injection, detection, reporting, response) disabled 1 ECC functionality enabled
FUM	**FIFO Update Mode** — This bit controls the FIFO update behavior when the interrupt flags FR_GIFER[FAFAIF] and FR_GIFER[FAFBIF] are written by the application (see Section 26.6.9.8, “FIFO Update”) 0 FIFOA/FIFOB is updated on writing 1 to FR_GIFER[FAFAIF] /FR_GIFER[FAFBIF] 1 FIFOA/FIFOB) is *not* updated on writing 1 to FR_GIFER[FAFAIF]/FR_GIFER[FAFBIF]
FAM	**FIFO Address Mode** — This bit controls the location of the system memory base address for the FIFOs. (see Section 26.6.9.2, “FIFO Configuration”) 0 FIFO Base Address located in System Memory Base Address Register (FR_SYMBADR) 1 FIFO Base Address located in Receive FIFO System Memory Base Address Register (FR_RFSYMBADR)
CLKSEL	**Protocol Engine Clock Source Select** — This bit is used to select the clock source for the protocol engine. 0 PE clock source is generated by on-chip crystal oscillator. 1 PE clock source is generated by on-chip PLL.
BITRATE	**FlexRay Bus Bit Rate** — This bit field defines the FlexRay Bus Bit Rate. 000 10.0 Mbit/sec 001 5.0 Mbit/sec 010 2.5 Mbit/sec 011 8.0 Mbit/sec 100 reserved 101 reserved 110 reserved 111 reserved

Table 26-9. FlexRay Channel Selection

SCM	CHB	CHA	Description
Dual Channel Device Modes			
0	0	0	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN not driven by CC ports FR_B_RX, FR_B_TX, and FR_A_TX_EN not driven by CC
	0	1	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN driven by CC - connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and FR_A_TX_EN not driven by CC
	1	0	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN not driven by CC ports FR_B_RX, FR_B_TX, and FR_A_TX_EN driven by CC - connected to FlexRay channel B
	1	1	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN driven by CC - connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and FR_A_TX_EN driven by CC - connected to FlexRay channel B
Single Channel Device Mode			
1	0	0	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN not driven by CC ports FR_B_RX, FR_B_TX, and FR_A_TX_EN not driven by CC
	0	1	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN driven by CC - connected to FlexRay channel A ports FR_B_RX, FR_B_TX, and FR_A_TX_EN not driven by CC
	1	0	ports FR_A_RX, FR_A_TX, and FR_A_TX_EN driven by CC - connected to FlexRay channel B ports FR_B_RX, FR_B_TX, and FR_A_TX_EN not driven by CC
	1	1	reserved

26.5.2.5 System Memory Base Address Register (FR_SYMBADR)

Base + 0x0004 Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SMBA[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-4. System Memory Base Address High Register (FR_SYMBADHR)

Base + 0x0006 Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SMBA[15:4]												0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-5. System Memory Base Address Low Register (FR_SYMBADLR)

NOTE

The system memory base address must be set before the CC is enabled.

The system memory base address registers define the base address of the flexray memory area within the system memory. The base address is used by the BMIF to calculate the physical memory address for system memory accesses.

Table 26-10. FR_SYMBADR Field Descriptions

Field	Description
SMBA	**System Memory Base Address** — This is the value of the system memory base address for the individual message buffers and sync frame table. This is the value of the system memory base address for the receive FIFO if the FIFO address mode bit FR_MCR[FAM] is set to 1. It is defines as a byte address.

26.5.2.6 Strobe Signal Control Register (FR_STBSCR)

Base + 0x0008 16-bit write access required Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	SEL				0	0	0	ENB	0	0	STBPSEL	
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-6. Strobe Signal Control Register (FR_STBSCR)

This register is used to assign the individual protocol timing related strobe signals given in Table 26-12 to the external strobe ports. Each strobe signal can be assigned to at most one strobe port. Each write access to registers overwrites the previously written ENB and STBPSEL values for the signal indicated by SEL. If more than one strobe signal is assigned to one strobe port, the current values of the strobe signals are combined with a binary OR and presented at the strobe port. If no strobe signal is assigned to a strobe port, the strobe port carries logic 0. For more detailed and timing information refer to Section 26.6.16, “Strobe Signal Support”.

NOTE

In single channel device mode, channel B related strobe signals are undefined and should not be assigned to the strobe ports.

Table 26-11. FR_STBSCR Field Descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	**Strobe Signal Select** — This control field selects one of the strobe signals given in Table 26-12 to be enabled or disabled and assigned to one of the four strobe ports given in Table 26-12.
ENB	**Strobe Signal Enable** — The control bit is used to enable and to disable the strobe signal selected by STBSSEL. 0 Strobe signal is disabled and not assigned to any strobe port. 1 Strobe signal is enabled and assigned to the strobe port selected by STBPSEL.
STBPSEL	**Strobe Port Select** — This field selects the strobe port that the strobe signal selected by the SEL is assigned to. All strobe signals that are enabled and assigned to the same strobe port are combined with a binary OR operation. 00 assign selected signal to FR_DBG[0] 01 assign selected signal to FR_DBG[1] 10 assign selected signal to FR_DBG[2] 11 assign selected signal to FR_DBG[3]

Table 26-12. Strobe Signal Mapping

SEL		Description	Channel	Type	Offset[1]	Reference
dec	hex					
0	0x0	arm	-	value	+1	MT start
1	0x1	mt	-	value	+1	MT start
2	0x2	cycle start	-	pulse	0	MT start
3	0x3	minislot start	-	pulse	0	MT start
4	0x4	slot start	A	pulse	0	MT start
5	0x5		B			
6	0x6	receive data after glitch filtering	A	value	+4	FR_A_RX
7	0x7		B			FR_B_RX
8	0x8	channel idle indicator	A	level	+5	FR_A_RX
9	0x9		B			FR_B_RX
10	0xA	syntax error detected	A	pulse	+4	FR_A_RX
11	0xB		B			FR_B_RX
12	0xC	content error detected	A	level	+4	FR_A_RX
13	0xD		B			FR_B_RX

Table 26-12. Strobe Signal Mapping

SEL		Description	Channel	Type	Offset[1]	Reference
dec	hex					
14	0xE	receive FIFO almost-full interrupt signals	A	value	n.a.	RX FIFO A Almost Full Interrupt
15	0xF		B			RX FIFO B Almost Full Interrupt

[1] Given in PE clock cycles

26.5.2.7 Message Buffer Data Size Register (FR_MBDSR)

Base + 0x000C — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	MBSEG2DS							0	MBSEG1DS						
W	▨								▨							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-7. Message Buffer Data Size Register (FR_MBDSR)

This register defines the size of the message buffer data section for the two message buffer segments in a number of two-byte entities.

The CC provides two independent segments for the individual message buffers. All individual message buffers within one segment have to have the same size for the message buffer data section. This size can be different for the two message buffer segments.

Table 26-13. FR_MBDSR Field Descriptions

Field	Description
MBSEG2DS	**Message Buffer Segment 2 Data Size** — The field defines the size of the message buffer data section in two-byte entities for message buffers within the *second* message buffer segment.
MBSEG1DS	**Message Buffer Segment 1 Data Size** — The field defines the size of the message buffer data section in two-byte entities for message buffers within the *first* message buffer segment.

26.5.2.8 Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)

Base + 0x000E — Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	LAST_MB_SEG1							0	LAST_MB_UTIL						
W	▨								▨							
Reset	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1

Figure 26-8. Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)

This register is used to define the last individual message buffer that belongs to the first message buffer segment and the number of the last used individual message buffer.

Table 26-14. FR_MBSSUTR Field Descriptions

Field	Description
LAST_MB_SEG1	**Last Message Buffer In Segment 1** — This field defines the message buffer number of the last individual message buffer that is assigned to the *first* message buffer segment. The individual message buffers in the *first* segment correspond to the message buffer control registers FR_MBCCSRn, FR_MBCCFRn, FR_MBFIDRn, FR_MBIDXRn with n <= LAST_MB_SEG1. The first message buffer segment contains LAST_MB_SEG1+1 individual message buffers. **Note:** The *first* message buffer segment contains *at least* one individual message buffer. The individual message buffers in the *second* message buffer segment correspond to the message buffer control registers FR_MBCCSRn, FR_MBCCFRn, FR_MBFIDRn, FR_MBIDXRn with LAST_MB_SEG1 < n < 128. **Note:** If LAST_MB_SEG1 = 127 all individual message buffers belong to the *first* message buffer segment and the *second* message buffer segment is empty.
LAST_MB_UTIL	**Last Message Buffer Utilized** — This field defines the message buffer number of last utilized individual message buffer. The message buffer search engine examines all individual message buffer with a message buffer number n <= LAST_MB_UTIL. **Note:** If LAST_MB_UTIL=LAST_MB_SEG1 all individual message buffers belong to the *first* message buffer segment and the *second* message buffer segment is empty.

26.5.2.9 PE DRAM Access Register (FR_PEDRAR)

Base + 0x0010 16-bit write access required Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	INST				ADDR											DAD
W																▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-9. PE DRAM Access Register (FR_PEDRAR)

This register is used to trigger write and read operations on the PE data memory (PE DRAM). These operations are used for memory error injection and memory error observation.

Each write access to this registers initiates a read or write operation on the PE DRAM. The access done status bit DAD is cleared after the write access and is set if the PE DRAM access has been finished.

In case of an PE DRAM write access, the data provided in PE DRAM Data Register (FR_PEDRDR) are written into the PE DRAM, read back from the PE DRAM and are stored into the PE DRAM Data Register (FR_PEDRDR).

In case of an PE DRAM read access, the requested data are read from PE DRAM and stored into the PE DRAM Data Register (FR_PEDRDR).

For a detailed description refer to Section 26.6.24, “Memory Content Error Detection”

Table 26-15. FR_PEDRAR Field Descriptions

Field	Description
INST	**PE DRAM Access Instruction** — This field defines the operation to be executed on the PE DRAM. 0011 PE DRAM write: Write FR_PEDRDR[DATA] to PE DRAM address ADDR (16 bit) 0101 PE DRAM read: Read Data from PE DRAM address ADDR (16 bit) into FR_PEDRDR[DATA] other reserved
ADDR	**PE DRAM Access Address** — This field defines the address in the PE DRAM to be written to or read from.
DAD	**PE DRAM Access Done** — This status bit is cleared when the application has written to this register and is set when the PE DRAM access has finished. 0 PE DRAM access running 1 PE DRAM access done

26.5.2.10 PE DRAM Data Register (FR_PEDRDR)

Base + 0x0012 16-bit write access required Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DATA															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-10. PE DRAM Data Register (FR_PEDRDR)

This register provides the data to be written to or read from the PE DRAM by the access initiated by write access to the PE DRAM Access Register (FR_PEDRAR).

26.5.2.11 Protocol Operation Control Register (FR_POCR)

Base + 0x0014 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	EOC_AP		ERC_AP		BSY	0	0	0	POCCMD			
W	WME								WMC							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-11. Protocol Operation Control Register (FR_POCR)

The application uses this register to issue

- protocol control commands
- external clock correction commands

Protocol control commands are issued by writing to the POCCMD field. For more information on protocol control commands, see Section 26.7.6, “Protocol Control Command Execution”.

External clock correction commands are issued by writing to the EOC_AP and ERC_AP fields. For more information on external clock correction, refer to Section 26.6.11, “External Clock Synchronization”.

Table 26-16. FR_POCR Field Descriptions

Field	Description
WME	**Write Mode External Correction** — This bit controls the write mode of the EOC_AP and ERC_AP fields. 0 Write to EOC_AP and ERC_AP fields on register write. 1 No write to EOC_AP and ERC_AP fields on register write.
EOC_AP	**External Offset Correction Application** — This field is used to trigger the application of the external offset correction value defined in the Protocol Configuration Register 29 (FR_PCR29). 00 do not apply external offset correction value 01 reserved 10 subtract external offset correction value 11 add external offset correction value
ERC_AP	**External Rate Correction Application** — This field is used to trigger application of the external rate correction value defined in the Protocol Configuration Register 21 (FR_PCR21) 00 do not apply external rate correction value 01 reserved 10 subtract external rate correction value 11 add external rate correction value
BSY	**Protocol Control Command Write Busy** — This status bit indicates the acceptance of the protocol control command issued by the application via the POCCMD field. The CC sets this status bit when the application has issued a protocol control command via the POCCMD field. The CC clears this status bit when protocol control command was accepted by the PE.When the application issues a protocol control command while the BSY bit is asserted, the CC ignores this command, sets the protocol command ignored error flag PCMI_EF in the CHI Error Flag Register (FR_CHIERFR), and will not change the value of the POCCMD field. 0 Command write idle, command accepted and ready to receive new protocol command. 1 Command write busy, command not yet accepted, not ready to receive new protocol command.
WMC	**Write Mode Command** — This bit controls the write mode of the POCCMD field. 0 Write to POCCMD field on register write. 1 Do not write to POCCMD field on register write.
POCCMD	**Protocol Control Command** — The application writes to this field to issue a protocol control command to the PE. The CC sends the protocol command to the PE immediately. While the transfer is running, the BSY bit is set. 0000 ALLOW_COLDSTART — Immediately activate capability of node to cold start cluster. 0001 ALL_SLOTS — Delayed[1] transition to the all slots transmission mode. 0010 CONFIG — Immediately transition to the *POC:config* state. 0011 FREEZE — Immediately transition to the *POC:halt* state. 0100 READY, CONFIG_COMPLETE — Immediately transition to the *POC:ready* state. 0101 RUN — Immediately transition to the *POC:startup start* state. 0110 DEFAULT_CONFIG — Immediately transition to the *POC:default config* state. 0111 HALT — Delayed transition to the *POC:halt* state 1000 WAKEUP — Immediately initiate the wakeup procedure. 1001 reserved 1010 reserved 1011 reserved 1100 reserved 1101 reserved 1110 reserved 1111 reserved

[1] Delayed means on completion of current communication cycle.

26.5.2.12 Global Interrupt Flag and Enable Register (FR_GIFER)

Base + 0x0016

Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MIF	PRIF	CHIF	WUPIF	FAFBIF	FAFAIF	RBIF	TBIF	MIE	PRIE	CHIE	WUPIE	FAFBIE	FAFAIE	RBIE	TBIE
W	▨	▨	▨	w1c	w1c	w1c	▨	▨								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-12. Global Interrupt Flag and Enable Register (FR_GIFER)

This register provides the means to control some of the interrupt request lines and provides the corresponding interrupt flags. The interrupt flags MIF, PRIF, CHIF, RBIF, and TBIF are the outcome of a binary OR of the related individual interrupt flags and interrupt enables. The generation scheme for these flags is depicted in Figure 26-157. For more details on interrupt generation, see Section 26.6.20, “Interrupt Support. These flags are cleared automatically when all of the corresponding interrupt flags or interrupt enables in the related interrupt flag and enable registers are cleared by the application.

Table 26-17. FR_GIFER Field Descriptions (Sheet 1 of 3)

Field	Description
MIF	**Module Interrupt Flag** — This interrupt flag is set if at least one of the other interrupt flags in this register and the related interrupt enable bit are set. 0 No interrupt flag and related interrupt enable bit are set 1 At least one of the other interrupt flags in this register and the related interrupt bit are set.
PRIF	**Protocol Interrupt Flag** — This interrupt flag is set if at least one of the individual flags in the Protocol Interrupt Flag Register 0 (FR_PIFR0) and Protocol Interrupt Flag Register 1 (FR_PIFR1) and the related interrupt enable bit are set. 0 No individual protocol interrupt flag and related interrupt enable bit are set. 1 At least one of the individual protocol interrupt flags and the related interrupt enable bit are set.
CHIF	**CHI Interrupt Flag** — This interrupt flag is set if at least one of the error flags in the CHI Error Flag Register (FR_CHIERFR) and the chi error interrupt enable bit FR_GIFER[CHIE] are set. 0 All CHI error flags are equal to 0 or the chi error interrupt is disabled. 1 At least one CHI error flag and the chi error interrupt enable are is set.
WUPIF	**Wakeup Interrupt Flag** — This interrupt flag is set when the CC has received a wakeup symbol on the FlexRay bus. The application can determine on which channel the wakeup symbol was received by reading the related wakeup flags WUB and WUA in the Protocol Status Register 3 (FR_PSR3). 0 No Wakeup symbol received on FlexRay bus 1 Wakeup symbol received on FlexRay bus
FAFBIF	**Receive FIFO Channel B Almost Full Interrupt Flag** — This interrupt flag is set when one of the following events occurs a) the current number of FIFO B entries is equal to or greater than the watermark defined by the WM field in the Receive FIFO Watermark and Selection Register (FR_RFWMSR), and the CC writes a received message into the FIFO B, or b) the current number of FIFO B entries is at least 1 and the periodic timer as defined by Receive FIFO Periodic Timer Register (FR_RFPTR) expires. 0 no such event 1 FIFO B almost full event has occurred

Table 26-17. FR_GIFER Field Descriptions (Sheet 1 of 3)

Field	Description
FAFAIF	**Receive FIFO Channel A Almost Full Interrupt Flag** — This interrupt flag is set when one of the following events occurs a) the current number of FIFO A entries is equal to or greater than the watermark defined by the WM field in the Receive FIFO Watermark and Selection Register (FR_RFWMSR), and the CC writes a received message into the FIFO A, or b) the current number of FIFO B entries is at least 1 and the periodic timer as defined by Receive FIFO Periodic Timer Register (FR_RFPTR) expires. 0 no such event 1 FIFO A almost full event has occurred
RBIF	**Receive Message Buffer Interrupt Flag** — This interrupt flag is set if for at least one of the individual receive message buffers (FR_MBCCSRn[MTD] = 0) both the interrupt flag MBIF and the interrupt enable bit MBIE in the corresponding Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn) are asserted. The application cannot clear this interrupt flag directly, instead it is cleared by the CC when all of the interrupt flags MBIF of the individual receive message buffers are cleared by the application or if the application has cleared the related interrupt enables bit MBIE. 0 None of the individual receive message buffers has the MBIF and MBIE flag set. 1 At least one individual receive message buffer has the MBIF and MBIE flag set.
TBIF	**Transmit Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual message buffers (FR_MBCCSRn[MTD] = 1) both the interrupt flag MBIF and the interrupt enable bit MBIE in the corresponding Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn) are equal to 1. The application cannot clear this interrupt flag directly, instead, this interrupt flag is cleared by the CC when either all of the individual interrupt flags MBIF of the individual transmit message buffers are cleared by the application or the application has cleared the related interrupt enables bit MBIE. 0 None of the individual transmit message buffers has the MBIF and MBIE flag set. 1 At least one individual transmit message buffer has the MBIF and MBIE flag set.
MIE	**Module Interrupt Enable** — This bit controls if the Module Interrupt line is asserted when the MIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
PRIE	**Protocol Interrupt Enable** — This bit controls if the Protocol Interrupt line is asserted when the PRIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
CHIE	**CHI Interrupt Enable** — This bit controls if the CHI Interrupt line is asserted when the CHIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
WUPIE	**Wakeup Interrupt Enable** — This bit controls if the Wakeup Interrupt line is asserted when the WUPIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
FAFBIE	**Receive FIFO Channel B Almost Full Interrupt Enable** — This bit controls if the RX FIFO B Almost Full Interrupt line is asserted when the FAFBIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
FAFAIE	**Receive FIFO Channel A Almost Full Interrupt Enable** — This bit controls if the RX FIFO A Almost Full Interrupt line is asserted when the FAFAIF flag is set. 0 Disable interrupt line 1 Enable interrupt line

Table 26-17. FR_GIFER Field Descriptions (Sheet 1 of 3)

Field	Description
RBIE	**Receive Message Buffer Interrupt Enable** — This bit controls if the Receive Message Buffer Interrupt line is asserted when the RBIF flag is set. 0 Disable interrupt line 1 Enable interrupt line
TBIE	**Transmit Message Buffer Interrupt Enable** — This bit controls if the Transmit Message Buffer Interrupt line is asserted when the TBIF flag is set. 0 Disable interrupt line 1 Enable interrupt line

26.5.2.13 Protocol Interrupt Flag Register 0 (FR_PIFR0)

Base + 0x0018 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FATL_IF	INTL_IF	ILCF_IF	CSA_IF	MRC_IF	MOC_IF	CCL_IF	MXS_IF	MTX_IF	LTXB_IF	LTXA_IF	TBVB_IF	TBVA_IF	TI2_IF	TI1_IF	CYS_IF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-13. Protocol Interrupt Flag Register 0 (FR_PIFR0)

The register holds one set of the protocol-related individual interrupt flags.

Table 26-18. FR_PIFR0 Field Descriptions (Sheet 1 of 3)

Field	Description
FATL_IF	**Fatal Protocol Error Interrupt Flag** — This flag is set when the protocol engine has detected a fatal protocol error. In this case, the protocol engine goes into the *POC:halt* state immediately. The fatal protocol errors are: 1) *pLatestTx* violation, as described in the MAC process of the FlexRay protocol 2) transmission across slot boundary violation, as described in the FSP process of the FlexRay protocol 0 No such event. 1 Fatal protocol error detected.
INTL_IF	**Internal Protocol Error Interrupt Flag** — This flag is set when the protocol engine has detected an internal protocol error. In this case, the protocol engine goes into the *POC:halt* state immediately. An internal protocol error occurs when the protocol engine has not finished a calculation and a new calculation is requested. This can be caused by a hardware error. 0 No such event. 1 Internal protocol error detected.
ILCF_IF	**Illegal Protocol Configuration Interrupt Flag** — This flag is set when the protocol engine has detected an illegal protocol configuration parameter setting. In this case, the protocol engine goes into the *POC:halt* state immediately. The protocol engine checks the *listen_timeout* value programmed into the Protocol Configuration Register 14 (FR_PCR14) and Protocol Configuration Register 15 (FR_PCR15) when the CONFIG_COMPLETE command was sent by the application via the Protocol Operation Control Register (FR_POCR). If the value of *listen_timeout* is equal to zero, the protocol configuration setting is considered as illegal. 0 No such event. 1 Illegal protocol configuration detected.

Table 26-18. FR_PIFR0 Field Descriptions (Sheet 1 of 3)

Field	Description
CSA_IF	**Cold Start Abort Interrupt Flag** — This flag is set when the configured number of allowed cold start attempts is reached and none of these attempts was successful. The number of allowed cold start attempts is configured by the coldstart_attempts field in the Protocol Configuration Register 3 (FR_PCR3). 0 No such event. 1 Cold start aborted and no more coldstart attempts allowed.
MRC_IF	**Missing Rate Correction Interrupt Flag** — This flag is set when an insufficient number of measurements is available for rate correction at the end of the communication cycle. 0 No such event 1 Insufficient number of measurements for rate correction detected
MOC_IF	**Missing Offset Correction Interrupt Flag** — This flag is set when an insufficient number of measurements is available for offset correction. This is related to the MISSING_TERM event in the CSP process for offset correction in the FlexRay protocol. 0 No such event. 1 Insufficient number of measurements for offset correction detected.
CCL_IF	**Clock Correction Limit Reached Interrupt Flag** — This flag is set when the internal calculated offset or rate calculation values have reached or exceeded its configured thresholds as given by the *offset_coorection_out* field in the Protocol Configuration Register 9 (FR_PCR9) and the *rate_correction_out* field in the Protocol Configuration Register 14 (FR_PCR14). 0 No such event. 1 Offset or rate correction limit reached.
MXS_IF	**Max Sync Frames Detected Interrupt Flag** — This flag is set when the number of synchronization frames detected in the current communication cycle exceeds the value of the *node_sync_max* field in the Protocol Configuration Register 30 (FR_PCR30). 0 No such event. 1 More than *node_sync_max* sync frames detected. **Note:** Only synchronization frames that have passed the synchronization frame acceptance and rejection filters are taken into account.
MTX_IF	**Media Access Test Symbol Received Interrupt Flag** — This flag is set when the MTS symbol was received on channel A or channel B. 0 No such event. 1 MTS symbol received.
LTXB_IF	*pLatestTx* **Violation on Channel B Interrupt Flag** — This flag is set when the frame transmission on channel B in the dynamic segment exceeds the dynamic segment boundary. This is related to the *pLatestTx* violation, as described in the MAC process of the FlexRay protocol. 0 No such event. 1 *pLatestTx* violation occurred on channel B.
LTXA_IF	*pLatestTx* **Violation on Channel A Interrupt Flag** — This flag is set when the frame transmission on channel A in the dynamic segment exceeds the dynamic segment boundary. This is related to the *pLatestTx* violation as described in the MAC process of the FlexRay protocol. 0 No such event. 1 *pLatestTx* violation occurred on channel A.
TBVB_IF	**Transmission across boundary on channel B Interrupt Flag** — This flag is set when the frame transmission on channel B crosses the slot boundary. This is related to the transmission across slot boundary violation as described in the FSP process of the FlexRay protocol. 0 No such event. 1 Transmission across boundary violation occurred on channel B.

Table 26-18. FR_PIFR0 Field Descriptions (Sheet 1 of 3)

Field	Description
TBVA_IF	**Transmission across boundary on channel A Interrupt Flag** — This flag is set when the frame transmission on channel A crosses the slot boundary. This is related to the transmission across slot boundary violation as described in the FSP process of the FlexRay protocol. 0 No such event. 1 Transmission across boundary violation occurred on channel A.
TI2_IF	**Timer 2 Expired Interrupt Flag** — This flag is set whenever timer 2 expires. 0 No such event. 1 Timer 2 has reached its time limit.
TI1_IF	**Timer 1 Expired Interrupt Flag** — This flag is set whenever timer 1 expires. 0 No such event 1 Timer 1 has reached its time limit
CYS_IF	**Cycle Start Interrupt Flag** — This flag is set when a communication cycle starts. 0 No such event 1 Communication cycle started.

26.5.2.14 Protocol Interrupt Flag Register 1 (FR_PIFR1)

Base + 0x001A Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EMC_IF	IPC_IF	PECF_IF	PSC_IF	SSI3_IF	SSI2_IF	SSI1_IF	SSI0_IF	0	0	EVT_IF	ODT_IF	0	0	0	0
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c			w1c	w1c				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-14. Protocol Interrupt Flag Register 1 (FR_PIFR1)

The register holds one set of the protocol-related individual interrupt flags.

Table 26-19. FR_PIFR1 Field Descriptions (Sheet 1 of 2)

Field	Description
EMC_IF	**Error Mode Changed Interrupt Flag** — This flag is set when the value of the ERRMODE bit field in the Protocol Status Register 0 (FR_PSR0) is changed by the CC. 0 No such event. 1 ERRMODE field changed.
IPC_IF	**Illegal Protocol Control Command Interrupt Flag** — This flag is set when the PE tries to execute a protocol control command, which was issued via the POCCMD field of the Protocol Operation Control Register (FR_POCR), and detects that this protocol control command is not allowed in the current protocol state. In this case the command is not executed. For more details, see Section 26.7.6, “Protocol Control Command Execution”. 0 No such event. 1 Illegal protocol control command detected.
PECF_IF	**Protocol Engine Communication Failure Interrupt Flag** — This flag is set if the CC has detected a communication failure between the PE and the CHI. 0 No such event. 1 Protocol Engine Communication Failure detected.

Table 26-19. FR_PIFR1 Field Descriptions (Sheet 1 of 2)

Field	Description
PSC_IF	**Protocol State Changed Interrupt Flag** — This flag is set when the protocol state in the PROTSTATE field in the Protocol Status Register 0 (FR_PSR0) has changed. 0 No such event. 1 Protocol state changed.
SSI3_IF SSI2_IF SSI1_IF SSI0_IF	**Slot Status Counter Incremented Interrupt Flag** — Each of these flags is set when the SLOTSTATUSCNT field in the corresponding Slot Status Counter Registers (FR_SSCR0–FR_SSCR3) is incremented**.** 0 No such event. 1 The corresponding slot status counter has incremented.
EVT_IF	**Even Cycle Table Written Interrupt Flag** — This flag is set if the CC has written the sync frame measurement / ID tables into the flexray memory area for the even cycle. 0 No such event. 1 Sync frame measurement table written
ODT_IF	**Odd Cycle Table Written Interrupt Flag** — This flag is set if the CC has written the sync frame measurement / ID tables into the flexray memory area for the odd cycle. 0 No such event. 1 Sync frame measurement table written

26.5.2.15 Protocol Interrupt Enable Register 0 (FR_PIER0)

Base + 0x001C Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	FATL_IE	INTL_IE	ILCF_IE	CSA_IE	MRC_IE	MOC_IE	CCL_IE	MXS_IE	MTX_IE	LTXB_IE	LTXA_IE	TBVB_IE	TBVA_IE	TI2_IE	TI1_IE	CYS_IE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-15. Protocol Interrupt Enable Register 0 (FR_PIER0)

This register defines whether or not the individual interrupt flags defined in the Protocol Interrupt Flag Register 0 (FR_PIFR0) can generate a protocol interrupt request.

Table 26-20. FR_PIER0 Field Descriptions

Field	Description
FATL_IE	**Fatal Protocol Error Interrupt Enable** — This bit controls FATL_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
INTL_IE	**Internal Protocol Error Interrupt Enable** — This bit controls INTL_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
ILCF_IE	**Illegal Protocol Configuration Interrupt Enable** — This bit controls ILCF_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled

Table 26-20. FR_PIER0 Field Descriptions

Field	Description
CSA_IE	**Cold Start Abort Interrupt Enable** — This bit controls CSA_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
MRC_IE	**Missing Rate Correction Interrupt Enable** — This bit controls MRC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
MOC_IE	**Missing Offset Correction Interrupt Enable** — This bit controls MOC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
CCL_IE	**Clock Correction Limit Reached Interrupt Enable** — This bit controls CCL_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
MXS_IE	**Max Sync Frames Detected Interrupt Enable** — This bit controls MXS_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
MTX_IE	**Media Access Test Symbol Received Interrupt Enable** — This bit controls MTX_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
LTXB_IE	*pLatestTx* **Violation on Channel B Interrupt Enable** — This bit controls LTXB_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
LTXA_IE	*pLatestTx* **Violation on Channel A Interrupt Enable** — This bit controls LTXA_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
TBVB_IE	**Transmission across boundary on channel B Interrupt Enable** — This bit controls TBVB_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
TBVA_IE	**Transmission across boundary on channel A Interrupt Enable** — This bit controls TBVA_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
TI2_IE	**Timer 2 Expired Interrupt Enable** — This bit controls TI1_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled

Table 26-20. FR_PIER0 Field Descriptions

Field	Description
TI1_IE	**Timer 1 Expired Interrupt Enable** — This bit controls TI1_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
CYS_IE	**Cycle Start Interrupt Enable** — This bit controls CYC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled

26.5.2.16 Protocol Interrupt Enable Register 1 (FR_PIER1)

Base + 0x001E Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	EMC_IE	IPC_IE	PECF_IE	PSC_IE	SSI3_IE	SSI2_IE	SSI1_IE	SSI0_IE	0	0	EVT_IE	ODT_IE	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-16. Protocol Interrupt Enable Register 1 (FR_PIER1)

This register defines whether or not the individual interrupt flags defined in Protocol Interrupt Flag Register 1 (FR_PIFR1) can generate a protocol interrupt request.

Table 26-21. FR_PIER1 Field Descriptions

Field	Description
EMC_IE	**Error Mode Changed Interrupt Enable** — This bit controls EMC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
IPC_IE	**Illegal Protocol Control Command Interrupt Enable** — This bit controls IPC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
PECF_IE	**Protocol Engine Communication Failure Interrupt Enable** — This bit controls PECF_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
PSC_IE	**Protocol State Changed Interrupt Enable** — This bit controls PSC_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
SSI3_IE SSI2_IE SSI1_IE SSI0_IE	**Slot Status Counter Incremented Interrupt Enable** — This bit controls SSI[3:0]_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled

Table 26-21. FR_PIER1 Field Descriptions

Field	Description
EVT_IE	**Even Cycle Table Written Interrupt Enable** — This bit controls EVT_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled
ODT_IE	**Odd Cycle Table Written Interrupt Enable** — This bit controls ODT_IF interrupt request generation. 0 interrupt request generation disabled 1 interrupt request generation enabled

26.5.2.17 CHI Error Flag Register (FR_CHIERFR)

Base + 0x0020 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FRLB_EF	FRLA_EF	PCMI_EF	FOVB_EF	FOVA_EF	MBS_EF	MBU_EF	LCK_EF	0	SBCF_EF	FID_EF	DPL_EF	SPL_EF	NML_EF	NMF_EF	ILSA_EF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c		w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-17. CHI Error Flag Register (FR_CHIERFR)

This register holds the CHI related error flags. The interrupt generation for each of these error flags is controlled by the CHI interrupt enable bit CHIE in the Global Interrupt Flag and Enable Register (FR_GIFER).

Table 26-22. FR_CHIERFR Field Descriptions (Sheet 1 of 2)

Field	Description
FRLB_EF	**Frame Lost Channel B Error Flag** — This flag is set if a complete frame was received on channel B but could not be stored in the selected individual message buffer because this message buffer is currently locked by the application. In this case, the frame and the related slot status information are lost. 0 No such event 1 Frame lost on channel B detected
FRLA_EF	**Frame Lost Channel A Error Flag** — This flag is set if a complete frame was received on channel A but could not be stored in the selected individual message buffer because this message buffer is currently locked by the application. In this case, the frame and the related slot status information are lost. 0 No such error 1 Frame lost on channel A detected
PCMI_EF	**Protocol Command Ignored Error Flag** — This flag is set if the application has issued a POC command by writing to the POCCMD field in the Protocol Operation Control Register (FR_POCR) while the BSY flag is equal to 1. In this case the command is ignored by the CC and is lost. 0 No such error 1 POC command ignored
FOVB_EF	**Receive FIFO Overrun Channel B Error Flag** — This flag is set when an overrun of the FIFO for channel B occurred. This error occurs if a semantically valid frame was received on channel B and matches the all criteria to be appended to the FIFO for channel B but the FIFO is full. In this case, the received frame and its related slot status information is lost. 0 No such error 1 FIFO overrun on channel B has been detected

Table 26-22. FR_CHIERFR Field Descriptions (Sheet 1 of 2)

Field	Description
FOVA_EF	**Receive FIFO Overrun Channel A Error Flag** — This flag is set when an overrun of the FIFO for channel A occurred. This error occurs if a semantically valid frame was received on channel A and matches the all criteria to be appended to the FIFO for channel A but the FIFO is full. In this case, the received frame and its related slot status information is lost. 0 No such error 1 FIFO overrun on channel B has been detected
MBS_EF	**Message Buffer Search Error Flag** — This flag is set if at least one of the following events occurs: a) The message buffer search engine is still running while the next search must be started due to the FlexRay protocol timing. b) A message buffer index greater than 131 is detected in the FR_MBIDXR[MBIDX] field of an found message buffer or in one of the FR_RSBIR[RSBIDX] fields. Refer to Section 26.6.7.4, “Message Buffer Search Error” for details. 0 No such event 1 Search engine active while search start appears or illegal message buffer index detected
MBU_EF	Message Buffer Utilization Error Flag — This flag is asserted if the application writes to a message buffer control field that is beyond the number of utilized message buffers programmed in the Message Buffer Segment Size and Utilization Register (FR_MBSSUTR). If the application writes to a FR_MBCCSRn register with n > LAST_MB_UTIL, the CC ignores the write attempt and asserts the message buffer utilization error flag MBU_EF in the CHI Error Flag Register (FR_CHIERFR). 0 No such event 1 Non-utilized message buffer enabled
LCK_EF	**Lock Error Flag** — This flag is set if the application tries to lock a message buffer that is already locked by the CC due to internal operations. In that case, the CC does not grant the lock to the application. The application must issue the lock request again. 0 No such error 1 Lock error detected
SBCF_EF	**System Bus Communication Failure Error Flag** — This flag is set if a system bus access was not finished within the required amount of time (see Section 26.6.19.1.2, “System Bus Access Timeout”). 0 No such event 1 System bus access not finished in time
FID_EF	**Frame ID Error Flag** — This flag is set if the frame ID stored in the message buffer header area differs from the frame ID stored in the message buffer control register. 0 No such error occurred 1 Frame ID error occurred
DPL_EF	**Dynamic Payload Length Error Flag** — This flag is set if the payload length written into the message buffer header field of a transmit message buffer assigned to the dynamic segment is greater than the maximum payload length for the dynamic segment as it is configured in the corresponding protocol configuration register field max_payload_length_dynamic in the Protocol Configuration Register 24 (FR_PCR24). 0 No such error occurred 1 Dynamic payload length error occurred
SPL_EF	**Static Payload Length Error** Flag — This flag is set if the payload length written into the message buffer header field of a transmit message buffer assigned to the static segment is different from the payload length for the static segment as it is configured in the corresponding protocol configuration register field payload_length_static in the Protocol Configuration Register 19 (FR_PCR19). 0 No such error occurred 1 Static payload length error occurred

Table 26-22. FR_CHIERFR Field Descriptions (Sheet 1 of 2)

Field	Description
NML_EF	**Network Management Length Error Flag** — This flag is set if the payload length written into the header structure of a receive message buffer assigned to the static segment is less than the configured length of the Network Management Vector as configured in the Network Management Vector Length Register (FR_NMVLR). In this case the received part of the Network Management Vector will be used to update the Network Management Vector. 0 No such error occurred 1 Network management length error occurred
NMF_EF	**Network Management Frame Error Flag** — This flag is set if a received message in the static segment with a Preamble Indicator flag PP asserted has its Null Frame indicator flag NF asserted as well. In this case, the Global Network Management Registers (see Network Management Vector Registers (FR_NMVR0–FR_NMVR5)) are not updated. 0 No such error occurred 1 Network management frame error occurred
ILSA_EF	**Illegal System Bus Address Error Flag** — This flag is set if the external system bus subsystem has detected an access to an illegal system bus address from the CC (see Section 26.6.19.1.1, “System Bus Illegal Address Access”). 0 No such event 1 Illegal system bus address accessed

26.5.2.18 Message Buffer Interrupt Vector Register (FR_MBIVEC)

Base + 0x0022

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	TBIVEC							0	RBIVEC						
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-18. Message Buffer Interrupt Vector Register (FR_MBIVEC)

This register indicates the lowest numbered receive message buffer and the lowest numbered transmit message buffer that have their interrupt status flag MBIF and interrupt enable MBIE bits asserted. This means that message buffers with lower message buffer numbers have higher priority.

Table 26-23. FR_MBIVEC Field Descriptions

Field	Description
TBIVEC	**Transmit Buffer Interrupt Vector** — This field provides the number of the lowest numbered enabled transmit message buffer that has its interrupt status flag MBIF and its interrupt enable bit MBIE set. If there is no transmit message buffer with the interrupt status flag MBIF and the interrupt enable MBIE bits asserted, the value in this field is set to 0.
RBIVEC	**Receive Buffer Interrupt Vector** — This field provides the message buffer number of the lowest numbered receive message buffer which has its interrupt flag MBIF and its interrupt enable bit MBIE asserted. If there is no receive message buffer with the interrupt status flag MBIF and the interrupt enable MBIE bits asserted, the value in this field is set to 0.

26.5.2.19 Channel A Status Error Counter Register (FR_CASERCR)

Base + 0x0024 Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	STATUS_ERR_CNT															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-19. Channel A Status Error Counter Register (FR_CASERCR)

This register provides the channel status error counter for channel A. The protocol engine generates a slot status vector for each static slot, each dynamic slot, the symbol window, and the NIT. The slot status vector contains the four protocol related error indicator bits *vSS!SyntaxError, vSS!ContentError, vSS!BViolation*, and *vSS!TxConflict*. The CC increments the status error counter by 1 if, for a slot or segment, at least one error indicator bit is set to 1. The counter wraps around after it has reached the maximum value. For more information on slot status monitoring, see Section 26.6.18, “Slot Status Monitoring”.

Table 26-24. FR_CASERCR Field Descriptions

Field	Description
STATUS_ERR_CNT	**Channel Status Error Counter** — This field provides the current value channel status error counter. The counter value is updated within the first macrotick of the following slot or segment.

26.5.2.20 Channel B Status Error Counter Register (FR_CBSERCR)

Base + 0x0026 Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	STATUS_ERR_CNT															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-20. Channel B Status Error Counter Register (FR_CBSERCR)

This register provides the channel status error counter for channel B. The protocol engine generates a slot status vector for each static slot, each dynamic slot, the symbol window, and the NIT. The slot status vector contains the four protocol related error indicator bits *vSS!SyntaxError*, *vSS!ContentError*, *vSS!BViolation*, and *vSS!TxConflict*. The CC increments the status error counter by 1 if, for a slot or segment, at least one error indicator bit is set to 1. The counter wraps around after it has reached the maximum value. For more information on slot status monitoring see Section 26.6.18, “Slot Status Monitoring”.

Table 26-25. FR_CBSERCR Field Descriptions

Field	Description
STATUS_ERR_CNT	**Channel Status Error Counter** — This field provides the current channel status error count. The counter value is updated within the first macrotick of the following slot or segment.

26.5.2.21 Protocol Status Register 0 (FR_PSR0)

Base + 0x0028

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	ERRMODE		SLOTMODE		0	PROTSTATE			STARTUPSTATE				0	WAKEUPSTATUS		
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-21. Protocol Status Register 0 (FR_PSR0)

This register provides information about the current protocol status.

Table 26-26. FR_PSR0 Field Descriptions

Field	Description
ERRMODE	**Error Mode** — protocol related variable: *vPOC!ErrorMode*. This field indicates the error mode of the protocol. 00 ACTIVE 01 PASSIVE 10 COMM_HALT 11 reserved
SLOTMODE	**Slot Mode** — protocol related variable: *vPOC!SlotMode*. This field indicates the slot mode of the protocol. 00 SINGLE 01 ALL_PENDING 10 ALL 11 reserved
PROTSTATE	**Protocol State** — protocol related variable: *vPOC!State.* This field indicates the state of the protocol. 000 *POC:default config* 001 *POC:config* 010 *POC:wakeup* 011 *POC:ready* 100 *POC:normal passive* 101 *POC:normal active* 110 *POC:halt* 111 *POC:startup*

Table 26-26. FR_PSR0 Field Descriptions

Field	Description
STARTUP STATE	**Startup State** — protocol related variable: *vPOC!StartupState.* This field indicates the current sub-state of the startup procedure. 0000 reserved 0001 reserved 0010 *POC:coldstart collision resolution* 0011 *POC:coldstart listen* 0100 *POC:integration consistency check* 0101 *POC:integrationi listen* 0110 reserved 0111 *POC:initialize schedule* 1000 reserved 1001 reserved 1010 *POC:coldstart consistency check* 1011 reserved 1100 reserved 1101 *POC:integration coldstart check* 1110 *POC:coldstart gap* 1111 *POC:coldstart join*
WAKEUP STATUS	**Wakeup Status** — protocol related variable: *vPOC!WakeupStatus*. This field provides the outcome of the execution of the wakeup mechanism. 000 UNDEFINED 001 RECEIVED_HEADER 010 RECEIVED_WUP 011 COLLISION_HEADER 100 COLLISION_WUP 101 COLLISION_UNKNOWN 110 TRANSMITTED 111 reserved

26.5.2.22 Protocol Status Register 1 (FR_PSR1)

Base + 0x002A

Additional Reset: CSAA, CSP, CPN: RUN Command

Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CSAA	CSP	0	REMCSAT					CPN	HHR	FRZ	APTAC				
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-22. Protocol Status Register 1 (FR_PSR1)

Table 26-27. FR_PSR1 Field Descriptions

Field	Description
CSAA	**Cold Start Attempt Aborted Flag** — protocol related event: ‘set coldstart abort indicator in CHI’ This flag is set when the CC has aborted a cold start attempt. 0 No such event 1 Cold start attempt aborted
CSP	**Leading Cold Start Path** — This status bit is set when the CC has reached the *POC:normal active* state via the leading cold start path. This indicates that this node has started the network 0 No such event 1 *POC:normal active* reached from *POC:startup* state via leading cold start path
REMCSAT	**Remaining Coldstart Attempts** — protocol related variable: *vRemainingColdstartAttempts* This field provides the number of remaining cold start attempts that the CC will execute.
CPN	**Leading Cold Start Path Noise** — protocol related variable: *vPOC!ColdstartNoise* This status bit is set if the CC has reached the *POC:normal active* state via the leading cold start path under noise conditions. This indicates there was some activity on the FlexRay bus while the CC was starting up the cluster. 0 No such event 1 *POC:normal active* state was reached from *POC:startup* state via noisy leading cold start path
HHR	**Host Halt Request Pending** — protocol related variable: *vPOC!CHIHaltRequest* This status bit is set when CC receives the HALT command from the application via the Protocol Operation Control Register (FR_POCR). The CC clears this status bit after a hard reset condition or when the protocol is in the *POC:default config* state. 0 No such event 1 HALT command received
FRZ	**Freeze Occurred** — protocol related variable: *vPOC!Freeze* This status bit is set when the CC has reached the *POC:halt* state due to the host FREEZE command or due to an internal error condition requiring immediate halt. The CC clears this status bit after a hard reset condition or when the protocol is in the *POC:default config* state. 0 No such event 1 Immediate halt due to FREEZE or internal error condition
APTAC	**Allow Passive to Active Counter** — protocol related variable: *vPOC!vAllowPassivetoActive* This field provides the number of consecutive even/odd communication cycle pairs that have passed with valid rate and offset correction terms, but the protocol is still in the *POC:normal passive* state due to an application configured delay to enter *POC:normal active* state. This delay is defined by the allow_passive_to_active field in the Protocol Configuration Register 12 (FR_PCR12).

26.5.2.23 Protocol Status Register 2 (FR_PSR2)

Base + 0x002C Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NBVB	NSEB	STCB	SBVB	SSEB	MTB	NBVA	NSEA	STCA	SBVA	SSEA	MTA	CLKCORRFAILCNT			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-23. Protocol Status Register 2 (FR_PSR2)

This register provides a snapshot of status information about the Network Idle Time NIT, the Symbol Window and the clock synchronization. The NIT related status bits NBVB, NSEB, NBVA, and NSEA are updated by the CC after the end of the NIT and before the end of the first slot of the next communication cycle. The Symbol Window related status bits STCB, SBVB, SSEB, MTB, STCA, SBVA, SSEB, and MTA are updated by the CC after the end of the symbol window and before the end of the current communication cycle. If no symbol window is configured, the symbol window related status bits remain in their reset state. The clock synchronization related CLKCORRFAILCNT is updated by the CC after the end of the static segment and before the end of the current communication cycle.

Table 26-28. FR_PSR2 Field Descriptions (Sheet 1 of 2)

Field	Description
NBVB	**NIT Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* for NIT on channel B This status bit is set when there was some media activity on the FlexRay bus channel B at the end of the NIT. 0 No such event 1 Media activity at boundaries detected
NSEB	**NIT Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* for NIT on channel B This status bit is set when a syntax error was detected during NIT on channel B. 0 No such event 1 Syntax error detected
STCB	**Symbol Window Transmit Conflict on Channel B** — protocol related variable: *vSS!TxConflict* for symbol window on channel B This status bit is set if there was a transmission conflict during the symbol window on channel B. 0 No such event 1 Transmission conflict detected
SBVB	**Symbol Window Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* for symbol window on channel B This status bit is set if there was some media activity on the FlexRay bus channel B at the start or at the end of the symbol window. 0 No such event 1 Media activity at boundaries detected
SSEB	**Symbol Window Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* for symbol window on channel B This status bit is set when a syntax error was detected during the symbol window on channel B. 0 No such event 1 Syntax error detected
MTB	**Media Access Test Symbol MTS Received on Channel B** — protocol related variable: *vSS!ValidMTS* for Symbol Window on channel B This status bit is set if the Media Access Test Symbol MTS was received in the symbol window on channel B. 0 No such event 1 MTS symbol received
NBVA	**NIT Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* for NIT on channel A This status bit is set when there was some media activity on the FlexRay bus channel A at the end of the NIT. 0 No such event 1 Media activity at boundaries detected
NSEA	**NIT Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* for NIT on channel A This status bit is set when a syntax error was detected during NIT on channel A. 0 No such event 1 Syntax error detected

Table 26-28. FR_PSR2 Field Descriptions (Sheet 1 of 2)

Field	Description
STCA	**Symbol Window Transmit Conflict on Channel A** — protocol related variable: *vSS!TxConflict* for symbol window on channel A This status bit is set if there was a transmission conflicts during the symbol window on channel A. 0 No such event 1 Transmission conflict detected
SBVA	**Symbol Window Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* for symbol window on channel A This status bit is set if there was some media activity on the FlexRay bus channel A at the start or at the end of the symbol window. 0 No such event 1 Media activity at boundaries detected
SSEA	**Symbol Window Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* for symbol window on channel A This status bit is set when a syntax error was detected during the symbol window on channel A. 0 No such event 1 Syntax error detected
MTA	**Media Access Test Symbol MTS Received on Channel A** — protocol related variable: *vSS!ValidMTS* for symbol window on channel A This status bit is set if the Media Access Test Symbol MTS was received in the symbol window on channel A. 1 MTS symbol received 0 No such event
CLKCORR-FAILCNT	**Clock Correction Failed Counter** — protocol related variable: *vClockCorrectionFailed* This field provides the number of consecutive even/odd communication cycle pairs that have passed without clock synchronization having performed an offset or a rate correction due to lack of synchronization frames. It is not incremented when it has reached the configured value of either max_without_clock_correction_fatal or max_without_clock_correction_passive as defined in the Protocol Configuration Register 8 (FR_PCR8). The CC resets this counter on a hard reset condition, when the protocol enters the *POC:normal active* state, or when both the rate and offset correction terms have been calculated successfully.

26.5.2.24 Protocol Status Register 3 (FR_PSR3)

Base + 0x002E Additional Reset: RUN Command Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	WUB	ABVB	AACB	ACEB	ASEB	AVFB	0	0	WUA	ABVA	AACA	ACEA	ASEA	AVFA
W			w1c	w1c	w1c	w1c	w1c	w1c			w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-24. Protocol Status Register 3 (FR_PSR3)

This register provides aggregated channel status information as an accrued status of channel activity for all communication slots, regardless of whether they are assigned for transmission or subscribed for reception. It provides accrued information for the symbol window, the NIT, and the wakeup status.

Table 26-29. FR_PSR3 Field Descriptions (Sheet 1 of 2)

Field	Description
WUB	**Wakeup Symbol Received on Channel B** — This flag is set when a wakeup symbol was received on channel B. 0 No wakeup symbol received 1 Wakeup symbol received
ABVB	**Aggregated Boundary Violation on Channel B** — This flag is set when a boundary violation has been detected on channel B. Boundary violations are detected in the communication slots, the symbol window, and the NIT. 0 No boundary violation detected 1 Boundary violation detected
AACB	**Aggregated Additional Communication on Channel B** — This flag is set when at least one valid frame was received on channel B in a slot that also contained an additional communication with either syntax error, content error, or boundary violations. 0 No additional communication detected 1 Additional communication detected
ACEB	**Aggregated Content Error on Channel B** — This flag is set when a content error has been detected on channel B. Content errors are detected in the communication slots, the symbol window, and the NIT. 0 No content error detected 1 Content error detected
ASEB	**Aggregated Syntax Error on Channel B** — This flag is set when a syntax error has been detected on channel B. Syntax errors are detected in the communication slots, the symbol window and the NIT. 0 No syntax error detected 1 Syntax errors detected
AVFB	**Aggregated Valid Frame on Channel B** — This flag is set when a syntactically correct valid frame has been received in any static or dynamic slot through channel B. 1 At least one syntactically valid frame received 0 No syntactically valid frames received
WUA	**Wakeup Symbol Received on Channel A** — This flag is set when a wakeup symbol was received on channel A. 0 No wakeup symbol received 1 Wakeup symbol received
ABVA	**Aggregated Boundary Violation on Channel A** — This flag is set when a boundary violation has been detected on channel A. Boundary violations are detected in the communication slots, the symbol window, and the NIT. 0 No boundary violation detected 1 Boundary violation detected
AACA	**Aggregated Additional Communication on Channel A** — This flag is set when a valid frame was received in a slot on channel A that also contained an additional communication with either syntax error, content error, or boundary violations. 0 No additional communication detected 1 Additional communication detected
ACEA	**Aggregated Content Error on Channel A** — This flag is set when a content error has been detected on channel A. Content errors are detected in the communication slots, the symbol window, and the NIT. 0 No content error detected 1 Content error detected

Table 26-29. FR_PSR3 Field Descriptions (Sheet 1 of 2)

Field	Description
ASEA	**Aggregated Syntax Error on Channel A** — This flag is set when a syntax error has been detected on channel A. Syntax errors are detected in the communication slots, the symbol window, and the NIT. 0 No syntax error detected 1 Syntax errors detected
AVFA	**Aggregated Valid Frame on Channel A** — This flag is set when a syntactically correct valid frame has been received in any static or dynamic slot through channel A. 0 No syntactically valid frames received 1 At least one syntactically valid frame received

26.5.2.25 Macrotick Counter Register (FR_MTCTR)

Base + 0x0030

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	MTCT													
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-25. Macrotick Counter Register (FR_MTCTR)

This register provides the macrotick count of the current communication cycle.

Table 26-30. FR_MTCTR Field Descriptions

Field	Description
MTCT	**Macrotick Counter** — protocol related variable: *vMacrotick* This field provides the macrotick count of the current communication cycle.

26.5.2.26 Cycle Counter Register (FR_CYCTR)

Base + 0x0032

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	CYCCNT					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-26. Cycle Counter Register (FR_CYCTR)

This register provides the number of the current communication cycle.

Table 26-31. FR_CYCTR Field Descriptions

Field	Description
CYCCNT	**Cycle Counter** — protocol related variable: *vCycleCounter* This field provides the number of the current communication cycle. If the counter reaches the maximum value of 63, the counter wraps and starts from zero again.

26.5.2.27 Slot Counter Channel A Register (FR_SLTCTAR)

Base + 0x0034

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	SLOTCNTA										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-27. Slot Counter Channel A Register (FR_SLTCTAR)

This register provides the number of the current slot in the current communication cycle for channel A.

Table 26-32. FR_SLTCTAR Field Descriptions

Field	Description
SLOTCNTA	**Slot Counter Value for Channel A** — protocol related variable: *vSlotCounter* for channel A This field provides the number of the current slot in the current communication cycle.

26.5.2.28 Slot Counter Channel B Register (FR_SLTCTBR)

Base + 0x0036

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	SLOTCNTB										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-28. Slot Counter Channel B Register (FR_SLTCTBR)

This register provides the number of the current slot in the current communication cycle for channel B.

Table 26-33. FR_SLTCTBR Field Descriptions

Field	Description
SLOTCNTA	**Slot Counter Value for Channel B** — protocol related variable: *vSlotCounter* for channel B This field provides the number of the current slot in the current communication cycle.

26.5.2.29 Rate Correction Value Register (FR_RTCORVR)

Base + 0x0038 Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	RATECORR															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-29. Rate Correction Value Register (FR_RTCORVR)

This register provides the sign extended rate correction value in microticks as it was calculated by the clock synchronization algorithm. The CC updates this register during the NIT of each odd numbered communication cycle.

Table 26-34. FR_RTCORVR Field Descriptions

Field	Description
RATECORR	**Rate Correction Value** — protocol related variable: *vRateCorrection* (before value limitation and external rate correction) This field provides the sign extended rate correction value in microticks as it was calculated by the clock synchronization algorithm. The value is represented in 2's complement format. This value does not include the value limitation and the application of the external rate correction. If the magnitude of the internally calculated rate correction value exceeds the limit given by rate_correction_out in the Protocol Configuration Register 13 (FR_PCR13), the clock correction reached limit interrupt flag CCL_IF is set in the Protocol Interrupt Flag Register 0 (FR_PIFR0). **Note:** If the CC was not able to calculate a new rate correction term due to a lack of synchronization frames, the RATECORR value is not updated.

26.5.2.30 Offset Correction Value Register (FR_OFCORVR)

Base + 0x003A Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	OFFSETCORR															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-30. Offset Correction Value Register (FR_OFCORVR)

This register provides the sign extended offset correction value in microticks as it was calculated by the clock synchronization algorithm. The CC updates this register during the NIT.

Table 26-35. FR_OFCORVR Field Descriptions

Field	Description
OFFSET-CORR	**Offset Correction Value** — protocol related variable: *vOffsetCorrection* (before value limitation and external offset correction) This field provides the sign extended offset correction value in microticks as it was calculated by the clock synchronization algorithm. The value is represented in 2’s complement format. This value does not include the value limitation and the application of the external offset correction. If the magnitude of the internally calculated rate correction value exceeds the limit given by offset_correction_out field in the Protocol Configuration Register 29 (FR_PCR29), the clock correction reached limit interrupt flag CCL_IF is set in the Protocol Interrupt Flag Register 0 (FR_PIFR0). **Note:** If the CC was not able to calculate an new offset correction term due to a lack of synchronization frames, the OFFSETCORR value is not updated.

26.5.2.31 Combined Interrupt Flag Register (FR_CIFR)

Base + 0x003C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	MIF	PRIF	CHIF	WUP IF	FAFB IF	FAFA IF	RBIF	TBIF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-31. Combined Interrupt Flag Register (FR_CIFR)

This register provides five combined interrupt flags and a copy of three individual interrupt flags. The combined interrupt flags are the result of a binary OR of the values of other interrupt flags regardless of the state of the interrupt enable bits. The generation scheme for the combined interrupt flags is depicted in Figure 26-160. The individual interrupt flags WUPIF, FAFBIF, and FAFAIF are copies of corresponding flags in the Global Interrupt Flag and Enable Register (FR_GIFER) and are provided here to simplify the application interrupt flag check. To clear the individual interrupt flags, the application must use the Global Interrupt Flag and Enable Register (FR_GIFER).

NOTE

The meanings of the combined status bits MIF, PRIF, CHIF, RBIF, and TBIF are different from those mentioned in the Global Interrupt Flag and Enable Register (FR_GIFER).

Table 26-36. FR_CIFR Field Descriptions

Field	Description
MIF	**Module Interrupt Flag** — This flag is set if there is at least one interrupt source that has its interrupt flag asserted. 0 No interrupt source has its interrupt flag asserted 1 At least one interrupt source has its interrupt flag asserted
PRIF	**Protocol Interrupt Flag** — This flag is set if at least one of the individual protocol interrupt flags in the Protocol Interrupt Flag Register 0 (FR_PIFR0) or Protocol Interrupt Flag Register 1 (FR_PIFR1) is equal to 1. 0 All individual protocol interrupt flags are equal to 0 1 At least one of the individual protocol interrupt flags is equal to 1
CHIF	**CHI Interrupt Flag** — This flag is set if at least one of the individual CHI error flags in the CHI Error Flag Register (FR_CHIERFR) is equal to 1. 0 All CHI error flags are equal to 0 1 At least one CHI error flag is equal to 1
WUPIF	**Wakeup Interrupt Flag** — Provides the same value as FR_GIFER[WUPIF]
FAFBIF	**Receive FIFO Channel B Almost Full Interrupt Flag** — Provides the same value as FR_GIFER[FAFBIF]
FAFAIF	**Receive FIFO Channel A Almost Full Interrupt Flag** — Provides the same value as FR_GIFER[FAFAIF]
RBIF	**Receive Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual receive message buffers (FR_MBCCSRn[MTD] = 0) the interrupt flag MBIF in the corresponding Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn) is equal to 1. 0 None of the individual receive message buffers has the MBIF flag asserted. 1 At least one individual receive message buffers has the MBIF flag asserted.
TBIF	**Transmit Message Buffer Interrupt Flag** — This flag is set if for at least one of the individual transmit message buffers (FR_MBCCSRn[MTD] = 1) the interrupt flag MBIF in the corresponding Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn) is equal to 1. 0 None of the individual transmit message buffers has the MBIF flag asserted. 1 At least one individual transmit message buffers has the MBIF flag asserted.

26.5.2.32 System Memory Access Time-Out Register (FR_SYMATOR)

Base + 0x003E

Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	TIMEOUT							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

Figure 26-32. System Memory Access Time-Out Register (FR_SYMATOR)

Table 26-37. FR_SYMATOR Field Descriptions

Field	Description
TIMEOUT	**System Memory Access Time-Out** — This value defines when a system bus access timeout is detected. For a detailed description see Section 26.7.1.1, “Configure System Memory Access Time-Out Register (FR_SYMATOR)” and Section 26.6.19.1.2, “System Bus Access Timeout”.

26.5.2.33 Sync Frame Counter Register (FR_SFCNTR)

Base + 0x0040 Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SFEVB				SFEVA				SFODB				SFODA			
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-33. Sync Frame Counter Register (FR_SFCNTR)

This register provides the number of synchronization frames that are used for clock synchronization in the last even and in the last odd numbered communication cycle. This register is updated after the start of the NIT and before 10 MT after offset correction start.

NOTE

If the application has locked the even synchronization table at the end of the static segment of an even communication cycle, the CC will not update the fields SFEVB and SFEVA.

If the application has locked the odd synchronization table at the end of the static segment of an odd communication cycle, the CC will not update the values SFODB and SFODA.

Table 26-38. FR_SFCNTR Field Descriptions

Field	Description
SFEVB	Sync Frames Channel B, even cycle — protocol related variable: size of (*vsSyncIdListB* for even cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFEVB	Sync Frames Channel A, even cycle — protocol related variable: size of (*vsSyncIdListA* for even cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFODB	Sync Frames Channel B, odd cycle — protocol related variable: size of (*vsSyncIdListB* for odd cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.
SFODA	Sync Frames Channel A, odd cycle — protocol related variable: size of (*vsSyncIdListA* for odd cycle) This field provides the size of the internal list of frame IDs of received synchronization frames used for clock synchronization.

26.5.2.34 Sync Frame Table Offset Register (FR_SFTOR)

Base + 0x0042 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SFT_OFFSET[15:1]															0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-34. Sync Frame Table Offset Register (FR_SFTOR)

This register defines the flexray memory area related offset for sync frame tables. For more details, see Section 26.6.12, “Sync Frame ID and Sync Frame Deviation Tables”.

Table 26-39. FR_SFTOR Field Description

Field	Description
SFT_OFFSET	**Sync Frame Table Offset** — The offset of the Sync Frame Tables in the flexray memory area. This offset is required to be 16-bit aligned. Thus STF_OFFSET[0] is always 0.

26.5.2.35 Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR)

Base + 0x0044 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	CYCNUM						ELKS	OLKS	EVAL	OVAL	0	0	SDVEN	SIDEN
W	ELKT	OLKT												OPT		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-35. Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR)

This register provides configuration, control, and status information related to the generation and access of the clock sync ID tables and clock sync measurement tables. For a detailed description, see Section 26.6.12, “Sync Frame ID and Sync Frame Deviation Tables”.

Table 26-40. FR_SFTCCSR Field Descriptions

Field	Description
ELKT	**Even Cycle Tables Lock/Unlock Trigger** — This trigger bit is used to lock and unlock the even cycle tables. 0 No effect 1 Triggers lock/unlock of the even cycle tables.
OLKT	**Odd Cycle Tables Lock/Unlock Trigger** — This trigger bit is used to lock and unlock the odd cycle tables. 0 No effect 1 Triggers lock/unlock of the odd cycle tables.
CYCNUM	**Cycle Number** — This field provides the number of the cycle in which the currently locked table was recorded. If none or both tables are locked, this value is related to the even cycle table.

Table 26-40. FR_SFTCCSR Field Descriptions

Field	Description
ELKS	**Even Cycle Tables Lock Status** — This status bit indicates whether the application has locked the even cycle tables. 0 Application has not locked the even cycle tables. 1 Application has locked the even cycle tables.
OLKS	**Odd Cycle Tables Lock Status** — This status bit indicates whether the application has locked the odd cycle tables. 0 Application has not locked the odd cycle tables. 1 Application has locked the odd cycle tables.
EVAL	**Even Cycle Tables Valid** — This status bit indicates whether the Sync Frame ID and Sync Frame Deviation Tables for the even cycle are valid. The CC clears this status bit when it starts updating the tables, and sets this bit when it has finished the table update. 0 Tables are not valid (update is ongoing) 1 Tables are valid (consistent).
OVAL	**Odd Cycle Tables Valid** — This status bit indicates whether the Sync Frame ID and Sync Frame Deviation Tables for the odd cycle are valid. The CC clears this status bit when it starts updating the tables, and sets this bit when it has finished the table update. 0 Tables are not valid (update is ongoing) 1 Tables are valid (consistent).
OPT	**One Pair Trigger** — This trigger bit controls whether the CC writes continuously or only one pair of Sync Frame Tables into the flexray memory area. If this trigger is set to 1 while SDVEN or SIDEN is set to 1, the CC writes only one pair of the enabled Sync Frame Tables corresponding to the next even-odd-cycle pair into the flexray memory area. In this case, the CC clears the SDVEN or SIDEN bits immediately. If this trigger is set to 0 while SDVEN or SIDEN is set to 1, the CC writes continuously the enabled Sync Frame Tables into the flexray memory area. 0 Write continuously pairs of enabled Sync Frame Tables into flexray memory area. 1 Write only one pair of enabled Sync Frame Tables into flexray memory area.
SDVEN	**Sync Frame Deviation Table Enable** — This bit controls the generation of the Sync Frame Deviation Tables. The application must set this bit to request the CC to write the Sync Frame Deviation Tables into the flexray memory area. 0 Do not write Sync Frame Deviation Tables 1 Write Sync Frame Deviation Tables into flexray memory area **Note:** If SDVEN is set to 1, then SIDEN must also be set to 1.
SIDEN	**Sync Frame ID Table Enable** — This bit controls the generation of the Sync Frame ID Tables. The application must set this bit to 1 to request the CC to write the Sync Frame ID Tables into the flexray memory area. 0 Do not write Sync Frame ID Tables 1 Write Sync Frame ID Tables into flexray memory area

26.5.2.36 Sync Frame ID Rejection Filter Register (FR_SFIDRFR)

Base + 0x0046 16-bit write access required Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	SYNFRID									
W	▨	▨	▨	▨	▨	▨										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-36. Sync Frame ID Rejection Filter Register (FR_SFIDRFR)

This register defines the Sync Frame Rejection Filter ID. The application must update this register outside of the static segment. If the application updates this register in the static segment, it can appear that the CC accepts the sync frame in the current cycle.

Table 26-41. FR_SFIDRFR Field Descriptions

Field	Description
SYNFRID	**Sync Frame Rejection ID** — This field defines the frame ID of a frame that must not be used for clock synchronization. For details see Section 26.6.15.2, “Sync Frame Rejection Filtering”.

26.5.2.37 Sync Frame ID Acceptance Filter Value Register (FR_SFIDAFVR)

Base + 0x0048 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	FVAL									
W	▨	▨	▨	▨	▨	▨										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-37. Sync Frame ID Acceptance Filter Value Register (FR_SFIDAFVR)

This register defines the sync frame acceptance filter value. For details on filtering, see Section 26.6.15, “Sync Frame Filtering”.

Table 26-42. FR_SFIDAFVR Field Descriptions

Field	Description
FVAL	**Filter Value** — This field defines the value for the sync frame acceptance filtering.

26.5.2.38 Sync Frame ID Acceptance Filter Mask Register (FR_SFIDAFMR)

Base + 0x004A Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	FMSK									
W	▨	▨	▨	▨	▨	▨										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-38. Sync Frame ID Acceptance Filter Mask Register (FR_SFIDAFMR)

This register defines the sync frame acceptance filter mask. For details on filtering see Section 26.6.15.1, “Sync Frame Acceptance Filtering”.

Table 26-43. FR_SFIDAFMR Field Descriptions

Field	Description
FMSK	**Filter Mask** — This field defines the mask for the sync frame acceptance filtering.

26.5.2.39 Network Management Vector Registers (FR_NMVR0–FR_NMVR5)

Base + 0x004C (FR_NMVR0)
Base + 0x004E (FR_NMVR1)
Base + 0x0050 (FR_NMVR2)
Base + 0x0052 (FR_NMVR3)
Base + 0x0054 (FR_NMVR4)
Base + 0x0056 (FR_NMVR5)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	NMVP[15:8]								NMVP[7:0]							
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-39. Network Management Vector Registers (FR_NMVR0–FR_NMVR5)

Each of these six registers holds one part of the Network Management Vector. The length of the Network Management Vector is configured in the Network Management Vector Length Register (FR_NMVLR). If FR_NMVLR is programmed with a value that is less than 12 bytes, the remaining bytes of the Network Management Vector Registers (FR_NMVR0–FR_NMVR5), which are not used for the Network Management Vector accumulating, will remain 0.

The NMVR provides accrued information over all received NMVs in the last communication cycle. All NMVs received in one cycle are ORed into the NMVR. The NMVR is updated at the end of the communication cycle.

Table 26-44. NMVR[0:5] Field Descriptions

Field	Description
NMVP	**Network Management Vector Part** — The mapping between the Network Management Vector Registers (FR_NMVR0–FR_NMVR5) and the receive message buffer payload bytes in NMV[0:11] is depicted in **Table 26-45**.

Table 26-45. Mapping of NMVRn to the Received Payload Bytes NMVn

NMVRn Register	NMVn Received Payload
FR_NMVR0[NMVP[15:8]]	NMV0
FR_NMVR0[NMVP[7:0]]	NMV1
FR_NMVR1[NMVP[15:8]]	NMV2
FR_NMVR1[NMVP[7:0]]	NMV3
...	
FR_NMVR5[NMVP[15:8]]	NMV10
FR_NMVR5[NMVP[7:0]]	NMV11

26.5.2.40 Network Management Vector Length Register (FR_NMVLR)

Base + 0x0058

Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	NMVL			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-40. Network Management Vector Length Register (FR_NMVLR)

This register defines the length of the network management vector in bytes.

Table 26-46. FR_NMVLR Field Descriptions

Field	Description
NMVL	**Network Management Vector Length** — protocol related variable: *gNetworkManagementVectorLength* This field defines the length of the Network Management Vector in bytes. Legal values are between 0 and 12.

26.5.2.41 Timer Configuration and Control Register (FR_TICCR)

Base + 0x005A

Write: T2_CFG: *POC:config*
T2_REP, T1_REP, T1SP, T2SP, T1TR, T2TR: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T2_CFG	T2_REP	0	0	0	T2ST	0	0	0	T1_REP	0	0	0	T1ST
W						T2SP	T2TR							T1SP	T1TR	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-41. Timer Configuration and Control Register (FR_TICCR)

This register is used to configure and control the two timers T1 and T2. For timer details, see Section 26.6.17, “Timer Support”. The Timer T1 is an absolute timer. The Timer T2 can be configured as an absolute or relative timer.

Table 26-47. FR_TICCR Field Descriptions

Field	Description
T2_CFG	**Timer T2 Configuration** — This bit configures the timebase mode of Timer T2. 0 T2 is absolute timer. 1 T2 is relative timer.
T2_REP	**Timer T2 Repetitive Mode** — This bit configures the repetition mode of Timer T2. 0 T2 is non repetitive 1 T2 is repetitive
T2SP	**Timer T2 Stop** — This trigger bit is used to stop timer T2. 0 no effect 1 stop timer T2
T2TR	**Timer T2 Trigger** — This trigger bit is used to start timer T2. 0 no effect 1 start timer T2
T2ST	**Timer T2 State** — This status bit provides the current state of timer T2. 0 timer T2 is idle 1 timer T2 is running
T1_REP	**Timer T1 Repetitive Mode** — This bit configures the repetition mode of timer T1. 0 T1 is non repetitive 1 T1 is repetitive
T1SP	**Timer T1 Stop** — This trigger bit is used to stop timer T1. 0 no effect 1 stop timer T1
T1TR	**Timer T1 Trigger** — This trigger bit is used to start timer T1. 0 no effect 1 start timer T1
T1ST	**Timer T1 State** — This status bit provides the current state of timer T1. 0 timer T1 is idle 1 timer T1 is running

NOTE

Both timers are deactivated immediately when the protocol enters a state different from *POC:normal active* or *POC:normal passive*.

26.5.2.42 Timer 1 Cycle Set Register (FR_TI1CYSR)

Base + 0x005C Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T1_CYC_VAL						0	0	T1_CYC_MSK					
W	▨	▨							▨	▨						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-42. Timer 1 Cycle Set Register (FR_TI1CYSR)

This register defines the cycle filter value and the cycle filter mask for timer T1. For a detailed description of timer T1, refer to Section 26.6.17.1, “Absolute Timer T1”.

Table 26-48. FR_TI1CYSR Field Descriptions

Field	Description
T1_CYC_VAL	**Timer T1 Cycle Filter Value** — This field defines the cycle filter value for timer T1.
T1_CYC_MSK	**Timer T1 Cycle Filter Mask** — This field defines the cycle filter mask for timer T1.

NOTE

If the application modifies the value in this register while the timer is running, the change becomes effective immediately and timer T1 will expire according to the changed value.

26.5.2.43 Timer 1 Macrotick Offset Register (FR_TI1MTOR)

Base + 0x005E Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	T1_MTOFFSET													
W	▨	▨														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-43. Timer 1 Macrotick Offset Register (FR_TI1MTOR)

This register holds the macrotick offset value for timer T1. For a detailed description of timer T1, refer to Section 26.6.17.1, “Absolute Timer T1”.

Table 26-49. FR_TI1MTOR Field Descriptions

Field	Description
T1_MTOFFSET	**Timer 1 Macrotick Offset** — This field defines the macrotick offset value for timer 1.

NOTE

If the application modifies the value in this register while the timer is running, the change becomes effective immediately and timer T1 will expire according to the changed value.

26.5.2.44 Timer 2 Configuration Register 0 (FR_TI2CR0)

Base + 0x0060 Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	R*		T2_CYC_VAL						R*		T2_CYC_MSK					
R W	T2_MTCNT[31:16]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-44. Timer 2 Configuration Register 0 (FR_TI2CR0)

The content of this register depends on the value of the T2_CFG bit in the Timer Configuration and Control Register (FR_TICCR). For a detailed description of timer T2, refer to Section 26.6.17.2, “Absolute / Relative Timer T2”.

Table 26-50. FR_TI2CR0 Field Descriptions

Field	Description
Fields for absolute timer T2 (FR_TICCR[T2_CFG] = 0)	
T2_CYC_VAL	**Timer T2 Cycle Filter Value** — This field defines the cycle filter value for timer T2.
T2_CYC_MSK	**Timer T2 Cycle Filter Mask** — This field defines the cycle filter mask for timer T2.
Fields for relative timer T2 (FR_TICCR[T2_CFG = 1)	
T2_MTCNT[31:16]	**Timer T2 Macrotick High Word** — This field defines the high word of the macrotick count for timer T2.

NOTE

If timer T2 is configured as an *absolute* timer and the application modifies the values in this register while the timer is running, the change becomes effective immediately and timer T2 will expire according to the changed values.

If timer T2 is configured as a *relative* timer and the application changes the values in this register while the timer is running, the change becomes effective when the timer has expired according to the old values.

26.5.2.45 Timer 2 Configuration Register 1 (FR_TI2CR1)

Base + 0x0062

Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	R*		T2_MTOFFSET													
R W	T2_MTCNT[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-45. Timer 2 Configuration Register 1 (FR_TI2CR1)

The content of this register depends on the value of the T2_CFG bit in the Timer Configuration and Control Register (FR_TICCR). For a detailed description of timer T2, refer to Section 26.6.17.2, “Absolute / Relative Timer T2”.

Table 26-51. FR_TI2CR1 Field Descriptions

Field	Description
Fields for absolute timer T2 (FR_TICCR[T2_CFG] = 0)	
T2_MTOFFSET	**Timer T2 Macrotick Offset** — This field holds the macrotick offset value for timer T2.
Fields for relative timer T2 (FR_TICCR[T2_CFG] = 1)	
T2_MTCNT[15:0]	**Timer T2 Macrotick Low Word** — This field defines the low word of the macrotick value for timer T2.

NOTE

> If timer T2 is configured as an *absolute* timer and the application modifies the values in this register while the timer is running, the change becomes effective immediately and the timer T2 will expire according to the changed values.
>
> If timer T2 is configured as a *relative* timer and the application changes the values in this register while the timer is running, the change becomes effective when the timer has expired according to the old values.

26.5.2.46 Slot Status Selection Register (FR_SSSR)

Base + 0x0064

16-bit write access required

Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SEL		0	SLOTNUMBER										
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-46. Slot Status Selection Register (FR_SSSR)

This register is used to access the four internal non-memory mapped slot status selection registers FR_SSSR0 to FR_SSSR3. Each internal register selects a slot, or symbol window/NIT, whose status vector will be saved in the corresponding Slot Status Registers (FR_SSR0–FR_SSR7) according to Table 26-53. For a detailed description of slot status monitoring, refer to Section 26.6.18, “Slot Status Monitoring”.

Table 26-52. FR_SSSR Field Descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	**Selector** — This field selects one of the four internal slot status selection registers for access. 00 select FR_SSSR0. 01 select FR_SSSR1. 10 select FR_SSSR2. 11 select FR_SSSR3.
SLOTNUMBER	**Slot Number** — This field specifies the number of the slot whose status will be saved in the corresponding slot status registers. **Note:** If this value is set to 0, the related slot status register provides the status of the symbol window after the NIT start, and provides the status of the NIT after the cycle start.

Table 26-53. Mapping Between FR_SSSRn and FR_SSRn

Internal Slot Status Selection Register	Write the Slot Status of the Slot Selected by FR_SSSRn for each			
	Even Communication Cycle		Odd Communication Cycle	
	For Channel B to	For Channel A to	For Channel B to	For Channel A to
FR_SSSR0	FR_SSR0[15:8]	FR_SSR0[7:0]	FR_SSR1[15:8]	FR_SSR1[7:0]
FR_SSSR1	FR_SSR2[15:8]	FR_SSR2[7:0]	FR_SSR3[15:8]	FR_SSR3[7:0]
FR_SSSR2	FR_SSR4[15:8]	FR_SSR4[7:0]	FR_SSR5[15:8]	FR_SSR5[7:0]
FR_SSSR3	FR_SSR6[15:8]	FR_SSR6[7:0]	FR_SSR7[15:8]	FR_SSR7[7:0]

26.5.2.47 Slot Status Counter Condition Register (FR_SSCCR)

Base + 0x0066   16-bit write access required   Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SEL		0	CNTCFG		MCY	VFR	SYF	NUF	SUF	STATUSMASK[3:0]			
W	WMD	▨			▨											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-47. Slot Status Counter Condition Register (FR_SSCCR)

This register is used to access and program the four internal non-memory mapped Slot Status Counter Condition Registers FR_SSCCR0 to FR_SSCCR3. Each of these four internal slot status counter condition registers defines the mode and the conditions for incrementing the counter in the corresponding Slot Status

Counter Registers (FR_SSCR0–FR_SSCR3). The correspondence is given in Table 26-55. For a detailed description of slot status counters, refer to Section 26.6.18.4, “Slot Status Counter Registers”.

Table 26-54. FR_SSCCR Field Descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL field only on write access.
SEL	**Selector** — This field selects one of the four internal slot counter condition registers for access. 00 select FR_SSCCR0. 01 select FR_SSCCR1. 10 select FR_SSCCR2. 11 select FR_SSCCR3.
CNTCFG	**Counter Configuration** — These bit field controls the channel related incrementing of the slot status counter. 00 increment by 1 if condition is fulfilled on channel A. 01 increment by 1 if condition is fulfilled on channel B. 10 increment by 1 if condition is fulfilled on at least one channel. 11 increment by 2 if condition is fulfilled on both channels channel. increment by 1 if condition is fulfilled on only one channel.
MCY	**Multi Cycle Selection** — This bit defines whether the slot status counter accumulates over multiple communication cycles or provides information for the previous communication cycle only. 0 The Slot Status Counter provides information for the previous communication cycle only. 1 The Slot Status Counter accumulates over multiple communication cycles.
VFR	**Valid Frame Restriction** — This bit is used to restrict the counter to received valid frames. 0 The counter is not restricted to valid frames only. 1 The counter is restricted to valid frames only.
SYF	**Sync Frame Restriction** — This bit is used to restrict the counter to received frames with the sync frame indicator bit set to 1. 0 The counter is not restricted with respect to the sync frame indicator bit. 1 The counter is restricted to frames with the sync frame indicator bit set to 1.
NUF	**Null Frame Restriction** — This bit is used to restrict the counter to received frames with the null frame indicator bit set to 0. 0 The counter is not restricted with respect to the null frame indicator bit. 1 The counter is restricted to frames with the null frame indicator bit set to 0.
SUF	**Startup Frame Restriction** — This bit is used to restrict the counter to received frames with the startup frame indicator bit set to 1. 0 The counter is not restricted with respect to the startup frame indicator bit. 1 The counter is restricted to received frames with the startup frame indicator bit set to 1.
STATUS MASK[3:0]	**Slot Status Mask** — This bit field is used to enable the counter with respect to the four slot status error indicator bits. **STATUSMASK[3]** – This bit enables the counting for slots with the syntax error indicator bit set to 1. **STATUSMASK[2]** – This bit enables the counting for slots with the content error indicator bit set to 1. **STATUSMASK[1]** – This bit enables the counting for slots with the boundary violation indicator bit set to 1. **STATUSMASK[0]** – This bit enables the counting for slots with the transmission conflict indicator bit set to 1.

Table 26-55. Mapping between internal FR_SSCCRn and FR_SSCRn

Condition Register	Condition Defined for Register
FR_SSCCR0	FR_SSCR0

Table 26-55. Mapping between internal FR_SSCCRn and FR_SSCRn

Condition Register	Condition Defined for Register
FR_SSCCR1	FR_SSCR1
FR_SSCCR2	FR_SSCR2
FR_SSCCR3	FR_SSCR3

26.5.2.48 Slot Status Registers (FR_SSR0–FR_SSR7)

Base + 0x0068 (FR_SSR0)
Base + 0x006A (FR_SSR1)
Base + 0x006C (FR_SSR2)
Base + 0x006E (FR_SSR3)
Base + 0x0070 (FR_SSR4)
Base + 0x0072 (FR_SSR5)
Base + 0x0074 (FR_SSR6)
Base + 0x0076 (FR_SSR7)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-48. Slot Status Registers (FR_SSR0–FR_SSR7)

Each of these eight registers holds the status vector of the slot specified in the corresponding internal slot status selection register, which can be programmed using the Slot Status Selection Register (FR_SSSR). Each register is updated after the end of the corresponding slot as shown in Figure 26-155. The register bits are directly related to the protocol variables and described in more detail in Section 26.6.18, “Slot Status Monitoring”.

Table 26-56. FR_SSR0–FR_SSR7 Field Descriptions

Field	Description
VFB	**Valid Frame on Channel B** — protocol related variable: *vSS!ValidFrame* channel B 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYB	**Sync Frame Indicator Channel B** — protocol related variable: *vRF!Header!SyFIndicator* channel B 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFB	**Null Frame Indicator Channel B** — protocol related variable: *vRF!Header!NFIndicator* channel B 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUB	**Startup Frame Indicator Channel B** — protocol related variable: *vRF!Header!SuFIndicator* channel B 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1

Table 26-56. FR_SSR0–FR_SSR7 Field Descriptions

Field	Description
SEB	**Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* channel B 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1
CEB	**Content Error on Channel B** — protocol related variable: *vSS!ContentError* channel B 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVB	**Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* channel B 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1
TCB	**Transmission Conflict on Channel B** — protocol related variable: *vSS!TxConflict* channel B 0 *vSS!TxConflict* = 0 1 *vSS!TxConflict* = 1
VFA	**Valid Frame on Channel A** — protocol related variable: *vSS!ValidFrame* channel A 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYA	**Sync Frame Indicator Channel A** — protocol related variable: *vRF!Header!SyFIndicator* channel A 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFA	**Null Frame Indicator Channel A** — protocol related variable: *vRF!Header!NFIndicator* channel A 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUA	**Startup Frame Indicator Channel A** — protocol related variable: *vRF!Header!SuFIndicator* channel A 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEA	**Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* channel A 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1
CEA	**Content Error on Channel A** — protocol related variable: *vSS!ContentError* channel A 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVA	**Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* channel A 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1
TCA	**Transmission Conflict on Channel A** — protocol related variable: *vSS!TxConflict* channel A 0 *vSS!TxConflict* = 0 1 *vSS!TxConflict* = 1

26.5.2.49 Slot Status Counter Registers (FR_SSCR0–FR_SSCR3)

Base + 0x0078 (FR_SSCR0)
Base + 0x007A (FR_SSCR1)
Base + 0x007C (FR_SSCR2)
Base + 0x007E (FR_SSCR3)

Additional Reset: RUN Command

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SLOTSTATUSCNT															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-49. Slot Status Counter Registers (FR_SSCR0–FR_SSCR3)

Each of these four registers provides the slot status counter value for the previous communication cycle(s) and is updated at the cycle start. The provided value depends on the control bits and fields in the related internal slot status counter condition register FR_SSCCRn, which can be programmed by using the Slot Status Counter Condition Register (FR_SSCCR). For more details, see Section 26.6.18.4, “Slot Status Counter Registers”.

NOTE

If the counter has reached its maximum value 0xFFFF and is in the multicycle mode (FR_SSCCRn[MCY] = 1), the counter is not reset to 0x0000. The application can reset the counter by clearing the FR_SSCCRn[MCY] bit and waiting for the next cycle start, when the CC clears the counter. Subsequently, the counter can be set into the multicycle mode again.

Table 26-57. FR_SSCR0–FR_SSCR3 Field Descriptions

Field	Description
SLOTSTATUSCNT	**Slot Status Counter** — This field provides the current value of the Slot Status Counter.

26.5.2.50 MTS A Configuration Register (FR_MTSACFR)

Base + 0x0080

Write: MTE: Anytime
CYCCNTMSK,CYCCNTVAL:*POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MTE	0	CYCCNTMSK						0	0	CYCCNTVAL					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-50. MTS A Configuration Register (FR_MTSACFR)

This register controls the transmission of the Media Access Test Symbol MTS on channel A. For more details, see Section 26.6.13, “MTS Generation”.

Table 26-58. FR_MTSACFR Field Descriptions

Field	Description
MTE	**Media Access Test Symbol Transmission Enable** — This control bit is used to enable and disable the transmission of the Media Access Test Symbol in the selected set of cycles. 0 MTS transmission disabled 1 MTS transmission enabled
CYCCNTMSK	Cycle Counter Mask — This field provides the filter mask for the MTS cycle count filter.
CYCCNTVAL	**Cycle Counter Value** — This field provides the filter value for the MTS cycle count filter.

26.5.2.51 MTS B Configuration Register (MTSBCFR)

Base + 0x0082

Write: MTE: Anytime
CYCCNTMSK,CYCCNTVAL:*POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MTE	0	CYCCNTMSK						0	0	CYCCNTVAL					
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-51. MTS B Configuration Register (MTSBCFR)

This register controls the transmission of the Media Access Test Symbol MTS on channel B. For more details, see Section 26.6.13, “MTS Generation”.

Table 26-59. MTSBCFR Field Descriptions

Field	Description
MTE	**Media Access Test Symbol Transmission Enable** — This control bit is used to enable and disable the transmission of the Media Access Test Symbol in the selected set of cycles. 0 MTS transmission disabled 1 MTS transmission enabled
CYCCNTMSK	Cycle Counter Mask — This field provides the filter mask for the MTS cycle count filter.
CYCCNTVAL	**Cycle Counter Value** — This field provides the filter value for the MTS cycle count filter.

26.5.2.52 Receive Shadow Buffer Index Register (FR_RSBIR)

Base + 0x0084

16-bit write access required

Write: WMD, SEL: Any Time
RSBIDX: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	SEL		0	0	0	0	RSBIDX$_{A1}$/RSBIDX$_{A2}$ RSBIDX$_{B1}$/RSBIDX$_{B2}$							
W	WMD															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-52. Receive Shadow Buffer Index Register (FR_RSBIR)

This register is used to provide and retrieve the indices of the message buffer header fields currently associated with the receive shadow buffers. For more details on the receive shadow buffer concept, refer to Section 26.6.6.3.5, “Receive Shadow Buffers Concept”.

Table 26-60. FR_RSBIR Field Descriptions

Field	Description
WMD	**Write Mode** — This bit controls the write mode for this register. 0 update SEL and RSBIDX field on register write 1 update only SEL field on register write
SEL	**Selector** — This field is used to select the internal receive shadow buffer index register for access. 00 FR_RSBIR_A1 — receive shadow buffer index register for channel A, segment 1 01 FR_RSBIR_A2 — receive shadow buffer index register for channel A, segment 2 10 FR_RSBIR_B1 — receive shadow buffer index register for channel B, segment 1 11 FR_RSBIR_B2 — receive shadow buffer index register for channel B, segment 2
RSBIDX_{A1} RSBIDX_{A2} RSBIDX_{B1} RSBIDX_{B2}	**Receive Shadow Buffer Index** — This field contains the current index of the message buffer header field of the receive shadow message buffer selected by the SEL field. The CC uses this index to determine the physical location of the shadow buffer header field in the flexray memory area. The CC will update this field during receive operation.The application provides initial message buffer header index value in the configuration phase. CC: Updates the message buffer header index after successful reception. Application: Provides initial message buffer header index. Legal Values are 0 <= i <= 131. Illegal values will be detected during the message buffer search.

26.5.2.53 Receive FIFO Start Data Offset Register (FR_RFSDOR)

Base + 0x00E6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	$\text{SDO}_A/\text{SDO}_B$															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-53. Receive FIFO Start Data Offset Register (FR_RFSDOR)

Table 26-61. FR_RFSDOR Field Descriptions

Field	Description
SDO_A SDO_B	**Start Data Field Offset** — This field defines the data field offset of the header field of the first message buffer of the selected FIFO. The CC uses the value of the SDO field to determine the physical location of the receiver FIFO’s first message buffer header field. For configuration constraints see Section 26.7.1.2, “Configure Data Field Offsets”.

NOTE

Since all data fields of the FIFO are of equal length and are located at subsequent system memory addresses the content of the FR_RFSDOR register corresponds to the start address of payload area of the selected FIFO.

26.5.2.54 Receive FIFO System Memory Base Address Register (FR_RFSYMBADR)

Base + 0x00E8 Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SMBA[31:16]															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-54. Receive FIFO System Memory Base Address High Register (FR_RFSYMBADHR)

Base + 0x00EA Write: Disabled Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	SMBA[15:4]												0	0	0	0
W													▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-55. Receive FIFO System Memory Base Address Low Register (FR_RFSYMBADLR)

These registers define the system memory base address for the receive FIFO if the FIFO address mode bit FR_MCR[FAM] is set to 1. The system memory base address is used by the BMIF to calculate the physical memory address for system memory accesses for the FIFOs.

Table 26-62. FR_RFSYMBADR Field Descriptions

Field	Description
SMBA	**System Memory Base Address** — This is the value of the system memory base address for the receive FIFO if the FIFO address mode bit FR_MCR[FAM] is set to 1. It is defines as a byte address.

26.5.2.55 Receive FIFO Periodic Timer Register (FR_RFPTR)

Base + 0x00EC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	PTD													
W	▨	▨														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-56. Receive FIFO Periodic Timer Register (FR_RFPTR)

This register holds periodic timer duration for the periodic FIFO timer. The periodic timer applies to both FIFOs (see Section 26.6.9.3, “FIFO Periodic Timer”).

Table 26-63. FR_RFPTR Field Descriptions

Field	Description
PTD	**Periodic Timer Duration** — This value defines the periodic timer duration in terms of macroticks. 0000 timer stays expired 3FFF timer never expires other timer expires after specified number of macroticks, expires and is restarted at each cycle start

26.5.2.56 Receive FIFO Watermark and Selection Register (FR_RFWMSR)

Base + 0x0086

Write: *WM_A/WM_B: POC:config, SEL: Anytime*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	WM_A/WM_B								0	0	0	0	0	0	0	SEL
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-57. Receive FIFO Watermark and Selection Register (FR_RFWMSR)

This register is used to

- select a receiver FIFO for subsequent programming access through the receiver FIFO configuration registers summarized in Table 26-64.
- to define the watermark for the selected FIFO.

Table 26-64. SEL Controlled Receiver FIFO Registers

Register
Receive FIFO Start Index Register (FR_RFSIR)
Receive FIFO Depth and Size Register (RFDSR)
Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)
Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)
Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)
Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)
Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)
Receive FIFO Range Filter Control Register (FR_RFRFCTR)

Table 26-65. FR_RFWMSR Field Descriptions

Field	Description
WM_A WM_B	**Watermark** — This field defines the watermark value for the selected FIFO. This value is used to control the generation of the almost full interrupt flags.
SEL	**Select** — This control bit selects the receiver FIFO for subsequent programming. 0 Receiver FIFO for channel A selected 1 Receiver FIFO for channel B selected

26.5.2.57 Receive FIFO Start Index Register (FR_RFSIR)

Base + 0x0088 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	$SIDX_A/SIDX_B$									
W	▨	▨	▨	▨	▨	▨										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-58. Receive FIFO Start Index Register (FR_RFSIR)

This register defines the message buffer header index of the first message buffer of the selected FIFO.

Table 26-66. FR_RFSIR Field Descriptions

Field	Description
$SIDX_A$ $SIDX_B$	**Start Index** — This field defines the number of the message buffer header field of the first message buffer of the selected FIFO. The CC uses the value of the SIDX field to determine the physical location of the receiver FIFO's first message buffer header field.

26.5.2.58 Receive FIFO Depth and Size Register (RFDSR)

Base + 0x008A Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	$FIFO\_DEPTH_A/FIFO\_DEPTH_B$								0	$ENTRY\_SIZE_A/ENTRY\_SIZE_B$						
W									▨							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-59. Receive FIFO Depth and Size Register (RFDSR)

This register defines the structure of the selected FIFO, i.e., the number of entries and the size of each entry.

Table 26-67. RFDSR Field Descriptions

Field	Description
$FIFO\_DEPTH_A$ $FIFO\_DEPTH_B$	**FIFO Depth** — This field defines the depth of the selected FIFO, i.e., the number of entries. **Note**: If the FIFO_DEPTH is configured to 0, FR_RFFIDRFMR[FIDRFMSK] must be configured to 0 too, to ensure that no frames are received into the FIFO.
$ENTRY\_SIZE_A$ $ENTRY\_SIZE_B$	**Entry Size** — This field defines the size of the frame data sections for the selected FIFO in 2 byte entities.

26.5.2.59 Receive FIFO A Read Index Register (FR_RFARIR)

Base + 0x008C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	RDIDX									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-60. Receive FIFO A Read Index Register (FR_RFARIR)

This register provides the message buffer header index of the next available FIFO A entry that the application can read.

Table 26-68. FR_RFARIR Field Descriptions

Field	Description
RDIDX	**Read Index** — This field provides the message buffer header index of the next available FIFO message buffer that the application can read.

NOTE

If the FIFO is empty, the RDIDX field points to an physical message buffer with invalid content.

26.5.2.60 Receive FIFO B Read Index Register (FR_RFBRIR)

Base + 0x008E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	RDIDX									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-61. Receive FIFO B Read Index Register (FR_RFBRIR)

This register provides the message buffer header index of the next available FIFO B entry that the application can read.

Table 26-69. FR_RFBRIR Field Descriptions

Field	Description
RDIDX	**Read Index** — This field provides the message buffer header index of the next available FIFO message buffer that the application can read.

NOTE

If the FIFO is empty, the RDIDX field points to an physical message buffer with invalid content.

26.5.2.61 Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)

Base + 0x00EE

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	FLB								FLA							
W	PCB								PCA							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-62. Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)

This register provides the current fill level of the two receiver FIFOs and is used to pop a number of entries from the FIFOs.

Table 26-70. FR_RFFLPCR Field Descriptions

Field	Description
FLB	**Fill Level FIFO B** — This field provides the current number of entries in the FIFO B.
FLA	**Fill Level FIFO A**— This field provides the current number of entries in the FIFO A.
PCB	**Pop Count FIFO B** — This field defines the number of entries to be removed from FIFO B.
PCA	**Pop Count FIFO A**— This field defines the number of entries to be removed from FIFO A.

NOTE

If the pop count value PCA/PCB is greater than the current FIFO fill level FLB/FLA, than the FIFO is empty after the update. No notification is given that not the required number of entries was removed.

26.5.2.62 Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)

Base + 0x0090 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	$\text{MIDAFVAL}_A/\text{MIDAFVAL}_B$															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-63. Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)

This register defines the filter value for the message ID acceptance filter of the selected FIFO. For details on message ID filtering see Section 26.6.9.9, “FIFO Filtering”.

Table 26-71. FR_RFMIDAFVR Field Descriptions

Field	Description
MIDAFVAL_A MIDAFVAL_B	**Message ID Acceptance Filter Value** — Filter value for the message ID acceptance filter.

26.5.2.63 Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)

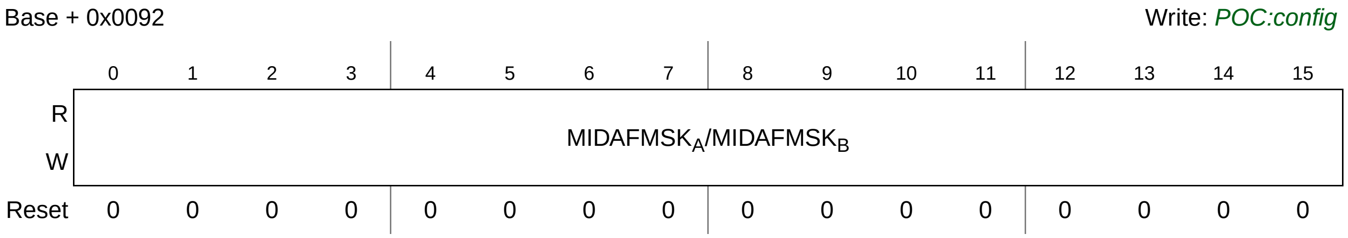


Figure 26-64. Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)

This register defines the filter mask for the message ID acceptance filter of the selected FIFO. For details on message ID filtering see Section 26.6.9.9, “FIFO Filtering”.

Table 26-72. FR_RFMIDAFMR Field Descriptions

Field	Description
MIDAFMSK$_A$ MIDAFMSK$_B$	**Message ID Acceptance Filter Mask** — Filter mask for the message ID acceptance filter.

26.5.2.64 Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)

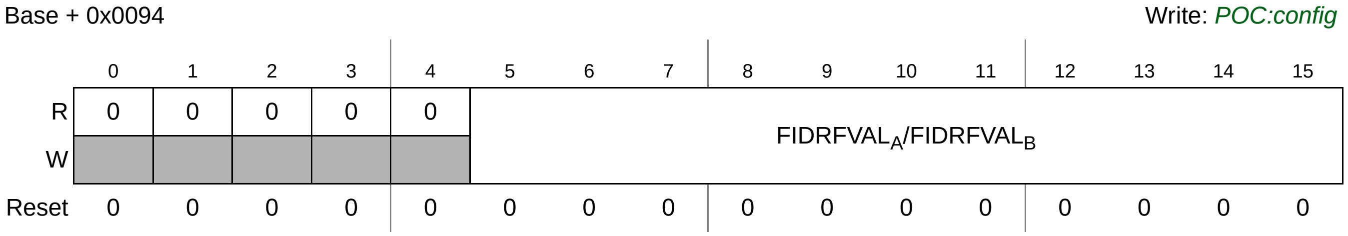


Figure 26-65. Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)

This register defines the filter value for the frame ID rejection filter of the selected FIFO. For details on frame ID filtering see Section 26.6.9.9, “FIFO Filtering”.

Table 26-73. FR_RFFIDRFVR Field Descriptions

Field	Description
FIDRFVAL$_A$ FIDRFVAL$_B$	**Frame ID Rejection Filter Value** — Filter value for the frame ID rejection filter.

26.5.2.65 Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)

Base + 0x0096 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	$\text{FIDRFMSK}_A/\text{FIDRFMSK}_B$										
W	▨	▨	▨	▨	▨											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-66. Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)

This register defines the filter mask for the frame ID rejection filter of the selected FIFO. For details on frame ID filtering see Section 26.6.9.9, “FIFO Filtering”.

Table 26-74. FR_RFFIDRFMR Field Descriptions

Field	Description
FIDRFMSK	**Frame ID Rejection Filter Mask** — Filter mask for the frame ID rejection filter.

26.5.2.66 Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)

Base + 0x0098 16-bit write access required Write: WMD, IBD, SEL: Any Time
SID: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	IBD	SEL		0	$\text{SID}_A/\text{SID}_B$										
W	WMD				▨											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-67. Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)

This register provides access to the four internal frame ID range filter boundary registers of the selected FIFO. For details on frame ID range filter see Section 26.6.9.9, “FIFO Filtering”.

Table 26-75. FR_RFRFCFR Field Descriptions

Field	Description
WMD	**Write Mode** — This control bit defines the write mode of this register. 0 Write to all fields in this register on write access. 1 Write to SEL and IBD field only on write access.
IBD	**Interval Boundary** — This control bit selects the interval boundary to be programmed with the SID value. 0 program lower interval boundary 1 program upper interval boundary

Table 26-75. FR_RFRFCFR Field Descriptions

Field	Description
SEL	**Filter Selector** — This control field selects the frame ID range filter to be accessed. 00 select frame ID range filter 0. 01 select frame ID range filter 1. 10 select frame ID range filter 2. 11 select frame ID range filter 3.
SID_A SID_B	**Slot ID —** Defines the IBD-selected frame ID boundary value for the SEL-selected range filter.

26.5.2.67 Receive FIFO Range Filter Control Register (FR_RFRFCTR)

Base + 0x009A Write: Anytime

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	F3MD	F2MD	F1MD	F0MD	0	0	0	0	F3EN	F2EN	F1EN	F0EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-68. Receive FIFO Range Filter Control Register (FR_RFRFCTR)

This register is used to enable and disable each frame ID range filter and to define whether it is running as acceptance or rejection filter.

Table 26-76. FR_RFRFCTR Field Descriptions (Sheet 1 of 2)

Field	Description
F3MD	**Range Filter 3 Mode** — This control bit defines the filter mode of the frame ID range filter 3. 0 range filter 3 runs as acceptance filter 1 range filter 3 runs as rejection filter
F2MD	**Range Filter 2 Mode** — This control bit defines the filter mode of the frame ID range filter 2. 0 range filter 2 runs as acceptance filter 1 range filter 2 runs as rejection filter
F1MD	**Range Filter 1 Mode** — This control bit defines the filter mode of the frame ID range filter 1. 0 range filter 1 runs as acceptance filter 1 range filter 1 runs as rejection filter
F0MD	**Range Filter 0 Mode** — This control bit defines the filter mode of the frame ID range filter 0. 0 range filter 0 runs as acceptance filter 1 range filter 0 runs as rejection filter
F3EN	**Range Filter 3 Enable** — This control bit is used to enable and disable the frame ID range filter 3. 0 range filter 3 disabled 1 range filter 3 enabled
F2EN	**Range Filter 2 Enable** — This control bit is used to enable and disable the frame ID range filter 2. 0 range filter 2 disabled 1 range filter 2 enabled

Table 26-76. FR_RFRFCTR Field Descriptions (Sheet 1 of 2)

Field	Description
F1EN	**Range Filter 1 Enable** — This control bit is used to enable and disable the frame ID range filter 1. 0 range filter 1 disabled 1 range filter 1 enabled
F0EN	**Range Filter 0 Enable** — This control bit is used to enable and disable the frame ID range filter 0. 0 range filter 0 disabled 1 range filter 0 enabled

26.5.2.68 Last Dynamic Transmit Slot Channel A Register (FR_LDTXSLAR)

Base + 0x009C

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	LASTDYNTXSLOTA										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-69. Last Dynamic Transmit Slot Channel A Register (FR_LDTXSLAR)

This register provides the number of the last transmission slot in the dynamic segment for channel A. This register is updated after the end of the dynamic segment and before the start of the next communication cycle.

Table 26-77. FR_LDTXSLAR Field Descriptions

Field	Description
LASTDYNTXSLOTA	**Last Dynamic Transmission Slot Channel A** — protocol related variable: *zLastDynTxSlot* channel A Number of the last transmission slot in the dynamic segment for channel A. If no frame was transmitted during the dynamic segment on channel A, the value of this field is set to 0.

26.5.2.69 Last Dynamic Transmit Slot Channel B Register (FR_LDTXSLBR)

Base + 0x009E

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	LASTDYNTXSLOTB										
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-70. Last Dynamic Transmit Slot Channel B Register (FR_LDTXSLBR)

This register provides the number of the last transmission slot in the dynamic segment for channel B. This register is updated after the end of the dynamic segment and before the start of the next communication cycle.

Table 26-78. FR_LDTXSLBR Field Descriptions

Field	Description
LASTDYNTX SLOTB	**Last Dynamic Transmission Slot Channel B** — protocol related variable: *zLastDynTxSlot* channel B Number of the last transmission slot in the dynamic segment for channel B. If no frame was transmitted during the dynamic segment on channel B the value of this field is set to 0.

26.5.2.70 Protocol Configuration Registers

The following configuration registers provide the necessary configuration information to the protocol engine. The individual values in the registers are described in Table 26-79. For more details about the FlexRay related configuration parameters and the allowed parameter ranges, see *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.

Table 26-79. Protocol Configuration Register Fields (Sheet 1 of 2)

Name	Description[1]	Min	Max	Unit	FR_PCR
coldstart_attempts	*gColdstartAttempts*			number	3
action_point_offset	*gdActionPointOffset* - 1			MT	0
cas_rx_low_max	*gdCASRxLowMax* - 1			*gdBit*	4
dynamic_slot_idle_phase	*gdDynamicSlotIdlePhase*			minislot	28
minislot_action_point_offset	*gdMinislotActionPointOffset* - 1			MT	3
minislot_after_action_point	*gdMinislot* - *gdMinislotActionPointOffset* - 1			MT	2
static_slot_length	*gdStaticSlot*			MT	0
static_slot_after_action_point	*gdStaticSlot* - *gdActionPointOffset* - 1			MT	13
symbol_window_exists	*gdSymbolWindow*!=0	0	1	bool	9
symbol_window_after_action_point	*gdSymbolWindow* - *gdActionPointOffset* - 1			MT	6
tss_transmitter	*gdTSSTransmitter*			*gdBit*	5
wakeup_symbol_rx_idle	*gdWakeupSymbolRxIdle*			*gdBit*	5
wakeup_symbol_rx_low	*gdWakeupSymbolRxLow*			*gdBit*	3
wakeup_symbol_rx_window	*gdWakeupSymbolRxWindow*			*gdBit*	4
wakeup_symbol_tx_idle	*gdWakeupSymbolTxIdle*			*gdBit*	8
wakeup_symbol_tx_low	*gdWakeupSymbolTxLow*			*gdBit*	5
noise_listen_timeout	(*gListenNoise* * *pdListenTimeout*) - 1			µT	16/17
macro_initial_offset_a	*pMacroInitialOffset[A]*			MT	6
macro_initial_offset_b	*pMacroInitialOffset[B]*			MT	16
macro_per_cycle	*gMacroPerCycle*			MT	10
macro_after_first_static_slot	*gMacroPerCycle* - *gdStaticSlot*			MT	1
macro_after_offset_correction	*gMacroPerCycle* - *gOffsetCorrectionStart*			MT	28
max_without_clock_correction_fatal	*gMaxWithoutClockCorrectionFatal*			cyclepairs	8
max_without_clock_correction_passive	*gMaxWithoutClockCorrectionPassive*			cyclepairs	8
minislot_exists	*gNumberOfMinislots*!=0	0	1	bool	9
minislots_max	*gNumberOfMinislots* - 1			minislot	29
number_of_static_slots	*gNumberOfStaticSlots*			static slot	2
offset_correction_start	*gOffsetCorrectionStart*			MT	11

Table 26-79. Protocol Configuration Register Fields (Sheet 1 of 2)

Name	Description[1]	Min	Max	Unit	FR_PCR
payload_length_static	*gPayloadLengthStatic*			2-bytes	19
max_payload_length_dynamic	*pPayloadLengthDynMax*			2-bytes	24
first_minislot_action_point_offset	max(*gdActionPointOffset*, *gdMinislotActionPointOffset*) - 1			MT	13
allow_halt_due_to_clock	*pAllowHaltDueToClock*			bool	26
allow_passive_to_active	*pAllowPassiveToActive*			cyclepairs	12
cluster_drift_damping	*pClusterDriftDamping*			µT	24
comp_accepted_startup_range_a	*pdAcceptedStartupRange* - *pDelayCompensation[A]*			µT	22
comp_accepted_startup_range_b	*pdAcceptedStartupRange* - *pDelayCompensation[B]*			µT	26
listen_timeout	*pdListenTimeout* - 1			µT	14/15
key_slot_id	*pKeySlotId*			number	18
key_slot_used_for_startup	*pKeySlotUsedForStartup*			bool	11
key_slot_used_for_sync	*pKeySlotUsedForSync*			bool	11
latest_tx	*gNumberOfMinislots* - *pLatestTx*			minislot	21
sync_node_max	*gSyncNodeMax*			number	30
micro_initial_offset_a	*pMicroInitialOffset[A]*			µT	20
micro_initial_offset_b	*pMicroInitialOffset[B]*			µT	20
micro_per_cycle	*pMicroPerCycle*			µT	22/23
micro_per_cycle_min	*pMicroPerCycle - pdMaxDrift*			µT	24/25
micro_per_cycle_max	*pMicroPerCycle + pdMaxDrift*			µT	26/27
micro_per_macro_nom_half	round(*pMicroPerMacroNom* / 2)			µT	7
offset_correction_out	*pOffsetCorrectionOut*			µT	9
rate_correction_out	*pRateCorrectionOut*			µT	14
single_slot_enabled	*pSingleSlotEnabled*			bool	10
wakeup_channel	*pWakeupChannel*	see Table 26-80			10
wakeup_pattern	*pWakeupPattern*			number	18
decoding_correction_a	*pDecodingCorrection* + *pDelayCompensation[A]* + 2			µT	19
decoding_correction_b	*pDecodingCorrection* + *pDelayCompensation[B]* + 2			µT	7
key_slot_header_crc	header CRC for key slot	0x000	0x7FF	number	12
extern_offset_correction	*pExternOffsetCorrection*			µT	29
extern_rate_correction	*pExternRateCorrection*			µT	21

[1] See *FlexRay Communications System Protocol Specification, Version 2.1 Rev A* for detailed protocol parameter definitions

Table 26-80. Wakeup Channel Selection

wakeup_channel	Wakeup Channel
0	A
1	B

26.5.2.70.1 Protocol Configuration Register 0 (FR_PCR0)

Base + 0x00A0 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	action_point_offset						static_slot_length									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-71. Protocol Configuration Register 0 (FR_PCR0)

26.5.2.70.2 Protocol Configuration Register 1 (FR_PCR1)

Base + 0x00A2 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	macro_after_first_static_slot													
W	▨	▨														
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-72. Protocol Configuration Register 1 (FR_PCR1)

26.5.2.70.3 Protocol Configuration Register 2 (FR_PCR2)

Base + 0x00A4 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	minislot_after_action_point						number_of_static_slots									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-73. Protocol Configuration Register 2 (FR_PCR2)

26.5.2.70.4 Protocol Configuration Register 3 (FR_PCR3)

Base + 0x00A6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	wakeup_symbol_rx_low						minislot_action_point_offset[4:0]					coldstart_attempts				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-74. Protocol Configuration Register 3 (FR_PCR3)

26.5.2.70.5 Protocol Configuration Register 4 (FR_PCR4)

Base + 0x00A8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	cas_rx_low_max							wakeup_symbol_rx_window								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-75. Protocol Configuration Register 4 (FR_PCR4)

26.5.2.70.6 Protocol Configuration Register 5 (FR_PCR5)

Base + 0x00AA Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	tss_transmitter				wakeup_symbol_tx_low						wakeup_symbol_rx_idle					
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-76. Protocol Configuration Register 5 (FR_PCR5)

26.5.2.70.7 Protocol Configuration Register 6 (FR_PCR6)

Base + 0x00AC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	symbol_window_after_action_point								macro_initial_offset_a						
W	▨															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-77. Protocol Configuration Register 6 (FR_PCR6)

26.5.2.70.8 Protocol Configuration Register 7 (FR_PCR7)

Base + 0x00AE Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	decoding_correction_b									micro_per_macro_nom_half						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-78. Protocol Configuration Register 7 (FR_PCR7)

26.5.2.70.9 Protocol Configuration Register 8 (FR_PCR8)

Base + 0x00B0 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	max_without_clock_ correction_fatal				max_without_clock_ correction_passive				wakeup_symbol_tx_idle							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-79. Protocol Configuration Register 8 (FR_PCR8)

26.5.2.70.10 Protocol Configuration Register 9 (FR_PCR9)

Base + 0x00B2 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	mini slot_ exists	sym bol_ win dow_ exists	offset_correction_out													
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-80. Protocol Configuration Register 9 (FR_PCR9)

26.5.2.70.11 Protocol Configuration Register 10 (FR_PCR10)

Base + 0x00B4 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	single _slot _en abled	wake up_ chan nel	macro_per_cycle													
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-81. Protocol Configuration Register 10 (FR_PCR10)

26.5.2.70.12 Protocol Configuration Register 11 (FR_PCR11)

Base + 0x00B6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	key_ slot_ used_ for_ start up	key_ slot_ used_ for_ sync	offset_correction_start													
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-82. Protocol Configuration Register 11 (FR_PCR11)

26.5.2.70.13 Protocol Configuration Register 12 (FR_PCR12)

Base + 0x00B8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	allow_passive_to_active					key_slot_header_crc										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-83. Protocol Configuration Register 12 (FR_PCR12)

26.5.2.70.14 Protocol Configuration Register 13 (FR_PCR13)

Base + 0x00BA Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	first_minislot_action_point_offset						static_slot_after_action_point									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-84. Protocol Configuration Register 13 (FR_PCR13)

26.5.2.70.15 Protocol Configuration Register 14 (FR_PCR14)

Base + 0x00BC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	rate_correction_out											listen_timeout[20:16]				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-85. Protocol Configuration Register 14 (FR_PCR14)

26.5.2.70.16 Protocol Configuration Register 15 (FR_PCR15)

Base + 0x00BE Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	listen_timeout[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-86. Protocol Configuration Register 15 (FR_PCR15)

26.5.2.70.17 Protocol Configuration Register 16 (FR_PCR16)

Base + 0x00C0 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	macro_initial_offset_b							noise_listen_timeout[24:16]								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-87. Protocol Configuration Register 16 (FR_PCR16)

26.5.2.70.18 Protocol Configuration Register 17 (FR_PCR17)

Base + 0x00C2 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	noise_listen_timeout[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-88. Protocol Configuration Register 17 (FR_PCR17)

26.5.2.70.19 Protocol Configuration Register 18 (FR_PCR18)

Base + 0x00C4 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	wakeup_pattern						key_slot_id									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-89. Protocol Configuration Register 18 (FR_PCR18)

26.5.2.70.20 Protocol Configuration Register 19 (FR_PCR19)

Base + 0x00C6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	decoding_correction_a									payload_length_static						
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-90. Protocol Configuration Register 19 (FR_PCR19)

26.5.2.70.21 Protocol Configuration Register 20 (FR_PCR20)

Base + 0x00C8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	micro_initial_offset_b								micro_initial_offset_a							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-91. Protocol Configuration Register 20 (FR_PCR20)

26.5.2.70.22 Protocol Configuration Register 21 (FR_PCR21)

Base + 0x00CA Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	extern_rate_correction			latest_tx												
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-92. Protocol Configuration Register 21 (FR_PCR21)

26.5.2.70.23 Protocol Configuration Register 22 (FR_PCR22)

Base + 0x00CC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	R*	comp_accepted_startup_range_a											micro_per_cycle[19:16			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-93. Protocol Configuration Register 22 (FR_PCR22)

26.5.2.70.24 Protocol Configuration Register 23 (FR_PCR23)

Base + 0x00CE Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	micro_per_cycle[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-94. Protocol Configuration Register 23 (FR_PCR23)

26.5.2.70.25 Protocol Configuration Register 24 (FR_PCR24)

Base + 0x00D0 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	cluster_drift_damping					max_payload_length_dynamic							micro_per_cycle_min [19:16]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-95. Protocol Configuration Register 24 (FR_PCR24)

26.5.2.70.26 Protocol Configuration Register 25 (FR_PCR25)

Base + 0x00D2 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	micro_per_cycle_min[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-96. Protocol Configuration Register 25 (FR_PCR25)

26.5.2.70.27 Protocol Configuration Register 26 (FR_PCR26)

Base + 0x00D4 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	allow_halt_due_to_clock	comp_accepted_startup_range_b											micro_per_cycle_max [19:16]			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-97. Protocol Configuration Register 26 (FR_PCR26)

26.5.2.70.28 Protocol Configuration Register 27 (FR_PCR27)

Base + 0x00D6 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	micro_per_cycle_max[15:0]															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-98. Protocol Configuration Register 27 (FR_PCR27)

26.5.2.70.29 Protocol Configuration Register 28 (FR_PCR28)

Base + 0x00D8 Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	dynamic_slot_idle_phase		macro_after_offset_correction													
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-99. Protocol Configuration Register 28 (FR_PCR28)

26.5.2.70.30 Protocol Configuration Register 29 (FR_PCR29)

Base + 0x00DA Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	extern_offset_correction			minislots_max												
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-100. Protocol Configuration Register 29 (FR_PCR29)

26.5.2.70.31 Protocol Configuration Register 30 (FR_PCR30)

Base + 0x00DC Write: *POC:config*

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	sync_node_max			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-101. Protocol Configuration Register 30 (FR_PCR30)

26.5.2.71 ECC Error Interrupt Flag and Enable Register (FR_EEIFER)

Base + 0x00F0 Write: Normal Mode

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	LRNE_OF	LRCE_OF	DRNE_OF	DRCE_OF	LRNE_IF	LRCE_IF	DRNE_IF	DRCE_IF	0	0	0	0	LRNE_IE	LRCE_IE	DRNE_IE	DRCE_IE
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-102. ECC Error Interrupt Flag and Enable Register (FR_EEIFER)

This register provides the means to control the ECC related interrupt request lines and provides the corresponding interrupt flags. The interrupt flags are cleared by writing 1, which resets the corresponding report registers. For a detailed description see Section 26.6.24.2, “Memory Error Reporting”.

Table 26-81. FR_EEIFER Field Descriptions

Field	Description
Error Overflow Flags	
LRNE_OF	**LRAM Non-Corrected Error Overflow Flag** — This flag is set to 1 when at least one of the following events appears: a) memory errors are detected *but not corrected* on CHI LRAM and interrupt flag LRNE_IF is already 1. b) memory errors are detected *but not corrected* on at least two banks of CHI LRAM 0 no such event 1 Non-Corrected Error overflow detected on CHI LRAM
LRCE_OF	**LRAM Corrected Error Overflow Flag** — This flag is set to 1 when at least one of the following events appears: a) memory errors are detected *and corrected* on CHI LRAM and interrupt flag LRCE_IF is already 1. b) memory errors are detected *and corrected* on at least two banks of CHI LRAM 0 no such event 1 Corrected Error overflow detected on CHI LRAM Note: Error Correction not implemented on CHI LRAM, flag will never be asserted.
DRNE_OF	**DRAM Non-Corrected Error Overflow Flag** — This flag is set to 1 when at least one of the following events appears: a) memory errors are detected *but not corrected* on PE DRAM and interrupt flag DRNE_IF is already 1. b) memory errors are detected *but not corrected* on at least two banks of the PE DRAM 0 no such event 1 Non-Corrected Error overflow detected on PE DRAM
DRCE_OF	**DRAM Corrected Error Overflow Flag** — This flag is set to 1 when at least one of the following events appears: a) memory errors are detected *and corrected* on PE DRAM and interrupt flag DRCE_IF is already 1. b) memory errors are detected *and corrected* on at least two banks of PE DRAM 0 no such event 1 Corrected Error overflow detected on PE DRAM

Table 26-81. FR_EEIFER Field Descriptions

Field	Description
Error Interrupt Flags	
LRNE_IF	**LRAM Non-Corrected Error Interrupt Flag** — This interrupt flag is set to 1 when a memory error is *detected but not corrected* on the CHI LRAM. 0 no such event 1 Non-Corrected Error detected on CHI LRAM
LRCE_IF	**LRAM Corrected Error Interrupt Flag** — This interrupt flag is set to 1 when a memory error is *detected and corrected* on the CHI LRAM. 0 no such event 1 Corrected Error detected on CHI LRAM Note: Error Correction not implemented on CHI LRAM, flag will never be asserted.
DRNE_IF	**DRAM Non-Corrected Error Interrupt Flag** — This interrupt flag is set to 1 when a memory error is *detected but not corrected* on PE DRAM. 0 no such event 1 Non-Corrected Error detected on PE DRAM
DRCE_IF	**DRAM Corrected Error Interrupt Flag** — This interrupt flag is set to 1 when a memory error is *detected and corrected* on PE DRAM. 0 no such event 1 Corrected Error detected on PE DRAM
Error Interrupt Enables	
LRNE_IE	**LRAM Non-Corrected Error Interrupt Enable** — This flag controls if the LRAM Non-Corrected Error Interrupt line is asserted when the LRNE_IF flag is set. 0 Disable interrupt line 1 Enable interrupt line
LRCE_IE	**LRAM Corrected Error Interrupt Enable** — This flag controls if the LRAM Corrected Error Interrupt line is asserted when the LRCE_IF flag is set. 0 Disable interrupt line 1 Enable interrupt line
DRNE_IE	**DRAM Non-Corrected Error Interrupt Enable** — This flag controls if the DRAM Non-Corrected Error Interrupt line is asserted when the DRNE_IF flag is set. 0 Disable interrupt line 1 Enable interrupt line
DRCE_IE	**DRAM Corrected Error Interrupt Enable** — This flag controls if the DRAM Corrected Error Interrupt line is asserted when the DRCE_IF flag is set. 0 Disable interrupt line 1 Enable interrupt line

26.5.2.72 ECC Error Report and Injection Control Register (FR_EERICR)

Base + 0x00F2

Write: ERS: Anytime
ERM, EIM, EIE: IDL

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	BSY	0	0	0	0	0	ERS		0	0	0	ERM	0	0	EIM	EIE
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-103. ECC Error Report and Injection Control Register (FR_EERICR)

This register configures the error injection and error reporting and provides the selector for the content of the report registers.

Table 26-82. FR_EERICR Field Descriptions

Field	Description
BSY	**Register Update Busy**— This field indicates the current state of the ECC configuration update and controls the register write access condition IDL specified in "Section 26.5.2.2, "Register Write Access" 0 ECC configuration is idle 1 ECC configuration is running
ERS	**Error Report Select** — This field selects the content of the ECC Error reporting registers. 00 show PE DRAM non-corrected error information 01 show PE DRAM corrected error information 10 show CHI LRAM non-corrected error information 11 show CHI LRAM corrected error information
ERM	**Error Report Mode** — This bit configures the type of data written into the internal error report registers on the detection of a memory error. 0 store data and code as delivered by ecc decoding logic. 1 store data and code as read from the memory.
EIM	**Error Injection Mode** — This bit configures the ECC error injection mode. 0 use FR_EEIDR[DATA] and FR_EEICR[CODE] as XOR distortion pattern for error injection. 1 use FR_EEIDR[DATA] and FR_EEICR[CODE] as write value for error injection.
EIE	**Error Injection Enable** — This bit configures the ECC error injection on the memories. 0 Error injection disabled 1 Error injection enabled

26.5.2.73 ECC Error Report Address Register (FR_EERAR)

Base + 0x00F4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MID	BANK			ADDR											
W																
Reset	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-104. ECC Error Report Address Register (FR_EERAR)

This register provides the memory identifier, bank, and address for which the memory error is reported.

Table 26-83. FR_EERAR Field Descriptions

Field	Description
MID	**Memory Identifier** — This flag provides the memory instance for which the memory error is reported. 0 PE DRAM 1 CHI LRAM
BANK	**Memory Bank** — This field provides the BANK for which the memory error is reported. 111 reset value, indicates no error found after reset. For MID=0: 000 PE DRAM [7:0] 001 PE DRAM [15:8] others - not used For MID=1: Refer to Table 26-84 for the assignment of the LRAM banks.
ADDR	**Memory Address** — This field provides the address of the failing memory location.

Table 26-84. LRAM Bank Value for MID = 1

BANK	Register		
000	FR_MBCCFR(2n)	FR_MBDOR(6n)	FR_LEETR0
001	FR_MBFIDR(2n)	FR_MBDOR(6n + 1)	FR_LEETR1
010	FR_MBIDXR(2n)	FR_MBDOR(6n + 2)	FR_LEETR2
011	FR_MBCCFR(2n+1)	FR_MBDOR(6n + 3)	FR_LEETR3
100	FR_MBFIDR(2n+1)	FR_MBDOR(6n + 4)	FR_LEETR4
101	FR_MBIDXR(2n+1)	FR_MBDOR(6n + 5)	FR_LEETR5
110	Not Used	Not Used	Not Used
111			

26.5.2.74 ECC Error Report Data Register (FR_EERDR)

Base + 0x00F6

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	DATA															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-105. ECC Error Report Data Register (FR_EERDR)

This register provides the data related information of the reported memory read access. The assignment of the bits depends on the selected memory and memory bank as shown in Table 26-86.

Table 26-85. FR_EERDR Field Descriptions

Field	Description
DATA	**Data** — The content of this field depends on the report mode selected by FR_EERICR[ERM] ERM=0: Ecc Data, shows data as generated by the ecc decoding logic. ERM=1: Memory Data, shows data as read from the memory.

Table 26-86. Valid Bits in FR_EERDR[DATA] / FR_EEIDR[DATA] field

MEM	BANK	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PE DRAM	0	▨	▨	▨	▨	▨	▨	▨	▨	PE DRAM[7:0]							
PE DRAM	1	▨	▨	▨	▨	▨	▨	▨	▨	PE DRAM[15:8]							
CHI LRAM	0	FR_MBCCFR(2n)															
CHI LRAM	0	FR_MBDOR(6n)															
CHI LRAM	0	FR_LEETR0															
CHI LRAM	1	▨	▨	▨	▨	▨	FR_MBFIDR(2n)[FID]										
CHI LRAM	1	FR_MBDOR(6n+1)															
CHI LRAM	1	FR_LEETR1															
CHI LRAM	2	▨	▨	▨	▨	▨	▨	▨	▨	FR_MBIDXR(2n)[MBIDX]							
CHI LRAM	2	FR_MBDOR(6n+2)															
CHI LRAM	2	FR_LEETR2															
CHI LRAM	3	FR_MBCCFR(2n+1)															
CHI LRAM	3	FR_MBDOR(6n+3)															
CHI LRAM	3	FR_LEETR3															
CHI LRAM	4	▨	▨	▨	▨	▨	FR_MBFIDR(2n+1)[FID]										
CHI LRAM	4	FR_MBDOR(6n+4)															
CHI LRAM	4	FR_LEETR4															
CHI LRAM	5	▨	▨	▨	▨	▨	▨	▨	▨	FR_MBIDXR(2n+1)[MBIDX]							
CHI LRAM	5	FR_MBDOR(6n+5)															
CHI LRAM	5	FR_LEETR5															

26.5.2.75 ECC Error Report Code Register (FR_EERCR)

Base + 0x00F8

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	CODE				
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-106. ECC Error Report Code Register (FR_EERCR)

This register provides the ecc related information of the reported memory read access.

Table 26-87. FR_EERSR Field Descriptions

Field	Description
CODE	**Code** — The content of this field depends on the report mode selected by FR_EERICR[ERM] ERM=0: Syndrome. Shows the ecc syndrome generated by the ecc decoding logic. The coding of the PE DRAM syndrome is shown in Section 26.6.24.2.2, “PE DRAM Syndrome” The coding of the CHI LRAM syndrome is shown in Section 26.6.24.2.4, “CHI LRAM Syndrome”. ERM=1: Checkbits. Shows the ecc checkbits read from the memory.

26.5.2.76 ECC Error Injection Address Register (FR_EEIAR)

Base + 0x00FA Write: IDL

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	MID	BANK			ADDR											
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-107. ECC Error Injection Address Register (FR_EEIAR)

This register defines the memory module, bank, and address where the ECC error has to be injected.

Table 26-88. FR_EEIAR Field Descriptions

Field	Description
MID	**Memory Identifier** — This flag defines the memory instance for ECC error injection. 0 PE DRAM 1 CHI LRAM
BANK	**Memory Bank** — This field defines the memory bank for ECC error injection. For MID=0: 000 BANK0: PE DRAM [7:0] 001 BANK1: PE DRAM [15:8] others reserved For MID=1: Refer to Table 26-84 for the assignment of the LRAM banks.
ADDR	**Memory Address** — This flag defines the memory address for ECC error injection.

26.5.2.77 ECC Error Injection Data Register (FR_EEIDR)

Base + 0x00FC Write: IDL

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	DATA															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-108. ECC Error Injection Data Register (FR_EEIDR)

This register defines the data distortion pattern for the error injection write. The number of valid bits depends on the selected memory and memory bank as shown in Table 26-86.

Table 26-89. FR_EEIDR Field Descriptions

Field	Description
DATA	**Data** — The content of this field depends on the error injection mode selected by FR_EERICR[EIM]. EIM=0: This field defines the XOR distortion pattern for the data written into the memory. EIM=1: This field defines the data to be written into the memory.

NOTE

The effect of the error injected depends from the LRAM content at the address accessed and from the module internal usage of the data. Refer to Section 26.6.24.3, “Memory Error Response” for details.

26.5.2.78 ECC Error Injection Code Register (FR_EEICR)

Base + 0x00FE Write: IDL

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	CODE				
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-109. ECC Error Injection Code Register (FR_EEICR)

This register defines the ecc code distortion pattern for the error injection write.

Table 26-90. FR_EEICR Field Descriptions

Field	Description
CODE	**Code** — The content of this field depends on the error injection mode selected by FR_EERICR[EIM]. EIM=0: This field defines the XOR distortion pattern for the ecc checkbits written into the memory. EIM=1: This field defines the ecc checkbits written into the memory.

26.5.2.79 Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)

Base + 0x0800 (FR_MBCCSR0)
Base + 0x0808 (FR_MBCCSR1)
...
Base + 0x0BF8 (FR_MBCCSR127)

Write: MTD: *POC:config* or MB_DIS
CMT: MB_LCK or MB_DIS
EDT, LCKT, MBIE, MBIF: Normal Mode

Additional Reset: CMT, DUP, DVAL, MBIF: Message Buffer Disable

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	MTD	CMT	0	0	MBIE	0	0	0	DUP	DVAL	EDS	LCKS	MBIF
W					rwm	EDT	LCKT									w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-110. Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)

The content of these registers comprises message buffer configuration data, message buffer control data, message buffer status information, and message buffer interrupt flags. A detailed description of all flags can be found in Section 26.6.6, “Individual Message Buffer Functional Description”

If the application writes 1 to the EDT bit, no write access to the other register bits is performed.

If the application writes 0 to the EDT bit and 1 to the LCKT bit, no write access to the other bits is performed.

Table 26-91. FR_MBCCSRn Field Descriptions (Sheet 1 of 2)

Field	Description
Message Buffer Configuration	
MTD	**Message Buffer Transfer Direction** — This bit configures the transfer direction of a message buffer. 0 Receive message buffer 1 Transmit message buffer
Message Buffer Control	
CMT	**Commit for Transmission** — This bit indicates if the transmit message buffer data are ready for transmission. 0 Message buffer data not ready for transmission 1 Message buffer data ready for transmission
EDT	**Enable/Disable Trigger** — If the application writes 1 to this bit, a message buffer enable or disable is triggered, depending on the current value of the EDS status bit. 0 No effect 1 Message buffer enable or disable is triggered
LCKT	**Lock/Unlock Trigger** — If the application writes 1 to this bit and writes 0 to the EDT bit, a message buffer lock or unlock is triggered, depending on the current value of the LCKS status bit. 0 No effect 1 Message buffer lock or unlock is triggered
MBIE	**Message Buffer Interrupt Enable** — This control bit defines whether the message buffer will generate an interrupt request when its MBIF flag is set. 0 Interrupt request generation disabled 1 Interrupt request generation enabled

Table 26-91. FR_MBCCSRn Field Descriptions (Sheet 1 of 2)

Field	Description
Message Buffer Status	
DUP	**Data Updated** — This status bit indicates whether the frame header in the message buffer header field and the data in the message buffer data field were updated after a frame reception. 0 Frame Header and Message buffer data field not updated 1 Frame Header and Message buffer data field updated
DVAL	**Data Valid** — For receive message buffers this status bit indicates whether the message buffer data field contains valid frame data. For transmit message buffers the status bit indicates if a message is transferred again due to the state transmission mode of the message buffer. 0 receive message buffer contains no valid frame data / message is transmitted for the first time 1 receive message buffer contains valid frame data / message will be transferred again
EDS	**Enable/Disable Status** — This status bit indicates whether the message buffer is enabled or disabled. 0 Message buffer is disabled. 1 Message buffer is enabled.
LCKS	**Lock Status** — This status bit indicates the current lock status of the message buffer. 0 Message buffer is not locked by the application. 1 Message buffer is locked by the application.
MBIF	**Message Buffer Interrupt Flag** — This flag is set when the slot status field of the message buffer was updated after frame transmission or reception, or when a transmit message buffer was just enabled by the application. 0 No such event 1 Slot status field updated or transmit message buffer just enabled

26.5.2.80 Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)

Base + 0x0802 (FR_MBCCFR0)	16-bit write access required	Write: *POC:config* or MB_DIS
Base + 0x080A (FR_MBCCFR1)		
...		
Base + 0x0BFA (FR_MBCCFR127)		

<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr>
<tr><td>R
W</td><td>MTM</td><td>CHA</td><td>CHB</td><td>CCFE</td><td colspan="6">CCFMSK</td><td colspan="6">CCFVAL</td></tr>
<tr><td>Reset</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td><td>-</td></tr>
</table>

Figure 26-111. Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)

This register contains message buffer configuration data for the transmission mode, the channel assignment, and for the cycle counter filtering. For detailed information on cycle counter filtering, refer to Section 26.6.7.1, "Message Buffer Cycle Counter Filtering".

Table 26-92. FR_MBCCFRn Field Descriptions

Field	Description
MTM	**Message Buffer Transmission Mode** — This control bit applies only to transmit message buffers and defines the transmission mode. 0 Event transmission mode 1 State transmission mode
CHA CHB	**Channel Assignment** — These control bits define the channel assignment and control the receive and transmit behavior of the message buffer according to **Table 26-93**.
CCFE	**Cycle Counter Filtering Enable** — This control bit is used to enable and disable the cycle counter filtering. 0 Cycle counter filtering disabled 1 Cycle counter filtering enabled
CCFMSK	Cycle Counter Filtering Mask — This field defines the filter mask for the cycle counter filtering.
CCFVAL	**Cycle Counter Filtering Value** — This field defines the filter value for the cycle counter filtering.

Table 26-93. Channel Assignment Description

CHA	CHB	Transmit Message Buffer		Receive Message Buffer	
		static segment	dynamic segment	static segment	dynamic segment
1	1	transmit on both channel A and channel B	Reserved (Function not available)	store first valid frame received on either channel A or channel B	Reserved (Function not available)
0	1	transmit on channel B	transmit on channel B	store first valid frame received on channel B	store first valid frame received on channel B
1	0	transmit on channel A	transmit on channel A	store first valid frame received on channel A	store first valid frame received on channel A
0	0	no frame transmission	no frame transmission	no frame stored	no frame stored

NOTE

If at least one message buffer assigned to a certain slot is assigned to both channels, then all message buffers assigned to this slot have to be assigned to both channels. Otherwise, the message buffer configuration is illegal and the result of the message buffer search is not defined.

26.5.2.81 Message Buffer Frame ID Registers (FR_MBFIDRn)

Base + 0x0804 (FR_MBFIDR0) Base + 0x080C (FR_MBFIDR1) ... Base + 0x0BFC (FR_MBFIDR127)	16-bit write access required	Write: *POC:config* or MB_DIS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	FID										
W																
Reset	0	0	0	0	0	-	-	-	-	-	-	-	-	-	-	-

Figure 26-112. Message Buffer Frame ID Registers (FR_MBFIDRn)

Table 26-94. FR_MBFIDRn Field Descriptions

Field	Description
FID	**Frame ID** — The semantic of this field depends on the message buffer transfer type. • *Receive Message Buffer:* This field is used as a filter value to determine if the message buffer is used for reception of a message received in a slot with the slot ID equal to FID. • *Transmit Message Buffer:* This field is used to determine the slot in which the message in this message buffer should be transmitted.

26.5.2.82 Message Buffer Index Registers (FR_MBIDXRn)

Base + 0x0806 (FR_MBIDXR0) Base + 0x080E (FR_MBIDXR1) ... Base + 0x0BFE (FR_MBIDXR127)	16-bit write access required	Write: *POC:config* or MB_DIS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	MBIDX							
W																
Reset	0	0	0	0	0	0	0	0	-	-	-	-	-	-	-	-

Figure 26-113. Message Buffer Index Registers (FR_MBIDXRn)

Table 26-95. FR_MBIDXRn Field Descriptions

Field	Description
MBIDX	**Message Buffer Index** — This field provides the index of the message buffer header field of the physical message buffer that is currently associated with this message buffer. The application writes the index of the initially associated message buffer header field into this register. The CC updates this register after frame reception or transmission. Legal Values are 0 <= i <= 131. Illegal values will be detected during the message buffer search.

26.5.2.83 Message Buffer Data Field Offset Registers (FR_MBDORn)

Base + 0x1000 (FR_MBDOR0) 16-bit write access required Write: *POC:config*
Base + 0x1002 (FR_MBDOR1)
...
Base + 0x1106 (FR_MBDOR131)

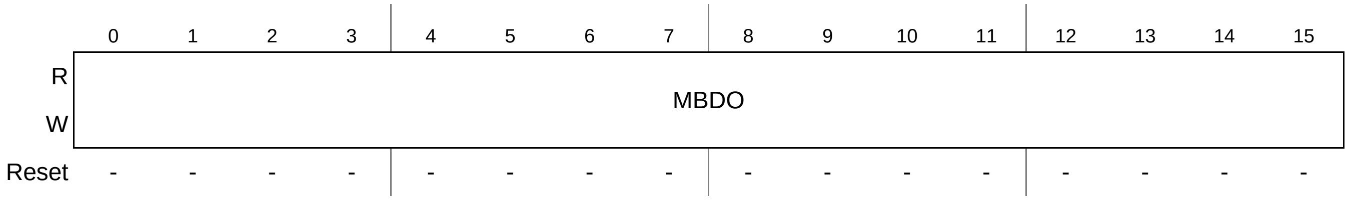


Figure 26-114. Message Buffer Data Field Offset Registers (FR_MBDORn)

Table 26-96. FR_MBDORn Field Descriptions

Field	Description
MBDO	**Message Buffer Data Field Offset** — This field provides the data field offset belonging to a particular Message Buffer Index. For configuration constraints see Section 26.7.1.2, “Configure Data Field Offsets”.

26.5.2.84 LRAM ECC Error Test Registers (FR_LEETRn)

Base + 0x1108 (FR_LEETR0) 16-bit write access required Write: Anytime
...
Base + 0x1112 (FR_LEETR5)

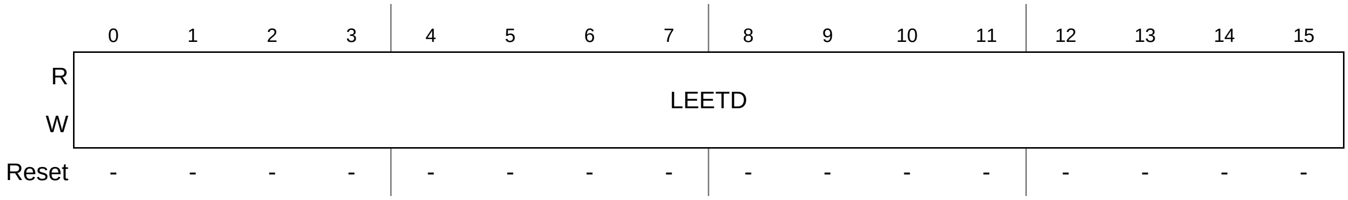


Figure 26-115. LRAM ECC Error Test Registers (FR_LEETRn)

Table 26-97. FR_LEETRn Field Descriptions

Field	Description
LEETD	**LRAM ECC Error Test Data** — This field contains the LRAM data belonging to the test register located in LRAM Bank n.

26.6 Functional Description

This section provides a detailed description of the functionality implemented in the CC.

26.6.1 Message Buffer Concept

The CC uses a data structure called *message buffer* to store frame data, configuration, control, and status data. Each message buffer consists of two parts, the *message buffer control data* and the *physical message buffer.* The message buffer control data are located in dedicated registers. The structure of the message buffer control data depends on the message buffer type and is described in Section 26.6.3, “Message Buffer Types”. The physical message buffer is located in the flexray memory area and is described in Section 26.6.2, “Physical Message Buffer”.

26.6.2 Physical Message Buffer

All FlexRay messages and related frame and slot status information of received frames and of frames to be transmitted to the FlexRay bus are stored in data structures called *physical message buffers*. The physical message buffers are located in the flexray memory area.The structure of a physical message buffer is depicted in Figure 26-116.

A physical message buffer consists of two fields, the *message buffer header field* and the *message buffer data field*. The message buffer header field contains the *frame header* and the *slot status*.The message buffer data field contains the *frame data*.

The connection between the two fields is established by the *data field offset*.

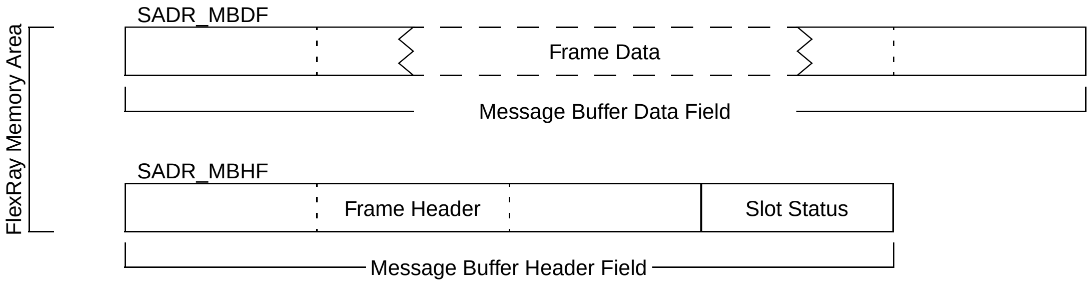


Figure 26-116. Physical Message Buffer Structure

26.6.2.1 Message Buffer Header Field

The message buffer header field is a contiguous region in the flexray memory area and occupies eight bytes. It contains the frame header, and the slot status. Its structure is shown in Figure 26-116. The physical start address *SADR_MBHF* of the message buffer header field must be 16-bit aligned.

26.6.2.1.1 Frame Header

The frame header occupies the first six bytes in the message buffer header field. It contains all FlexRay frame header related information according to the *FlexRay Communications System Protocol*

Specification, Version 2.1 Rev A. A detailed description of the usage and the content of the frame header is provided in Section 26.6.5.2.1, "Frame Header Description".

26.6.2.1.2 Slot Status

The slot status occupies the last two bytes of the message buffer header field. It provides the slot and frame status related information according to the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*. A detailed description of the content and usage of the slot status is provided in Section 26.6.5.2.2, "Slot Status Description".

26.6.2.2 Message Buffer Data Field

The message buffer data field is a contiguous area of 2-byte entities. This field contains the frame payload data, or a part of it, of the frame to be transmitted to or received from the FlexRay bus. The minimum length of this field depends on the specific message buffer configuration and is specified in the message buffer descriptions given in Section 26.6.3, "Message Buffer Types".

26.6.3 Message Buffer Types

The CC provides three different types of message buffers.

- Individual Message Buffers
- Receive Shadow Buffers
- Receive FIFO Buffers

For each message buffer type the structure of the physical message buffer is identical. The message buffer types differ only in the structure and content of message buffer control data, which control the related physical message buffer. The message buffer control data are described in the following sections.

26.6.3.1 Individual Message Buffers

The individual message buffers are used for all types of frame transmission and for dedicated frame reception based on individual filter settings for each message buffer. The CC supports three types of individual message buffers, which are described in Section 26.6.6, "Individual Message Buffer Functional Description".

Each individual message buffer consists of two parts, the physical message buffer, which is located in the flexray memory area, and the message buffer control data, which are located in dedicated registers. The structure of an individual message buffer is given in Figure 26-117.

Each individual message buffer has a message buffer number n assigned, which determines the set of message buffer control registers associated to this individual message buffer. The individual message buffer with message buffer number *n* is controlled by the registers FR_MBCCSRn, FR_MBCCFRn, FR_MBFIDRn, and FR_MBIDXRn.

The connection between the message buffer control registers and the physical message buffer is established by the message buffer index field MBIDX in the Message Buffer Index Registers

(FR_MBIDXRn). The start address SADR_MBHF of the related message buffer header field in the flexray memory area is determined according to Equation 1.

$$\text{SADR\_MBHF} = (\text{FR\_MBIDXRn[MBIDX]} * 8) + \text{SMBA} \qquad \textit{Eqn. 1}$$

The data field belonging to a particular physical message buffer is characterized by the data field offset. For each physical message buffer with MBIDX i the FR_MBDORi contains the offset of the corresponding message buffer data field with respect to the CC flexray memory area base address as provided by SMBA field in the System Memory Base Address Register (FR_SYMBADR)".

The data field offset is used to determine the start address *SADR_MBDF* of the corresponding message buffer data field in the flexray memory area according to Equation 2.

$$\text{SADR\_MBDF} = [\text{Data Field Offset}] + \text{SMBA} \qquad \textit{Eqn. 2}$$

The FR_MBDORn are stored in the module internal memory LRAM. Refer to Section 26.7.2.3, "CHI LRAM Initialization" for the setup of the data field offset values.

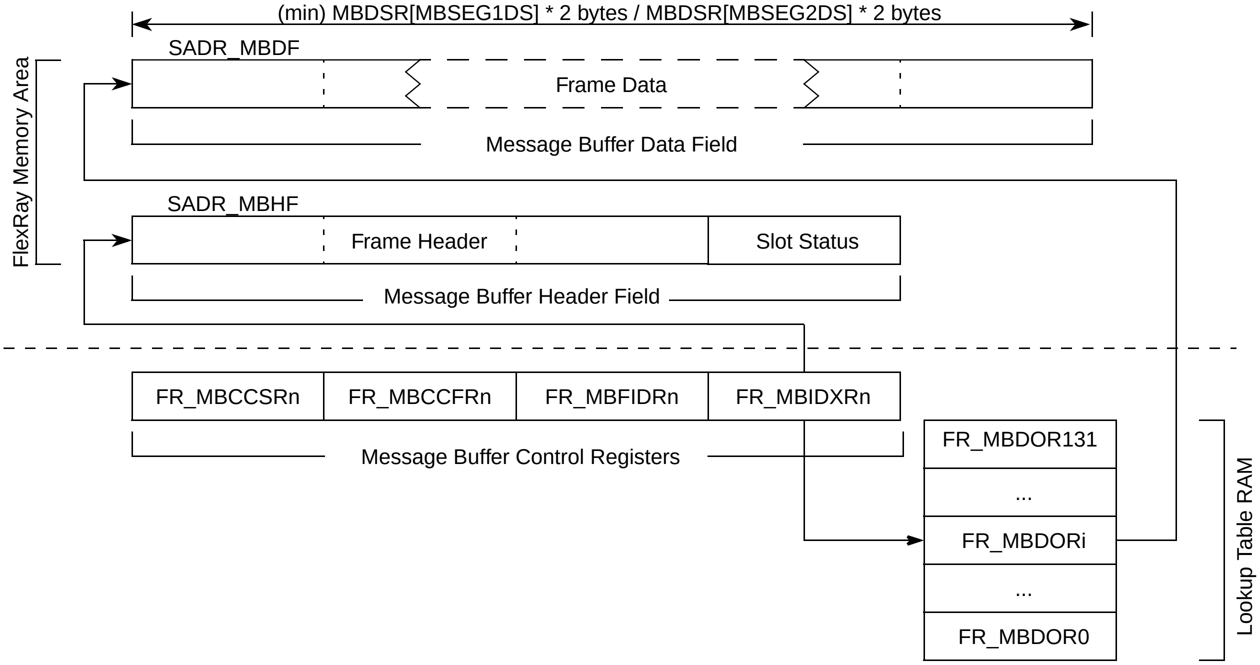


Figure 26-117. Individual Message Buffer Structure

26.6.3.1.1 Individual Message Buffer Segments

The set of the individual message buffers can be split up into two message buffer segments using the Message Buffer Segment Size and Utilization Register (FR_MBSSUTR). All individual message buffers with a message buffer number n <= FR_MBSSUTR[LAST_MB_SEG1] belong to the first message buffer segment. All individual message buffers with a message buffer number n > FR_MBSSUTR[LAST_MB_SEG1] belong to the second message buffer segment. The following rules apply to the length of the message buffer data field:

- all physical message buffers associated to individual message buffers that belong to the same message buffer segment must have message buffer data fields of the same length
- the minimum length of the message buffer data field for individual message buffers in the first message buffer segment is 2 * FR_MBDSR[MBSEG1DS] bytes
- the minimum length of the message buffer data field for individual message buffers assigned to the second segment is 2 * FR_MBDSR[MBSEG2DS] bytes.

26.6.3.2 Receive Shadow Buffers

The receive shadow buffers are required for the frame reception process for individual message buffers. The CC provides four receive shadow buffers, one receive shadow buffer per channel and per message buffer segment.

Each receive shadow buffer consists of two parts, the physical message buffer located in the flexray memory area and the receive shadow buffer control registers located in dedicated registers. The structure of a receive shadow buffer is shown in Figure 26-118. The four internal shadow buffer control registers can be accessed by the Receive Shadow Buffer Index Register (FR_RSBIR).

The connection between the receive shadow buffer control register and the physical message buffer for the selected receive shadow buffer is established by the receive shadow buffer index field RSBIDX in the Receive Shadow Buffer Index Register (FR_RSBIR). The start address SADR_MBHF of the related message buffer header field in the flexray memory area is determined according to Equation 3.

$$\text{SADR\_MBHF} = (\text{FR\_RSBIR[RSBIDX]} * 8) + \text{SMBA} \qquad \textit{Eqn. 3}$$

The length required for the message buffer data field depends on the message buffer segment that the receive shadow buffer is assigned to. For the receive shadow buffers assigned to the first message buffer segment, the length must be the same as for the individual message buffers assigned to the first message buffer segment. For the receive shadow buffers assigned to the second message buffer segment, the length must be the same as for the individual message buffers assigned to the second message buffer segment. The receive shadow buffer assignment is described in Receive Shadow Buffer Index Register (FR_RSBIR).

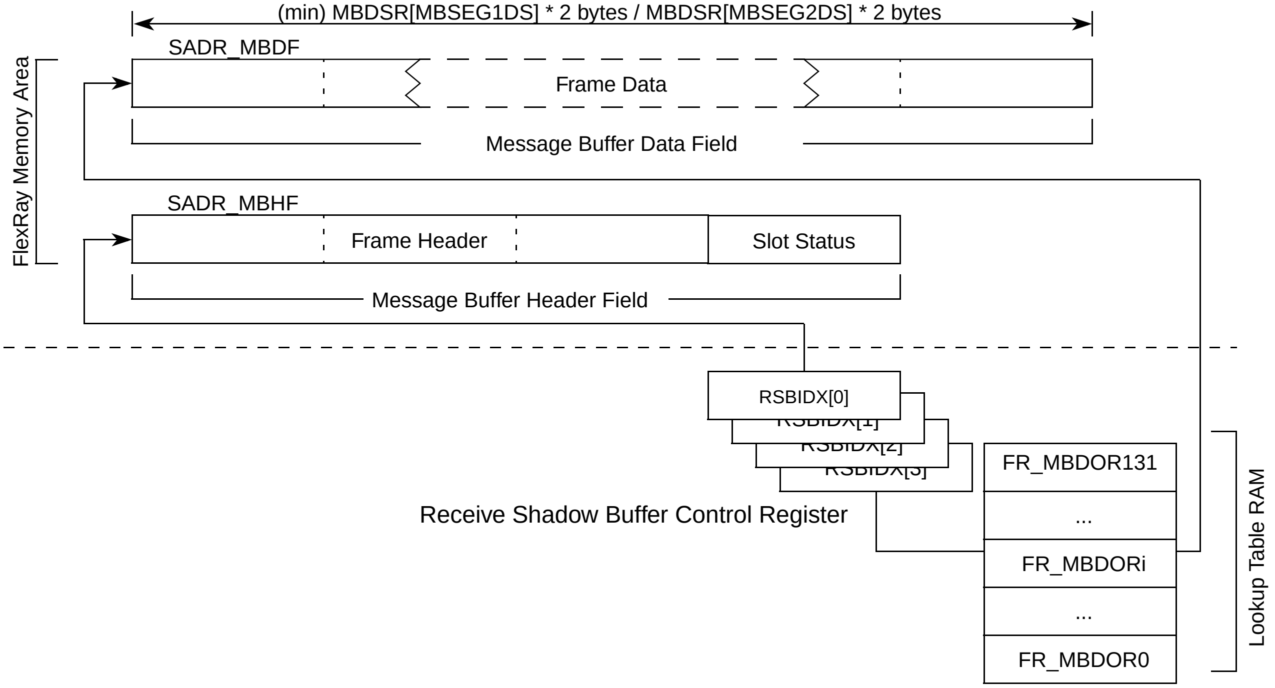


Figure 26-118. Receive Shadow Buffer Structure

26.6.3.3 Receive FIFO

The receive FIFO implements a frame reception system based on the FIFO concept. The CC provides two independent receive FIFOs, one per channel.

A receive FIFO consists of a set of physical message buffers in the flexray memory area and a set of receive FIFO control registers located in dedicated registers. The structure of a receive FIFO is given in Figure 26-119.

The connection between the receive FIFO control registers and the set of physical message buffers is established by the Receive FIFO Start Index Register (FR_RFSIR), the Receive FIFO Depth and Size Register (RFDSR), and the Receive FIFO A Read Index Register (FR_RFARIR) / Receive FIFO B Read Index Register (FR_RFBRIR).

The system memory base address SMBA valid for the receive FIFOs is defined by the system memory base address register selected by the FIFO address mode bit FR_MCR[FAM], refer to Section 26.5.2.4, “Module Configuration Register (FR_MCR)”.

The start byte address SADR_MBHF[1] of the first message buffer header field that belongs to the receive FIFO is determined according to Equation 4.

$$\text{SADR\_MBHF[1]} = (8 * \text{FR\_RFSIR[SIDX]}) + \text{SMBA} \qquad \textit{Eqn. 4}$$

The start byte address SADR_MBHF[n] of the last message buffer header field that belongs to the receive FIFO in the flexray memory area is determined according to Equation 5.

SADR_MBHF[n] = (8 * (FR_RFSIR[SIDX] + RFDSR[FIFO_DEPTH])) + SMBA ***Eqn. 5***

The required information to access the current entry of the FIFO is given in the following registers:

- The registers Receive FIFO A Read Index Register (FR_RFARIR) and Receive FIFO B Read Index Register (FR_RFBRIR) provide the index of the physical message buffer belonging to the current entry.

  The data field offset belonging to the current FIFO entry RF_DFO[X] must be calculated using the current read index i according to the following formula:

RF_DFO[X] = FR_RFSDOR[X] + (FR_RFDSR[X][ENTRY_SIZE] * 2) * i - FR_RFSIDX[X}) ***Eqn. 6***

NOTE

The current read index loops up starting at the number given in the FR_RD[A/B]RDIDX register for the required number of entries.

Refer to Section 26.6.9.8, “FIFO Update” for details about updating the FIFO read pointer.

All message buffer header fields assigned to a receive FIFO are within a contiguous region defined by FR_RFSIR[SIDX] and RFDSR[FIFO_DEPTH].

The data sections of all FIFO entries within on receive FIFO are of the same length defined by RFDSR[FIFO_SIZE].

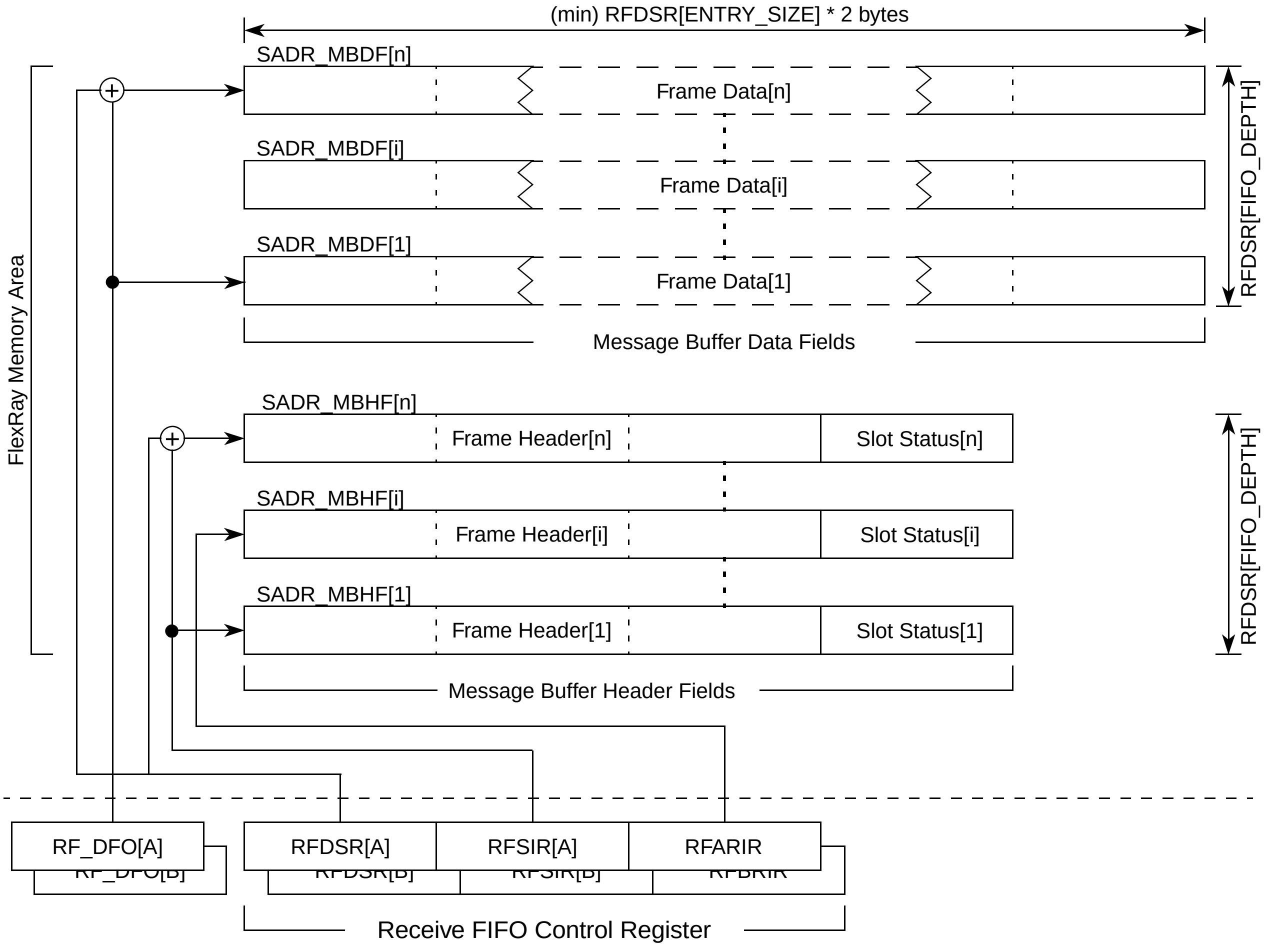


Figure 26-119. Receive FIFO Structure

NOTE

The actual values of the data field offsets RF_DFO[A/B] need to be calculated according to Equation 6. They are not stored in a register.

26.6.3.4 Message Buffer Configuration and Control Data

This section describes the configuration and control data for each message buffer type.

26.6.3.4.1 Individual Message Buffer Configuration Data

Before an individual message buffer can be used for transmission or reception, it must be configured. There is a set of common configuration parameters that applies to all individual message buffers and a set of configuration parameters that applies to each message buffer individually.

Common Configuration Data

The set of common configuration data for individual message buffers is located in the following registers.

- Message Buffer Data Size Register (FR_MBDSR)
  The MBSEG2DS and MBSEG1DS fields define the minimum length of the message buffer data field with respect to the message buffer segment.
- Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)
  The LAST_MB_SEG1 and LAST_MB_UTIL fields define the segmentation of the individual message buffers and the number of individual message buffers that are used. For more details, see Section 26.6.3.1.1, "Individual Message Buffer Segments"

Specific Configuration Data

The set of message buffer specific configuration data for individual message buffers is located in the following registers.

- Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)
  The MTD bit configures the message buffer type.
- Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)
  The MTM, CHA, CHB bits configure the transmission mode and the channel assignment. The CCFE, CCFMSK, and CCFVAL bits and fields configure the cycle counter filter.
- Message Buffer Frame ID Registers (FR_MBFIDRn)
  For a transmit message buffer, the FID field is used to determine the slot in which the message in this message buffer will be transmitted.
- Message Buffer Index Registers (FR_MBIDXRn)
  This MBIDX field provides the index of the message buffer header field of the physical message buffer that is currently associated with this message buffer.

26.6.3.5 Individual Message Buffer Control Data

During normal operation, each individual message buffer can be controlled by the control and trigger bits CMT, LCKT, EDT, and MBIE in the Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn).

26.6.3.6 Receive Shadow Buffer Configuration Data

Before frame reception into the individual message buffers can be performed, the receive shadow buffers must be configured. The configuration data are provided by the Receive Shadow Buffer Index Register (FR_RSBIR). For each receive shadow buffer, the application provides the message buffer header index. When the protocol is in the *POC:normal active* or *POC:normal passive* state, the receive shadow buffers are under full CC control.

26.6.3.7 Receive FIFO Control and Configuration Data

This section describes the configuration and control data for the two receive FIFOs.

26.6.3.7.1 Receive FIFO Configuration Data

The CC provides two functional independent receive FIFOs, one per channel. The FIFOs have a common subset of configuration data:

- Receive FIFO Periodic Timer Register (FR_RFPTR)

Each FIFO has its own set of configuration data. The configuration data are located in the following registers:

- Receive FIFO Watermark and Selection Register (FR_RFWMSR)
- Receive FIFO Start Index Register (FR_RFSIR)
- Receive FIFO Start Data Offset Register (FR_RFSDOR)
- Receive FIFO Depth and Size Register (RFDSR)
- Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR)
- Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR)
- Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR)
- Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR)
- Receive FIFO Range Filter Configuration Register (FR_RFRFCFR)

26.6.3.7.2 Receive FIFO Control Data

The application can access the FIFOs at any time using the control bits in the following registers:

- Global Interrupt Flag and Enable Register (FR_GIFER)
- Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)

26.6.3.7.3 Receive FIFO Status Data

The current status of the receive FIFO is provided in the following register:

- Global Interrupt Flag and Enable Register (FR_GIFER)
- Receive FIFO A Read Index Register (FR_RFARIR)
- Receive FIFO B Read Index Register (FR_RFBRIR)
- Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR)

26.6.4 Flexray Memory Area Layout

The CC supports a wide range of possible layouts for the flexray memory area. Two basic layout modes can be selected by the FIFO address mode bit FR_MCR[FAM].

26.6.4.1 Flexray Memory Area Layout (FR_MCR[FAM] = 0)

Figure 26-120 shows an example layout for the FIFO address mode FR_MCR[FAM]=0. In this mode, the following set of rules applies to the layout of the flexray memory area:

- The flexray memory area is one contiguous region.
- The flexray memory area size is maximum 64 KB.
- The flexray memory area starts at a 16 byte boundary

The flexray memory area contains three areas: the *message buffer header area*, the *message buffer data area*, and the *sync frame table area*.

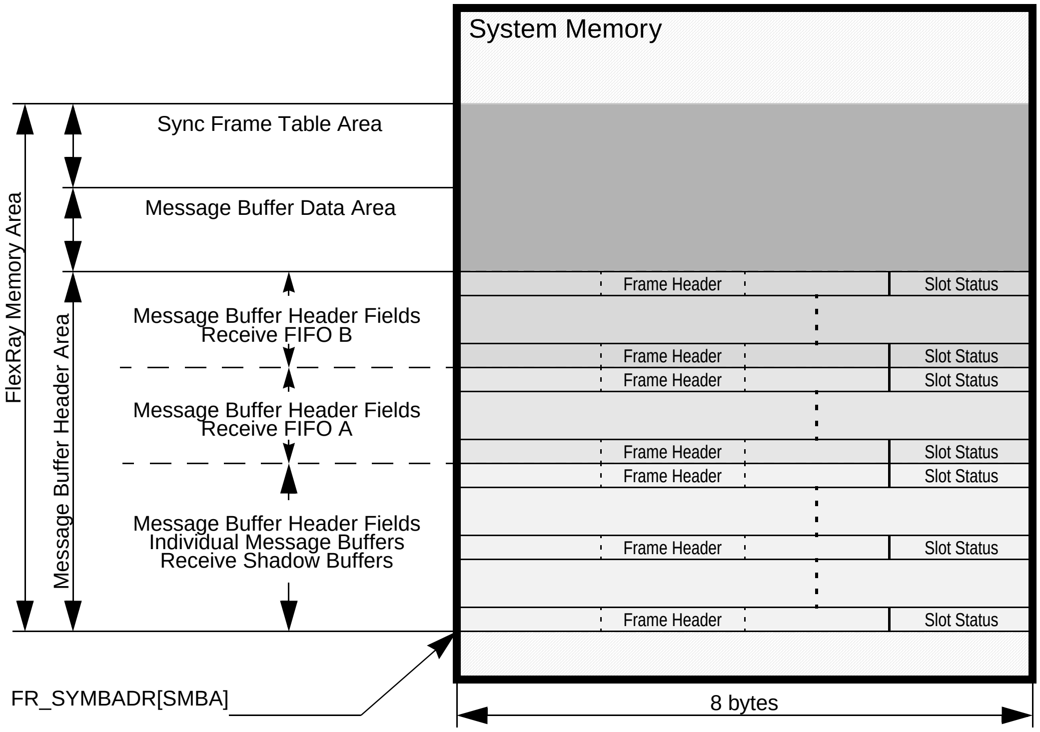


Figure 26-120. Example of FlexRay Memory Area Layout (FR_MCR[FAM] = 0)

26.6.4.2 FlexRay Memory Area Layout (FR_MCR[FAM] = 1)

Figure 26-121 shows an example layout for the FIFO address mode FR_MCR[FAM]=1. The following set of rules applies to the layout of the flexray memory area:

- The flexray memory area consists of two contiguous regions.
- The size of each region is maximum 64 KB.
- Each region start at a 16 byte boundary.

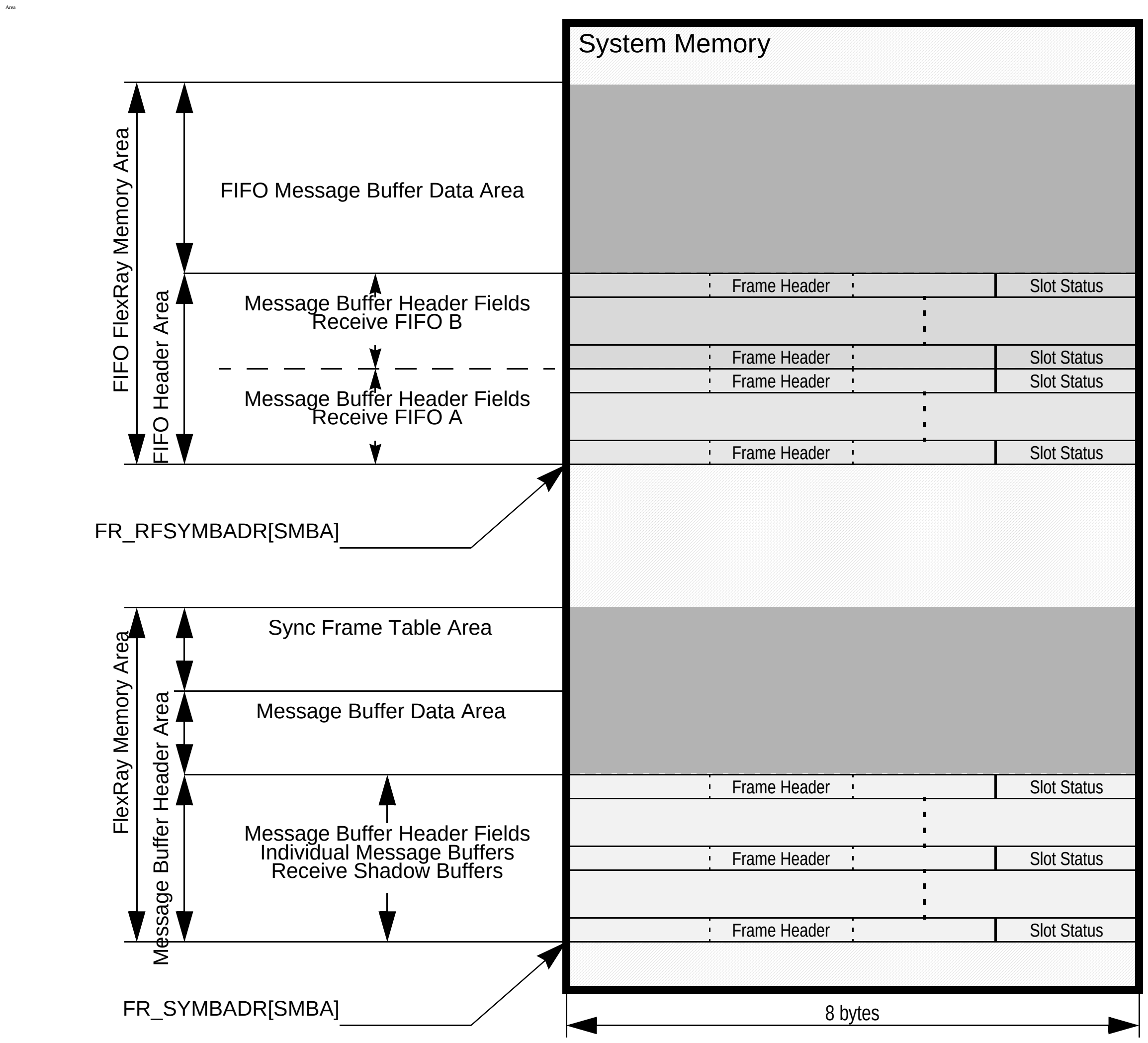


Figure 26-121. Example of FlexRay Memory Area Layout (FR_MCR[FAM] = 1)

26.6.4.3 Message Buffer Header Area (FR_MCR[FAM] = 0)

The message buffer header area contains all message buffer header fields of the physical message buffers for all message buffer types. The following rules apply to the message buffer header fields for the three type of message buffers.

1. The start byte address SADR_MBHF of each message buffer header field for *individual message buffers* and *receive shadow buffers* must fulfill Equation 7.

$$\mathrm{SADR\_MBHF} = (i * 8) + \mathrm{FR\_SYMBADR[SMBA]};\ (0 <= i <= 131) \qquad \textit{Eqn. 7}$$

2. The start byte address SADR_MBHF of each message buffer header field for the *FIFO* must fulfill Equation 8.

$$\mathrm{SADR\_MBHF} = (i * 8) + \mathrm{FR\_SYMBADR[SMBA]};\ (0 <= i <= 1023) \qquad \textit{Eqn. 8}$$

$$\mathrm{SADR\_MBHF} = (i * 8) + \mathrm{FR\_SYMBADR[SMBA]};\ (0 <= i <= 1023) \qquad \textit{Eqn. 9}$$

3. The message buffer header fields for each FIFO have to be a contiguous area.

26.6.4.4 Message Buffer Header Area (FR_MCR[FAM] = 1)

The message buffer header area contains all message buffer header fields of the physical message buffers for the individual message buffers and receiver shadow buffers. The following rules apply to the message buffer header fields for the two type of message buffers.

1. The start address SADR_MBHF of each message buffer header field for *individual message buffers* and *receive shadow buffers* must fulfill Equation 10.

SADR_MBHF = (i * 8) + FR_SYMBADR[SMBA]; (0 <= i <= 131) ***Eqn. 10***

26.6.4.5 FIFO Message Buffer Header Area (FR_MCR[FAM] = 1)

The FIFO message buffer header area contains all message buffer header fields of the physical message buffers for the FIFO. The following rules apply to the FIFO message buffer header fields.

1. The start byte address SADR_MBHF of each message buffer header field for the *FIFO* must fulfill Equation 11.

SADR_MBHF = (i * 8) + FR_RFSYMBADR[SMBA]; (0 <= i <= 1023) ***Eqn. 11***

2. The message buffer header fields for each FIFO have to be a contiguous area.

26.6.4.6 Message Buffer Data Area

The message buffer data area contains all the message buffer data fields of the physical message buffers. Each message buffer data field must start at a 16-bit boundary.

26.6.4.7 Sync Frame Table Area

The sync frame table area is used to provide a copy of the internal sync frame tables for application access. Refer to Section 26.6.12, “Sync Frame ID and Sync Frame Deviation Tables” for the description of the sync frame table area.

26.6.5 Physical Message Buffer Description

This section provides a detailed description of the usage and the content of the two parts of a physical message buffer, the message buffer header field and the message buffer data field.

26.6.5.1 Message Buffer Protection and Data Consistency

The physical message buffers are located in the flexray memory area. The CC provides no means to protect the flexray memory area from uncontrolled or illegal host or other client write access. To ensure data consistency of the physical message buffers, the application must follow the write access scheme that is given in the description of each of the physical message buffer fields.

26.6.5.2 Message Buffer Header Field Description

This section provides a detailed description of the usage and content of the message buffer header field. A description of the structure of the message buffer header fields is given in Section 26.6.2.1, “Message Buffer Header Field”. Each message buffer header field consists of two sections: the frame header section and the slot status section.

26.6.5.2.1 Frame Header Description

Frame Header Content

The semantic and content of the frame header section depends on the message buffer type.

For individual receive message buffers and receive FIFOs, the frame header receives the frame header data of the *first valid frame* received on the assigned channels.

For receive shadow buffers, the frame header receives the frame header data of the current frame received regardless of whether the frame is valid or not.

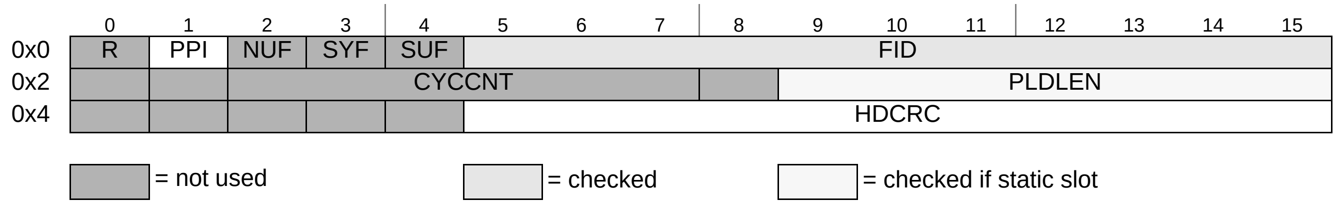
For transmit message buffers, the application writes the frame header of the frame to be transmitted into this location. The frame header will be read out when the frame is transferred to the FlexRay bus.

The structure of the frame header in the message buffer header field for receive message buffers and the receive FIFO is given in Figure 26-122. A detailed description is given in Table 26-99.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0x0	R	PPI	NUF	SYF	SUF	FID										
0x2	0	0	CYCCNT						0	PLDLEN						
0x4	0	0	0	0	0	HDCRC										

Figure 26-122. Frame Header Structure (Receive Message Buffer and Receive FIFO)

The structure of the frame header in the message buffer header field for transmit message buffers is given in Figure 26-123. A detailed description is given in Table 26-100. The checks that will be performed are described in Frame Header Checks.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0x0	R	PPI	NUF	SYF	SUF	FID										
0x2			CYCCNT							PLDLEN						
0x4						HDCRC										

▨ = not used ▒ = checked ☐ = checked if static slot

Figure 26-123. Frame Header Structure (Transmit Message Buffer)

The structure of the frame header in the message buffer header field for transmit message buffers assigned to key slot is given in Figure 26-124.

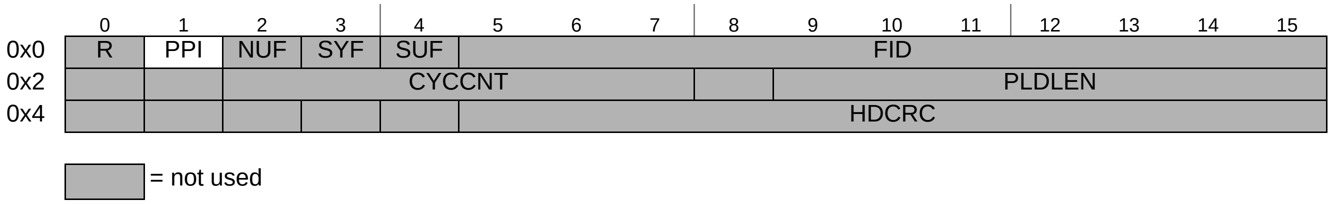


Figure 26-124. Frame Header Structure (Transmit Message Buffer for Key Slot)

Frame Header Access

The frame header is located in the flexray memory area. To ensure data consistency, the application must follow the write access scheme described below.

For receive message buffers, receive shadow buffers, and receive FIFOs, the application must not write to the frame header field.

For transmit message buffers, the application must follow the write access restrictions given in Table 26-98. This table shows the condition under which the application can write to the frame header entries without corrupting the FlexRay message transmission.

Table 26-98. Frame Header Write Access Constraints (Transmit Message Buffer)

<table>
<tr><th>Field</th><th>Static Segment</th><th>Dynamic Segment</th></tr>
<tr><td>FID</td><td colspan="2"><i>POC:config</i> or MB_DIS</td></tr>
<tr><td rowspan="2">PPI,
PLDLEN,
HDCRC</td><td colspan="2"><i>POC:config</i> or MB_DIS or</td></tr>
<tr><td></td><td>MB_LCK</td></tr>
</table>

Frame Header Checks

As shown in Figure 26-123 and Figure 26-124 not all fields in the message buffer frame header are used for transmission. Some fields in the message buffer frame header are ignored, some are used for transmission, and some of them are checked for correct values. All checks that will be performed are described below.

For message buffers assigned to the key slot, no checks will be performed.

The value of the FID field must be equal to the value of the corresponding Message Buffer Frame ID Registers (FR_MBFIDRn). If the CC detects a mismatch while transmitting the frame header, it will set the frame ID error flag FID_EF in the CHI Error Flag Register (FR_CHIERFR). The value of the FID field will be ignored and replaced by the value provided in the Message Buffer Frame ID Registers (FR_MBFIDRn).

For transmit message buffers assigned to the *static* segment, the PLDLEN value must be equal to the value of the payload_length_static field in the Protocol Configuration Register 19 (FR_PCR19). If this is not fulfilled, the static payload length error flag SPL_EF in the CHI Error Flag Register (FR_CHIERFR) is set when the message buffer is under transmission. A syntactically and semantically correct frame is generated with payload_length_static payload words and the payload length field in the transmitted frame header set to payload_length_static.

For transmit message buffers assigned to the *dynamic* segment, the PLDLEN value must be less than or equal to the value of the max_payload_length_dynamic field in the Protocol Configuration Register 24 (FR_PCR24). If this is not fulfilled, the dynamic payload length error flag DPL_EF in the CHI Error Flag Register (FR_CHIERFR) is set when the message buffer is under transmission. A syntactically and semantically correct dynamic frame is generated with PLDLEN payload words and the payload length field in the frame header set to PLDLEN.

Table 26-99. Frame Header Field Descriptions (Receive Message Buffer and Receive FFO)

Field	Description
R	**Reserved Bit** — This is the value of the *Reserved bit* of the received frame stored in the message buffer
PPI	**Payload Preamble Indicator** — This is the value of the *Payload Preamble Indicator* of the received frame stored in the message buffer.
NUF	**Null Frame Indicator** — This is the value of the *Null Frame Indicator* of the received frame stored in the message buffer.
SYF	**Sync Frame Indicator** — This is the value of the *Sync Frame Indicator* of the received frame stored in the message buffer.
SUF	**Startup Frame Indicator** — This is the value of the *Startup Frame Indicator* of the received frame stored in the message buffer.
FID	**Frame ID** — This is the value of the *Frame ID* field of the received frame stored in the message buffer.
CYCCNT	**Cycle Count** — This is the number of the communication cycle in which the frame stored in the message buffer was received.
PLDLEN	**Payload Length** — This is the value of the *Payload Length* field of the received frame stored in the message buffer.
HDCRC	**Header CRC** — This is the value of the *Header CRC* field of the received frame stored in the message buffer.

Table 26-100. Frame Header Field Descriptions (Transmit Message Buffer)

Field	Description
R	**Reserved Bit** — This bit is not used, the value of the *Reserved bit* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.
PPI	**Payload Preamble Indicator** — This bit provides the value of the *Payload Preamble Indicator* for the frame transmitted from the message buffer.
NUF	**Null Frame Indicator** — This bit is not used, the value of the *Null Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.
SYF	**Sync Frame Indicator** — This bit is not used, the value of the *Sync Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.
SUF	**Startup Frame Indicator** — This bit is not used, the value of the *Startup Frame Indicator* is generated internally according to *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*.
FID	**Frame ID** — This field is checked as described in Frame Header Checks.

Table 26-100. Frame Header Field Descriptions (Transmit Message Buffer)

Field	Description
CYCCNT	**Cycle Count** — This field is not used, the value of the transmitted *Cycle Count* field is taken from the internal communication cycle counter.
PLDLEN	**Payload Length** — This field is checked and used as described in Frame Header Checks.
HDCRC	**Header CRC** — This field provides the value of the *Header CRC* field for the frame transmitted from the message buffer.

26.6.5.2.2 Slot Status Description

The slot status is a read-only structure for the application and a write-only structure for the CC. The meaning and content of the slot status in the message buffer header field depends on the message buffer type.

Receive Message Buffer and Receive FIFO Slot Status Description

This section describes the slot status structure for the individual receive message buffers and receive FIFOs. The content of the slot status structure for receive message buffers depends on the message buffer type and on the channel assignment for individual receive message buffers as given by Table 26-101.

Table 26-101. Receive Message Buffer Slot Status Content

Receive Message Buffer Type	Slot Status Content
Individual Receive Message Buffer assigned to both channels FR_MBCCFRn[CHA]=1 and FR_MBCCFRn[CHB]=1	see Figure 26-125
Individual Receive Message Buffer assigned to channel A FR_MBCCFRn[CHA]=1 and FR_MBCCFRn[CHB[=0	see Figure 26-126
Individual Receive Message Buffer assigned to channel B FR_MBCCFRn[CHA]=0 and FR_MBCCFRn[CHB]=1	see Figure 26-127
Receive FIFO Channel A Message Buffer	see Figure 26-126
Receive FIFO Channel B Message Buffer	see Figure 26-127

The meaning of the bits in the slot status structure is explained in Table 26-102.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	CH	VFA	SYA	NFA	SUA	SEA	CEA	BVA	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 26-125. Receive Message Buffer Slot Status Structure (ChAB)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	VFA	SYA	NFA	SUA	SEA	CEA	BVA	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 26-126. Receive Message Buffer Slot Status Structure (ChA)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	1	0	0	0	0	0	0	0	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 26-127. Receive Message Buffer Slot Status Structure (ChB)

Table 26-102. Receive Message Buffer Slot Status Field Description

Field	Description
	Common Message Buffer Status Bits
VFB	**Valid Frame on Channel B** — protocol related variable: *vSS!ValidFrame* channel B 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYB	**Sync Frame Indicator Channel B** — protocol related variable: *vRF!Header!SyFIndicator* channel B 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFB	**Null Frame Indicator Channel B** — protocol related variable: *vRF!Header!NFIndicator* channel B 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUB	**Startup Frame Indicator Channel B** — protocol related variable: *vRF!Header!SuFIndicator* channel B 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEB	**Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* channel B 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1
CEB	**Content Error on Channel B** — protocol related variable: *vSS!ContentError* channel B 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVB	**Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* channel B 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1
CH	**Channel first valid received** — This status bit applies only to receive message buffers assigned to the static segment and to both channels. It indicates the channel that has received the *first valid* frame in the slot. This flag is set to 0 if no valid frame was received at all in the subscribed slot. 0 first valid frame received on channel A, or no valid frame received at all 1 first valid frame received on channel B
VFA	**Valid Frame on Channel A** — protocol related variable: *vSS!ValidFrame* channel A 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYA	**Sync Frame Indicator Channel A** — protocol related variable: *vRF!Header!SyFIndicator* channel A 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFA	**Null Frame Indicator Channel A** — protocol related variable: *vRF!Header!NFIndicator* channel A 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1

Table 26-102. Receive Message Buffer Slot Status Field Description

Field	Description
SUA	**Startup Frame Indicator Channel A** — protocol related variable: *vRF!Header!SuFIndicator* channel A 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEA	**Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* channel A 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1
CEA	**Content Error on Channel A** — protocol related variable: *vSS!ContentError* channel A 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVA	**Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* channel A 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1

Transmit Message Buffer Slot Status Description

This section describes the slot status structure for transmit message buffers. Only the TCA and TCB status bits are directly related to the transmission process. All other status bits in this structure are related to a receive process that may have occurred. The content of the slot status structure for transmit message buffers depends on the channel assignment as given by Table 26-103.

Table 26-103. Transmit Message Buffer Slot Status Content

Transmit Message Buffer Type	Slot Status Content
Individual Transmit Message Buffer assigned to both channels FR_MBCCFRn[CHA]=1 and FR_MBCCFRn[CHB]=1	see Figure 26-128
Individual Transmit Message Buffer assigned to channel A FR_MBCCFRn[CHA]=1 and FR_MBCCFRn[CHB]=0	see Figure 26-129
Individual Transmit Message Buffer assigned to channel B FR_MBCCFRn[CHA]=0 and FR_MBCCFRn[CHB]=1	see Figure 26-130

The meaning of the bits in the slot status structure is described in Table 26-102.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 26-128. Transmit Message Buffer Slot Status Structure (ChAB)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	VFA	SYA	NFA	SUA	SEA	CEA	BVA	TCA
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 26-129. Transmit Message Buffer Slot Status Structure (ChA)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VFB	SYB	NFB	SUB	SEB	CEB	BVB	TCB	0	0	0	0	0	0	0	0
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 26-130. Transmit Message Buffer Slot Status Structure (ChB)

Table 26-104. Transmit Message Buffer Slot Status Structure Field Descriptions

Field	Description
VFB	**Valid Frame on Channel B** — protocol related variable: *vSS!ValidFrame* channel B 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYB	**Sync Frame Indicator Channel B** — protocol related variable: *vRF!Header!SyFIndicator* channel B 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFB	**Null Frame Indicator Channel B** — protocol related variable: *vRF!Header!NFIndicator* channel B 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUB	**Startup Frame Indicator Channel B** — protocol related variable: *vRF!Header!SuFIndicator* channel B 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEB	**Syntax Error on Channel B** — protocol related variable: *vSS!SyntaxError* channel B 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1
CEB	**Content Error on Channel B** — protocol related variable: *vSS!ContentError* channel B 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVB	**Boundary Violation on Channel B** — protocol related variable: *vSS!BViolation* channel B 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1
TCB	**Transmission Conflict on Channel B** — protocol related variable: *vSS!TxConflict* channel B 0 *vSS!TxConflict* = 0 1 *vSS!TxConflict* = 1
VFA	**Valid Frame on Channel A** — protocol related variable: *vSS!ValidFrame* channel A 0 *vSS!ValidFrame* = 0 1 *vSS!ValidFrame* = 1
SYA	**Sync Frame Indicator Channel A** — protocol related variable: *vRF!Header!SyFIndicator* channel A 0 *vRF!Header!SyFIndicator* = 0 1 *vRF!Header!SyFIndicator* = 1
NFA	**Null Frame Indicator Channel A** — protocol related variable: *vRF!Header!NFIndicator* channel A 0 *vRF!Header!NFIndicator* = 0 1 *vRF!Header!NFIndicator* = 1
SUA	**Startup Frame Indicator Channel A** — protocol related variable: *vRF!Header!SuFIndicator* channel A 0 *vRF!Header!SuFIndicator* = 0 1 *vRF!Header!SuFIndicator* = 1
SEA	**Syntax Error on Channel A** — protocol related variable: *vSS!SyntaxError* channel A 0 *vSS!SyntaxError* = 0 1 *vSS!SyntaxError* = 1

Table 26-104. Transmit Message Buffer Slot Status Structure Field Descriptions

Field	Description
CEA	**Content Error on Channel A** — protocol related variable: *vSS!ContentError* channel A 0 *vSS!ContentError* = 0 1 *vSS!ContentError* = 1
BVA	**Boundary Violation on Channel A** — protocol related variable: *vSS!BViolation* channel A 0 *vSS!BViolation* = 0 1 *vSS!BViolation* = 1
TCA	**Transmission Conflict on Channel A** — protocol related variable: *vSS!TxConflict* channel A 0 *vSS!TxConflict* = 0 1 *vSS!TxConflict* = 1

26.6.5.3 Message Buffer Data Field Description

The message buffer data field is used to store the frame payload data, or a part of it, of the frame to be transmitted to or received from the FlexRay bus. The minimum required length of this field depends on the message buffer type that the physical message buffer is assigned to and is given in Table 26-105. The structure of the message buffer data field is given in Figure 26-131.

Table 26-105. Message Buffer Data Field Minimum Length

physical message buffer assigned to	minimum length defined by
Individual Message Buffer in Segment 1	FR_MBDSR[MBSEG1DS]
Receive Shadow Buffer in Segment 1	FR_MBDSR[MBSEG1DS]
Individual Message Buffer in Segment 2	FR_MBDSR[MBSEG2DS]
Receive Shadow Buffer in Segment 2	FR_MBDSR[MBSEG2DS]
Receive FIFO for channel A	FR_RFDSR[ENTRY_SIZE] (FR_RFWMSR[SEL] = 0)
Receive FIFO for channel B	FR_RFDSR[ENTRY_SIZE] (FR_RFWMSR[SEL] = 1)

NOTE

The CC will not access any locations outside the message buffer data field boundaries given by Table 26-105.

	0 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
0x0	DATA0 / MID0 / NMV0	DATA1 / MID1 / NMV1
0x2	DATA2 / NMV2	DATA3 / NMV3
...	...	...
0xN-2	DATA N-2	DATA N-1

Figure 26-131. Message Buffer Data Field Structure

The message buffer data field is located in the flexray memory area; thus, the CC has no means to control application write access to the field. To ensure data consistency, the application must follow a write and read access scheme.

26.6.5.3.1 Message Buffer Data Field Read Access

For transmit message buffers, the CC will not modify the content of the Message Buffer Data Field. Thus the application can read back the data at any time without any impact on data consistency.

For receive message buffers the application must lock the related receive message buffer and retrieve the message buffer header index from the Message Buffer Index Registers (FR_MBIDXRn). While the message buffer is locked, the CC will not update the Message Buffer Data Field.

For receive FIFOs, the application can read the message buffer indicated by the Receive FIFO A Read Index Register (FR_RFARIR) or the Receive FIFO B Read Index Register (FR_RFBRIR) when the related fill levels in the Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR) indicate an non-empty FIFO.

26.6.5.3.2 Message Buffer Data Field Write Access

For receive message buffers, receive shadow buffers, and receive FIFOs, the application must not write to the message buffer data field.

For transmit message buffers, the application must follow the write access restrictions given in Table 26-106.

Table 26-106. Frame Data Write Access Constraints

Field	CC/MB State
DATA, MID, NMV	*POC:config* or MB_DIS or MB_LCK

Table 26-107. Frame Data Field Descriptions

Field	Description
DATA 0, DATA 1, ... DATA N-1	**Message Data** — Provides the message data received or to be transmitted. For receive message buffer and receive FIFOs, this field provides the message data received for this message buffer. For transmit message buffers, the field provides the message data to be transmitted.
MID 0, MID 1	Message Identifier — If the payload preamble bit PPI is set in the message buffer frame header, the MID field holds the message ID of a dynamic frame located in the message buffer. The receive FIFO filter uses the received message ID for message ID filtering.
NMV 0, NMV 1, ... NMV 11	Network Management Vector — If the payload preamble bit PPI is set in the message buffer frame header, the network management vector field holds the network management vector of a static frame located in the message buffer. **Note:** The MID and NMV bytes replace the corresponding DATA bytes.

26.6.6 Individual Message Buffer Functional Description

The CC provides three basic types of individual message buffers:

1. Transmit Message Buffers
2. Receive Message Buffers

Before an individual message buffer can be used, it must be configured by the application. After the initial configuration, the message buffer can be reconfigured later. The set of the configuration data for individual message buffers is given in Section 26.6.3.4.1, “Individual Message Buffer Configuration Data”.

26.6.6.1 Individual Message Buffer Configuration

The individual message buffer configuration consists of two steps. The first step is the allocation of the required amount of memory for the flexray memory area. The second step is the programming of the message buffer configuration registers, which is described in this section.

26.6.6.1.1 Common Configuration Data

One part of the message buffer configuration data is common to all individual message buffers and the receive shadow buffers. These data can only be set when the protocol is in the *POC:config* state.

The application configures the number of utilized individual message buffers by writing the message buffer number of the last utilized message buffer into the LAST_MB_UTIL field in the Message Buffer Segment Size and Utilization Register (FR_MBSSUTR).

The application configures the size of the two segments of individual message buffers by writing the message buffer number of the last message buffer in the first segment into the LAST_MB_SEG1 field in the Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)

The application configures the length of the message buffer data fields for both of the message buffer segments by writing to the MBSEG2DS and MBSEG1DS fields in the Message Buffer Data Size Register (FR_MBDSR).

Depending on the current receive functionality of the CC, the application must configure the receive shadow buffers. For each segment and for each channel with at least one individual receive message buffer assigned, the application must configure the related receive shadow buffer using the Receive Shadow Buffer Index Register (FR_RSBIR).

26.6.6.1.2 Specific Configuration Data

The second part of the message buffer configuration data is specific for each message buffer.

These data can be changed only when either

- the protocol is in the *POC:config* state or
- the message buffer is disabled (FR_MBCCSR*n*[EDS] = 0)

The individual message buffer type is defined by the MTD and MBT bits in the Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn) as given in Table 26-108.

Table 26-108. Individual Message Buffer Types

FR_MBCCSRn		Individual Message Buffer Description
MTD	**MBT**	
0	0	Receive Message Buffer
0	1	Reserved

Table 26-108. Individual Message Buffer Types

FR_MBCCSRn		Individual Message Buffer Description
MTD	**MBT**	
1	0	Transmit Message Buffer
1	1	Reserved

The message buffer specific configuration data are

1. MTD bits in Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn)
2. all fields and bits in Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn)
3. all fields and bits in Message Buffer Frame ID Registers (FR_MBFIDRn)
4. all fields and bits in Message Buffer Index Registers (FR_MBIDXRn)

The meaning of the specific configuration data depends on the message buffer type, as given in the detailed message buffer type descriptions Section 26.6.6.2, “Transmit Message Buffers” and Section 26.6.6.3, “Receive Message Buffers”.

26.6.6.2 Transmit Message Buffers

The section provides a detailed description of the functionality of single buffered transmit message buffers.

A transmit message buffer is used by the application to provide message data to the CC that will be transmitted over the FlexRay Bus. The CC uses the transmit message buffers to provide information about the transmission process and status information about the slot in which message was transmitted.

The individual message buffer with message buffer number *n* is configured to be a transmit message buffer by the following settings:

- FR_MBCCSRn[MBT] = 0 (single buffered message buffer)
- FR_MBCCSRn[MTD] = 1 (transmit message buffer)

26.6.6.2.1 Access Regions

To certain message buffer fields, both the application and the CC have access. To ensure data consistency, a message buffer locking scheme is implemented, which is used to control the access to the data, control, and status bits of a message buffer. The access regions for transmit message buffers are depicted in Figure 26-132. A description of the regions is given in Table 26-109. If an region is active as indicated in Table 26-110, the access scheme given for that region applies to the message buffer.

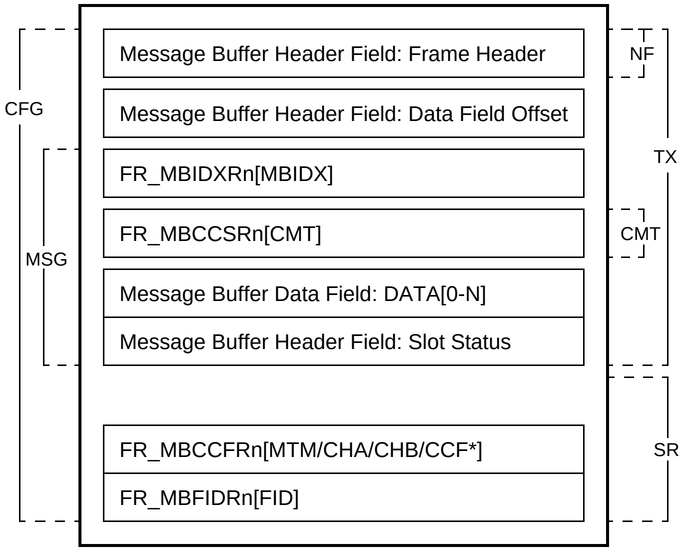


Figure 26-132. Transmit Message Buffer Access Regions

Table 26-109. Transmit Message Buffer Access Regions Description

Region	Access from		Region used for
	Application	Module	
CFG	read/write	-	Message Buffer Configuration
MSG	read/write	-	Message Data and Slot Status Access
NF	-	read-only	Message Header Access for Null Frame Transmission
TX	-	read/write	Message Transmission and Slot Status Update
CM	-	read-only	Message Buffer Validation
SR	-	read-only	Message Buffer Search

The trigger bits FR_MBCCSRn[EDT] and FR_MBCCSRn[LCKT], and the interrupt enable bit FR_MBCCSRn[MBIE] are not under access control and can be accessed from the application at any time. The status bits FR_MBCCSRn[EDS] and FR_MBCCSRn[LCKS] are not under access control and can be accessed from the CC at any time.

The interrupt flag FR_MBCCSRn[MBIF] is not under access control and can be accessed from the application and the CC at any time. CC clear access has higher priority.

The CC restricts its access to the regions depending on the current state of the message buffer. The application must adhere to these restrictions in order to ensure data consistency. The transmit message buffer states are given in Figure 26-133. A description of the states is given in Table 26-110, which also provides the access scheme for the access regions.

The status bits FR_MBCCSRn[EDS] and FR_MBCCSRn[LCKS] provide the application with the required message buffer status information. The internal status information is not visible to the application.

26.6.6.2.2 Message Buffer States

This section describes the transmit message buffer states and provides a state diagram.

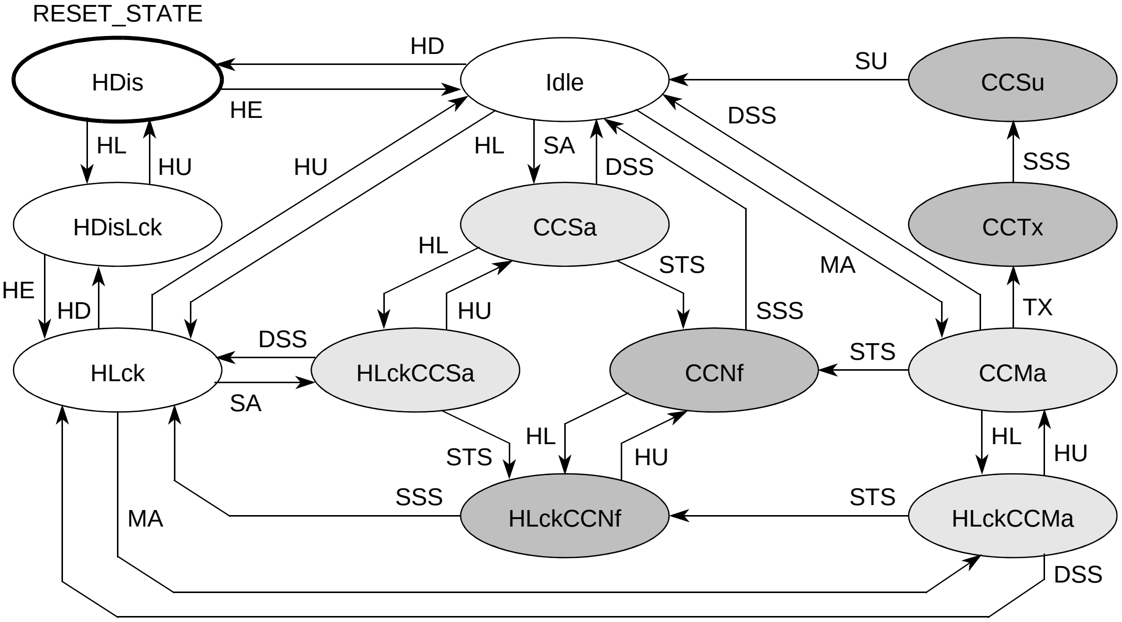


Figure 26-133. Transmit Message Buffer States

Table 26-110. Transmit Message Buffer State Description (Sheet 1 of 2)

State	FR_MBCCSRn		Access Region		Description
	EDS	LCKS	Appl.	Module	
Idle	1	0	–	CM, SR	**Idle** - Message Buffer is idle. Included in message buffer search.
HDis	0	0	CFG	–	**Disabled** - Message Buffer under configuration. Excluded from message buffer search.
HDisLck	0	1	CFG	–	**Disabled and Locked** - Message Buffer under configuration. Excluded from message buffer search.
HLck	1	1	MSG	SR	Locked - Applications access to data, control, and status. Included in message buffer search.
CCSa	1	0	–	–	Slot Assigned - Message buffer assigned to next static slot. Ready for Null Frame transmission.
HLckCCSa	1	1	MSG	–	Locked and Slot Assigned - Applications access to data, control, and status.Message buffer assigned to next static slot
CCNf	1	0	–	NF	Null Frame Transmission Header is used for null frame transmission.
HLckCCNf	1	1	MSG	NF	Locked and Null Frame Transmission - Applications access to data, control, and status. Header is used for null frame transmission.
CCMa	1	0	–	CM	Message Available - Message buffer is assigned to next slot and cycle counter filter matches.
HLckCCMa	1	1	MSG	–	Locked and Message Available - Applications access to data, control, and status. Message buffer is assigned to next slot and cycle counter filter matches.
CCTx	1	0	–	TX	Message Transmission - Message buffer data transmit. Payload data from buffer transmitted

Table 26-110. Transmit Message Buffer State Description (Sheet 1 of 2)

State	FR_MBCCSRn		Access Region		Description
	EDS	LCKS	Appl.	Module	
CCSu	1	0	–	TX	Status Update - Message buffer status update. Update of status flags, the slot status field, and the header index.

26.6.6.2.3 Message Buffer Transitions

Application Transitions

The application transitions can be triggered by the application using the commands described in Table 26-111. The application issues the commands by writing to the Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn). Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

Message Buffer Enable and Disable

The enable and disable commands issued by writing 1 to the trigger bit FR_MBCCSRn[EDT]. The transition that will be triggered by each of these command depends on the current value of the status bit FR_MBCCSRn[EDS]. If the command triggers the disable transition HD and the message buffer is in one of the states CCSa, HLckCCSa, CCMa, HLckCCMa, CCNf, HLckCCNf, or CCTx, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

If the communication controller is started as a non-coldstart node, and the message buffers are configured and enabled in the POC config state for Slot 1, then the message buffer cannot be disabled in the INTEGRATION_LISTEN state by directly writing 1 to the EDT bit. To facilitate this, a FREEZE command needs to be issued just before running the message buffer disable for slot 1. Executing this command enables the message buffer disable during the LISTEN states.

Message Buffer Lock and Unlock

The lock and unlock commands issued by writing 1 to the trigger bit FR_MBCCSRn[LCKT]. The transition that will be triggered by each of these commands depends on the current value of the status bit FR_MBCCSRn[LCKS]. If the command triggers the lock transition HL and the message buffer is in the state CCTx, the lock transition has no effect (command is ignored) and message buffer state is not changed. In this case, the message buffer lock error flag LCK_EF in the CHI Error Flag Register (FR_CHIERFR) is set.

Table 26-111. Transmit Message Buffer Application Transitions

Transition	Command	Condition	Description
HE	FR_MBCCSRn[EDT]:= 1	FR_MBCCSRn[EDS] = 0	Application triggers message buffer enable.
HD		FR_MBCCSRn[EDS] = 1	Application triggers message buffer disable.
HL	FR_MBCCSRn[LCKT]:= 1	FR_MBCCSRn[LCKS] = 0	Application triggers message buffer lock.
HU		FR_MBCCSRn[LCKS] = 1	Application triggers message buffer unlock.

Module Transitions

The module transitions that can be triggered by the CC are described in Table 26-112. Each transition will be triggered for certain message buffers when the related condition is fulfilled.

Table 26-112. Transmit Message Buffer Module Transitions

Transition	Condition	Description
SA	slot match and static slot	Slot Assigned - Message buffer is assigned to next static slot.
MA	slot match and CycleCounter match	Message Available - Message buffer is assigned to next slot and cycle counter filter matches.
TX	slot start and FR_MBCCSRn[CMT] = 1	Transmission Slot Start - Slot Start and commit bit CMT is set. In case of a dynamic slot, pLatestTx is not exceeded.
SU	status updated	Status Updated - Slot Status field and message buffer status flags updated. Interrupt flag set.
STS	static slot start	Static Slot Start - Start of static slot.
DSS	dynamic slot start or symbol window start or NIT start	Dynamic Slot or Segment Start. - Start of dynamic slot or symbol window or NIT.
SSS	slot start or symbol window start or NIT start	Slot or Segment Start - Start of static slot or dynamic slot or symbol window or NIT.

Transition Priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in the first part of Table 26-113, the module transitions have a higher priority than the application transitions. For all states except the CCMa state, both a lock/unlock transition HL/HD and a module transition can be executed at the same time. The result state is reached by first applying the application transition and subsequently the module transition to the intermediately reached state. For example, if the message buffer is in the HLck state and the application unlocks the message buffer by the HU transition and the module triggers the slot assigned transition SA, the intermediate state is Idle and the resulting state is CCSa.

The priorities among the module transitions is given in the second part of Table 26-113.

Table 26-113. Transmit Message Buffer Transition Priorities

State	Priorities	Description
module vs. application		
Idle, HLck	SA > HD MA > HD	Slot Assigned > Message Buffer Disable Message Available > Message Buffer Disable
CCMa	TX > HL	Transmission Start > Message Buffer Lock
module internal		
Idle, HLck	MA > SA	Message Available > Slot Assigned
CCMa	TX > STS TX > DSS	Transmission Slot Start > Static Slot Start Transmission Slot Start > Dynamic Slot Start

26.6.6.2.4 Transmit Message Setup

To transmit a message over the FlexRay bus, the application writes the message data into the message buffer data field and sets the commit bit CMT in the Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn). The physical access to the message buffer data field is described in Section 26.6.3.1, “Individual Message Buffers”.

As indicated by Table 26-110, the application shall write to the message buffer data field and change the commit bit CMT only if the transmit message buffer is in one of the states HDis, HDisLck, HLck, HLckCCSa, HLckCCMa, or HLckCCMa. The application can change the state of a message buffer if it issues the appropriate commands shown in Table 26-111. The state change is indicated through the FR_MBCCSRn[EDS] and FR_MBCCSRn[LCKS] status bits.

If the transmit message buffer enters one of the states HDis, HDisLck, HLck, HLckCCSa, HLckCCMa, or HLckCCMa the FR_MBCCSRn[DVAL] flag is negated.

26.6.6.2.5 Message Transmission

As a result of the message buffer search described in Section 26.6.7, “Individual Message Buffer Search”, the CC triggers the message available transition MA for up to two transmit message buffers. This changes the message buffer state from Idle to CCMa and the message buffers can be used for message transmission in the next slot.

The CC transmits a message from a message buffer if both of the following two conditions are fulfilled at the start of the transmission slot:

1. the message buffer is in the message available state CCMa
2. the message data are still valid, (FR_MBCCSRn[CMT] = 1)

In this case, the CC triggers the TX transition and changes the message buffer state to CCTx. A transmit message buffer timing and state change diagram for message transmission is given in Figure 26-134. In this example, the message buffer with message buffer number n is Idle at the start of the search slot, matches the slot and cycle number of the next slot, and message buffer data are valid (FR_MBCCSRn[CMT] = 1).

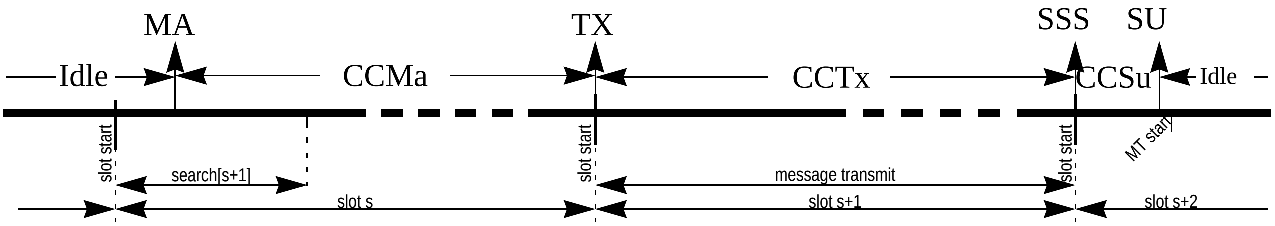


Figure 26-134. Message Transmission Timing

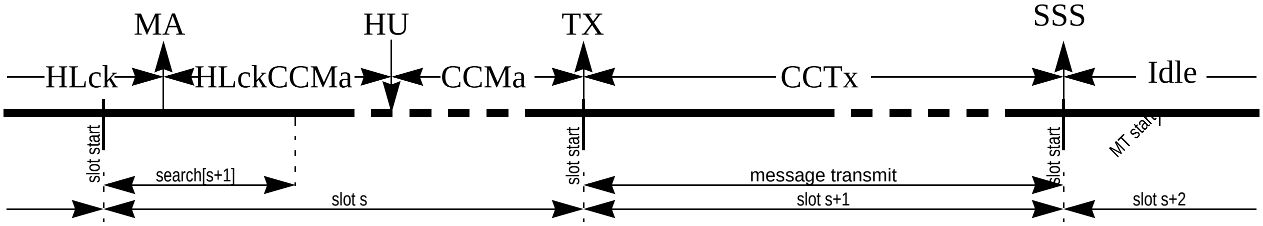


Figure 26-135. Message Transmission from HLck state with unlock

The amount of message data read from the flexray memory area and transferred to the FlexRay bus is determined by the following three items

1. the message buffer segment that the message buffer is assigned to, as defined by the Message Buffer Segment Size and Utilization Register (FR_MBSSUTR).
2. the message buffer data field size, as defined by the related field of the Message Buffer Data Size Register (FR_MBDSR)
3. the value of the PLDLEN field in the message buffer header field, as described in Section 26.6.5.2.1, “Frame Header Description”

If a message buffer is assigned to message buffer segment 1, and PLDLEN > MBSEG1DS, then 2 * MBSEG1DS bytes will be read from the message buffer data field and zero padding is used for the remaining bytes for the FlexRay bus transfer. If PLDLEN <= MBSEG1DS, the CC reads and transfers 2*PLDLEN bytes. The same holds for segment 2 and MBSEG2DS.

26.6.6.2.6 Null Frame Transmission

A static slot with slot number S is assigned to the CC for channel A, if at least one transmit message buffer is configured with the FR_MBFIDRn[FID] set to S and FR_MBCCFRn[CHA] set to 1. A Null Frame is transmitted in the static slot S on channel A, if this slot is assigned to the CC for channel A, and all transmit message buffers with FR_MBFIDRn[FID] = s and FR_MBCCFRn[CHA] = 1 are either not committed, i.e FR_MBCCSRn[CMT] = 0, or locked by the application (FR_MBCCSRn[LCKS] = 1), or the cycle counter filter is enabled and does not match.

Additionally, the application can clear the commit bit of a message buffer that is in the CCMa state, which is called *uncommit* or *transmit abort*. This message buffer will be used for null frame transmission.

As a result of the message buffer search described in Section 26.6.7, “Individual Message Buffer Search”, the CC triggers the slot assigned transition SA for up to two transmit message buffers if at least one of the conditions mentioned above is fulfilled for these message buffers. The transition SA changes the message buffer states from either Idle to CCSa or from HLck to HLckCCSa. In each case, these message buffers will be used for null frame transmission in the next slot. A message buffer timing and state change diagram for null frame transmission from Idle state is given in Figure 26-136.

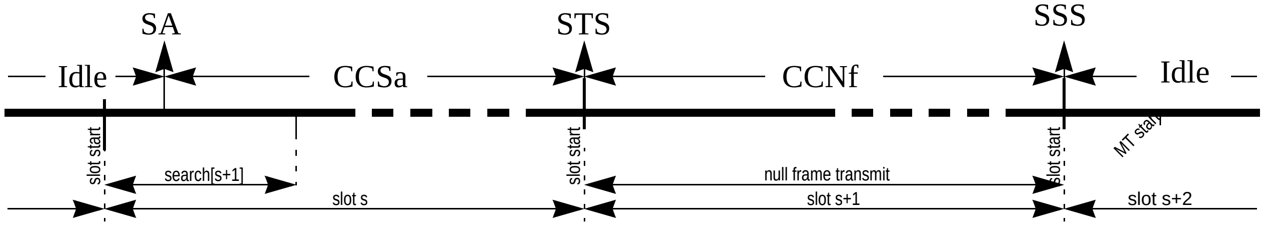


Figure 26-136. Null Frame Transmission from Idle state

A message buffer timing and state change diagram for null frame transmission from HLck state is given in Figure 26-137.

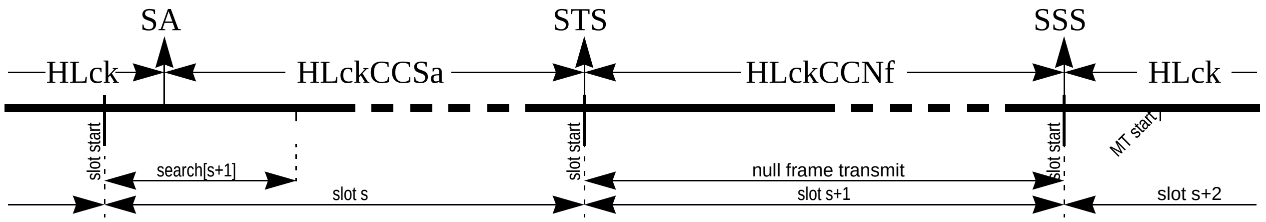


Figure 26-137. Null Frame Transmission from HLck state

If a transmit message buffer is in the CCSa or HLckCCSa state at the start of the transmission slot, a null frame is transmitted in any case, even if the message buffer is unlocked or committed before the transmission slot starts. A transmit message buffer timing and state change diagram for null frame transmission for this case is given in Figure 26-138.

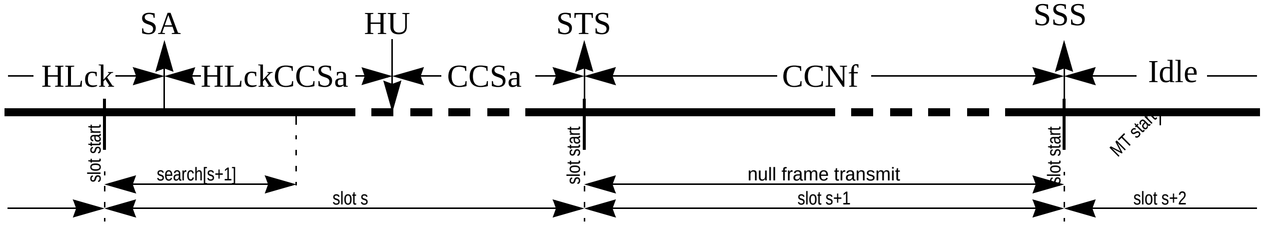


Figure 26-138. Null Frame Transmission from HLck state with unlock

Since the null frame transmission will not use the message buffer data, the application can lock/unlock the message buffer during null frame transmission. A transmit message buffer timing and state change diagram for null frame transmission for this case is given in Figure 26-139.

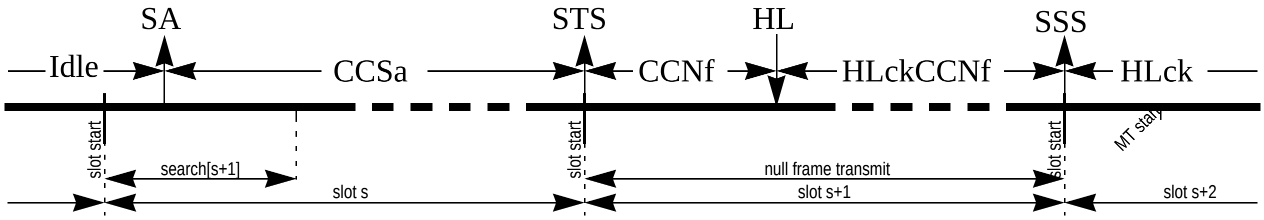


Figure 26-139. Null Frame Transmission from Idle state with locking

26.6.6.2.7 Message Buffer Status Update

After the end of each slot, the PE generates the slot status vector. Depending on the this status, the transmitted frame type, and the amount of transmitted data, the message buffer status is updated.

Message Buffer Status Update after Complete Message Transmission

The term complete message transmission refers to the fact that all payload data stored in the message buffer were send to FlexRay bus. In this case, the CC updates the slot status field of the message buffer and triggers the status updated transition SU. With the SU transition, the CC sets the message buffer interrupt flag FR_MBCCSRn[MBIF] to indicate the successful message transmission.

Depending on the transmission mode flag FR_MBCCFRn[MTM], the CC changes the commit flag FR_MBCCSRn[CMT] and the valid flag FR_MBCCSRn[DVAL]. If the FR_MBCCFRn[MTM] flag is negated, the message buffer is in the *event transmission mode*. In this case, each committed message is transmitted only once. The commit flag FR_MBCCSRn[CMT] is cleared with the SU transition. If the FR_MBCCFRn[MTM] flag is asserted, the message buffer is in the *state transmission mode*. In this case, each committed message is transmitted as long as the application provides new data or locks the message buffers. The CC will not clear the FR_MBCCSRn[CMT] flag at the end of transmission and will set the valid flag FR_MBCCSRn[DVAL] to indicate that the message will be transmitted again.

Message Buffer Status Update after Incomplete Message Transmission

The term incomplete message transmission refers to the fact that not all payload data that should be transmitted were send to FlexRay bus. This may be caused by the following regular conditions in the dynamic segment:

1. The transmission slot starts in a minislot with a minislot number greater than *pLatestTx*.
2. The transmission slot did not exist in the dynamic segment at all.

Additionally, an incomplete message transmission can be caused by internal communication errors. If those error occur, the Protocol Engine Communication Failure Interrupt Flag PECF_IF is set in the Protocol Interrupt Flag Register 1 (FR_PIFR1).

In any of these two cases, the status of the message buffer is not changed at all with the SU transition. The slot status field is not updated, the status and control flags are not changed, and the interrupt flag is not set.

Message Buffer Status Update after Null Frame Transmission

After the transmission of a null frame, the status of the message buffer that was used for the null frame transmission is not changed at all. The slot status field is not updated, the status and control flags are not changed, and the interrupt flag is not set.

26.6.6.3 Receive Message Buffers

The section provides a detailed description of the functionality of the receive message buffers. If receive message buffers are used it is required to configure the related receive shadow buffer as described in Section 26.6.3.2, “Receive Shadow Buffers”

A receive message buffer is used to receive a message from the FlexRay Bus based on individual filter criteria. The CC uses the receive message buffer to provide the following data to the application

1. message data received
2. information about the reception process
3. status information about the slot in which the message was received

A individual message buffer with message buffer number *n* is configured as a receive message buffer by the following configuration settings

- FR_MBCCSRn[MTD] = 0 (receive message buffer)

To certain message buffer fields, both the application and the CC have access. To ensure data consistency, a message buffer locking scheme is implemented that is used to control the access to the data, control, and status bits of a message buffer. The access regions for receive message buffers are depicted in Figure 26-140. A description of the regions is given in Table 26-114. If an region is active as indicated in Table 26-115, the access scheme given for that region applies to the message buffer.

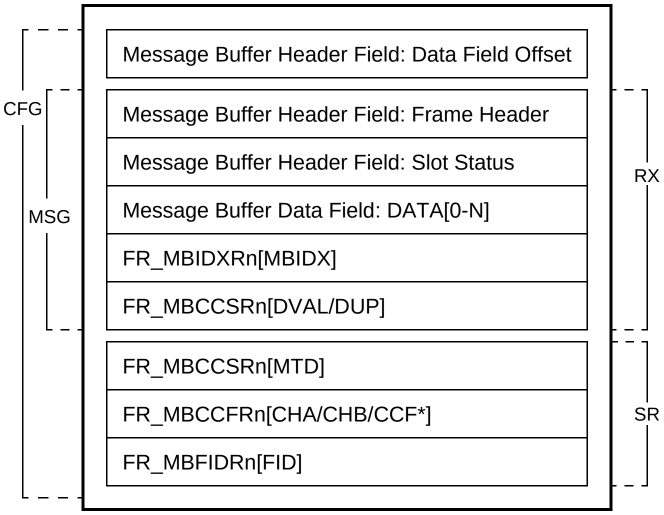


Figure 26-140. Receive Message Buffer Access Regions

Table 26-114. Receive Message Buffer Access Region Description

Region	Access from		Region used for
	Application	Module	
CFG	read/write	-	Message Buffer Configuration, Message Data and Status Access
MSG	read/write	-	Message Data, Header, and Status Access
RX	-	write-only	Message Reception and Status Update
SR	-	read-only	Message Buffer Search Data

The trigger bits FR_MBCCSRn[EDT] and FR_MBCCSRn[LCKT] and the interrupt enable bit FR_MBCCSRn[MBIE] are not under access control and can be accessed from the application at any time. The status bits FR_MBCCSRn[EDS] and FR_MBCCSRn[LCKS] are not under access control and can be accessed from the CC at any time.

The interrupt flag FR_MBCCSRn[MBIF] is not under access control and can be accessed from the application and the CC at any time. CC set access has higher priority.

The CC restricts its access to the regions depending on the current state of the message buffer. The application must adhere to these restrictions in order to ensure data consistency. The receive message buffer states are given in Figure 26-141. A description of the message buffer states is given in Table 26-110, which also provides the access scheme for the access regions.

The status bits FR_MBCCSRn[EDS] and FR_MBCCSRn[LCKS] provide the application with the required status information. The internal status information is not visible to the application.

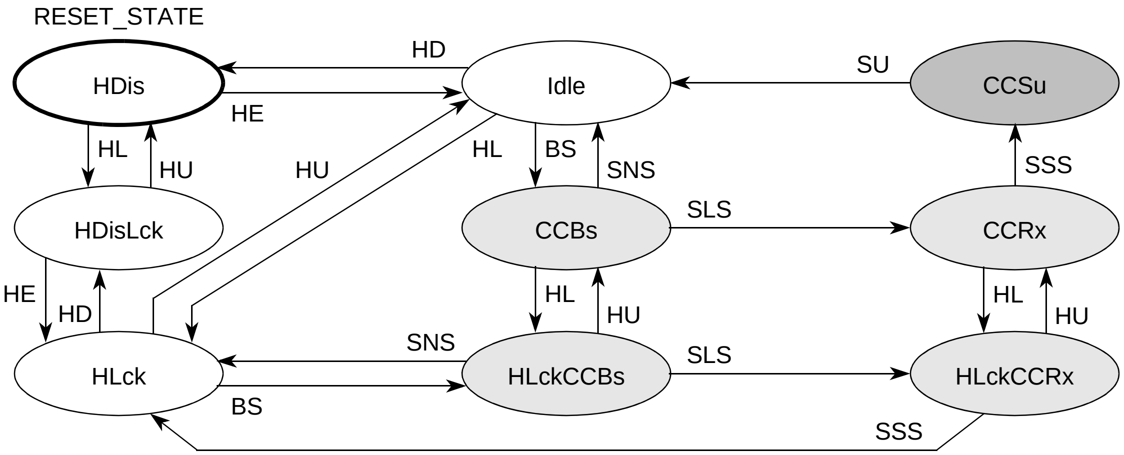


Figure 26-141. Receive Message Buffer States

Table 26-115. Receive Message Buffer States and Access (Sheet 2 of 2)

State	FR_MBCCSRn		Access from		Description
	EDS	LCKS	Appl.	Module	
Idle	1	0	–	SR	**Idle** - Message Buffer is idle. Included in message buffer search.
HDis	0	0	CFG	–	**Disabled** - Message Buffer under configuration. Excluded from message buffer search.

Table 26-115. Receive Message Buffer States and Access (Sheet 2 of 2)

State	FR_MBCCSRn		Access from		Description
	EDS	LCKS	Appl.	Module	
HDisLck	0	1	CFG	–	**Disabled and Locked** - Message Buffer under configuration. Excluded from message buffer search.
HLck	1	1	MSG	–	**Locked** - Applications access to data, control, and status. Included in message buffer search.
CCBs	1	0	–	–	**Buffer Subscribed** - Message buffer subscribed for reception. Filter matches next (slot, cycle, channel) tuple.
HLckCCBs	1	1	MSG	–	**Locked and Buffer Subscribed** - Applications access to data, control, and status. Message buffer subscribed for reception.
CCRx	1	0	–	–	**Message Receive** - Message data received into related shadow buffer.
HLckCCRx	1	1	MSG	–	Locked and Message Receive - Applications access to data, control, and status. Message data received into related shadow buffer.
CCSu	1	0	–	RX	**Status Update** - Message buffer status update. Update of status flags, the slot status field, and the header index.

26.6.6.3.1 Message Buffer Transitions

Application Transitions

The application transitions that can be triggered by the application using the commands described in Table 26-116. The application issues the commands by writing to the Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn). Only one command can be issued with one write access. Each command is executed immediately. If the command is ignored, it must be issued again.

Message Buffer Enable and Disable

The enable and disable commands issued by writing 1 to the trigger bit FR_MBCCSRn[EDT]. The transition that will be triggered by each of these command depends on the current value of the status bit FR_MBCCSRn[EDS]. If the command triggers the disable transition HD and the message buffer is in one of the states CCBs, HLckCCBs, or CCRx, the disable transition has no effect (command is ignored) and the message buffer state is not changed. No notification is given to the application.

If the communication controller is started as a non-coldstart node, and the message buffers are configured and enabled in the POC config state for Slot 1, then the message buffer cannot be disabled in the INTEGRATION_LISTEN state by directly writing 1 to the EDT bit. To facilitate this, a FREEZE command needs to be issued just before running the message buffer disable for slot 1. Executing this command enables the message buffer disable during the LISTEN states.

Message Buffer Lock and Unlock

The lock and unlock commands issued by writing 1 to the trigger bit FR_MBCCSRn[LCKT]. The transition that will be triggered by each of these commands depends on the current value of the status bit FR_MBCCSRn[LCKS]. If the command triggers the lock transition HL while the message buffer is in the state CCRx, the lock transition has no effect (command is ignored) and message buffer state is not changed. In this case, the message buffer lock error flag LCK_EF in the CHI Error Flag Register (FR_CHIERFR) is set.

Table 26-116. Receive Message Buffer Application Transitions

Transition	Host Command	Condition	Description
HE	FR_MBCCSRn[EDT]:= 1	FR_MBCCSRn[EDS] = 0	Application triggers message buffer enable.
HD		FR_MBCCSRn[EDS] = 1	Application triggers message buffer disable.
HL	FR_MBCCSRn[LCKT]:= 1	FR_MBCCSRn[LCKS] = 0	Application triggers message buffer lock.
HU		FR_MBCCSRn[LCKS] = 1	Application triggers message buffer unlock.

Module Transitions

The module transitions that can be triggered by the CC are described in Table 26-117. Each transition will be triggered for certain message buffers when the related condition is fulfilled.

Table 26-117. Receive Message Buffer Module Transitions

Transition	Condition	Description
BS	slot match and Cycle Counter match	Buffer Subscribed - The message buffer filter matches next slot and cycle.
SLS	slot start	Slot Start - Start of either Static Slot or Dynamic Slot.
SNS	symbol window start or NIT start	Symbol Window or NIT Start - Start of either Symbol Window or NIT.
SSS	slot start or symbol window start or NIT start	Slot or Segment Start - Start of either Static Slot, Dynamic Slot, Symbol Window, or NIT.
SU	status updated	Status Updated - Slot Status field, message buffer status flags, header index updated. Interrupt flag set.

Transition Priorities

The application can trigger only one transition at a time. There is no need to specify priorities among them.

As shown in Table 26-118, the module transitions have a higher priority than the application transitions. For all states except the CCRx state, a module transition and the application lock/unlock transition HL/HU and can be executed at the same time. The result state is reached by first applying the module transition and subsequently the application transition to the intermediately reached state. For example, if the message buffer is in the buffer subscribed state CCBs and the module triggers the slot start transition SLS at the same time as the application locks the message buffer by the HL transition, the intermediate state is CCRx and the resulting state is locked buffer subscribed state HLckCCRx.

Table 26-118. Receive Message Buffer Transition Priorities

State	Priorities	Description
module vs. application		
Idle	BS > HD	Buffer Subscribed > Message Buffer Disable
HLck	BS > HD	Buffer Subscribed > Message Buffer Disable
CCRx	SSS > HL	Slot or Segment Start > Message Buffer Lock

26.6.6.3.2 Message Reception

As a result of the message buffer search, the CC changes the state of up to two enabled receive message buffers from either idle state Idle or locked state HLck to the either subscribed state CCBs or locked buffer subscribed state HLckCCBs by triggering the buffer subscribed transition BS.

If the receive message buffers for the next slot are assigned to both channels, then at most one receive message buffer is changed to a buffer subscribed state.

If more than one matching message buffers assigned to a certain channel, then only the message buffer with the lowest message buffer number is in one of the states mentioned above.

With the start of the next static or dynamic slot the module trigger the slot start transition SLS. This changes the state of the subscribed receive message buffers from either CCBs to CCRx or from HLckCCBs to HLckCCRx, respectively.

During the reception slot, the received frame data are written into the shadow buffers. For details on receive shadow buffers, see Section 26.6.6.3.5, "Receive Shadow Buffers Concept". The data and status of the receive message buffers that are the CCRx or HLckCCRx are not modified in the reception slot.

26.6.6.3.3 Message Buffer Update

With the start of the next static or dynamic slot or with the start of the symbol window or NIT, the module triggers the slot or segment start transition SSS. This transition changes the state of the receiving receive message buffers from either CCRx to CCSu or from HLckCCRx to HLck, respectively.

If a message buffer was in the locked state HLckCCRx, no update will be performed. The received data are lost. This is indicated by setting the Frame Lost Channel A/B Error Flag FRLA_EF/FRLB_EF in the CHI Error Flag Register (FR_CHIERFR).

If a message buffer was in the CCRx state it is now in the CCSu state. After the evaluation of the slot status provided by the PE the message buffer is updated. The message buffer update depends on the slot status bits and the segment the message buffer is assigned to. This is described in Table 26-119.

Table 26-119. Receive Message Buffer Update (Continued)

vSS!ValidFrame	*vRF!Header!NFIndicator*	Update description
1	1	Valid non-null frame received. - Message Buffer Data Field updated. - Frame Header Field updated. - Slot Status Field updated. - DUP:= 1 - DVAL:= 1 - MBIF:= 1

Table 26-119. Receive Message Buffer Update (Continued)

vSS!ValidFrame	*vRF!Header!NFIndicator*	**Update description**
1	0	Valid null frame received. - Message Buffer Data Field *not* updated. - Frame Header Field *not* updated. - Slot Status Field updated. - DUP:= 0 - DVAL *not* changed - MBIF:= 1
0	x	No valid frame received. - Message Buffer Data Field not updated. - Frame Header Field not updated. - Slot Status Field updated. - DUP:= 0 - DVAL *not* changed. - MBIF:= 1, if the slot was not an empty dynamic slot. **Note:** An empty dynamic slot is indicated by the following frame and slot status bit values: *vSS!ValidFrame* = 0 and *vSS!SyntaxError* = 0 and *vSS!ContentError* = 0 and *vSS!BViolation* = 0.

NOTE

If the number of the last slot in the current communication cycle on a given channel is *n*, then all receive message buffers assigned to this channel with FR_MBFIDRn[FID] > *n* will not be updated at all.

When the receive message buffer update has finished the status updated transition SU is triggered, which changes the buffer state from CCSu to Idle. An example receive message buffer timing and state change diagram for a normal frame reception is given in Figure 26-142.

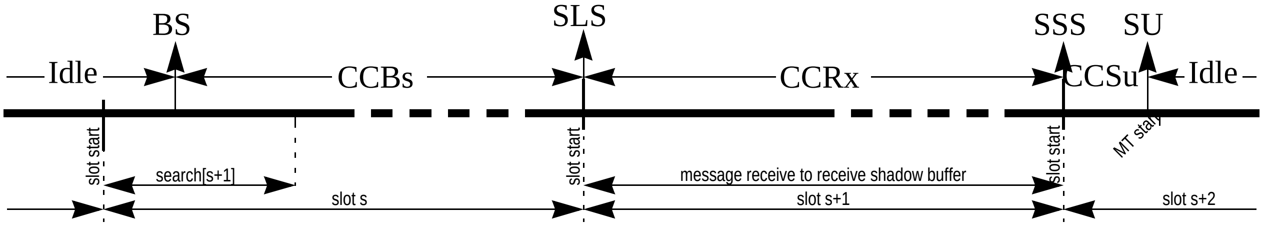


Figure 26-142. Message Reception Timing

The amount of message data written into the message buffer data field of the receive shadow buffer is determined by the following two items:

1. the message buffer segment that the message buffer is assigned to, as defined by the Message Buffer Segment Size and Utilization Register (FR_MBSSUTR).
2. the message buffer data field size, as defined by the related field of the Message Buffer Data Size Register (FR_MBDSR)
3. the number of bytes received over the FlexRay bus

If the message buffer is assigned to the message buffer segment 1, and the number of received bytes is greater than 2*FR_MBDSR.MBSEG1DS, the CC writes only 2*FR_MBDSR.MBSEG1DS bytes into the message buffer data field of the receive shadow buffer. If the number of received bytes is less than 2*FR_MBDSR.MBSEG1DS, the CC writes only the received number of bytes and will not change the trailing bytes in the message buffer data field of the receive shadow buffer. The same holds for the message buffer segment 2 with FR_MBDSR.MBSEG2DS.

26.6.6.3.4 Received Message Access

To access the message data received over the FlexRay bus, the application reads the message data stored in the message buffer data field of the corresponding receive message buffer. The access to the message buffer data field is described in Section 26.6.3.1, “Individual Message Buffers”.

The application can read the message buffer data field if the receive message buffer is one of the states HDis, HDisLck, or HLck. If the message buffer is in one of these states, the CC will not change the content of the message buffer.

26.6.6.3.5 Receive Shadow Buffers Concept

The receive shadow buffer concept applies only to individual receive message buffers. The intention of this concept is to ensure that only syntactically and semantically valid received non-null frames are presented to the application in a receive message buffer. The basic structure of a receive shadow buffer is described in Section 26.6.3.2, “Receive Shadow Buffers”.

The receive shadow buffers temporarily store the received frame header and message data. After the slot boundary the slot status information is generated. If the slot status information indicates the reception of the valid non-null frame (see Table 26-119), the CC writes the slot status into the slot status field of the receive shadow buffer and exchanges the content of the Message Buffer Index Registers (FR_MBIDXRn) with the content of the corresponding internal shadow buffer index register. In all other cases, the CC writes the slot status into the identified receive message buffer, depending on the slot status and the FlexRay segment the message buffer is assigned to.

The shadow buffer concept, with its index exchange, results in the fact that the flexray memory area located message buffer associated to an individual receive message buffer changes after successful reception of a valid frame. This means that the message buffer area in the flexray memory area accessed by the application for reading the received message is different from the initial setting of the message buffer. Therefore, the application must not rely on the index information written initially into the Message Buffer Index Registers (FR_MBIDXRn). Instead, the index of the message buffer header field must be fetched from the Message Buffer Index Registers (FR_MBIDXRn).

26.6.7 Individual Message Buffer Search

This section provides a detailed description of the message buffer search algorithm.

The message buffer search determines for each enabled channel if a slot *s* in a communication cycle *c* is assigned for frame or null frame transmission or if it is subscribed for frame reception on that channel.

The message buffer search is a sequential algorithm which is invoked at the following protocol related events:

1. NIT start
2. slot start in the static segment
3. minislot start in the dynamic segment

The message buffer search within the NIT searches for message buffers assigned or subscribed to slot 1. The message buffer search within slot *n* searches for message buffers assigned or subscribed to slot *n+1*.

In general, the message buffer search for the next slot *n* considers only message buffers that are

- Enabled (FR_MBCCSRn[EDS] = 1) and
- Matches the next slot *n* (FR_MBFIDRn[FID] = *n*)

On top of that, for the static segment only those message buffers are considered, that match the condition of at least one row of Table 26-120. For the dynamic segment only those message buffers are considered, that match the condition of at least one row of Table 26-121. These message buffers are called *matching* message buffers.

For each enabled channel the message buffer search may identify multiple *matching* message buffers. Among all matching message buffers the message buffers with highest priority according to Table 26-120 for the static segment and according to Table 26-121 for the dynamic segment are selected.

Table 26-120. Message Buffer Search Priority (static segment)

Priority	MTD	LCKS	CMT	CCFM[1]	Description	Transition
(highest) 0	1	0	1	1	transmit buffer, matches cycle count, not locked and committed	MA
1	1	-	0	1	transmit buffer, matches cycle count, not committed	SA
	1	1	-	1	transmit buffer, matches cycle count, locked	SA
2	1	-	-	-	transmit buffer	SA
3	0	0	n/a	1	receive buffer, matches cycle count, not locked	SB
(lowest) 4	0	1	n/a	1	receive buffer, matches cycle count, locked	SB

[1] Cycle Counter Filter Match, see Section 26.6.7.1, “Message Buffer Cycle Counter Filtering”

Table 26-121. Message Buffer Search Priority (dynamic segment)

Priority	MTD	LCKS	CMT	CCFM[1]	Description	Transition
(highest) 0	1	0	1	1	transmit buffer, matches cycle count, not locked and committed	MA
1	0	0	n/a	1	receive buffer, matches cycle count, not locked	SB
(lowest) 2	0	1	n/a	1	receive buffer, matches cycle count, locked	SB

[1] Cycle Counter Filter Match, see Section 26.6.7.1, “Message Buffer Cycle Counter Filtering”

If there are multiple message buffer with highest priority, the message buffer with the lowest message buffer number is selected. All message buffer which have the highest priority must have a consistent channel assignment as specified in Section 26.6.7.2, “Message Buffer Channel Assignment Consistency”.

Depending on the message buffer channel assignment the same message buffer can be found for both channel A and channel B. In this case, this message buffer is used as described in Section 26.6.3.1, “Individual Message Buffers”.

26.6.7.1 Message Buffer Cycle Counter Filtering

The message buffer cycle counter filter is a value-mask filter defined by the CCFE, CCFMSK, and CCFVAL fields in the Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn). This filter determines a set of communication cycles in which the message buffer is considered for message reception or message transmission. If the cycle counter filter is disabled (CCFE = 0), this set of cycles consists of all communication cycles.

If the cycle counter filter of a message buffer does not match a certain communication cycle number, this message buffer is not considered for message transmission or reception in that communication cycle. In case of a transmit message buffer assigned to a slot in the static segment, though, this buffer is added to the matching message buffers to indicate the slot assignment and to trigger the null frame transmission.

The cycle counter filter of a message buffer matches the communication cycle with the number CYCCNT if at least one of the following conditions evaluates to true:

$$\mathrm{MBCCFRn[CCFE]} = 0 \qquad \textbf{\textit{Eqn. 12}}$$

$$\mathrm{YCCNT}\ \&\ \mathrm{MBCCFRn[CCFMSK]} = \mathrm{MBCCFRn[CCFVAL]}\ \&\ \mathrm{MBCCFRn[CCFMSK} \qquad \textbf{\textit{Eqn. 13}}$$

26.6.7.2 Message Buffer Channel Assignment Consistency

The message buffer channel assignment given by the CHA and CHB bits in the Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn) defines the channels on which the message buffer will receive or transmit. The message buffer with number *n* transmits or receives on channel A if FR_MBCCFRn[CHA] = 1 and transmits or receives on channel B if FR_MBCCFRn[CHB] = 1.

To ensure correct message buffer operation, all message buffers assigned to the same slot and with the same priority must have a *consistent* channel assignment. That means they must be either assigned to one channel only, or must be assigned to *both* channels. The behavior of the message buffer search is not defined, if both types of channel assignments occur for one slot and priority. An inconsistent channel assignment for message buffer 0 and message buffer 1 is depicted in Figure 26-143.

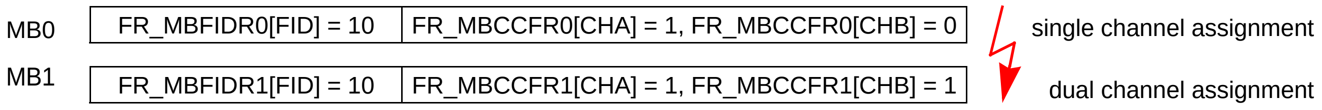


Figure 26-143. Inconsistent Channel Assignment

26.6.7.3 Node Related Slot Multiplexing

The term *Node Related Slot Multiplexing* applies to the dynamic segment only and refers to the functionality if there are transmit as well as receive message buffers are configured for the same slot.

According to Table 26-121 the transmit buffer is only found if the cycle counter filter matches, and the buffer is not locked and committed. In all other cases, the receive buffer will be found. Thus, if the block has no data to transmit in a dynamic slot, it is able to receive frames on that slot.

26.6.7.4 Message Buffer Search Error

There are two kinds of errors which may occur during message buffer search[1].

26.6.7.4.1 Message Buffer Search Start while Running

If the message buffer search is running in slot *n-1* and the next message buffer search start event appears due to the start of slot *n*, the message buffer search engine is stopped and the Message Buffer Search Error Flag MBS_EF is set in the CHI Error Flag Register (FR_CHIERFR). As a result of this stop, no individual message buffer is identified for transmission or reception in slot *n*. Additionally, the search engine will not be started in slot *n*, and consequently no individual message buffer is identified for transmission or reception in slot *n+1*.

A message buffer search error appears only if the CHI frequency is too slow to allow the search through all message buffers to be completed within the NIT or a minislot.

For more details of minimum required CHI frequency see Section 26.7.5, “Number of Usable Message Buffers”.

26.6.7.4.2 Illegal Message Buffer Index Found

If the message buffer search has finished the message buffer search in slot *n-1*, it retrieves the data offset values for the found message buffers and the receive shadow buffers. If one of these message buffers contains an illegal message buffer index, the Message Buffer Search Error Flag MBS_EF is set in the CHI Error Flag Register (FR_CHIERFR) is set and no individual message buffer is identified for transmission or reception in slot *n*. The legal message buffer index values for the individual and receive shadow buffers are specified in Section 26.5.2.52, “Receive Shadow Buffer Index Register (FR_RSBIR)” and Section 26.5.2.82, “Message Buffer Index Registers (FR_MBIDXRn)”.

26.6.8 Individual Message Buffer Reconfiguration

The initial configuration of each individual message buffer can be changed even when the protocol is not in the *POC:config* state. This is referred to as individual message buffer *reconfiguration*. The configuration bits and fields that can be changed are given in the section on Specific Configuration Data. The common configuration data given in the section on Specific Configuration Data cannot be reconfigured when the protocol is out of the *POC:config* state.

26.6.8.1 Reconfiguration Schemes

Depending on the target and destination basic state of the message buffer that is to be reconfigured, there are three reconfiguration schemes.

---

[1] The FIFO reception is not affected by the search errors. Additionally, if no rx buffer has been found due to an search error, the received frame is considered for FIFO reception.

26.6.8.1.1 Basic Type Not Changed (RC1)

A reconfiguration will not change the basic type of the individual message buffer, if the message buffer transfer direction bit FR_MBCCSRn[MTD] are not changed. This type of reconfiguration is denoted by RC1 in Figure 26-144. Transmit and receive message buffers can be RC1-reconfigured when in the HDis or HDisLck state.

26.6.8.1.2 Buffer Type Not Changed (RC2)

A reconfiguration will not change the buffer type of the individual message buffer. This type of reconfiguration is denoted by RC2 in Figure 26-144. It applies to transmit and receive message buffers. Transmit and receive message buffers can be RC2-reconfigured when in the HDis or HDisLck state.

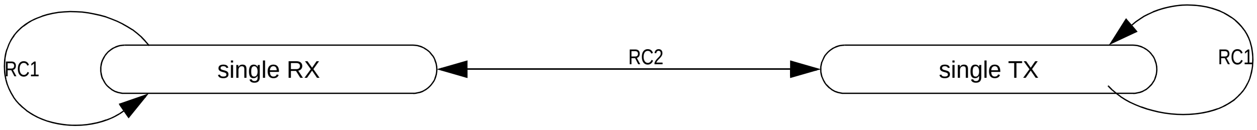


Figure 26-144. Message Buffer Reconfiguration Scheme

26.6.9 Receive FIFOs

This section provides the functional description of the two receive FIFOs.

26.6.9.1 Overview

The two receive FIFOs implement the queued message buffer concept defined by the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*. One FIFO is assigned to channel A, the other FIFO is assigned to channel B. Both FIFOs work completely independent from each other.

The message buffer structure of each FIFO is described in Section 26.6.3.3, “Receive FIFO”. The area in the flexray memory area for each of the two FIFOs is characterized by:

- The FIFO system memory base address
- The index of the first FIFO entry given by Receive FIFO Start Index Register (FR_RFSIR)
- The data field offset of the data field belonging to the first FIFO entry given by Receive FIFO Start Data Offset Register (FR_RFSDOR)
- The number of FIFO entries and the length of each FIFO entry as given by Receive FIFO Depth and Size Register (RFDSR)

26.6.9.2 FIFO Configuration

The FIFOs can be configured for two different locations of the system memory base address via the FIFO address mode bit FAM in the Module Configuration Register (FR_MCR).

26.6.9.2.1 Single System Memory Base Address Mode

This mode is configured, when the FIFO address mode flag FR_MCR[FAM] is set to 0. In this mode, the location of the system memory base address for the FIFO buffers is System Memory Base Address Register (FR_SYMBADR).

26.6.9.2.2 Dual System Memory Base Address Mode

This mode is configured, when the FIFO address mode flag FR_MCR[FAM] is set to 1. In this mode, the location of the system memory base address for the FIFO buffers is Receive FIFO System Memory Base Address Register (FR_RFSYMBADR).

The FIFO control and configuration data are given in Section 26.6.3.7, “Receive FIFO Control and Configuration Data”. The configuration of the FIFOs consists of two steps.

The first step is the allocation of the required amount of flexray memory area for the FlexRay window. This includes the allocation of the message buffer header area and the allocation of the message buffer data fields. For more details see Section 26.6.4, “Flexray Memory Area Layout”.

The second step is the programming of the configuration data register while the PE is in *POC:config.*

The following steps configure the layout of the FIFO.

- Configure the FIFO update and address modes in Module Configuration Register (FR_MCR)
- Configure the FIFO system memory base address
- Configure the Receive FIFO Start Index Register (FR_RFSIR) with the first message buffer header index that belongs to the FIFO
- Configure the Receive FIFO Start Data Offset Register (FR_RFSDOR) with the data field offset of the data field belonging to the first message buffer that belongs to the FIFO
- Configure the Receive FIFO Depth and Size Register (RFDSR) with FIFO entry size
- Configure the Receive FIFO Depth and Size Register (RFDSR) with FIFO depth
- Configure the FIFO Filters

26.6.9.3 FIFO Periodic Timer

The FIFO periodic timer is used to generate an FIFO almost-full interrupt at certain point in time, if the almost-full watermark is not reached, but the FIFO is not empty. This can be used to prevent frames from get stuck in the FIFO for a long time.

The FIFO periodic timer is configured via the Receive FIFO Periodic Timer Register (FR_RFPTR). If the periodic timer duration FR_RFPTR[PTD] is configured to 0x0000, the periodic timer is continuously expired. If the periodic timer duration FR_RFPTR[PTD] is configured to 0x3FFF, the periodic timer never expires. If the periodic timer is configured to a value *ptd*, greater than 0x0000 and smaller 0x3FFF, the periodic timer expires and is restarted at the start of every communication cycle, and expires and is restarted after *ptd* macroticks have been elapsed.

26.6.9.4 FIFO Reception

The FIFO reception is a CC internal operation.

A message frame reception is directed into the FIFO, if no individual message buffer is assigned for transmission or subscribed for reception for the current slot. In this case the FIFO filter path shown in Figure 26-145 is activated.

If the FIFO filter path indicates that the received frame has to be appended to the FIFO and the FIFO is not full, the CC writes the received frame header into the message buffer header field indicated by the CC internal FIFO write index. The frame payload data are written into the corresponding message buffer data field. If the status of the received frame indicates a valid non-null frame, the slot status information is written into the message buffer header field and the CC internal FIFO write index is updated by 1 and the FIFO fill level FLA (FLB) in the Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR) is incremented.If the status of the received frame indicates an invalid or null frame, the frame is not appended to the FIFO.

26.6.9.5 FIFO Almost-Full Interrupt Generation

If the FIFO fill level FLA (FLB) is updated after a frame reception and exceeds the FIFO watermark level WM, i.e., $FLA > WM_A$ ($FLB > WM_B$), then the FIFO almost-full interrupt flag FR_GIFER[FAFAIF] (FR_GIFER[FAFBIF]) is asserted.If the periodic timer expires, and FIFOA (FIFOB) is not empty, i.e., $FLA > 0$ ($FLB > 0$), then the FIFO almost-full interrupt flag FR_GIFER[FAFAIF] (FR_GIFER[FAFBIF]) is asserted.

26.6.9.6 FIFO Overflow Error Generation

If the FIFOA (FIFOB) is full, i.e., $FLA = FIFO\_DEPTH_A$ ($FLB = FIFO\_DEPTH_B$) and the conditions for a FIFO reception as described in Section 26.6.9.4, “FIFO Reception” are fulfilled, then the FIFO overflow error flag FR_CHIERFR[FOVA_EF] (FR_CHIERFR[FOVB_EF]) is asserted.

26.6.9.7 FIFO Message Access

The FIFOA (FIFOB) contains valid messages if the FIFO fill level given in the fields FLA (FLB) in the Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR) is greater than 0. The Receive FIFO A Read Index Register (FR_RFARIR) and the (Receive FIFO B Read Index Register (FR_RFBRIR)) point to a message buffer with valid content and the oldest frames stored in the FIFO.The respective read data field offsets can be calculated according to Equation 6.

If the FIFO fill level FLA (FLB) in the Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR) is 0, than the FIFOA (FIFOB) contains no valid messages and the corresponding read index register Receive FIFO A Read Index Register (FR_RFARIR) or (Receive FIFO B Read Index Register (FR_RFBRIR) point to a message buffer with invalid content. In this case the application must not read data from this FIFO.

To access the oldest message in the FIFOA (FIFOB), the application first reads the read index RDIDX out of the Receive FIFO A Read Index Register (FR_RFARIR) (Receive FIFO B Read Index Register (FR_RFBRIR)). This read index points to the message buffer header field of the oldest message buffer that contains valid received message data. The data field offset belonging to this message buffer must be calculated by the application according to Equation 6. The application can access the message data as described in Section 26.6.3.3, “Receive FIFO”. When the application has read the message buffer data and status information, it can update the FIFO as described in Section 26.6.9.8, “FIFO Update”.

26.6.9.8 FIFO Update

The application updates the FIFOA (FIFOB) by writing a pop count value *pc* different from 0 to the PCA (PCB) field in the Receive FIFO Fill Level and POP Count Register (FR_RFFLPCR).

As a result of the this operation, the CC removes the oldest *pc* entries from FIFOA (FIFOB).

If the specified pop count value *pc* is greater than the current fill level *fl* provided in FLA (FAB) field, then only *fl* entries are removed from the FIFOA (FIFOB), the remaining *fl-pc* requested pop operations are discarded without any notification. In this case FIFOA (FIFOB) is empty after the update operation.

The read index in the Receive FIFO A Read Index Register (FR_RFARIR) (Receive FIFO B Read Index Register (FR_RFBRIR)) is incremented by the number of removed items. If the read index reaches the top of the FIFO, it wraps around to the FIFO start index defined in Receive FIFO Start Index Register (FR_RFSIR) automatically.

26.6.9.8.1 FIFO Interrupt Flag Update

The FIFO Interrupt Flag Update mode is configured, when the FIFO update mode flag FR_MCR[FUM] is set to 0. In this mode FIFOA (FIFOB) will be updated by 1 entry, when the interrupt flag FR_GIFER[FAFAIF] (FR_GIFER[FAFBIF]) is written with 1 by the application.

If the FIFO is empty, the update request is ignored without any notification.

The read index in the Receive FIFO A Read Index Register (FR_RFARIR) (Receive FIFO B Read Index Register (FR_RFBRIR)) is incremented by 1, if the FIFO was not empty. If the read index reaches the top of the FIFO, it wraps around to the FIFO start index automatically.

26.6.9.9 FIFO Filtering

The FIFO filtering is activated after all enabled individual receive message buffers have been searched without success for a message buffer to receive the current frame.

The CC provides three sets of FIFO filters. The FIFO filters are applied to valid non-null frames only. The FIFO will not receive invalid or null-frames. For each FIFO filter, the pass criteria is specified in the related section given below. Only frames that have passed all filters will be appended to the FIFO. The FIFO filter path is depicted in Figure 26-145.

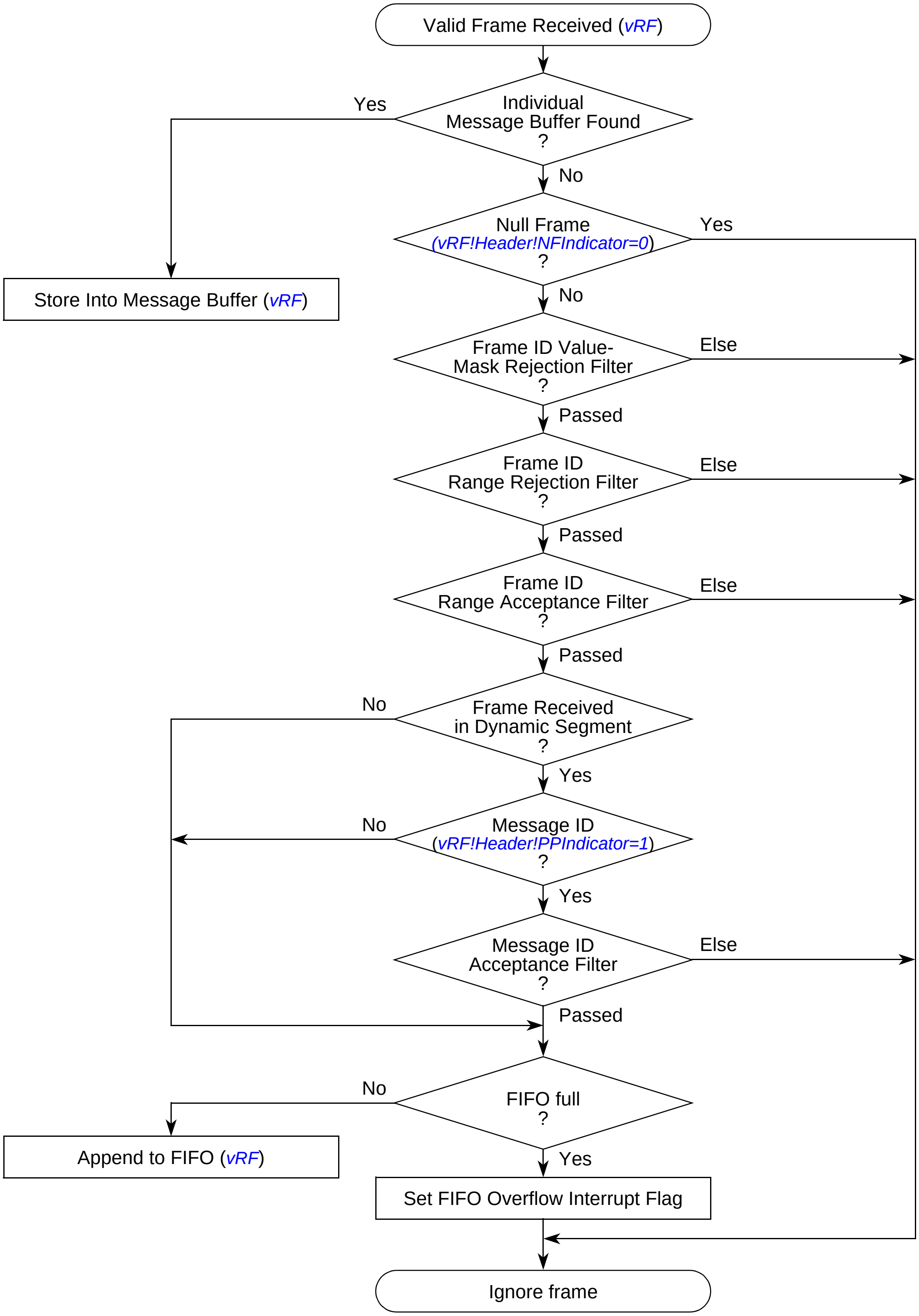


Figure 26-145. Received Frame FIFO Filter Path

A received frame passes the FIFO filtering if it has passed all three type of filter.

26.6.9.9.1 RX FIFO Frame ID Value-Mask Rejection Filter

The frame ID value-mask rejection filter is a value-mask filter and is defined by the fields in the Receive FIFO Frame ID Rejection Filter Value Register (FR_RFFIDRFVR) and the Receive FIFO Frame ID Rejection Filter Mask Register (FR_RFFIDRFMR). Each received frame with a frame ID FID that does not match the value-mask filter value passes the filter, i.e., is not rejected.

Consequently, a received valid frame with the frame ID FID passes the RX FIFO Frame ID Value-Mask Rejection Filter if Equation 14 is fulfilled.

$$\mathrm{ID\ \&\ FR\_RFFIDRFMR[FIDRFMSK]} \neq \mathrm{FR\_RFFIDRFVR[FIDRFVAL]\ \&\ FR\_RFFIDRFMR[FIDRFMSK} \qquad \textbf{\textit{Eqn. 14}}$$

The RX FIFO Frame ID Value-Mask Rejection Filter can be configured to pass all frames by the following settings.

- FR_RFFIDRFVR[FIDRFVAL]:= 0x000 and FR_RFFIDRFMR[FIDRFMSK]:= 0x7FF

Using the settings above, only the frame with frame ID 0 will be rejected, which is an invalid frame. All other frames will pass.

The RX FIFO Frame ID Value-Mask Rejection Filter can be configured to reject all frames by the following settings.

- FR_RFFIDRFMR[FIDRFMSK]:= 0x000

Using the settings above, Equation 14 can never be fulfilled (0!= 0) and thus all frames are rejected; no frame will pass. This is the reset value for the RX FIFO.

26.6.9.9.2 RX FIFO Frame ID Range Rejection Filter

Each of the four RX FIFO Frame ID Range filters can be configured as a rejection filter. The filters are configured by the Receive FIFO Range Filter Configuration Register (FR_RFRFCFR) and controlled by the Receive FIFO Range Filter Control Register (FR_RFRFCTR). The RX FIFO Frame ID range filters apply to all received valid frames. A received frame with the frame ID FID passes the RX FIFO Frame ID Range rejection filters if either no rejection filter is enabled, or, for all of the enabled RX FIFO Frame ID Range rejection filters, i.e., FR_RFRFCTR[FiMD] = 1 and FR_RFRFCTR[FiEN] = 1, Equation 15 is fulfilled.

$$\mathrm{FID} < \mathrm{FR\_RFRFCFR}_{\mathrm{SEL}}[\mathrm{SID}_{\mathrm{IBD\,=\,0}}]) \text{ or } (\mathrm{FR\_RFRFCFR}_{\mathrm{SEL}}[\mathrm{SID}_{\mathrm{IBD\,=\,1}}] < \mathrm{FID} \qquad \textbf{\textit{Eqn. 15}}$$

Consequently, all frames with a frame ID that fulfills Equation 16 for at least one of the enabled rejection filters will be rejected and thus not pass.

$$\mathrm{R\_RFRFCFR}_{\mathrm{SEL}}[\mathrm{SID}_{\mathrm{IBD\,=\,0}}] \leq \mathrm{FID} \leq \mathrm{FR\_RFRFCFR}_{\mathrm{SEL}}[\mathrm{SID}_{\mathrm{IBD\,=\,1}} \qquad \textbf{\textit{Eqn. 16}}$$

26.6.9.9.3 RX FIFO Frame ID Range Acceptance filter

Each of the four RX FIFO Frame ID Range filters can be configured as an acceptance filter. The filters are configured by the Receive FIFO Range Filter Configuration Register (FR_RFRFCFR) and controlled by the Receive FIFO Range Filter Control Register (FR_RFRFCTR). The RX FIFO Frame ID range filters apply to all received valid frames. A received frame with the frame ID FID passes the RX FIFO Frame ID Range acceptance filters if either no acceptance filter is enabled, or, for at least one of the enabled RX FIFO Frame ID Range acceptance filters, i.e., FR_RFRFCTR[FiMD] = 0 and FR_RFRFCTR[FiEN] = 1, Equation 17 is fulfilled.

$$\text{R\_RFRFCFR}_{\text{SEL}}[\text{SID}_{\text{IBD = 0}}] \le \text{FID} \le \text{FR\_RFRFCFR}_{\text{SEL}}[\text{SID}_{\text{IBD = 1}} \qquad \textit{Eqn. 17}$$

26.6.9.9.4 RX FIFO Message ID Acceptance Filter

The RX FIFO Message ID Acceptance Filter is a value-mask filter and is defined by the Receive FIFO Message ID Acceptance Filter Value Register (FR_RFMIDAFVR) and the Receive FIFO Message ID Acceptance Filter Mask Register (FR_RFMIDAFMR). This filter applies only to valid frames received in the dynamic segment with the payload preamble indicator bit PPI set to 1. All other frames will pass this filter.

A received valid frame in the dynamic segment with the payload preamble indicator bit PPI set to 1 and with the message ID MID (the first two bytes of the payload) will pass the RX FIFO Message ID Acceptance Filter if Equation 18 is fulfilled.

$$\text{ID \& FR\_RFMIDAFMR[MIDAFMSK]} = \text{FR\_RFMIDAFMR[MIDAFVAL] \& FR\_RFMIDAFMR[MIDAFMSK]} \qquad \textit{Eqn. 18}$$

The RX FIFO Message ID Acceptance Filter can be configured to accept all frames by setting

- FR_RFMIDAFMR[MIDAFMSK]:= 0x000

Using the settings above, Equation 18 is always fulfilled and all frames will pass.

26.6.10 Channel Device Modes

This section describes the two FlexRay channel device modes that are supported by the CC.

26.6.10.1 Dual Channel Device Mode

In the dual channel device mode, both FlexRay ports are connected to physical FlexRay bus lines. The FlexRay port consisting of FR_A_RX, FR_A_TX, and $\overline{\text{FR\_A\_TX\_EN}}$ is connected to the physical bus channel A and the FlexRay port consisting of FR_B_RX, FR_B_TX, and $\overline{\text{FR\_B\_TX\_EN}}$ is connected to the physical bus channel B. The dual channel system is shown in Figure 26-146.

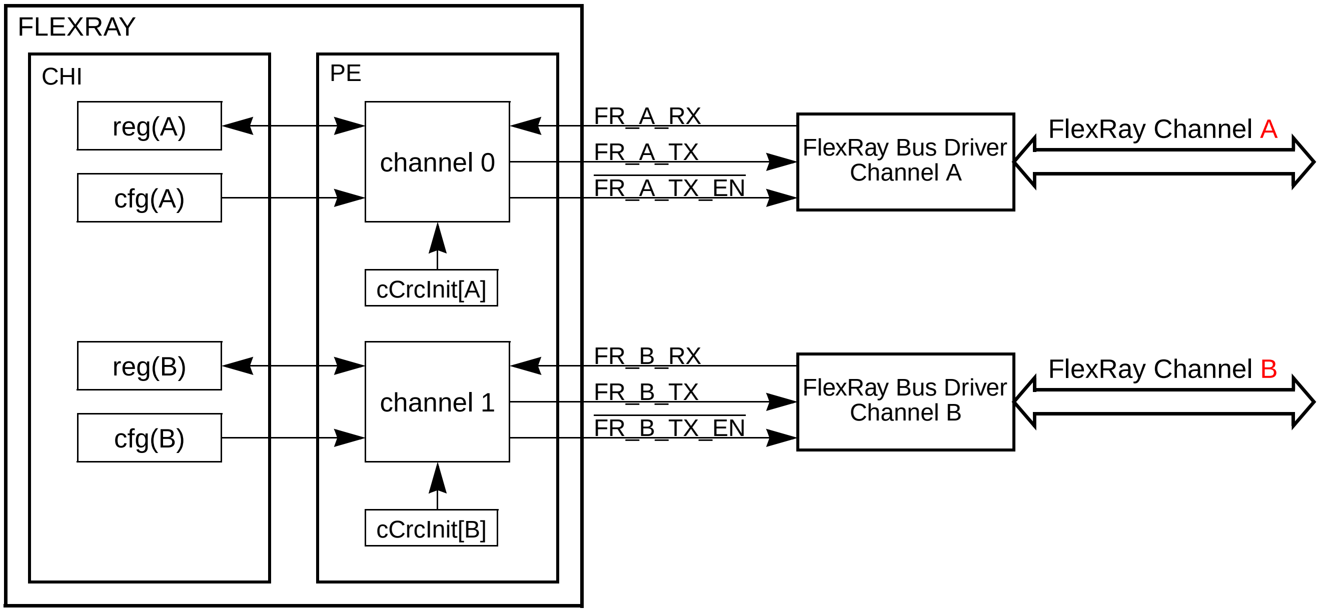


Figure 26-146. Dual Channel Device Mode

26.6.10.2 Single Channel Device Mode

The single channel device mode supports devices that have only one FlexRay port available. This FlexRay port consists of the signals FR_A_RX, FR_A_TX, and FR_A_TX_EN and can be connected to either the physical bus channel A (shown in Figure 26-147) or the physical bus channel B (shown in Figure 26-148).

If the device is configured as a single channel device by setting FR_MCR[SCM] to 1, only the internal channel A and the FlexRay Port A is used. Depending on the setting of FR_MCR[CHA] and FR_MCR[CHB], the internal channel A behaves either as a FlexRay Channel A or FlexRay Channel B. The bit FR_MCR[CHA] must be set, if the FlexRay Port A is connected to a FlexRay Channel A. The bit FR_MCR[CHB] must be set if the FlexRay Port A is connected to a FlexRay Channel B. The two FlexRay channels differ only in the initial value for the frame CRC *cCrcInit*. For a single channel device, the application can access and configure only the registers related to internal channel A.

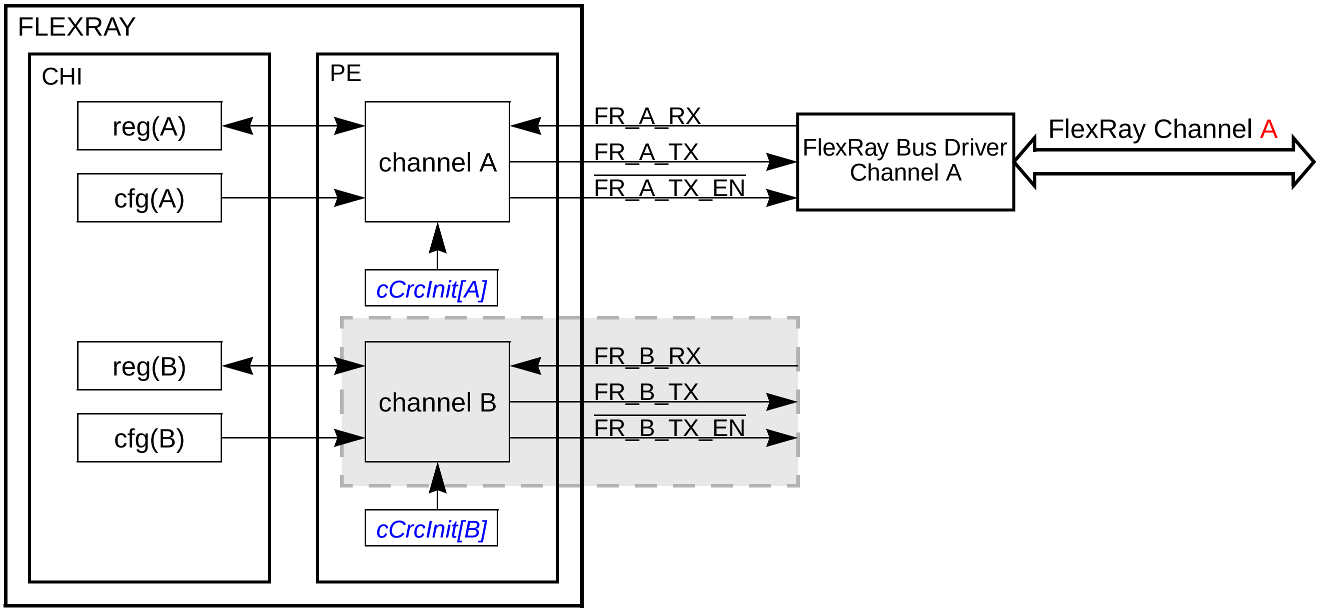


Figure 26-147. Single Channel Device Mode (Channel A)

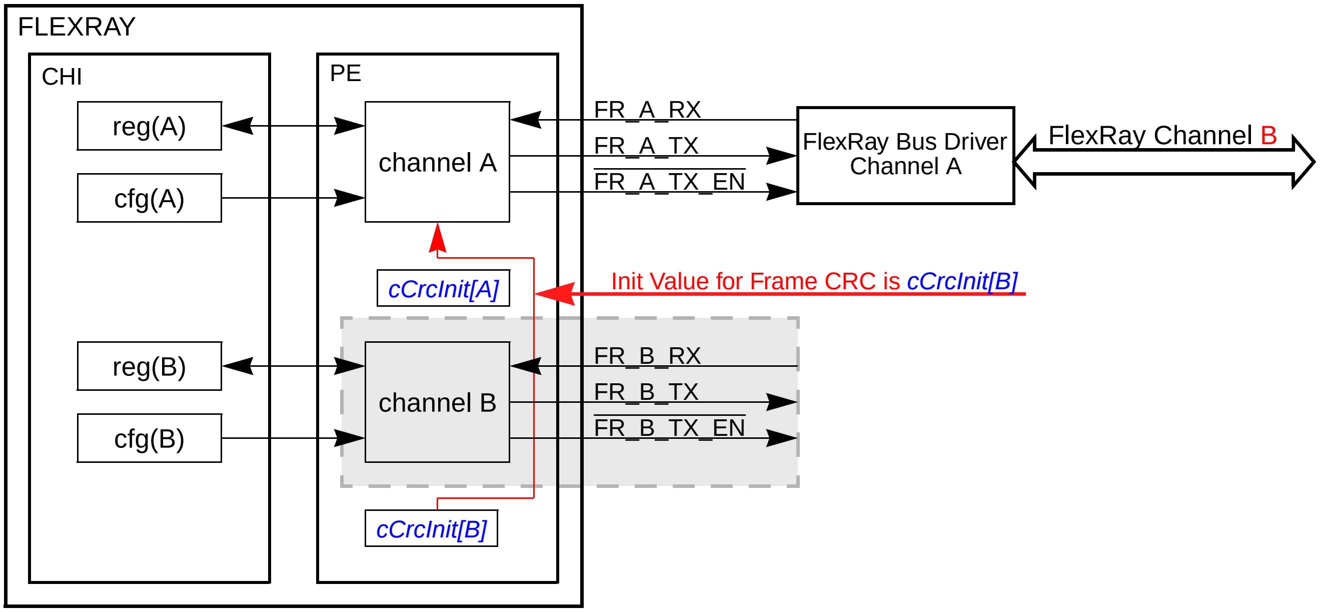


Figure 26-148. Single Channel Device Mode (Channel B)

26.6.11 External Clock Synchronization

The application of the external rate and offset correction is triggered when the application writes to the EOC_AP and ERC_AP fields in the Protocol Operation Control Register (FR_POCR). The PE applies the external correction values in the next even-odd cycle pair as shown in Figure 26-149 and Figure 26-150.

NOTE

> The values provided in the EOC_AP and ERC_AP fields are the values that were written from the application most recently. If these value were already applied, they will not be applied in the current cycle pair again.

If the offset correction applied in the NIT of cycle 2n+1 shall be affect by the external offset correction, the EOC_AP field must be written to after the start of cycle 2n and before the end of the static segment of cycle 2n+1. If this field is written to after the end of the static segment of cycle 2n+1, it is not guaranteed that the external correction value is applied in cycle 2n+1. If the value is not applied in cycle 2n+1, then the value will be applied in the cycle 2n+3. Refer to Figure 26-149 for timing details.

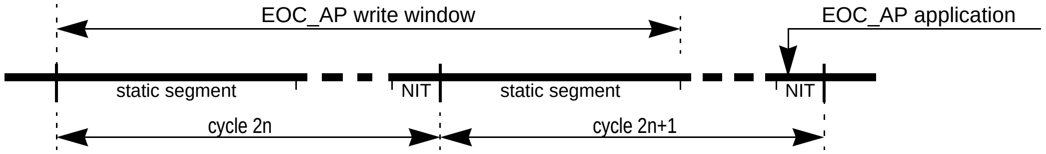


Figure 26-149. External Offset Correction Write and Application Timing

If the rate correction for the cycle pair [2n+2, 2n+3] shall be affect by the external offset correction, the ERC_AP field must be written to after the start of cycle 2n and before the end of the static segment start of cycle 2n+1. If this field is written to after the end of the static segment of cycle 2n+1, it is not guaranteed that the external correction value is applied in cycle pair [2n+2, 2n+3]. If the value is not applied for cycle pair [2n+2, 2n+3], then the value will be applied for cycle pair [2n+4, 2n+5]. Refer to Figure 26-150 for details.

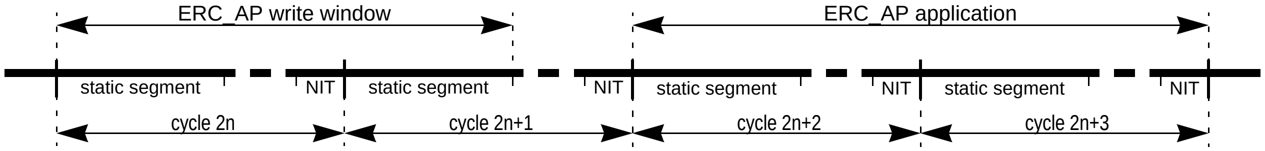


Figure 26-150. External Rate Correction Write and Application Timing

26.6.12 Sync Frame ID and Sync Frame Deviation Tables

The FlexRay protocol requires the provision of a snapshot of the Synchronization Frame ID tables for the even and odd communication cycle for both channels. The CC provides the means to write a copy of these internal tables into the flexray memory area and ensures application access to consistent tables by means of table locking. Once the application has locked the table successfully, the CC will not overwrite these tables and the application can read a consistent snapshot.

NOTE

Only synchronization frames that have passed the synchronization frame filters are considered for clock synchronization and appear in the sync frame tables.

26.6.12.1 Sync Frame ID Table Content

The Sync Frame ID Table is a snapshot of the protocol related variables *vsSyncIdListA* and *vsSyncIdListB* for each even and odd communication cycle. This table provides a list of the frame IDs of the synchronization frames received on the corresponding channel and cycle that are used for the clock synchronization.

26.6.12.2 Sync Frame Deviation Table Content

The Sync Frame Deviation Table is a snapshot of the protocol related variable zsDev(id)(oe)(ch)!Value. Each Sync Frame Deviation Table entry provides the deviation value for the sync frame, with the frame ID presented in the corresponding entry in the Sync Frame ID Table.

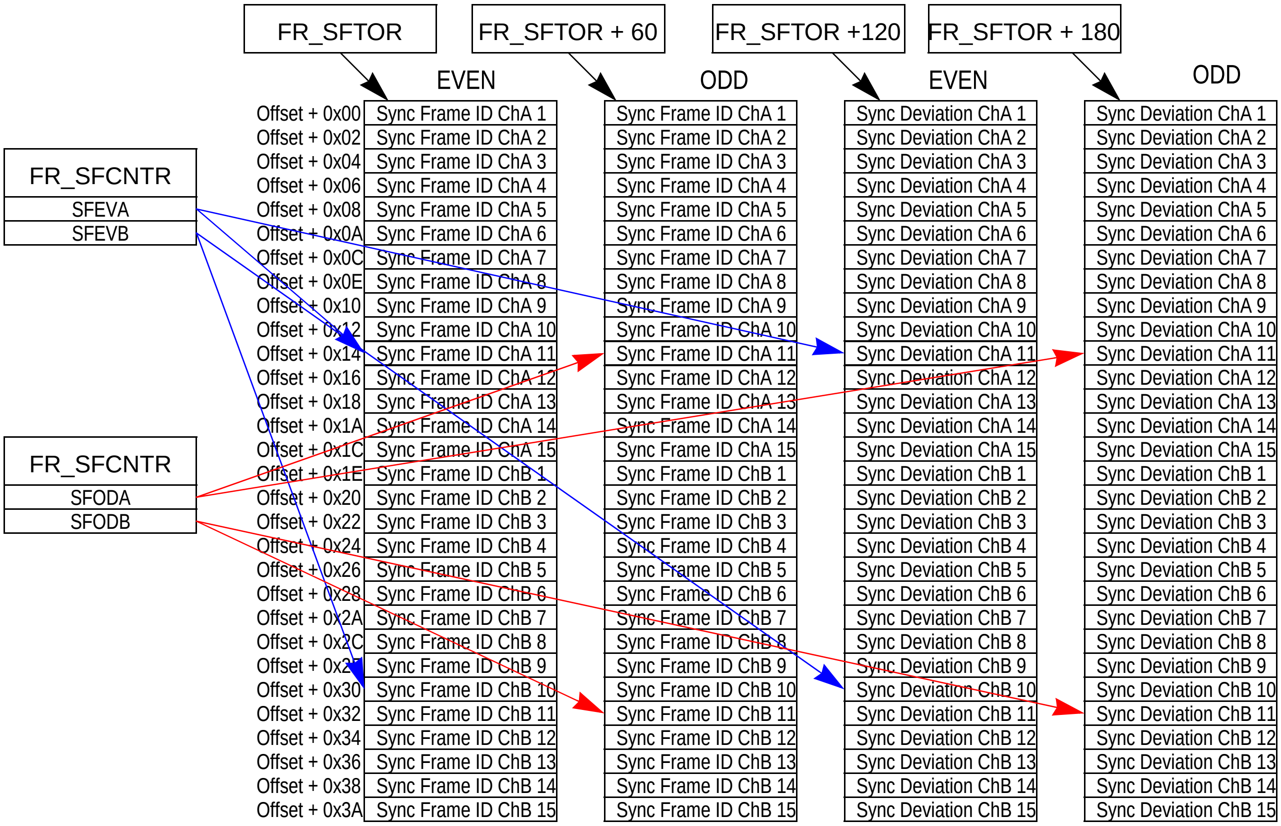


Figure 26-151. Sync Table Memory Layout

26.6.12.3 Sync Frame ID and Sync Frame Deviation Table Setup

The CC writes a copy of the internal synchronization frame ID and deviation tables into the flexray memory area if requested by the application. The application must provide the appropriate amount of flexray memory area for the tables. The memory layout of the tables is given in Figure 26-151. Each table occupies 120 16-bit entries.

While the protocol is in *POC:config* state, the application must program the offsets for the tables into the Sync Frame Table Offset Register (FR_SFTOR).

26.6.12.4 Sync Frame ID and Sync Frame Deviation Table Generation

The application controls the generation process of the Sync Frame ID and Sync Frame Deviation Tables into the flexray memory area using the Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR). A summary of the copy modes is given in Table 26-122.

Table 26-122. Sync Frame Table Generation Modes

FR_SFTCCSR			Description
OPT	SDVEN	SIDEN	
0	0	0	No Sync Frame Table copy
0	0	1	Sync Frame ID Tables will be copied continuously
0	1	0	Reserved
0	1	1	Sync Frame ID Tables and Sync Frame Deviation Tables will be copied continuously
1	0	0	No Sync Frame Table copy
1	0	1	Sync Frame ID Tables for next even-odd-cycle pair will be copied
1	1	0	Reserved
1	1	1	Sync Frame ID Tables and Sync Frame Deviation Tables for next even-odd-cycle pair will be copied

The Sync Frame Table generation process is described in the following for the even cycle. The same sequence applies to the odd cycle.

If the application has enabled the sync frame table generation by setting FR_SFTCCSR[SIDEN] to 1, the CC starts the update of the even cycle related tables after the start of the NIT of the next even cycle. The CC checks if the application has locked the tables by reading the FR_SFTCCSR[ELKS] lock status bit. If this bit is set, the CC will not update the table in this cycle. If this bit is cleared, the CC locks this table and starts the table update. To indicate that these tables are currently updated and may contain inconsistent data, the CC clears the even table valid status bit FR_SFTCCSR[EVAL]. Once all table entries related to the even cycle have been transferred into the flexray memory area, the CC sets the even table valid bit FR_SFTCCSR[EVAL] and the Even Cycle Table Written Interrupt Flag EVT_IF in the Protocol Interrupt Flag Register 1 (FR_PIFR1). If the interrupt enable flag EVT_IE is set, an interrupt request is generated.

To read the generated tables, the application must lock the tables to prevent the CC from updating these tables. The locking is initiated by writing a 1 to the even table lock trigger FR_SFTCCSR[ELKT]. When the even table is not currently updated by the CC, the lock is granted and the even table lock status bit FR_SFTCCSR[ELKS] is set. This indicates that the application has successfully locked the even sync tables and the corresponding status information fields SFRA, SFRB in the Sync Frame Counter Register

(FR_SFCNTR). The value in the FR_SFTCCSR[CYCNUM] field provides the number of the cycle that this table is related to.

The number of available table entries per channel is provided in the FR_SFCNTR[SFEVA] and FR_SFCNTR[SFEVB] fields. The application can now start to read the sync table data from the locations given in Figure 26-151.

After reading all the data from the locked tables, the application must unlock the table by writing to the even table lock trigger FR_SFTCCSR[ELKT] again. The even table lock status bit FR_SFTCCSR[ELKS] is reset immediately.

If the sync frame table generation is disabled, the table valid bits FR_SFTCCSR[EVAL] and FR_SFTCCSR[EVAL] are reset when the counter values in the Sync Frame Counter Register (FR_SFCNTR) are updated. This is done because the tables stored in the flexray memory area are no longer related to the values in the Sync Frame Counter Register (FR_SFCNTR).

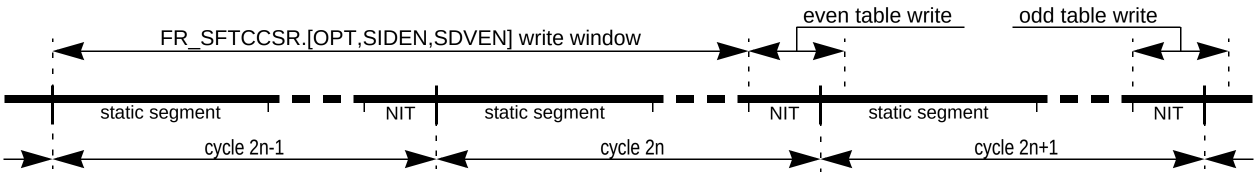


Figure 26-152. Sync Frame Table Trigger and Generation Timing

26.6.12.5 Sync Frame Table Access

The sync frame tables will be transferred into the flexray memory area during the table write windows shown in Figure 26-152. During the table write, the application cannot lock the table that is currently written. If the application locks the table outside of the table write window, the lock is granted immediately.

26.6.12.5.1 Sync Frame Table Locking and Unlocking

The application locks the even/odd sync frame table by writing 1 to the lock trigger bit ELKT/OLKT in the Sync Frame Table Configuration, Control, Status Register (FR_SFTCCSR). If the affected table is not currently written to the flexray memory area, the lock is granted immediately, and the lock status bit ELKS/OLKS is set. If the affected table is currently written to the flexray memory area, the lock is not granted. In this case, the application must issue the lock request again until the lock is granted.

The application unlocks the even/odd sync frame table by writing 1 to the lock trigger bit ELKT/OLKT. The lock status bit ELKS/OLKS is cleared immediately.

26.6.13 MTS Generation

The CC provides a flexible means to request the transmission of the Media Access Test Symbol MTS in the symbol window on channel A or channel B.

The application can configure the set of communication cycles in which the MTS will be transmitted over the FlexRay bus by programming the CYCCNTMSK and CYCCNTVAL fields in the MTS A Configuration Register (FR_MTSACFR) and MTS B Configuration Register (MTSBCFR).

The application enables or disables the generation of the MTS on either channel by setting or clearing the MTE control bit in the MTS A Configuration Register (FR_MTSACFR) or MTS B Configuration Register (MTSBCFR). If an MTS is to be transmitted in a certain communication cycle, the application must set the MTE control bit during the static segment of the preceding communication cycle.

The MTS is transmitted over channel A in the communication cycle with number CYCCNT, if Equation 20, Equation 21, and Equation 21 are fulfilled.

$$\mathrm{FR\_PSR0[PROTSTATE]} = \textit{POC:normal active}$$

Eqn. 19

$$\mathrm{FR\_MTSACRF[MTE]} = 1$$

Eqn. 20

$$\begin{gathered}\mathrm{CYCCNT\ \&\ FR\_MTSACFR[CYCCNTMSK]} = \\ \mathrm{FR\_MTSACFR[CYCCNTVAL]\ \&\ FR\_MTSACFR[CYCCNTMSK]}\end{gathered}$$

Eqn. 21

The MTS is transmitted over channel B in the communication cycle with number CYCCNT, if Equation 19, Equation 22, and Equation 23 are fulfilled.

$$\mathrm{FR\_MTSBCRF[MTE]} = 1$$

Eqn. 22

$$\begin{gathered}\mathrm{CYCCNT\ \&\ FR\_MTSBCFR[CYCCNTMSK]} = \\ \mathrm{FR\_MTSBCFR[CYCCNTVAL]\ \&\ FR\_MTSBCFR[CYCCNTMSK]}\end{gathered}$$

Eqn. 23

26.6.14 Key Slot Transmission

26.6.14.1 Key Slot Assignment

A key slot is assigned to the CC if the key_slot_id field in the Protocol Configuration Register 18 (FR_PCR18) is configured with a value greater than 0 and less or equal to number_of_static_slots in Protocol Configuration Register 2 (FR_PCR2), otherwise no key slot is assigned.

26.6.14.2 Key Slot Transmission in *POC:startup*

If a key slot is assigned and the CC is in the *POC:startup* state, startup null frames will be transmitted as specified by FlexRay Communications System Protocol Specification, Version 2.1 Rev A.

26.6.14.3 Key Slot Transmission in *POC:normal active*

If a key slot is assigned and the CC is in *POC:normal active*, a frame of the type as shown in Table 26-123 is transmitted. If a transmit message buffer is configured for the key slot and a valid message is available, a message frame is transmitted (see Section 26.6.6.2.5, “Message Transmission”). If no transmit message buffer is configured for the key slot or no valid message is available, a null frame is transmitted (see Section 26.6.6.2.6, “Null Frame Transmission”).

Table 26-123. Key Slot Frame Type

FR_PCR11[key_slot_used_for_sync]	FR_PCR11[key_slot_used_for_startup]	key slot frame type
0	0	normal frame
0	1	normal frame[1]
1	0	sync frame
1	1	startup frame

[1] The frame transmitted has an semantically incorrect header and will be detected as an invalid frame at the receiver.

26.6.15 Sync Frame Filtering

Each received synchronization frame must pass the Sync Frame Acceptance Filter and the Sync Frame Rejection Filter before it is considered for clock synchronization. If the synchronization frame filtering is globally disabled, i.e., the SFFE control bit in the Module Configuration Register (FR_MCR) is cleared, all received synchronization frames are considered for clock synchronization. If a received synchronization frame did not pass at least one of the two filters, this frame is processed as a normal frame and is not considered for clock synchronization.

26.6.15.1 Sync Frame Acceptance Filtering

The synchronization frame acceptance filter is implemented as a value-mask filter. The value is configured in the Sync Frame ID Acceptance Filter Value Register (FR_SFIDAFVR) and the mask is configured in the Sync Frame ID Acceptance Filter Mask Register (FR_SFIDAFMR). A received synchronization frame with the frame ID FID passes the sync frame acceptance filter, if Equation 24 or Equation 25evaluates to true.

$$\mathrm{FR\_MCR[SFFE]} = 0 \qquad \textit{Eqn. 24}$$

$$\mathrm{ID\ \&\ FR\_SFIDAFMR[FMSK]} = \mathrm{FR\_SFIDAFVR[FVAL]\ \&\ FR\_SFIDAFMR[FMSK} \qquad \textit{Eqn. 25}$$

NOTE

Sync frames are transmitted in the static segment only. Thus FID <= 1023.

26.6.15.2 Sync Frame Rejection Filtering

The synchronization frame rejection filter is a comparator. The compare value is defined by the Sync Frame ID Rejection Filter Register (FR_SFIDRFR). A received synchronization frame with the frame ID FID passes the sync frame rejection filter if Equation 26 or Equation 27 evaluates to true.

$$\mathrm{FR\_MCR[SFFE]} = 0 \qquad \textit{Eqn. 26}$$

$$\mathrm{FID} \neq \mathrm{FR\_SFIDRFR[SYNFRID]} \qquad \textit{Eqn. 27}$$

NOTE

Sync frames are transmitted in the static segment only. Thus FID <= 1023.

26.6.16 Strobe Signal Support

The CC provides a number of strobe signals for observing internal protocol timing related signals in the protocol engine. The signals are listed and described in Table 26-12.

26.6.16.1 Strobe Signal Assignment

Each of the strobe signals listed in Table 26-12 can be assigned to one of the four strobe ports using the Strobe Signal Control Register (FR_STBSCR). To assign multiple strobe signals, the application must write multiple times to the Strobe Signal Control Register (FR_STBSCR) with appropriate settings.

To read out the current settings for a strobe signal with number N, the application must execute the following sequence.

1. Write to FR_STBSCR with WMD = 1 and SEL = N. (updates SEL field only)
2. Read STBCSR.
   The SEL field provides N and the ENB and STBPSEL fields provides the settings for signal N.

26.6.16.2 Strobe Signal Timing

This section provides detailed timing information of the strobe signals with respect to the protocol engine clock.

The strobe signals display internal PE signals. Due to the internal architecture of the PE, some signals are generated several PE clock cycles before the actual action is performed on the FlexRay Bus. These signals are listed in Table 26-12 with a negative clock offset. An example waveform is given in Figure 26-153.

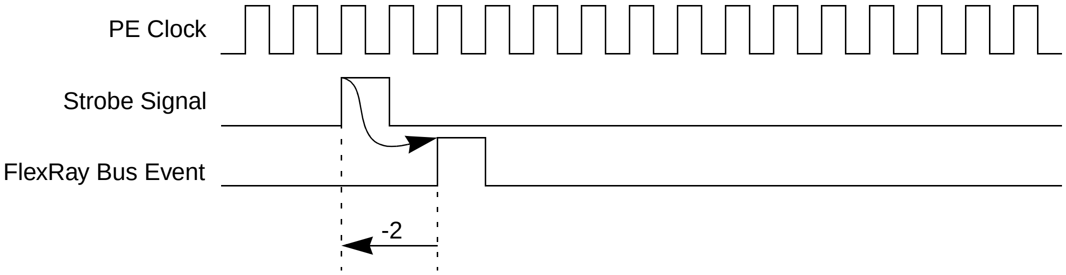


Figure 26-153. Strobe Signal Timing (type = pulse, clk_offset = -2)

Other signals refer to events that occurred on the FlexRay Bus some cycles before the strobe signal is changed. These signals are listed in Table 26-12 with a positive clock offset. An example waveform is given in Figure 26-154.

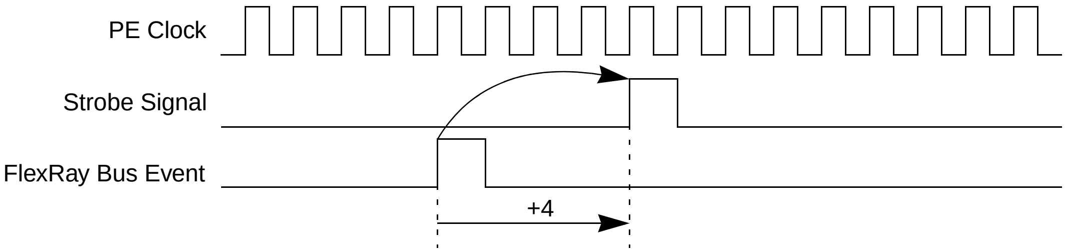


Figure 26-154. Strobe Signal Timing (type = pulse, clk_offset = +4)

26.6.17 Timer Support

The CC provides two timers, which run on the FlexRay time base. Each timer generates a maskable interrupt when it reaches a configured point in time. Timer T1 is an absolute timer. Timer T2 can be configured to be an absolute or a relative timer. Both timers can be configured to be repetitive. In the non-repetitive mode, timer stops if it expires. In repetitive mode, timer is restarted when it expires.

Both timers are active only when the protocol is in *POC:normal active* or *POC:normal passive* state. If the protocol is not in one of these modes, the timers are stopped. The application must restart the timers when the protocol has reached the *POC:normal active* or *POC:normal passive* state.

26.6.17.1 Absolute Timer T1

The absolute timer T1 has the protocol cycle count and the macrotick count as the time base. The timer 1 interrupt flag TI1_IF in the Protocol Interrupt Flag Register 0 (FR_PIFR0) is set at the macrotick start event, if Equation and Equation 29 are fulfilled

$$\text{YCTR[CTCCNT]} \,\&\, \text{FR\_TI1CYSR[T1\_CYC\_MSK]} = \text{FR\_TI1CYSR[T1\_CYC\_VAL]} \,\&\, \text{FR\_TI1CYSR[T1\_CYC\_MSK} \qquad \textit{Eqn. 28}$$

$$\text{FR\_MTCTR[MTCT]} = \text{FR\_TI1MTOR[T1\_MTOFFSET]} \qquad \textit{Eqn. 29}$$

If the timer 1 interrupt enable bit TI1_IE in the Protocol Interrupt Enable Register 0 (FR_PIER0) is asserted, an interrupt request is generated.

The status bit T1ST is set when the timer is triggered, and is cleared when the timer expires and is non-repetitive. If the timer expires but is repetitive, the T1ST bit is not cleared and the timer is restarted immediately. The T1ST is cleared when the timer is stopped.

26.6.17.2 Absolute / Relative Timer T2

The timer T2 can be configured to be an absolute or relative timer by setting the T2_CFG control bit in the Timer Configuration and Control Register (FR_TICCR). The status bit T2ST is set when the timer is triggered, and is cleared when the timer expires and is non-repetitive. If the timer expires but is repetitive, the T2ST bit is not cleared and the timer is restarted immediately. The T2ST is cleared when the timer is stopped.

26.6.17.2.1 Absolute Timer T2

If timer T2 is configured as an absolute timer, it has the same functionality timer T1 but the configuration from Timer 2 Configuration Register 0 (FR_TI2CR0) and Timer 2 Configuration Register 1 (FR_TI2CR1) is used. On expiration of timer T2, the interrupt flag TI2_IF in the Protocol Interrupt Flag Register 0 (FR_PIFR0) is set. If the timer 1 interrupt enable bit TI1_IE in the Protocol Interrupt Enable Register 0 (FR_PIER0) is asserted, an interrupt request is generated.

26.6.17.2.2 Relative Timer T2

If the timer T2 is configured as a relative timer, the interrupt flag TI2_IF in the Protocol Interrupt Flag Register 0 (FR_PIFR0) is set, when the programmed amount of macroticks MT[31:0], defined by Timer 2 Configuration Register 0 (FR_TI2CR0) and Timer 2 Configuration Register 1 (FR_TI2CR1), has expired since the trigger or restart of timer 2. The relative timer is implemented as a down counter and expires when it has reached 0. At the macrotick start event, the value of MT[31:0] is checked and then decremented. Thus, if the timer is started with MT[31:0] == 0, it expires at the next macrotick start.

26.6.18 Slot Status Monitoring

The CC provides several means for slot status monitoring. All slot status monitors use the same slot status vector provided by the PE. The PE provides a slot status vector for each static slot, for each dynamic slot, for the symbol window, and for the NIT, on a per channel base. The content of the slot status vector is described in Table 26-124. The PE provides the slot status vector within the first macrotick after the end of the related slot/window/NIT, as shown in Figure 26-155.

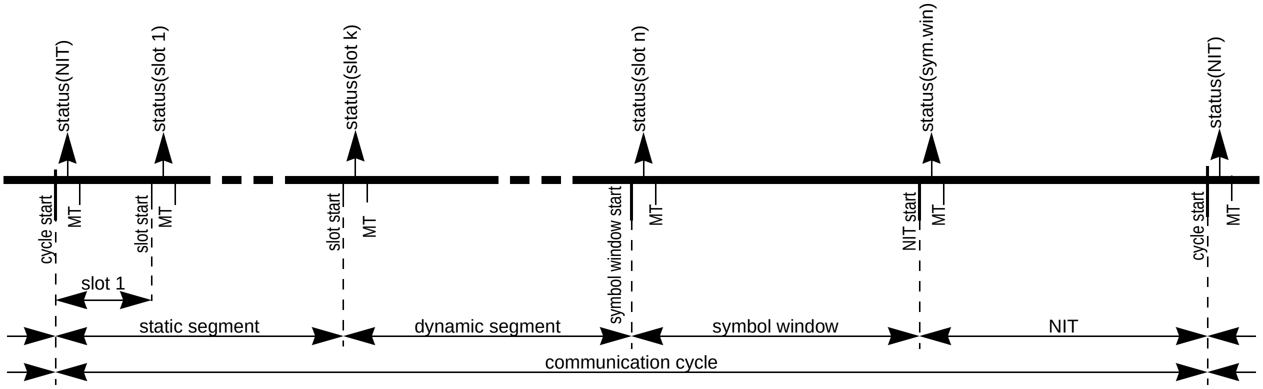


Figure 26-155. Slot Status Vector Update

NOTE

The slot status for the NIT of cycle n is provided after the start of cycle n+1.

Table 26-124. Slot Status Content

	Status Content

static / dynamic Slot	slot related status
	vSS!ValidFrame - valid frame received
	vSS!SyntaxError - syntax error occurred while receiving
	vSS!ContentError - content error occurred while receiving
	vSS!BViolation - boundary violation while receiving
	for slots in which the module transmits:
	vSS!TxConflict - reception ongoing while transmission starts
	for slots in which the module does not transmit:
	vSS!TxConflict - reception ongoing while transmission starts
	first valid - channel that has received the first valid frame
	received frame related status
	extracted from
	a) header of valid frame, if *vSS!ValidFrame* = 1
	b) last received header, if *vSS!ValidFrame* = 0
	c) set to 0, if nothing was received
	vRF!Header!NFIndicator - Null Frame Indicator (0 for null frame)
	vRF!Header!SuFIndicator - Startup Frame Indicator
	vRF!Header!SyFIndicator - Sync Frame Indicator

Symbol Window	window related status *vSS!ValidFrame* - always 0 *vSS!ContentError* - content error occurred while receiving *vSS!SyntaxError* - syntax error occurred while receiving *vSS!BViolation* - boundary violation while receiving *vSS!TxConflict* - reception ongoing while transmission starts received symbol related status *vSS!ValidMTS* - valid Media Test Access Symbol received received frame related status see static/dynamic slot
NIT	NIT related status *vSS!ValidFrame* - always 0 *vSS!ContentError* - content error occurred while receiving *vSS!SyntaxError* - syntax error occurred while receiving *vSS!BViolation* - boundary violation while receiving *vSS!TxConflict* - always 0 received frame related status see static/dynamic slot

26.6.18.1 Channel Status Error Counter Registers

The two channel status error counter registers, Channel A Status Error Counter Register (FR_CASERCR) and Channel B Status Error Counter Register (FR_CBSERCR), incremented by one, if at least one of four slot status error bits, *vSS!SyntaxError*, *vSS!ContentError*, *vSS!BViolation*, or *vSS!TxConflict* is set to 1. The status vectors for all slots in the static and dynamic segment, in the symbol window, and in the NIT are taken into account. The counters wrap round after they have reached the maximum value.

26.6.18.2 Protocol Status Registers

The Protocol Status Register 2 (FR_PSR2) provides slot status information about the Network Idle Time NIT and the Symbol Window. The Protocol Status Register 3 (FR_PSR3) provides aggregated slot status information.

26.6.18.3 Slot Status Registers

The eight slot status registers, Slot Status Registers (FR_SSR0–FR_SSR7), can be used to observe the status of static slots, dynamic slots, the symbol window, or the NIT without individual message buffers. These registers provide all slot status related and received frame / symbol related status information, as given in Table 26-124, except of the *first valid* indicator for non-transmission slots.

26.6.18.4 Slot Status Counter Registers

The CC provides four slot status error counter registers, Slot Status Counter Registers (FR_SSCR0–FR_SSCR3). Each of these slot status counter registers is updated with the value of an internal slot status counter at the start of a communication cycle. The internal slot status counter is incremented if its increment condition, defined by the Slot Status Counter Condition Register (FR_SSCCR), matches the status vector provided by the PE. All static slots, the symbol window, and the NIT status are taken into account. *Dynamic* slots are *excluded*. The internal slot status counting and update timing is shown in Figure 26-156.

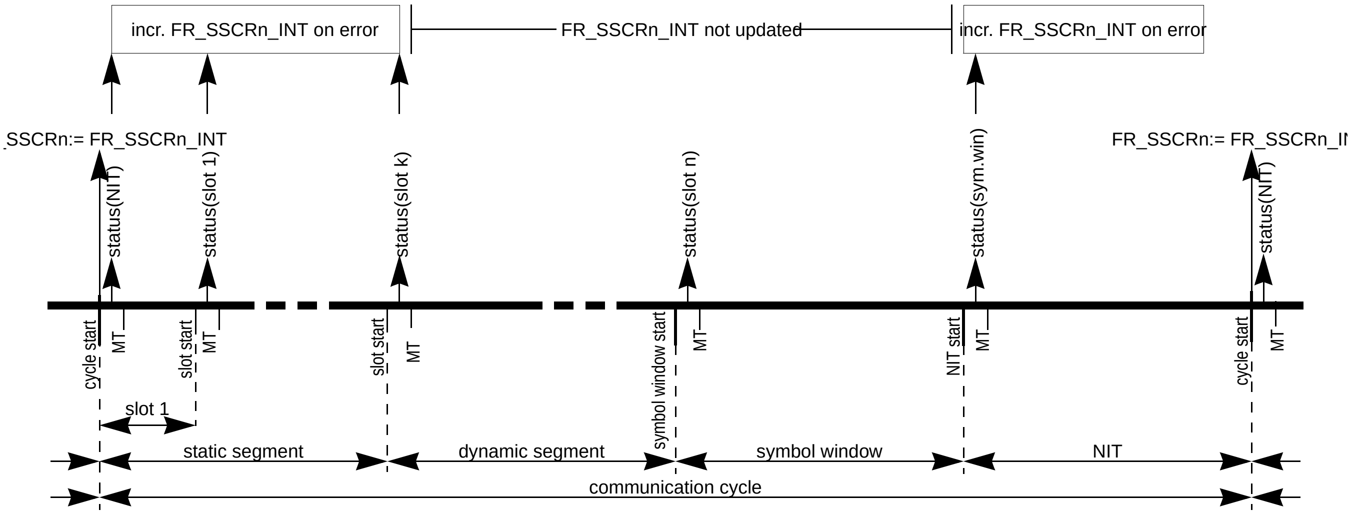


Figure 26-156. Slot Status Counting and FR_SSCRn Update

The PE provides the status of the NIT in the first slot of the next cycle. Due to these facts, the FR_SSCRn register reflects, in cycle n, the status of the NIT of cycle n-2, and the status of all static slots and the symbol window of cycle n-1.

The increment condition for each slot status counter consists of two parts, the frame related condition part and the slot related condition part. The internal slot status counter FR_SSCRn_INT is incremented if at least one of the conditions is fulfilled:

1. frame related condition:

- (FR_SSCCRn[VFR] | FR_SSCCRn[SYF] | FR_SSCCRn[NUF] | FR_SSCCRn[SUF]) // count on frame condition
  = 1;

and

- ((~FR_SSCCRn[VFR] | *vSS!ValidFrame*) & // valid frame restriction
  (~FR_SSCCRn[SYF] | *vRF!Header!SyFIndicator*) & // sync frame indicator restriction
  (~FR_SSCCRn[NUF] | ~*vRF!Header!NFIndicator*) & // null frame indicator restriction
  (~FR_SSCCRn[SUF] | *vRF!Header!SuFIndicator*)) // startup frame indicator restriction
  = 1;

NOTE

> The indicator bits SYF, NUF, and SUF are valid only when a valid frame was received. Thus it is required to set the VFR always, whenever count on frame condition is used.

2. slot related condition:

- ((FR_SSCCRn[STATUSMASK[3]] & *vSS!ContentError*) | // increment on content error
  (FR_SSCCRn[STATUSMASK[2]] & *vSS!SyntaxError*) | // increment on syntax error
  (FR_SSCCRn[STATUSMASK[1]] & *vSS!BViolation*) | // increment on boundary violation
  (FR_SSCCRn[STATUSMASK[0]] & *vSS!TxConflict*)) // increment on transmission conflict
  = 1;

If the slot status counter is in single cycle mode (FR_SSCCRn[MCY] = 0), the internal slot status counter FR_SSCRn_INT is reset at each cycle start. If the slot status counter is in the multicycle mode (FR_SSCCRn[MCY] = 1), the counter is not reset and incremented, until the maximum value is reached.

26.6.18.5 Message Buffer Slot Status Field

Each individual message buffer and each FIFO message buffer provides a slot status field, which provides the information shown in Table 26-124 for the static/dynamic slot. The update conditions for the slot status field depend on the message buffer type. Refer to the Message Buffer Update Sections in Section 26.6.6, “Individual Message Buffer Functional Description”.

26.6.19 System Bus Access

This section provides a description of the system bus accesses failures and the related CC behavior. System bus access failures may occur when the CC transfers data to or from the flexray memory area.

The system bus access failure types are described in Section 26.6.19.1, “System Bus Access Failure Types”.

The behavior of the CC after the occurrence of a system bus access failure is described in Section 26.6.19.2, "System Bus Access Failure Response".

26.6.19.1 System Bus Access Failure Types

This section describes the two types of system bus access failures.

26.6.19.1.1 System Bus Illegal Address Access

A system bus illegal address access is detected when the CC has used an illegal or invalid address to access the flexray system memory area. There are three conditions which are treated as a system bus illegal address access:

- The system bus subsystem detects an CC access to an illegal system memory address.
- The CC detects the usage of an data field offset with the value of 0.
- The CC detects a memory error while reading a data field offset from the CHI LRAM memory (see Section 26.6.24.3.1, "CHI LRAM Error Response after CC Read).

If a system bus illegal address access is detected, the CC sets the ILSA_EF flag in the CHI Error Flag Register (FR_CHIERFR).

26.6.19.1.2 System Bus Access Timeout

A system bus access timeout is detected if an access to the flexray memory area is not finished in time. The timeout value is derived from the SYMATOR[TIMEOUT] setting (see Section 26.7.1.1, "Configure System Memory Access Time-Out Register (FR_SYMATOR)"

If a system bus access timeout is detected, the CC sets the SBCF_EF flag in the CHI Error Flag Register (FR_CHIERFR).

26.6.19.2 System Bus Access Failure Response

This section describes the two types of behavior of the CC after the occurrence of a system bus access failure. The actual behavior is defined by the SBFF bit in the Module Configuration Register (FR_MCR).

26.6.19.2.1 Continue after System Bus Access Failure

If the SBFF bit in the Module Configuration Register (FR_MCR) is 0, the CC will continue its operation after the occurrence of the system bus access failure, but will not generate any system bus accesses until the start of the next communication cycle.Since no data are read from or written to the flexray memory area, no messages are received or transmitted. Consequently, none of the individual message buffers or receive FIFOs will be updated until the next communication cycle starts.

If a frame is under transmission when the system bus failure occurs, a correct frame is generated with the remaining header and frame data are replaced by all zeros. Depending on the point in time this can affect the PPI bit, the Header CRC, the Payload Length in case of an dynamic slot, and the payload data. Starting from the next slot in the current cycle, no frames will be transmitted and received, except for the key slot, where a sync or startup null-frame is transmitted, if the key slot is assigned.

If a frame is received when the system bus failure occurs, the reception is aborted and the related receive message buffer is not updated.

Normal operation is resumed after the start of next communication cycle.

26.6.19.2.2 Freeze after System Bus Access Failure

If the SBFF bit in the Module Configuration Register (FR_MCR) is set to 1, the CC will go into the freeze mode immediately after the occurrence of one of the system bus access failures.

26.6.20 Interrupt Support

The CC provides 172 individual interrupt sources and five combined interrupt sources.

26.6.20.1 Individual Interrupt Sources

26.6.20.1.1 Message Buffer Interrupts

The CC provides 128 message buffer interrupt sources.

Each individual message buffer provides an interrupt flag FR_MBCCSRn[MBIF] and an interrupt enable bit FR_MBCCSRn[MBIE]. The CC sets the interrupt flag when the slot status of the message buffer was updated. If the interrupt enable bit is asserted, an interrupt request is generated.

26.6.20.1.2 FIFO Interrupts

The CC provides 2 FIFO interrupt sources.

Each of the 2 FIFO provides a Receive FIFO Almost Full Interrupt Flag. The CC sets the Receive FIFO Almost Full Interrupt Flags (FR_GIFER[FAFBIF], FR_GIFER[FAFAIF]) in the Global Interrupt Flag and Enable Register (FR_GIFER) if the corresponding Receive FIFO fill level exceeds the defined watermark.

26.6.20.1.3 Wakeup Interrupt

The CC provides one interrupt source related to the wakeup.

The CC sets the Wakeup Interrupt Flag FR_GIFER[WUPIF] when it has received a wakeup symbol on the FlexRay bus. The CC generates an interrupt request if the interrupt enable bit FR_GIFER[WUPIE] is asserted.

26.6.20.1.4 Protocol Interrupts

The CC provides 25 interrupt sources for protocol related events. For details, see Protocol Interrupt Flag Register 0 (FR_PIFR0) and Protocol Interrupt Flag Register 1 (FR_PIFR1). Each interrupt source has its own interrupt enable bit.

26.6.20.1.5 CHI Interrupts

The CC provides 16 interrupt sources for CHI related error events. For details, see CHI Error Flag Register (FR_CHIERFR). There is one common interrupt enable bit FR_GIFER[CHIE] for all CHI error interrupt sources.

26.6.20.2 Combined Interrupt Sources

Each combined interrupt source generates an interrupt request only when at least one of the interrupt sources that is combined generates an interrupt request.

26.6.20.2.1 Receive Message Buffer Interrupt

The Receive Message Buffer Interrupt request is generated when at least one of the individual receive message buffers generates an interrupt request MBXIRQ[n] and the interrupt enable bit FR_GIFER[RBIE] is set.

26.6.20.2.2 Transmit Message Buffer Interrupt

The Transmit Message Buffer Interrupt request is generated when at least one of the individual transmit message buffers generates an interrupt request MBXIRQ[n] and the interrupt enable bit FR_GIFER[TBIE] is asserted.

26.6.20.2.3 Protocol Interrupt

The Protocol Interrupt request is generated when at least one of the individual protocol interrupt sources generates an interrupt request and the interrupt enable bit FR_GIFER[PRIE] is set.

26.6.20.2.4 CHI Interrupt

The CHI Interrupt request is generated when at least one of the individual chi error interrupt sources generates an interrupt request and the interrupt enable bit FR_GIFER[CHIE] is set.

26.6.20.2.5 Module Interrupt

The Module Interrupt request is generated if at least one of the combined interrupt sources generates an interrupt request and the interrupt enable bit FR_GIFER[MIE] is set.

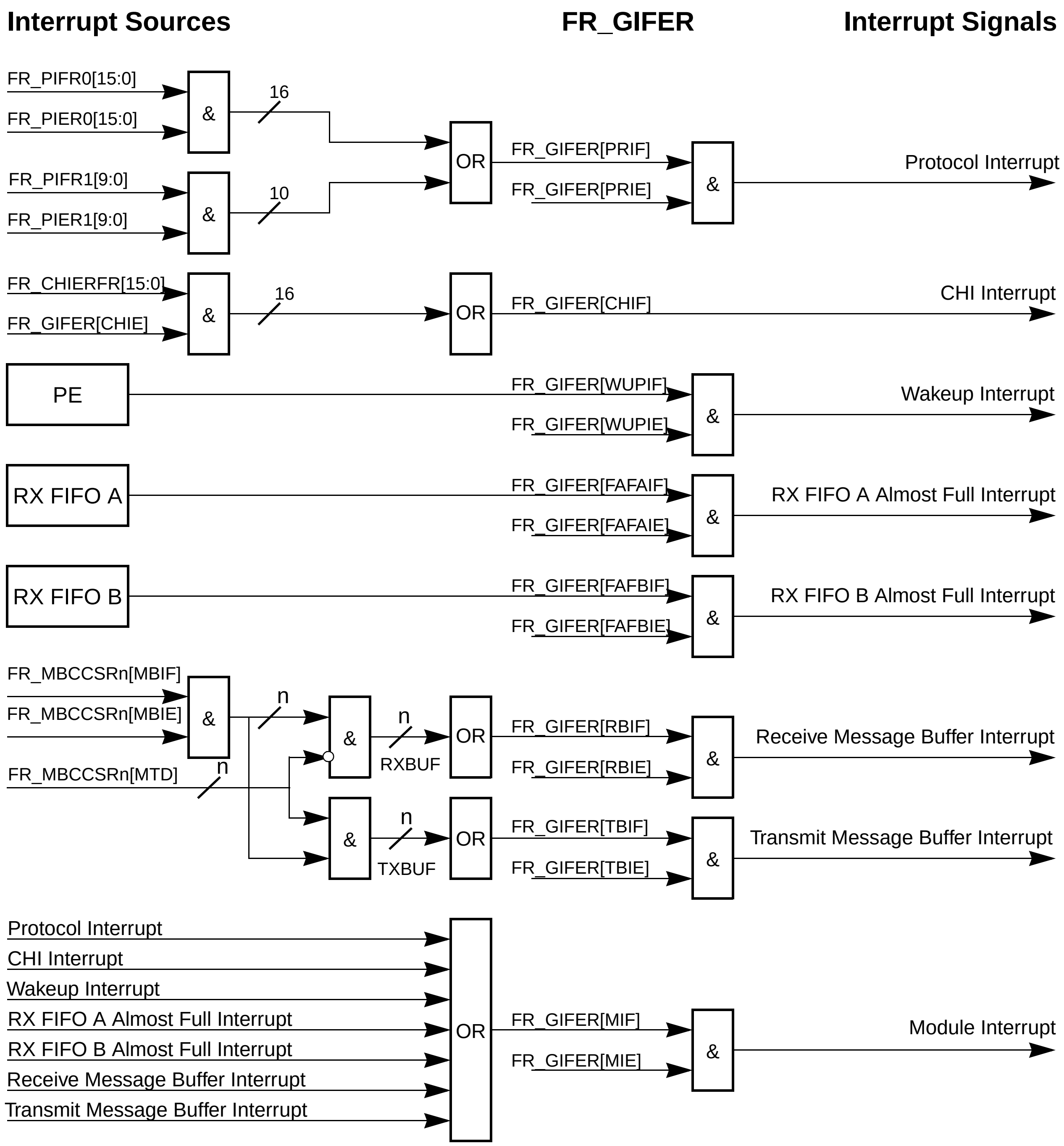


Figure 26-157. Scheme of FR_GIFER interrupt signal generation

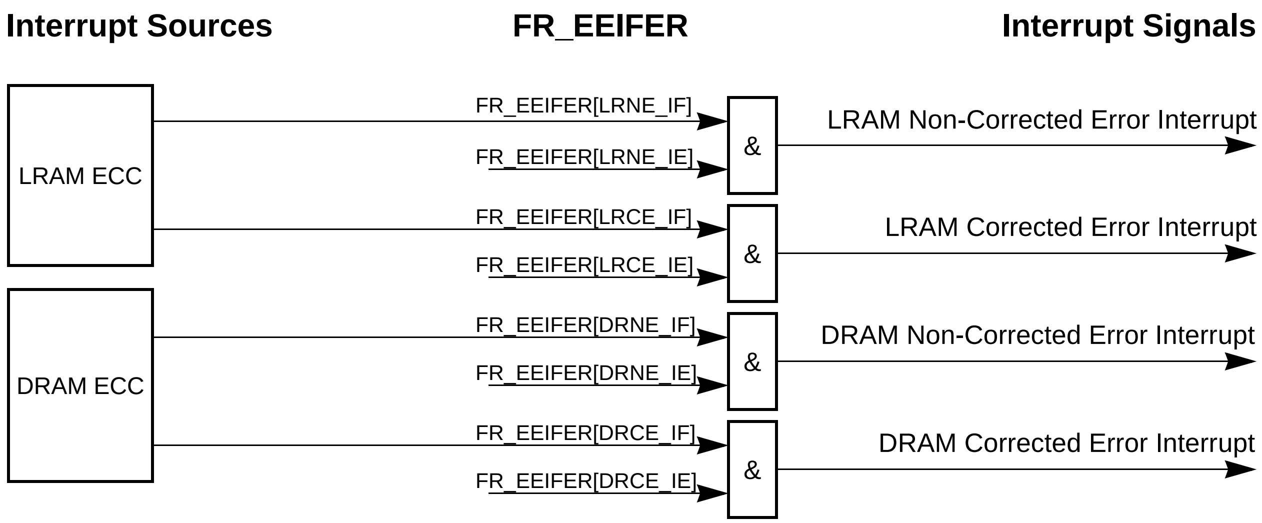


Figure 26-158. Scheme of FR_EEIFER interrupt signal generation

Figure 26-159.

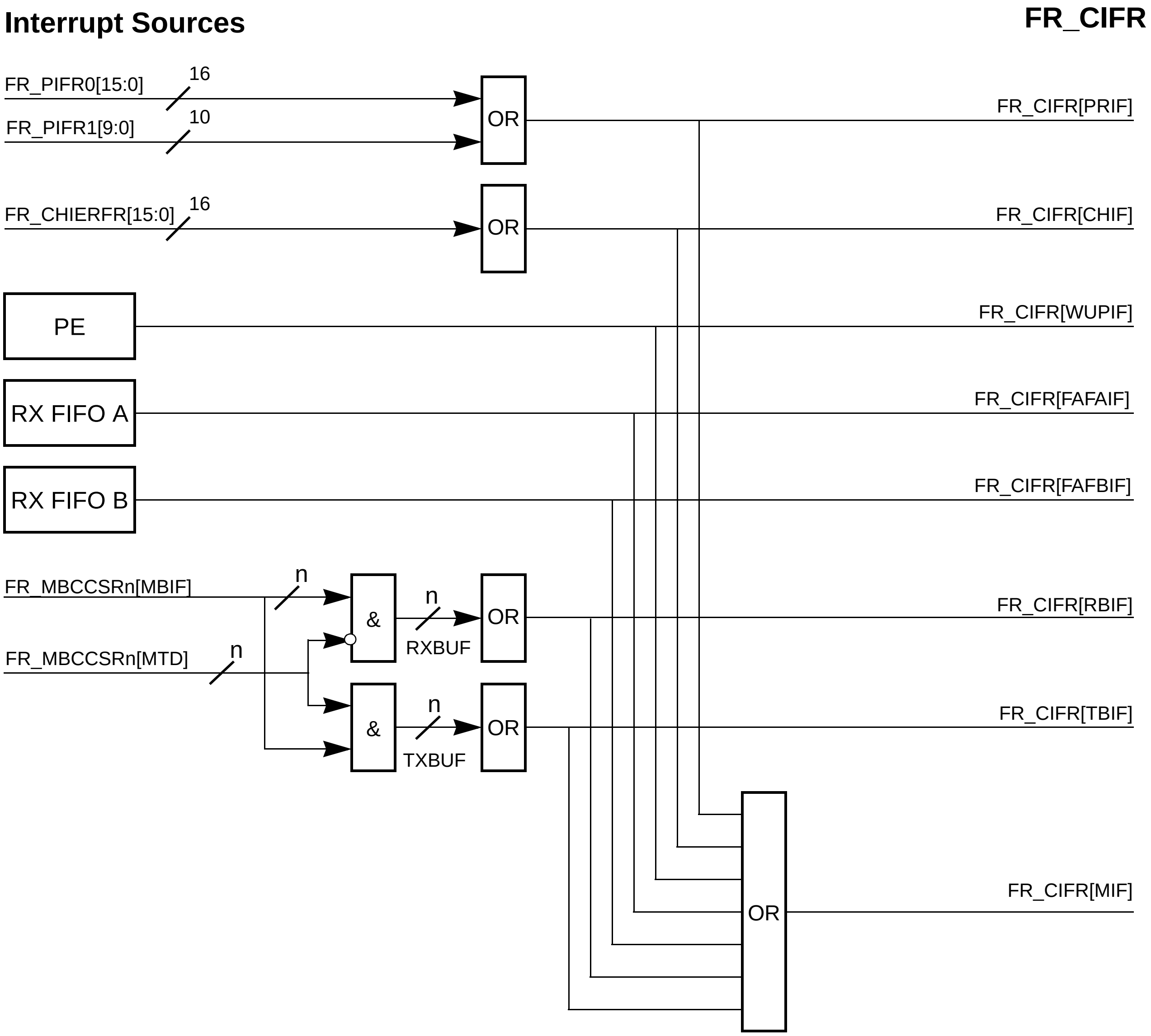


Figure 26-160. Scheme of FR_CIFR flags generation

26.6.21 Lower Bit Rate Support

The CC supports a number of lower bit rates on the FlexRay bus channels. The lower bit rates are implemented by modifying the duration of the microtick *pdMicrotick*, the number of samples per microtick *pSamplesPerMicrotick*, the number of samples per bit *cSamplesPerBit*, and the strobe offset *cStrobeOffset.* The application configures the FlexRay channel bit rate by setting the BITRATE field in the Module Configuration Register (FR_MCR). The protocol values are set internally. The available bit rates, the related BITRATE field configuration settings and related protocol parameter values are shown in Table 26-125.

Table 26-125. FlexRay Channel Bit Rate Control

FlexRay Channel Bit Rate [Mbit/s]	FR_MCR[BITRATE]	*pdMicrotick* [ns]	*gdSampleClockPeriod* [ns]	*pSamplesPerMicrotick*	*cSamplesPerBit*	*cStrobeOffset*
10.0	000	25.0	12.5	2	8	5
8.0	011	25.0	12.5	2	10	6
5.0	001	25.0	25.0	1	8	5
2.5	010	50.0	50.0	1	8	5

NOTE

The bit rate of 8 Mbit/s is not defined by the *FlexRay Communications System Protocol Specification, Version 2.1 Rev A.*

26.6.22 PE Data Memory (PE DRAM)

The PE Data Memory (PE DRAM) is 128 word, 16-bit wide memory with byte access, which contains the program data of the PE internal CPU. The PE DRAM is divided into two banks, 8-bit each. The memory data [7:0] are assigned to BANK0, the memory data [15:8] are assigned to BANK1.

Table 26-126. PE DRAM Layout

ADDR	BANK1	BANK0
0x00	byte1	byte0
0x01	byte3	byte2
...		
0x7F	byte255	byte254

The FlexRay module provides means to access the PE DRAM from the application. The PE DRAM application access is initiated and controlled via PE DRAM Access Register (FR_PEDRAR) and PE DRAM Data Register (FR_PEDRDR). This functionality is used to check the memory error detection.

26.6.22.1 PE DRAM Read Access

A read access from the PE DRAM can be initiated in any protocol state. The following sequence describes a read access from the PE DRAM address 0x70.

1. FR_PEDRAR:= 0x50E0;
   // INST=0x5; ADDR=070
2. wait until FR_PEDRAR[DAD] == 1;
   // wait for end of PE DRAM access

3. val = FR_PEDRDR[DATA];
   // read PE DRAM data

The read access is handled by the PE internal CPU with the lowest execution priority. This may cause an response delay with a maximum of 1000 PE clock cycle (25us).

26.6.22.2 PE DRAM Write Access

A write access into the PE DRAM can be initiated in any protocol state. The following sequence describes a write access to the PE DRAM address 0x70.

1. FR_PEDRDR:= DATA;
   // write value to be written into data register
2. FR_PEDRAR:= 0x30E0;
   // INST=0x3; ADDR=0x70
3. wait until FR_PEDRAR[DAD] == 1;
   // wait for end of PE DRAM access
4. val = FR_PEDRDR[DATA];
   // read back PE DRAM data

The write access is handled by the PE internal CPU with the lowest execution priority. This may causes an response delay with a maximum of 1000 PE clock cycle (25us).

If the conditions given in Section 26.6.22.3, “PE DRAM Write Access Limitations” are fulfilled, the data provided in PE DRAM Data Register (FR_PEDRDR) are written into the PE DRAM, read back in the next clock cycle and stored into the PE DRAM Data Register (FR_PEDRDR). Otherwise, data are not written into the PE DRAM and 0x0000 is stored into the PE DRAM Data Register (FR_PEDRDR).

26.6.22.3 PE DRAM Write Access Limitations

The PE DRAM is used by the protocol engine if the module is not in *POC:default config* state. The only address not used by the protocol engine is 0x70. To prevent the corruption of protocol engine data the following PE DRAM write access limitations apply for application writes.

1. When the module is in *POC:default config* state, all PE DRAM addresses are writable.
2. When the module is not in *POC:default config* state, only PE DRAM address 0x70 is writable.

26.6.23 CHI Lookup-Table Memory (CHI LRAM)

The CHI Lookup-Table Memory (CHI LRAM) is an CHI internal memory which contains the message buffer configuration data and the data field offsets for the physical message buffers. The configuration data for two message buffers or 6 data field offsets are contained in one memory row. The CHI LRAM is divided into 6 memory BANKs.

Table 26-127. CHI LRAM Layout

ADR	BANK5	BANK4	BANK3	BANK2	BANK1	BANK0
0x00	FR_MBIDXR1	FR_MBFIDR1	FR_MBCCFR1	FR_MBIDXR0	FR_MBFIDR0	FR_MBCCFR0
0x01	FR_MBIDXR3	FR_MBFIDR3	FR_MBCCFR3	FR_MBIDXR2	FR_MBFIDR2	FR_MBCCFR2
...						
0x3F	FR_MBIDXR127	FR_MBFIDR127	FR_MBCCFR127	FR_MBIDXR126	FR_MBFIDR126	FR_MBCCFR126
0x40	FR_MBDOR5	FR_MBDOR4	FR_MBDOR3	FR_MBDOR2	FR_MBDOR1	FR_MBDOR0
...						
0x55	FR_MBDOR131	FR_MBDOR130	FR_MBDOR129	FR_MBDOR128	FR_MBDOR127	FR_MBDOR126
0x56	FR_LEETR5	FR_LEETR4	FR_LEETR3	FR_LEETR2	FR_LEETR1	FR_LEETR0

26.6.23.1 CHI LRAM Read and Write Access

The CHI LRAM is accessed by the application via regular register read and write accesses.

26.6.24 Memory Content Error Detection

The FlexRay module provides integrated memory content error detection for both the CHI LRAM and PE DRAM, and memory content error correction for the PE DRAM. The memory error detection for the CHI LRAM uses an standard Hamming code with a Hamming distance of 3 and detects all single-bit and double-bit errors (SEDDED). The memory error detection and correction for the PE DRAM uses an enhanced Hamming code with a Hamming distance of 4 and detects and corrects all single-bit errors and detects all double-bit errors (SECDED).

This section describes the reporting of the occurrence of memory content errors, the reaction of the module on the occurrence, and how the application can inject memory errors in order to trigger the report and response behavior.

26.6.24.1 Memory Error Types

A memory error is the distortion of one or more bits read out of the memory. The reading of the values of all zeros and all ones is considered as an special case. The FlexRay module detects and indicates the memory errors as shown in Table 26-128. The entries on the top have higher priority.

Each memory read access reads out *all* banks of the addressed row, and runs error detection on *all* banks, even in the case that the application has triggered a read from only one bank. This may lead to the reporting of an memory error if at least one bank contains a memory error, even if an error free bank has been read.

Table 26-128. Detected Memory Error Types

Memory	Priority	Memory Data	Indication
CHI LRAM	0 (highest)	All Zero’s	No Error - Valid Data
PE DRAM			Non-Corrected Error
CHI LRAM		All One’s	Non-Corrected Error
PE DRAM			
CHI LRAM	1 (lowest)	One Bit Flipped	Non-Corrected Error
PE DRAM			Corrected Error
CHI LRAM		Two Bits Flipped	Non-Corrected Error
PE DRAM			
CHI LRAM		Three or more Bits Flipped	one out of {No error, Non-Corrected Error}, defined by coding given in Section 26.6.24.2.3, “CHI LRAM Checkbits” and Section 26.6.24.2.3, “CHI LRAM Checkbits”
PE DRAM			one out of {No error, Corrected Error, Non-Corrected Error}, defined by coding given in Section 26.6.24.2.1, “PE DRAM Checkbits” and Section 26.6.24.2.2, “PE DRAM Syndrome”

26.6.24.2 Memory Error Reporting

The memory error reporting is enabled only if the ECC functionality enable bit ECCE in the Module Configuration Register (FR_MCR) is set.

For each of the two memories exists two sets of internal registers to store the detection of one corrected and one non-corrected memory error.

If a memory error is detected, the module checks whether the related error interrupt flag in the ECC Error Interrupt Flag and Enable Register (FR_EEIFER) is set.

- If the error interrupt flag is set, the related internal error reporting register is not updated and the related error overflow flag is set to 1 to indicate a loss of error condition.
- If the error interrupt flag is not set, the internal reporting register is updated and the error interrupt flag is set to 1. If two or more memory errors of the same type are detected, the error for the bank with the lower bank number will be reported, and the error overflow flag will be set to 1.

If a memory error is detected for at least two banks of one memory, the related error overflow flag is set to 1 to indicate a loss of error condition.

26.6.24.2.1 PE DRAM Checkbits

The coding of the checkbits reported in ECC Error Report Code Register (FR_EERCR) for PE DRAM memory errors is shown in Table 26-130. This table shows the implemented enhanced Hamming code. If the error injection was applied to distort the checkbits, then the distorted checkbits are reported.

Table 26-129. PE DRAM checkbits coding

CODE	CODE				DATA							
	3	2	1	0	7	6	5	4	3	2	1	0
4[1]	X	X	X	X	X	X	X	X	X	X	X	X
3[2]	-	-	-	-	X	X	X	X	-	-	-	-
2	-	-	-	-	X	-	-	-	X	X	X	-
1	-	-	-	-	-	X	X	-	X	X	-	X
0	-	-	-	-	-	X	-	X	X	-	X	X

[1] The checkbit CODE[4] is set to 1 if and only if there is a even number of 1’s in columns with X.

[2] The checkbits CODE[3]... CODE[0] are set to 1 if and only if there is a odd number of 1’s in all columns with X.

This coding of the checkbit ensures that neither 0x000 nor 0xFFF are valid code words and written into the memory.

26.6.24.2.2 PE DRAM Syndrome

The coding of the syndrome reported in ECC Error Report Code Register (FR_EERCR) for PE DRAM memory errors is shown in Table 26-130.

Table 26-130. FR_EERCR[CODE] PE DRAM Syndrome Coding

FR_EERCR[CODE]		Description
[4]	[3:0]	
0x1	0x0	No Error (Never appears in error report registers)
0x0	0x0	If data == 0: Non Corrected Error (Dedicated Handling of All Zero Code Word) If data!= 0: Corrected Error (Parity Bit 4)
0x0	0x1	Corrected Error (Parity Bit 0)
0x0	0x2	Corrected Error (Parity Bit 1)
0x0	0x3	Corrected Error (Data Bit 0)
0x0	0x4	Corrected Error (Parity Bit 2)
0x0	0x5	Corrected Error (Data Bit 1)
0x0	0x6	Corrected Error (Data Bit 2)
0x0	0x7	Corrected Error (Data Bit 3)
0x0	0x8	Corrected Error (Parity Bit 3)
0x0	0x9	Corrected Error (Data Bit 4)
0x0	0xA	Corrected Error (Data Bit 5)
0x0	0xB	Corrected Error (Data Bit 6)
0x0	0xC	Corrected Error (Data Bit 7)
0x0	0xD-0xF	Non-Corrected Error
0x1	0x1-0xF	Non-Corrected Error

26.6.24.2.3 CHI LRAM Checkbits

The coding of the checkbits reported in ECC Error Report Code Register (FR_EERCR) for CHI LRAM memory errors is shown in Table 26-131. This table shows the implemented Hamming code. If the error injection was applied to distort the checkbits, then the distorted checkbits are reported.

Table 26-131. CHI LRAM checkbits coding

CODE[1]	DATA															
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
4	X	X	X	X	X	-	-	-	-	-	-	-	-	-	-	-
3	-	-	-	-	-	X	X	X	X	X	X	X	-	-	-	-
2	X	X	-	-	-	X	X	X	X	-	-	-	X	X	X	-
1	-	-	X	X	-	X	X	-	-	X	X	-	X	X	-	X
0	X	-	X	-	X	X	-	X	-	X	-	X	X	-	X	X

[1] The checkbit CODE[n] is set to 1 if and only if there is a odd number of 1's in all columns with X.

26.6.24.2.4 CHI LRAM Syndrome

The coding of the syndrome reported in ECC Error Report Code Register (FR_EERCR) for CHI LRAM memory errors is shown in Table 26-132.

Table 26-132. FR_EERCR[CODE] CHI LRAM Syndrome Coding

FR_EERCR[CODE]	Description
0x00	No Error (Never appears in error report registers)
0x01-0x1F	Non Corrected Error

26.6.24.3 Memory Error Response

The memory error response is enabled only when the ECC functionality enable bit ECCE in the Module Configuration Register (FR_MCR) is set.

In case of the detection of a *corrected* memory error, the FlexRay module continues its normal operation using the corrected data word. This section describes the behavior of the FlexRay module after the detection of a *non-corrected* memory error.

26.6.24.3.1 CHI LRAM Error Response after CC Read

When the CC is out of the *POC:default config* state, it reads the configuration data and the data field offsets of all utilized message buffers in every slot and in the NIT. If a non-corrected memory error is detected during this module read access the error response of the module depends from LRAM location where the error occurred.

- If the LRAM address belongs to physical message buffer configuration data the FlexRay module will consider the affected message buffer as disabled for the current search and will exclude this buffer from the search. The configuration of the affected message buffer is not changed.

If the affected message buffer is a tx message buffer, no frame will be transmitted from this message buffer in the next slot. If the affected message buffer is a rx message buffer, no frame will be received to this message buffer in the next slot.

- If the LRAM address belongs to the data field offset area and the related physical message buffer is used for Rx or Tx the first access to the system memory caused by payload read or write yields to the assertion of the FR_CHIERFR[ILSA_EF]. No memory access occurs w.r.t. payload access is performed for the complete frame.

26.6.24.3.2 CHI LRAM Error Response after Application Read

The application can read the content of the CHI LRAM via reading the FR_MBCCFRn, FR_MBFIDRn, FR_MBIDXRn, FR_MBDORn, and FR_LEETRn registers. If a non-corrected memory error is detected during this kind of read access, the module indicates the detected memory error, delivers the non-corrected data read and continues its normal operation.

26.6.24.3.3 PE DRAM Error Response after CC Read

If the CC detects an non-corrected memory error during internal read of program data which is contained in PE DRAM, this is considered as an fatal protocol error and the module enters the protocol freeze state immediately.

26.6.24.3.4 PE DRAM Error Response after Application Read in *POC:default config* state

If the CC detects an non-corrected memory error during an application triggered read from any PE DRAM address and the protocol is in the *POC:default config* state, this is considered as an fatal protocol error and the module enters the protocol freeze state. This behavior allows for checking the freeze functionality in case of the detection of non-corrected errors.

26.6.24.3.5 PE DRAM Error Response after Application Read out of *POC:default config*

If the CC detects an non-corrected memory error during an application triggered read from any PE DRAM address, and the protocol is not in the *POC:default config* state, this error is not considered as an fatal error and the protocol state is not changed. This prevents any interference of the running protocol by PE DRAM error injection reads.

26.6.25 Memory Error Injection

The error injection functionality is used by the application to inject data errors into the memories to trigger and check the memory error detection functionality.

The error injection is enabled only if the ECC functionality enable bit ECCE in the Module Configuration Register (FR_MCR) and the error injection enable control bit EIE in the ECC Error Report and Injection Control Register (FR_EERICR) are set.

The error injection mode is configured by the EIM configuration bit in the ECC Error Report and Injection Control Register (FR_EERICR).When the error injection is enabled, each write access to the configured memory location will be distorted.

The injector has the same behavior for FlexRay module memory writes and application memory writes.

26.6.25.1 CHI LRAM Error Injection

The following sequence describes an memory error injection sequence for the CHI LRAM memory. This sequence consists of the setup of the error injector followed by an application triggered write access to provoke an distortion of the memory content. The content of the CHI LRAM is described in Table 26-127.

When the CC is in *POC:default config*, there are no limitations for the error injection and no impacts of error injection to the application. For error injection out of *POC:default config* see Section 26.7.3, "Memory Error Injection out of POC:default config".

Injector Setup:

1. FR_MCR[ECCE]:= 1;
   // enable ecc functionality
2. FR_EERICE[EIE]:=I_MODE;
   // configure error injection mode
3. FR_EEIAR[MID]:= 1;
   // select CHI LRAM for error injection
4. FR_EEIAR[BANK]:= I_BANK;
   // select bank for error injection; I_BANK = {0,1,2,3,4,5}
5. FR_EEIAR[ADDR]:= I_ADDR;
   // select address for error injection; I_ADDR <= 0x56
6. FR_EEIDR[DATA]:= D_DIST;
   // define data distortion pattern
7. FR_EEICR[CODE]:= C_DIST;
   // define checkbit distortion pattern
8. FR_EERICE[EIE]:=1;
   // enable error injection

Application Write Access:

```
If (I_BANK==0) -> FR_MBCCFR(2n) / FR_MBDOR(6k) / FR_LEETR0 := DATA;
If (I_BANK==1) -> FR_MBFIDR(2n) / FR_MBDOR(6k+1) / FR_LEETR1 := DATA;
If (I_BANK==2) -> FR_MBIDXR(2n) / FR_MBDOR(6k+2) / FR_LEETR2 := DATA;
If (I_BANK==3) -> FR_MBCCFR(2n+1) / FR_MBDOR(6k+3) / FR_LEETR3 := DATA;
If (I_BANK==4) -> FR_MBFIDR(2n+1) / FR_MBDOR(6k+4) / FR_LEETR4 := DATA;
If (I_BANK==5) -> FR_MBIDXR(2n+1) / FR_MBDOR(6k+5) / FR_LEETR5 := DATA;
// write DATA to the defined injection bank and injection address (see Table 26-127).
```

26.6.25.2 PE DRAM Error Injection

The following sequence describes an memory error injection sequence for the PE DRAM memory. This sequence consists of the setup of the error injector followed by an application triggered write access to provoke an distortion of the memory content.

When the FlexRay module is in *POC:default config*, there are no limitations for the error injection and no impacts of error injection to the application. For error injection out of *POC:default config* see Section 26.7.3.2, “PE DRAM Error Injection out of POC:default config”.

Injector Setup:

1. FR_MCR[ECCE]:= 1;
   // enable ecc functionality
2. FR_EERICE[EIE]:=I_MODE;
   // configure error injection mode
3. FR_EEIAR[MID]:= 0;
   // select PE DRAM for error injection
4. FR_EEIAR[BANK]:= I_BANK;
   // define bank for error injection; I_BANK = {0,1}
5. FR_EEIAR[ADDR]:= I_ADDR;
   // define address for error injection; I_ADDR <= 0x7F
6. FR_EEIDR[DATA]:= D_DIST;
   // define data distortion pattern
7. FR_EEICR[CODE]:= C_DIST;
   // define checkbit distortion pattern
8. FR_EERICE[EIE]:=1;
   // enable error injection

Application Write Access (e.g. I_ADDR=0x70):

1. FR_PEDRAR:= 0x30E0;
   // INST=0x3; ADDR=0x70
2. wait until FR_PEDRAR[DAD] == 1;
   // wait for end of PE DRAM access
3. val = FR_PEDRDR[DATA]; |
   // get read back PE DRAM data

Note: The write access to the PE DRAM triggers an subsequent read access from PE DRAM in the next cycle, which triggers the detection of the distorted data.

26.7 Application Information

26.7.1 Module Configuration

This section describes essential parts of the module configuration.

26.7.1.1 Configure System Memory Access Time-Out Register (FR_SYMATOR)

To ensure reliable operation of the CC, the application must ensure that the TIMEOUT value in System Memory Access Time-Out Register (FR_SYMATOR) and the CHI clock frequency f_{CHI} in MHz fulfill Equation 30[1].

$$0 \le \text{SYMATOR[TIMEOUT]} \le \lfloor 0.45 \cdot f_{CHI} - 8 \rfloor \qquad \textit{Eqn. 30}$$

For a given SYMATOR[TIMEOUT] value, f_{CHI} can be increased without causing unreliable operation of the CC. The same holds for reducing the SYMATOR[TIMEOUT] value for a given f_{CHI}.

Some examples for maximum values of the SYMATOR[TIMEOUT] for a minimum CHI frequency are given in Table 26-133.

Table 26-133. Maximum SYMATOR[TIMEOUT] examples

f_{CHI}	SYMATOR[TIMEOUT]	f_{CHI}	SYMATOR[TIMEOUT]
>= 18 MHz	0	>= 100 MHz	<= 37
>= 23 MHz	<= 2	>= 120 MHz	<= 46
>= 27 MHz	<= 4	>= 140 MHz	<= 55
>= 32 MHz	<= 6	>= 160 MHz	<= 64
>= 60 MHz	<= 19	>= 180 MHz	<= 73
>= 80 MHz	<= 28	>= 200 MHz	<= 82

26.7.1.1.1 System Bus Wait State Constraints

The SYMATOR[TIMEOUT] value corresponds directly to a certain acceptable number of wait states on the system bus.

For single channel configurations and if the sync frame table generation functionality is *not* used (FR_SFTCCSR[SDVEN,SIDEN] = 0) no timeout will be detected if less than 2*SYMATOR[TIMEOUT]+1 wait states will be seen on the system bus for each system bus access.

For dual channel configurations, or if the sync frame table generation functionality is used, no timeout will be detected if less than SYMATOR[TIMEOUT]-1 wait states will be seen on the system bus for each system bus access.

26.7.1.2 Configure Data Field Offsets

The data field offsets are located in the Message Buffer Data Field Offset Registers (FR_MBDORn) and Receive FIFO Start Data Offset Register (FR_RFSDOR). The application has to configure the data field offset values for all message buffers which are used.

The reset value of all data field offsets FR_MBDORn[MBDO] and FR_RFSDOR[SDO] is 0. This value is considered to be illegal (see Section 26.6.19.1.1, "System Bus Illegal Address Access).

26.7.2 Initialization Sequence

This section describes the required steps to initialize the CC. The first subsection describes the steps required after a system reset, the second section describes the steps required after preceding shutdown of the CC.

---

1. see Section 26.3, "Controller Host Interface Clocking" for all constraints of minimum CHI clock frequency.

26.7.2.1 Module Initialization

This section describes the module related initialization steps after a system reset.

1. Configure CC.
    a) configure the control bits in the Module Configuration Register (FR_MCR)
    b) configure the system memory base address in System Memory Base Address Register (FR_SYMBADR)
2. Enable the CC.
    a) write 1 to the module enable bit MEN in the Module Configuration Register (FR_MCR)

The CC now enters the Normal Mode. The application can commence with the protocol initialization described in Section 26.7.2.2, "Protocol Initialization".

26.7.2.2 Protocol Initialization

This section describes the protocol related initialization steps.

1. Configure the Protocol Engine.
    a) issue CONFIG command via Protocol Operation Control Register (FR_POCR)
    b) wait for *POC:config* in Protocol Status Register 0 (FR_PSR0)
    c) configure the FR_PCR0,..., FR_PCR30 registers to set all protocol parameters
2. Configure the Message Buffers and FIFOs.
    a) set the number of message buffers used and the message buffer segmentation in the Message Buffer Segment Size and Utilization Register (FR_MBSSUTR)
    b) define the message buffer data size in the Message Buffer Data Size Register (FR_MBDSR)
    c) configure each message buffer by setting the configuration values in the Message Buffer Configuration, Control, Status Registers (FR_MBCCSRn), Message Buffer Cycle Counter Filter Registers (FR_MBCCFRn), Message Buffer Frame ID Registers (FR_MBFIDRn), Message Buffer Index Registers (FR_MBIDXRn)
    d) configure the FIFOs
    e) issue CONFIG_COMPLETE command via Protocol Operation Control Register (FR_POCR)
    f) wait for *POC:ready* in Protocol Status Register 0 (FR_PSR0)

After this sequence, the CC is configured as a FlexRay node and is ready to integrate into the FlexRay cluster.

26.7.2.3 CHI LRAM Initialization

The initialization of the CHI LRAM is performed by the CC when it leaves the Disabled Mode. The unitization runs for 87 CHI clock cycles. All fields in the FR_MBCCSRn, FR_MBCCFRn, FR_MBFIDRn, FR_MBDORn, and LEETRn registers are initialized to 0. All application read or write accesses to these registers are delayed until the initialization is finished.

26.7.2.4 PE DRAM Initialization

The PE DRAM initialization is performed by the CC in the *POC:default config* state. This initialization runs for 4.8 µs, and will delay the state transition from *POC:default config* into *POC:config*.

26.7.3 Memory Error Injection out of *POC:default config*

This section provides information for application driven memory error injection out if *POC:default config*. The CC provides means to inject memory errors from the application without any impacts to the internal protocol operation of the CC.

26.7.3.1 CHI LRAM Error Injection out of *POC:default config*

The CC will never perform any internal read access from the LRAM ECC Error Test Registers (FR_LEETRn). Any memory errors injected into these CHI LRAM locations will never be detected by internal access, independent from the protocol state.

The application should use these registers and related CHI LRAM location to inject memory errors into the CHI LRAM. The injection sequence is described in Section 26.6.25.1, “CHI LRAM Error Injection”.

26.7.3.2 PE DRAM Error Injection out of *POC:default config*

The CC will never perform any internal read access from the PE DRAM address 0x70. This is the only one PE DRAM address writable by the application out of the *POC:default config* state.

The application should use these PE DRAM location to inject memory errors into the PE DRAM. The injection sequence is described in Section 26.6.25.2, “PE DRAM Error Injection”.

26.7.4 Shut Down Sequence

This section describes a secure shut down sequence to stop the CC gracefully. The main targets of this sequence are

- finish all ongoing reception and transmission
- do not corrupt FlexRay bus and do not disturb ongoing FlexRay bus communication

For a graceful shutdown the application shall perform the following tasks:

1. Disable all enabled message buffers.
   a) repeatedly write 1 to FR_MBCCSRn[EDT] until FR_MBCCSRn[EDS] == 0.
2. Stop Protocol Engine.
   a) issue HALT command via Protocol Operation Control Register (FR_POCR)
   b) wait for *POC:halt* in Protocol Status Register 0 (FR_PSR0)

26.7.5 Number of Usable Message Buffers

This section describes the required minimum CHI clock frequency for a specified number of utilized message buffers configured in the Message Buffer Segment Size and Utilization Register

(FR_MBSSUTR), a configured minislot length *gdMinislot*, and a configured nominal macrotick length *gdMacrotick*[1].

Additional constraints for the minimum CHI clock frequency are given in Section 26.3, “Controller Host Interface Clocking”.

The CC uses a sequential search algorithm to determine the individual message buffer assigned or subscribed to the next slot. This search is started at the start of slot and must be finished before the start of the next slot.

The shortest FlexRay slot is an corrected empty dynamic slot. An corrected empty dynamic slot is a minislot and consists of *gdMinislot* corrected macroticks with a duration of *gdMacrotick*. The minimum duration of an corrected macrotick is $\mathit{gdMacrotick}_{\mathrm{min}}$ = 39 µµT. This results in a minimum length of an correct slot

$$\Delta_{\mathrm{slotmin}} = 39 \cdot \mathit{pdMicrotick} \cdot \mathit{gdMinislot} \qquad \textbf{\textit{Eqn. 31}}$$

The message buffer search engine runs on the CHI clock and evaluates one individual message buffer per CHI clock cycle. For internal status update operations and to account for clock domain crossing jitter, an additional amount of 27 CHI clock cycles is required to ensure correct search engine operation.

For a given number of utilized message buffers FR_MBSSUTR[LAST_MB_UTIL] + 1 and for a given CHI clock frequency f_{chi}, this results in a search duration of

$$\Delta_{\mathrm{search}} = \frac{1}{f_{\mathrm{chi}}} \cdot (\mathrm{FR\_MBSSUTR[LAST\_MB\_UTIL]+27}) \qquad \textbf{\textit{Eqn. 32}}$$

The message buffer search must be finished within one slot which requires that Equation 33 must be fulfilled:

$$\Delta_{\mathrm{search}} \le \Delta_{\mathrm{slotmin}} \qquad \textbf{\textit{Eqn. 33}}$$

This results in the formula given in Equation 34 which determines the required minimum CHI frequency for a given number of message buffers that are utilized.

$$f_{\mathrm{chi}} \ge \frac{(\mathrm{FR\_MBSSUTR[LAST\_MB\_UTIL]+27})}{39 \cdot \mathit{pdMicrotick} \cdot \mathit{gdMinislot}} \qquad \textbf{\textit{Eqn. 34}}$$

The required minimum CHI Clock frequency for a selected set of relevant protocol parameters and for the LAST_MB_UTIL field in the Message Buffer Segment Size and Utilization Register (FR_MBSSUTR) set to 127 is given in Table 26-134.

---

[1] see Section 26.3, “Controller Host Interface Clocking” for all constraints of minimum CHI clock frequency.

Table 26-134. Minimum f_{chi} [MHz] examples (128 message buffers used)

pdMicrotick [ns]	*gdMinislot*					
	2	3	4	5	6	7
25.0	79.5	53	39.8	31.8	26.5	22.8
50.0	39.8	26.5	19.9	15.9	13.3	11.4

NOTE

If the minimum CHI frequency is not met the CHIERFR[MBS_EF] flag is set. Refer to Section 26.5.2.17, “CHI Error Flag Register (FR_CHIERFR)” for details.

26.7.6 Protocol Control Command Execution

This section considers the issues of the protocol control command execution.

The application issues any of the protocol control commands listed in the POCCMD field of Table 26-16 by writing the command to the POCCMD field of the Protocol Operation Control Register (FR_POCR). As a result the CC sets the BSY bit while the command is transferred to the PE. When the PE has accepted the command, the BSY flag is cleared. All commands are accepted by the PE.

The PE maintains a protocol command vector. For each command that was accepted by the PE, the PE sets the corresponding command bit in the protocol command vector. If a command is issued while the corresponding command bit is set, the command is not queued and is lost.

If the command execution block of the PE is idle, it selects the next accepted protocol command with the highest priority from the current protocol command vector according to the protocol control command priorities given in Table 26-135. If the current protocol state does not allow the execution of this protocol command (see POC state changes in *FlexRay Communications System Protocol Specification, Version 2.1 Rev A*) the CC asserts the illegal protocol command interrupt flag IPC_IF in the Protocol Interrupt Flag Register 1 (FR_PIFR1). The protocol command is not executed in this case.

Some protocol commands may be interrupted by other commands or the detection of a fatal protocol error as indicated by Table 26-135. If the application issues the FREEZE or READY command, or if the PE detects a fatal protocol error, some commands already stored in the command vector will be removed from this vector.

Table 26-135. Protocol Control Command Priorities

Protocol Command	Priority	Interrupted By	Cleared and Terminated By
FREEZE	(highest) 1	none	
READY	2		
CONFIG_COMPLETE	3		

Table 26-135. Protocol Control Command Priorities

Protocol Command	Priority	Interrupted By	Cleared and Terminated By
ALL_SLOTS	4	FREEZE, READY, CONFIG_COMPLET, fatal protocol error	FREEZE, READY, CONFIG_COMPLETE, fatal protocol error
ALLOW_COLDSTART	5		
RUN	6		FREEZE, fatal protocol error
WAKEUP	7		FREEZE, fatal protocol error
DEFAULT_CONFIG	8		FREEZE, fatal protocol error
CONFIG	9		
HALT	(lowest) 10		FREEZE, READY, CONFIG_COMPLETE, fatal protocol error

26.7.7 Message Buffer Search on Simple Message Buffer Configuration

This sections describes the message buffer search behavior for a simplified message buffer configuration. The FIFO behavior is not considered in this section.

26.7.7.1 Simple Message Buffer Configuration

A simple message buffer configuration is a configuration that has at most one transmit message buffer and at most one receive message buffer assigned to a slot *S*. The simple configuration used in this section utilizes two message buffers, one single buffered transmit message buffer and one receive message buffer.

The transmit message buffer has the message buffer number *t* and has following configuration

Table 26-136. Transmit Buffer Configuration

Register	Field	Value	Description
FR_MBCCSR*t*	MTD	1	transmit buffer
FR_MBCCFR*t*	MTM	0	event transition mode
	CHA	1	assigned to channel A
	CHB	0	not assigned to channel B
	CCFE	1	cycle counter filter enabled
	CCFMSK	000011	cycle set = {4n} = {0,4,8,12,...}
	CCFVAL	000000	
FR_MBFIDR*t*	FID	S	assigned to slot S

The availability of data in the transmit buffer is indicated by the commit bit FR_MBCCSRt[CMT] and the lock bit FR_MBCCSRt[LCKS].

The receive message buffer has the message buffer number *r* and has following configuration

Table 26-137. Receive Buffer Configuration

Register	Field	Value	Description
FR_MBCCSR*r*	MTD	0	receive buffer
FR_MBCCFR*r*	MTM	-	n/a
	CHA	1	assigned to channel A
	CHB	0	not assigned to channel B
	CCFE	1	cycle counter filter enabled
	CCFMSK	000001	cycle set = {2n} = {0,2,4,6,...}
	CCFVAL	000000	
FR_MBFIDR*r*	FID	S	subscribed slot

Furthermore the assumption is that both message buffers are enabled (FR_MBCCSR*t*[EDS] = 1 and FR_MBCCSR*r*[EDS] = 1)

NOTE

The cycle set {4n+2} = {2,6,10,...} is assigned to the receive buffer only.

The cycle set {4n} = {0,4,8,12,...} is assigned to both buffers.

26.7.7.2 Behavior in static segment

In this case, both message buffers are assigned to a slot *S* in the *static* segment.

The configuration of a transmit buffer for a static slot *S* assigns this slot to the node as a transmit slot. The FlexRay protocol requires:

- When a slot occurs, if the slot is assigned to a node on a channel that node must transmit either a normal frame or a null frame on that channel. Specifically, a null frame will be sent if there is no data ready, or if there is no match on a transmit filter (cycle counter filtering, for example).

Regardless of the availability of data and the cycle counter filter, the node will transmit a frame in the static slot *S*. In any case, the result of the message buffer search will be the transmit message buffer *t*. The receive message buffer *r* will not be found, no reception is possible.

26.7.7.3 Behavior in dynamic segment

In this case, both message buffers are assigned to a slot *S* in the *dynamic* segment. The FlexRay protocol requires:

- When a slot occurs, if a slot is assigned to a node on a channel that node only transmits a frame on that channel if there is data ready and there is a match on relevant transmit filters (no null frames are sent).

The transmission of a frame in the dynamic segment is determined by the availability of data and the match of the cycle counter filter of the transmit message buffer.

26.7.7.3.1 Transmit Data Not Available

If transmit data are *not available*, i.e., the transmit buffer is not committed FR_MBCCSR*t*[CMT]=0 and/or locked FR_MBCCSR*t*[LCKS]=1,

a) for the cycles in the set {4n}, which is assigned to both buffers, the receive buffer will be found and the node can receive data, and

b) for the cycles in the set {4n+2}, which is assigned to the receive buffer only, the receive buffer will be found and the node can receive data.

The receive cycles are shown in Figure 26-161

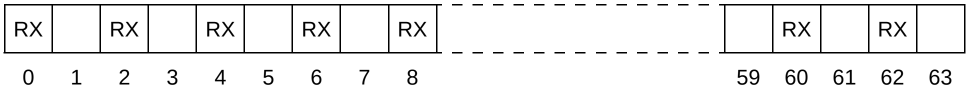


Figure 26-161. Transmit Data Not Available

26.7.7.3.2 Transmit Data Available

If transmit data are *available*, i.e., the transmit buffer is committed FR_MBCCSR*t*[CMT]=1 and not locked FR_MBCCSR*t*[LCKS]=0,

a) for the cycles in the set {4n}, which is assigned to both buffers, the transmit buffer will be found and the node transmits data.

b) for the cycles in the set {4n+2}, which is assigned to the receive buffer only, the receive buffer will be found and the node can receive data.

The receive and transmit cycles are shown in Figure 26-161.

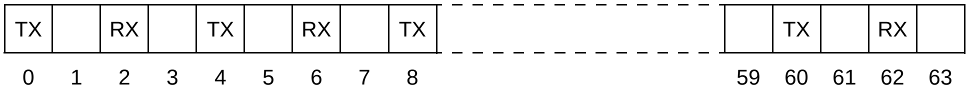


Figure 26-162. Transmit Data Not Available

Chapter 27
Interrupts and Interrupt Controller (INTC)

27.1 Introduction

This chapter describes the interrupts and the interrupt controller (INTC), which schedules interrupt requests (IRQs) from software and internal peripherals to the Core 0 and Core 1 cores. The INTC provides interrupt prioritization and preemption, interrupt masking, interrupt priority elevation, and protocol support. The INTC supports 316 interrupt requests.

The INTC has two independent sets of priority arbitration/comparison, request selection, vector encoder and acknowledge logic—one set for each CPU. This allows each CPU to handle its software-assigned interrupt requests independently of the other CPU’s operation, and provides flexibility for the user to decide which core should handle which interrupt sources in the application. This flexibility comes from a set of configuration bits that allows any interrupt source to generate an interrupt request to either the Core 0 or Core 1 or both the cores.

When multiple tasks share a resource, coherent accesses to that resource need to be supported. The INTC supports the priority ceiling protocol for coherent accesses. By providing a modifiable priority mask, the priority can be raised temporarily so that all tasks which share the resource cannot preempt each other.

Multiple processors can assert interrupt requests to each other through software settable interrupt requests, i.e., by using application software to assert an interrupt request. These same software settable interrupt requests also can be used to break the work involved in servicing an interrupt request into a high priority portion and a low priority portion. The high priority portion is initiated by a peripheral interrupt request, but then the ISR can assert a software settable interrupt request to finish the servicing in a lower priority ISR.

27.1.1 Block Diagram

Interrupts implemented by the MCU are defined in the *e200z7 PowerPC™ Core Reference Manual*.

Figure 27-1 shows a block diagram of the interrupt controller (INTC).

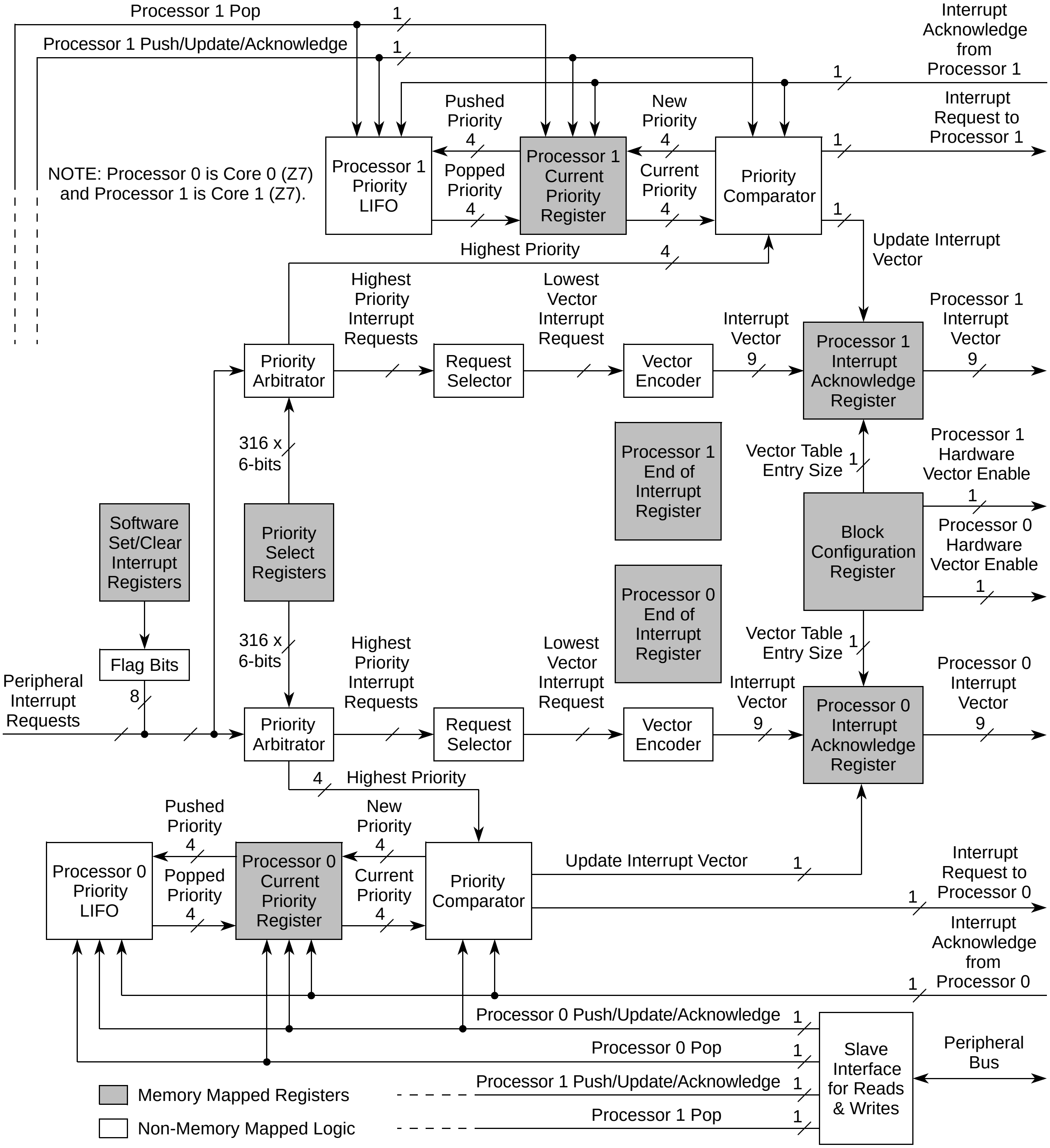


Figure 27-1. INTC Block Diagram

27.1.2 Interrupt Controller Features

- Supports all peripheral and eight software-settable interrupt request sources.
- Each interrupt source can be steered by software to processor 0 (Core 0), processor 1 (Core 1), or both processors interrupt request outputs.

NOTE

> By default, Core 0 receives all interrupt requests, so backward compatibility with single processor systems is maintained.

- 9-bit unique vector for each interrupt request source in hardware vector mode.
- Each interrupt source can be programmed to one of 16 priorities
- Preemption
  - Preemptive prioritized interrupt requests to processor
  - ISR at a higher priority preempts ISRs or tasks at lower priorities
  - Automatic pushing or popping of preempted priority to or from a LIFO
  - Ability to modify the ISR or task priority; modifying the priority can be used to implement the priority ceiling protocol for accessing shared resources.
- Low latency—three clocks from receipt of interrupt request from peripheral to interrupt request to processor.

27.1.3 Modes of Operation

The interrupt controller has two handshaking modes with the processor: software vector mode and hardware vector mode. The state of the hardware vector enable bit, INTC_MCR[HVEN_PRC*n*], independently determines which mode is used for each CPU.

In debug mode the interrupt controller operation is identical to its normal operation of software vector mode or hardware vector mode.

27.1.3.1 Software Vector Mode

In software vector mode, as shown in Figure 27-2, the CPU branches to a common interrupt exception handler whose location is determined by an address derived from special purpose registers IVPR and IVOR4. The interrupt exception handler reads the INTC_IACKR to determine the vector of the interrupt request source.

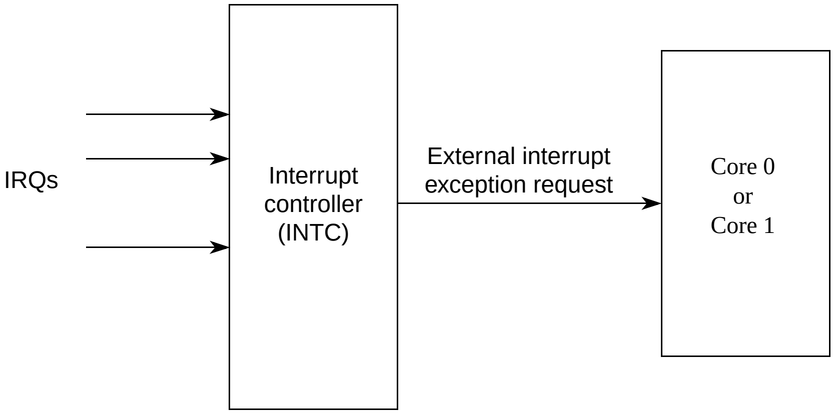
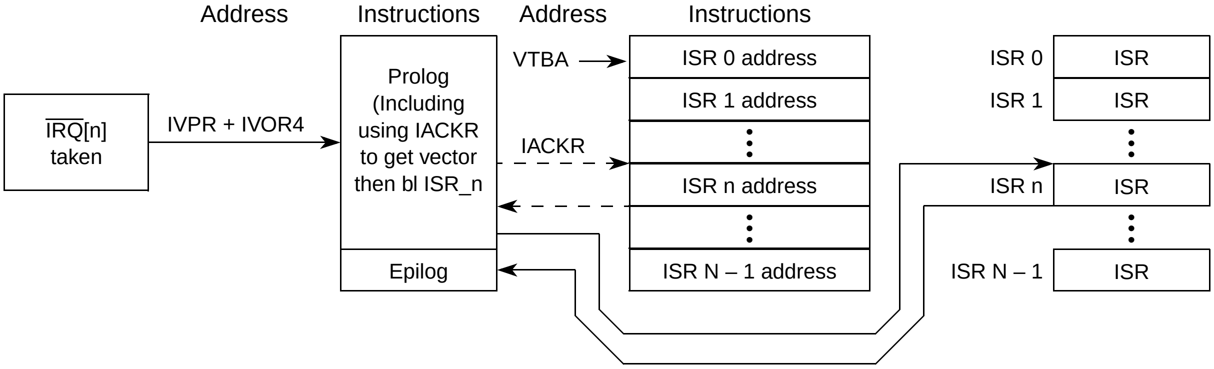


Figure 27-2. INTC Software Vector Mode

Typical program flow for software vector mode is shown in Figure 27-3.



N is the maximum number of usable interrupt vectors and includes eight software-settable IRQ vectors.

Figure 27-3. Program Flow–Software Vector Mode

The common interrupt exception handler address is calculated by hardware as shown in Figure 27-4. The upper half of the interrupt vector prefix register (IVPR) is added to the offset contained in the external input interrupt vector offset register (IVOR4). Note that since bits IVOR4[28:31] are not part of the offset value, the vector offset must be located on a quad-word (16-byte) aligned location in memory.

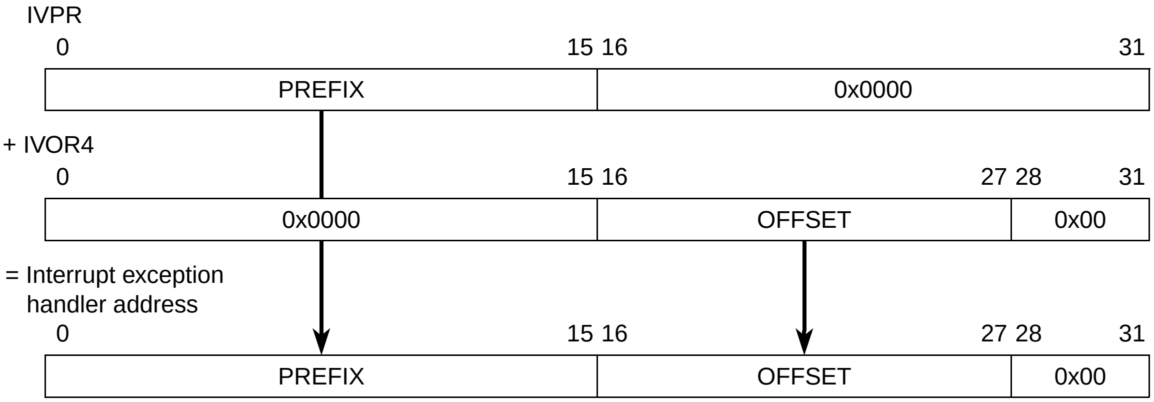


Figure 27-4. Software Vector Mode: Interrupt Exception Handler Address Calculation

As shown in Figure 27-3, the common interrupt exception handler reads the INTC_IACKR_PRC*n* to determine the vector of the interrupt request source. The INTC_IACKR_PRC*n* register contains a 32-bit address for a vector table base address (VTBA) plus an offset to access the interrupt vector (INTVEC). The address is then used to branch to the corresponding routine for that peripheral or software interrupt source.

Reading the INTC_IACKR_PRC*n* acknowledges the INTC's interrupt request and negates the interrupt request to the processor. The interrupt request to the processor does not clear if a higher priority interrupt request arrives. Even in this case, INTVEC does not update to the higher priority request until the lower priority interrupt request is acknowledged by reading the INTC_IACKR_PRC*n*. The reading also pushes the PRI value in the INTC current priority register (INTC_CPR_PRC*n*) onto the LIFO and updates PRI in the INTC_CPR_PRC*n* with the priority of the interrupt request. The INTC_CPR_PRC*n* masks any peripheral or software settable interrupt request at the same or lower priority of the current value of the PRI field in INTC_CPR_PRC*n* from generating an interrupt request to the processor.

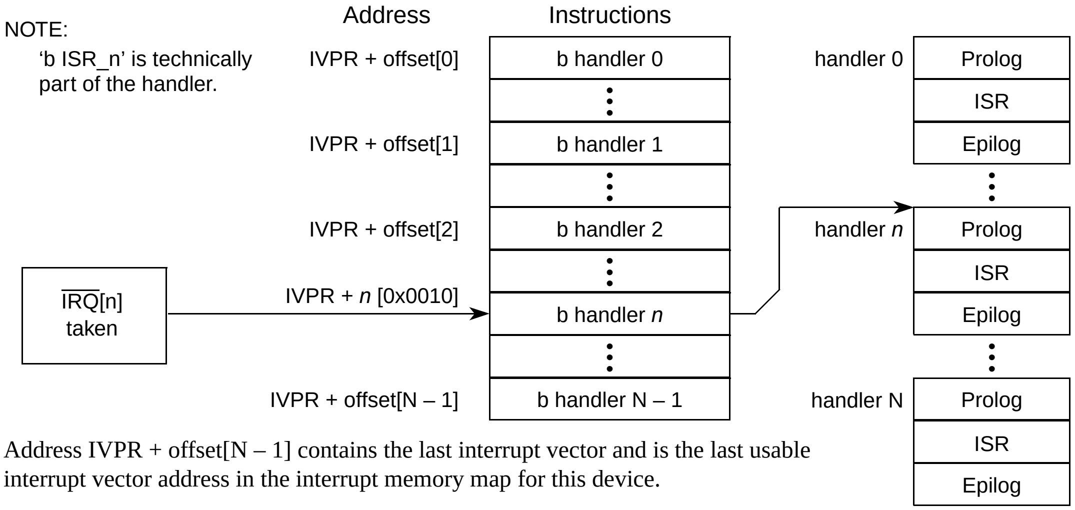
The interrupt exception handler must write to the end-of-interrupt register (INTC_EOIR_PRC*n*) to complete the operation. Writing to the INTC_EOIR_PRC*n* ends the servicing of the interrupt request. The INTC's LIFO is popped into the INTC_CPR_PRC*n*'s PRI field by writing to the INTC_EOIR_PRC*n*, and the size of a write does not affect the operation of the write. Those values and sizes written to this register neither update the INTC_EOIR_PRC*n* contents nor affect whether the LIFO pops. For possible future compatibility, write four bytes of all 0s to the INTC_EOIR_PRC*n*. The timing relationship between popping the LIFO and disabling recognition of external input has no restriction. The writes can happen in either order.

However, disabling recognition of the external input before popping the LIFO eases the calculation of the maximum stack depth at the cost of postponing the servicing of the next interrupt request.

27.1.3.2 Hardware Vector Mode

For high priority interrupt requests, the time from the assertion of the interrupt request from the peripheral to when the processor is performing useful work to service the interrupt request needs to be minimized. The INTC can be optimized to support this goal through the hardware vector mode, where a unique vector is provided for each interrupt request source. It also provides 16 priorities so that lower priority ISRs do not delay the execution of higher priority ISRs. Since each individual application has different priorities for each source of interrupt request, the priority of each interrupt request is configurable.

Typical program flow for hardware vector mode is shown in Figure 27-5.



Address IVPR + offset[N – 1] contains the last interrupt vector and is the last usable interrupt vector address in the interrupt memory map for this device.

N is the maximum number of usable interrupt vectors and includes eight software-settable IRQ vectors.

Figure 27-5. Program Flow–Hardware Vector Mode

In hardware vector mode, the interrupt exception handler address is specific to the peripheral or software settable interrupt source rather than being common to all of them. No IVOR is used. The interrupt exception handler address is calculated by hardware as shown in Figure 27-6. The upper half of the interrupt vector prefix register (IVPR) is added to an offset which corresponds to the peripheral or software interrupt source which caused the interrupt request. The offset matches the value in the Interrupt Vector field, INTC_IACKR_PRC*n*[INTVEC]. Each interrupt exception handler address is aligned on a quad word (16-byte) boundary. IVOR4 is not used in this mode, and software does not need to read INTC_IACKR_PRC*n* to get the interrupt vector number.

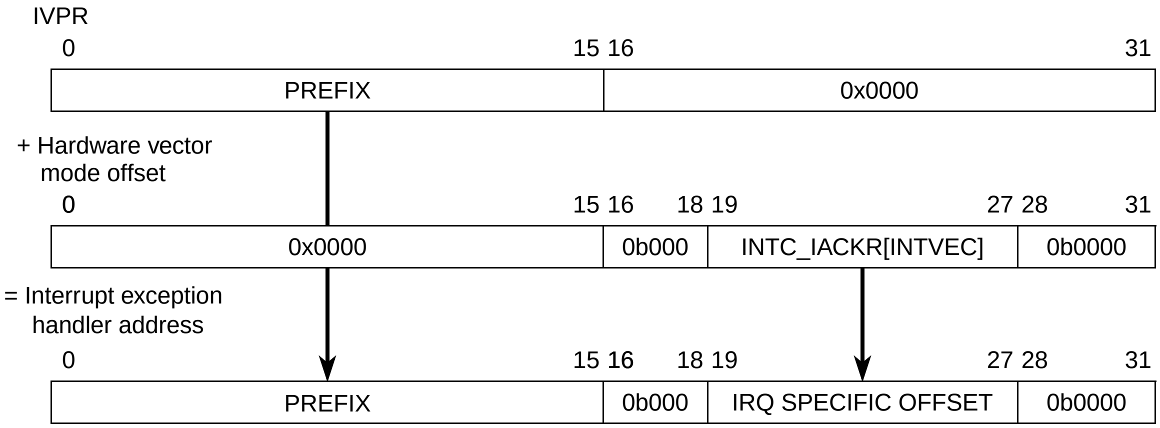


Figure 27-6. Hardware Vector Mode: Interrupt Exception Handler Address Calculation

The processor negates INTC’s interrupt request when automatically acknowledging the interrupt request. However, the interrupt request to the processor do not negate if a higher priority interrupt request arrives. Even in this case, the interrupt vector number does not update to the higher priority request until the lower priority request is acknowledged by the processor.

The assertion of the interrupt acknowledge signal pushes the PRI value in the INTC_CPR_PRC*n* onto the LIFO and updates PRI in the INTC_CPR_PRC*n* with the new priority.

27.2 External Signal Description

The INTC has no direct external MCU signals. However, there are external pins that can be configured in the SIU as external interrupt request input pins. When configured in this function, an interrupt on the pin sets an external interrupt flag. These flags can cause one of five peripheral interrupt requests to the interrupt controller.

For more information on external interrupts, the pins used, and how to configure them, refer to Chapter 2, “Signal Descriptions,” and Chapter 3, “System Integration Units (SIU, SIU_B),” for more information on these pins.

27.3 Memory Map and Registers

27.3.1 INTC Memory Map

Table 27-1 shows the INTC memory map.

Table 27-1. INTC Memory Map

Offset from INTC_BASE_ADDR (0xFFF4_8000)	Register	Access	Reset Value	Section/Page
0x0000	INTC_MCR—INTC module configuration register	R/W	0x0000_0000	27.3.2.1/27-8
0x0004	Reserved			
0x0008	INTC_CPR_PRC0—INTC current priority register for processor 0 (Core 0)	R/W	0x0000_000F	27.3.2.2/27-10
0x00C	INTC_CPR_PRC1—INTC current priority register for processor 1 (Core 1)	R/W	0x0000_000F	27.3.2.3/27-11
0x0010	INTC_IACKR_PRC0—INTC interrupt acknowledge register for processor 0 (Core 0)	R[1]/W	0x0000_0000	27.3.2.4/27-11
0x0014	INTC_IACKR_PRC1—INTC interrupt acknowledge register for processor 1 (Core 1)	R[1]/W	0x0000_0000	27.3.2.5/27-13
0x0018	INTC_EOIR_PRC0—INTC end of interrupt register for processor 0 (Core 0)	W	0x0000_0000	27.3.2.6/27-14
0x001C	INTC_EOIR_PRC1—INTC end of interrupt register for processor 1 (Core 1)	W	0x0000_0000	27.3.2.7/27-14
0x0020	INTC_SSCIR0_3—INTC software set/clear interrupt register 0–3	R/W	0x0000_0000	27.3.2.8/27-15

Table 27-1. INTC Memory Map (continued)

Offset from INTC_BASE_ADDR (0xFFF4_8000)	Register	Access	Reset Value	Section/Page
0x0024	INTC_SSCIR4_7—INTC software set/clear interrupt register 4–7	R/W	0x0000_0000	27.3.2.8/27-15
0x0028 – 0x003F	Reserved			
0x0040 – 0x023F	INTC_PSR0_3—INTC priority select register 0 – 3 to INTC_PSR508_511 — INTC priority select register 508 – 511	R/W	0x0000_0000	27.3.2.9/27-16
0x240 – 0x3FFF	Reserved			

---

[1] When the HVEN bit in the INTC module configuration register (INTC_MCR) is asserted, a read of the INTC_IACKR_PRC*n* has no side effects.

27.3.2 Register Descriptions

With the exception of the INTC_SSCI*n* and INTC_PSR*n* registers, all registers are 32 bits in width. Any combination of accessing the four bytes of a register with a single access is supported, provided that the access does not cross a register boundary. These supported accesses include types and sizes of eight bits, aligned 16 bits, misaligned 16 bits to the middle two bytes, and aligned 32 bits.Although INTC_SSCI*n* and INTC_PSR*n* are 8 bits wide, they can be accessed with a single 16-bit or 32-bit access, provided that the access does not cross a 32-bit boundary.

In software vector mode, the side effects of a read of INTC_IACKR_PRC0 and INTC_IACR_PRC1 are the same regardless of the size of the read. In either software or hardware vector mode, the size of a write to either INTC_EOIR_PRC0 or INTC_EOIR_PRC1 does not affect the operation of the write.

27.3.2.1 INTC Module Configuration Register (INTC_MCR)

The module configuration register is used to configure options of the INTC.

Offset: INTC_BASE_ADDR + 0x0000

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	VTES_PRC1	0	0	0	0	HVEN_PRC1	0	0	VTES_PRC0	0	0	0	0	HVEN_PRC0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-7. INTC Module Configuration Register (INTC_MCR)

Table 27-2. INTC_MCR Field Descriptions

Field	Description
VTES_PRC1	For software mode only, the Vector Table Entry Size for Processor 1 (Core 1). The VTES_PRC1 bit controls the number of 0s to the right of INTVEC_PRC1 in INTC_IACKR_PRC1. If the contents of INTC_IACKR_PRC1 are used as an address of an entry in a vector table, then the number of right-most 0s will determine the size of each vector table entry. 0 4 bytes. 1 8 bytes.
HVEN_PRC1	Hardware Vector Enable for Processor 1 (Core 1). The HVEN bit controls whether the INTC is in hardware vector mode or software vector mode. Refer to Section 27.1.3, “Modes of Operation,” for details of handshaking with the processor in each mode. 0 Software vector mode. 1 Hardware vector mode.
VTES_PRC0	For software mode only, the Vector Table Entry Size for Processor 0 (Core 0). The VTES_PRC0 bit controls the number of 0s to the right of INTVEC_PRC0 in INTC_IACKR_PRC0. If the contents of INTC_IACKR_PRC0 are used as an address of an entry in a vector table, then the number of right-most 0s will determine the size of each vector table entry. 0 4 bytes. 1 8 bytes.
HVEN_PRC0	Hardware Vector Enable for Processor 0 (Core 0). The HVEN bit controls whether the INTC is in hardware vector mode or software vector mode. Refer to Section 27.1.3, “Modes of Operation,” for details of handshaking with the processor in each mode. 0 Software vector mode. 1 Hardware vector mode.

27.3.2.2 INTC Current Priority Register for Processor 0 (Core 0) (INTC_CPR_PRC0)

The current priority register masks any peripheral or software settable interrupt request at the same or lower priority of the current value than the PRI field in INTC_CPR_PRC0 from generating an interrupt request to processor 0 (Core 0). When INTC_IACKR_PRC0 is read in software vector mode, or the interrupt acknowledge signal from the processor is asserted in hardware vector mode, the value of PRI is pushed onto the LIFO, and PRI is updated with the priority of the preempting interrupt request. When INTC_EOIR_PRC0 is written, the LIFO is popped into the INTC_CPR_PRC0's PRI field. An exception case in hardware vector mode to this behavior is described in Section 27.1.3.2, "Hardware Vector Mode."

The masking priority can be raised or lowered by writing to the PRI field, supporting the PCP. Refer to Section 27.5.5, "Priority Ceiling Protocol."

NOTE

The PRI register address space must be configured as guarded and cache-inhibited prior to writing to the register.

NOTE

The INTC does not clear pending interrupts once they have been asserted to the core. This means an elevated priority mask (PRI) does not absolutely guarantee a lower priority interrupt will not be taken. The condition depends on the arrival time of the lower priority interrupt. If it occurs after PRI has been updated in the INTC_CPR_PRCx register, then the lower priority ISR will not be taken. If it occurs before the PRI update in INTC_CPR_PRCx, then the lower priority ISR will be taken. Refer to Section 27.5.5.2, "Ensuring Coherency," for example code to ensure coherency.

Offset: INTC_BASE_ADDR + 0x0008 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	PRI			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 27-8. INTC Current Priority Register for Processor 0 (Core 0) (INTC_CPR_PRC0)

Table 27-3. INTC_CPR_PRC0 Field Descriptions

Field	Description
PRI	Priority. PRI is the priority of the currently executing Core 0 ISR according to the following field values.1111Priority 15 (highest) 1110 Priority 14 ... 0001 Priority 1 0000 Priority 0 (lowest)

27.3.2.3 INTC Current Priority Register for Processor 1 (Core 1) (INTC_CPR_PRC1)

The function of this register is the same as described for processor 0 (Core 0) in Section 27.3.2.2, “INTC Current Priority Register for Processor 0 (Core 0) (INTC_CPR_PRC0).”

Offset: INTC_BASE_ADDR + 0x000C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	PRI			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 27-9. INTC Current Priority Register for Processor 1 (Core 1) (INTC_CPR_PRC1)

Table 27-4. INTC_CPR_PRC1 Field Descriptions

Field	Description
PRI	Priority. The function of this field is the same as described for processor 0 (Core 0) in Section 27.3.2.2, “INTC Current Priority Register for Processor 0 (Core 0) (INTC_CPR_PRC0).”

27.3.2.4 INTC Interrupt Acknowledge Register for Processor 0 (Core 0) (INTC_IACKR_PRC0)

The INTC_IACKR_PRC*n* provides a value that can be used to load the address of an ISR from a vector table. The vector table can be composed of addresses of the ISRs specific to their respective interrupt vectors.

In software vector mode, reading the INTC_IACKR_PRC0 acknowledges the INTC’s interrupt request. Refer to Section 27.1.3, “Modes of Operation,” for a detailed description of the effect on the interrupt request to the processor. The reading also pushes the PRI value in the INTC current priority register (INTC_CPR_PRC*n*) onto the LIFO and updates PRI in the INTC_CPR_PRC*n* with the priority of the interrupt request. The side effect from the reads in software vector mode, that is, the effect on the interrupt request to the processor, the current priority, and the LIFO, are the same regardless of the size of the read

Reading the INTC_IACKR_PRC*n* does not have side effects in hardware vector mode.

NOTE

The INTC_IACKR_PRC*n* must not be read speculatively while in software vector mode. Therefore, for future compatibility, the TLB entry covering the INTC_IACKR_PRC*n* must be configured to be guarded.

In software vector mode, the INTC_IACKR_PRC*n* must be read before setting MSR[EE]. No synchronization instruction is needed after reading the INTC_IACKR_PRC*n* and before setting MSR[EE].

However, the time for the processor to recognize the assertion or negation of the external input to it is not defined by the book E architecture and can be greater than 0. Therefore, insert instructions between the reading of the INTC_IACKR_PRC*n* and the setting of MSR[EE] that consumes at least two processor clock cycles. This length of time allows the interrupt request negation to propagate through the processor before MSR[EE] is set.

Offset: INTC_BASE_ADDR + 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	VTBA_PRC0 (most significant 16 bits)															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	VTBA_PRC0 (least significant five bits)					INTVEC_PRC0[1]									0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] When the VTES_PRC0 bit in INTC_MCR is asserted, INTVEC_PRC0 is shifted to the left one bit. Bit 29 is read as a 0. VTBA_PRC0 is narrowed to 20 bits in width.

Figure 27-10. INTC Interrupt Acknowledge Register for Processor 0 (Core 0) (INTC_IACKR_PRC0)

Table 27-5. INTC_IACKR_PRC0 Field Descriptions

Field	Description
VTBA_PRC0	Vector Table Base Address for Processor 0 (Core 0). VTBA_PRC0 can be the base address of a vector table of addresses of ISRs for processor 0 (Core 0). The VTBA_PRC0 only uses the left-most 20 bits when the VTES_PRC0 bit in INTC_MCR is asserted.
INTVEC_PRC0	Interrupt Vector for Processor 0 (Core 0). INTVEC_PRC0 is the vector of the peripheral or software settable interrupt request that caused the interrupt request to the processor. When the interrupt request to the processor asserts, the INTVEC_PRC0 is updated, whether the INTC is in software or hardware vector mode.

27.3.2.5 INTC Interrupt Acknowledge Register for Processor 1 (Core 1) (INTC_IACKR_PRC1)

The function of this register is the same as described for processor 0 (Core 0) in Section 27.3.2.4, "INTC Interrupt Acknowledge Register for Processor 0 (Core 0) (INTC_IACKR_PRC0)", applied to processor 1 (Core 1).

Offset: INTC_BASE_ADDR + 0x0014 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	VTBA_PRC1 (most significant 16 bits)															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	VTBA_PRC1 (5 least-significant bits)					INTVEC_PRC1[1]									0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[1] When the VTES_PRC1 bit in INTC_MCR is asserted, INTVEC_PRC1 is shifted to the left one bit. Bit 29 is read as 0. VTBA_PRC1 is narrowed to 20 bits wide

Figure 27-11. INTC Interrupt Acknowledge Register for Processor 1 (Core 1) (INTC_IACKR_PRC1)

Table 27-6. INTC_IACKR_PRC1 Field Descriptions

Field	Description
VTBA_PRC1	Vector Table Base Address for Processor 1 (Core 1). VTBA_PRC1 can be the base address of a vector table of addresses of ISRs for processor 1 (Core 1). The VTBA_PRC1 only uses the left-most 20 bits when the VTES_PRC1 bit in INTC_MCR is asserted.
INTVEC_PRC1	Interrupt Vector for Processor 1 (Core 1). INTVEC_PRC1 is the vector of the peripheral or software settable interrupt request that caused the interrupt request to the processor. When the interrupt request to the processor asserts, the INTVEC_PRC1 is updated, whether the INTC is in software or hardware vector mode.

27.3.2.6 INTC End-of-Interrupt Register for Processor 0 (Core 0) (INTC_EOIR_PRC0)

Writing to the end-of-interrupt register signals the end of the servicing of the interrupt request. When the INTC_EOIR_PRC0 is written, the priority last pushed on the LIFO is popped into INTC_CPR_PRC0. An exception to this behavior is described in Section 27.1.3.2, “Hardware Vector Mode.” The values and size of data written to the INTC_EOIR_PRC0 are ignored. The values and sizes written to this register neither update the INTC_EOIR_PRC0 contents or affect whether the LIFO pops. For possible future compatibility, write four bytes of all 0s to the INTC_EOIR_PRC0.

Offset: INTC_BASE_ADDR + 0x0018 Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R																
W	INTC_EOIR_PRC0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R																
W	INTC_EOIR_PRC0															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-12. INTC End-of-Interrupt Register for Processor 0 (Core 0) (INTC_EOIR_PRC0)

27.3.2.7 INTC End-of-Interrupt Register for Processor 1 (Core 1) (INTC_EOIR_PRC1)

The register’s function is the same as for processor 0 (Core 0) as described in Section 27.3.2.6, “INTC End-of-Interrupt Register for Processor 0 (Core 0) (INTC_EOIR_PRC0)”, applied to processor 1 (Core 1).

Offset: INTC_BASE_ADDR + 0x001C Access: User write-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R																
W	INTC_EOIR_PRC1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R																
W	INTC_EOIR_PRC1															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-13. INTC End-of-Interrupt Register for Processor 1 (Core 1) (INTC_EOIR_PRC1)

27.3.2.8 INTC Software Set/Clear Interrupt Registers (INTC_SSCIR0_3–INTC_SSCIR4_7)

The software set/clear interrupt registers support the setting or clearing of software settable interrupt request. These registers contain eight independent sets of bits to set and clear a corresponding flag bit by software. Excepting being set by software, this flag bit behaves the same as a flag bit set within a peripheral. This flag bit generates an interrupt request within the INTC like a peripheral interrupt request. Writing a 1 to SET*n* leaves SET*n* unchanged at 0 but sets CLR*n*. Writing a 0 to SET*n* has no effect. CLR*n* is the flag bit. Writing a 1 to CLR*n* clears it. Writing a 0 to CLR*n* has no effect. If a 1 is written simultaneously to a pair of SET*n* and CLR*n* bits, CLR*n* is asserted, regardless of whether CLR*n* was asserted before the write.

Offset: INTC_BASE_ADDR + 0x0020 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	CLR0	0	0	0	0	0	0	0	CLR1
W							SET0								SET1	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	CLR2	0	0	0	0	0	0	0	CLR3
W							SET2								SET3	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-14. INTC Software Set/Clear Interrupt Register 0–3 (INTC_SSCIR[0:3])

Offset: INTC_BASE_ADDR + 0x0024 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	CLR4	0	0	0	0	0	0	0	CLR5
W							SET4								SET5	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	CLR6	0	0	0	0	0	0	0	CLR7
W							SET6								SET7	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-15. INTC Software Set/Clear Interrupt Register 4–7 (INTC_SSCIR[4:7])

Table 27-7. INTC_SSCIR[0:7] Field Descriptions

Field	Description
SET	Set Flag Bits. Writing a 1 sets the corresponding CLR*n* bit. Writing a 0 has no effect. Each SET*n* is always read as a 0.
CLR	Clear Flag Bits. CLR*n* is the flag bit. Writing a 1 to CLR*nx* clears it provided that a 1 is not written simultaneously to its corresponding SET*n* bit. Writing a 0 to CLR*n* has no effect. 0 Interrupt request not pending within INTC. 1 Interrupt request pending within INTC.

27.3.2.9 INTC Priority Select Registers (INTC_PSR0_3–INTC_PSR508_511)

The priority select registers support the selection of an individual priority for each source of interrupt request, and whether the interrupt request is to be sent to processor 0 (Core 0), processor 1 (Core 1) or both. The unique vector of each peripheral or software settable interrupt request determines which INTC_PSR*n*_*m* is assigned to that interrupt request. The software settable interrupt requests 0–7 are assigned vectors 0–7, and their priorities are configured in INTC_PSR0_3 and INTC_PSR4_7, respectively. The peripheral interrupt requests are assigned to vectors in the range 8 to 511, and their priorities are configured in the range of registers INTC_PSR8_11 to INTC_PSR508_511, respectively (see Section 27.4.1, "External Interrupt Request Sources,"). Vectors in this range that are not assigned to interrupts are reserved. The PSR registers that contain one or more reserved vector priority are listed in Table 27-10.

Offset: INTC_BASE_ADDR + 0x0040 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PRC_SEL0		0	0	PRI0				PRC_SEL1		0	0	PRI1			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PRC_SEL2		0	0	PRI2				PRC_SEL3		0	0	PRI3			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-16. INTC Priority Select Register 0–3 (INTC_PSR0–3)

Offset: INTC_BASE_ADDR + 0x023C Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	PRC_ SEL508		0	0	PRI508				PRC_ SEL509		0	0	PRI509			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PRC_ SEL510		0	0	PRI510				PRC_ SEL511		0	0	PRI511			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 27-17. INTC Priority Select Register 508–511 (INTC_PSR508–511)

Table 27-8. INTC_PSR0_3–INTC_PSR508_511 Field Descriptions

Field	Description
PRC_SEL0– PRC_SEL511	Processor Select. If an interrupt source is enabled, PRC_SEL*n* selects whether the interrupt request is to be sent to processor 0 (Core 0), processor 1 (Core 1), or both. See Table 27-9.
PRI0– PRI511	Priority Select. PRI*n* selects the priority for interrupt requests. Refer to Section 27.4.2, “Priority Management.” 1111 Priority 15 (highest) 1110 Priority 14 ... 0001 Priority 1 0000 Priority 0 (lowest)

NOTE

The PRC_SEL*n* or PRI*n* field of an INTC_PSR*n*_*n* must not be modified while the corresponding peripheral or software settable interrupt request is asserted.

Table 27-9. Selected Processor for Interrupt Request

PRC_SEL*n*	Meaning
00	Interrupt request sent to processor 0 (Core 0)
01	Interrupt request sent to both processors
10	Reserved
11	Interrupt request sent to processor 1 (Core 1)

Table 27-10. Reserved Vectors and PSR numbers

Vector Number	Contained in PSR number
154	INTC_PSR152_155
175	INTC_PSR172_175
194	INTC_PSR192_195
195	INTC_PSR192_195
196	INTC_PSR196_199
282	INTC_PSR280_283
310	INTC_PSR308_311
329	INTC_PSR328_331
330	INTC_PSR328_331
331	INTC_PSR328_331
332	INTC_PSR332_335
333	INTC_PSR332_335
334	INTC_PSR332_335

Table 27-10. Reserved Vectors and PSR numbers

Vector Number	Contained in PSR number
335	INTC_PSR332_335
336	INTC_PSR336_339
337	INTC_PSR336_339
338	INTC_PSR336_339
339	INTC_PSR336_339
340	INTC_PSR340_343
341	INTC_PSR340_343
342	INTC_PSR340_343
343	INTC_PSR340_343
344	INTC_PSR344_347
345	INTC_PSR344_347
346	INTC_PSR344_347
347	INTC_PSR344_347
348	INTC_PSR348_351
349	INTC_PSR348_351
358	INTC_PSR356_359
359	INTC_PSR356_359
360	INTC_PSR360_363
361	INTC_PSR360_363
362	INTC_PSR360_363
363	INTC_PSR360_363
364	INTC_PSR364_367
365	INTC_PSR364_367
474	INTC_PSR472_475
475	INTC_PSR472_475
484	INTC_PSR484_487
485	INTC_PSR484_487
486	INTC_PSR484_487
487	INTC_PSR484_487
504	INTC_PSR504_507
505	INTC_PSR504_507

27.4 Functional Description

27.4.1 External Interrupt Request Sources

The INTC has two types of interrupt requests, peripheral and software settable. The assignments between the interrupt requests from the modules to the vectors for input to the CPU are shown in Table 27-11. The ‘Offset’ column lists the IRQ-specific offsets when using hardware vector mode.

Table 27-11. INTC: Interrupt Request Sources

Offset	Vector	Source	Module	Description
0x0000	0	INTC_SSCIR0[CLR0]	Software	INTC software settable Clear flag 0
0x0010	1	INTC_SSCIR1[CLR1]		INTC software settable Clear flag 1
0x0020	2	INTC_SSCIR2[CLR2]		INTC software settable Clear flag 2
0x0030	3	INTC_SSCIR3[CLR3]		INTC software settable Clear flag 3
0x0040	4	INTC_SSCIR4[CLR4]		INTC software settable Clear flag 4
0x0050	5	INTC_SSCIR5[CLR5]		INTC software settable Clear flag 5
0x0060	6	INTC_SSCIR6[CLR6]		INTC software settable Clear flag 6
0x0070	7	INTC_SSCIR7[CLR7]		INTC software settable Clear flag 7
0x0080	8	SWTA_IR[TIF]	Watchdog A	Software Watchdog A Interrupt flag
0x0090	9	ECSM_ESR[RNCE]|ECSM_ESR[FNCE]	ECSM	Internal SRAM Non-Correctable Error or Flash Non-Correctable Error
0x00A0	10	EDMA_ERL[ERR31:ERR0]	eDMA2_A	eDMA channel Error flags 0 - 31
0x00B0	11	EDMA_IRQRL[INT00]		eDMA channel Interrupt 0
0x00C0	12	EDMA_IRQRL[INT01]		eDMA channel Interrupt 1
0x00D0	13	EDMA_IRQRL[INT02]		eDMA channel Interrupt 2
0x00E0	14	EDMA_IRQRL[INT03]		eDMA channel Interrupt 3
0x00F0	15	EDMA_IRQRL[INT04]		eDMA channel Interrupt 4
0x0100	16	EDMA_IRQRL[INT05]		eDMA channel Interrupt 5
0x0110	17	EDMA_IRQRL[INT06]		eDMA channel Interrupt 6
0x0120	18	EDMA_IRQRL[INT07]		eDMA channel Interrupt 7
0x0130	19	EDMA_IRQRL[INT08]		eDMA channel Interrupt 8
0x0140	20	EDMA_IRQRL[INT09]		eDMA channel Interrupt 9
0x0150	21	EDMA_IRQRL[INT10]		eDMA channel Interrupt 10
0x0160	22	EDMA_IRQRL[INT11]		eDMA channel Interrupt 11
0x0170	23	EDMA_IRQRL[INT12]		eDMA channel Interrupt 12
0x0180	24	EDMA_IRQRL[INT13]		eDMA channel Interrupt 13
0x0190	25	EDMA_IRQRL[INT14]		eDMA channel Interrupt 14
0x01A0	26	EDMA_IRQRL[INT15]		eDMA channel Interrupt 15

Table 27-11. INTC: Interrupt Request Sources (continued)

Offset	Vector	Source	Module	Description
0x01B0	27	EDMA_IRQRL[INT16]	eDMA2_A	eDMA channel Interrupt 16
0x01C0	28	EDMA_IRQRL[INT17]		eDMA channel Interrupt 17
0x01D0	29	EDMA_IRQRL[INT18]		eDMA channel Interrupt 18
0x01E0	30	EDMA_IRQRL[INT19]		eDMA channel Interrupt 19
0x01F0	31	EDMA_IRQRL[INT20]		eDMA channel Interrupt 20
0x0200	32	EDMA_IRQRL[INT21]		eDMA channel Interrupt 21
0x0210	33	EDMA_IRQRL[INT22]		eDMA channel Interrupt 22
0x0220	34	EDMA_IRQRL[INT23]		eDMA channel Interrupt 23
0x0230	35	EDMA_IRQRL[INT24]		eDMA channel Interrupt 24
0x0240	36	EDMA_IRQRL[INT25]		eDMA channel Interrupt 25
0x0250	37	EDMA_IRQRL[INT26]		eDMA channel Interrupt 26
0x0260	38	EDMA_IRQRL[INT27]		eDMA channel Interrupt 27
0x0270	39	EDMA_IRQRL[INT28]		eDMA channel Interrupt 28
0x0280	40	EDMA_IRQRL[INT29]		eDMA channel Interrupt 29
0x0290	41	EDMA_IRQRL[INT30]		eDMA channel Interrupt 30
0x02A0	42	EDMA_IRQRL[INT31]		eDMA channel Interrupt 31
0x02B0	43	FMPLL_SYNSR[LOCF]	PLL	FMPLL Loss of Clock Flag
0x02C0	44	FMPLL_SYNSR[LOLF]		FMPLL Loss of Lock Flag
0x02D0	45	SIU_OSR[OVF15:OVF0]	SIU_A	SIU combined overrun interrupt requests of the external interrupt Overrun Flags
0x02E0	46	SIU_EIISR[EIF0]		SIU External Interrupt Flag 0
0x02F0	47	SIU_EIISR[EIF1]		SIU External Interrupt Flag 1
0x0300	48	SIU_EIISR[EIF2]		SIU External Interrupt Flag 2
0x0310	49	SIU_EIISR[EIF3]		SIU External Interrupt Flag 3
0x0320	50	SIU_EIISR[EIF15:EIF4]		SIU External Interrupt Flags 15–4

Table 27-11. INTC: Interrupt Request Sources (continued)

Offset	Vector	Source	Module	Description
0x0330	51	EMIOS_GFR[F0]	eMIOS_A	eMIOS_A channel 0 Flag
0x0340	52	EMIOS_GFR[F1]		eMIOS_A channel 1 Flag
0x0350	53	EMIOS_GFR[F2]		eMIOS_A channel 2 Flag
0x0360	54	EMIOS_GFR[F3]		eMIOS_A channel 3 Flag
0x0370	55	EMIOS_GFR[F4]		eMIOS_A channel 4 Flag
0x0380	56	EMIOS_GFR[F5]		eMIOS_A channel 5 Flag
0x0390	57	EMIOS_GFR[F6]		eMIOS_A channel 6 Flag
0x03A0	58	EMIOS_GFR[F7]		eMIOS_A channel 7 Flag
0x03B0	59	EMIOS_GFR[F8]		eMIOS_A channel 8 Flag
0x03C0	60	EMIOS_GFR[F9]		eMIOS_A channel 9 Flag
0x03D0	61	EMIOS_GFR[F10]		eMIOS_A channel 10 Flag
0x03E0	62	EMIOS_GFR[F11]		eMIOS_A channel 11 Flag
0x03F0	63	EMIOS_GFR[F12]		eMIOS_A channel 12 Flag
0x0400	64	EMIOS_GFR[F13]		eMIOS_A channel 13 Flag
0x0410	65	EMIOS_GFR[F14]		eMIOS_A channel 14 Flag
0x0420	66	EMIOS_GFR[F15]		eMIOS_A channel 15 Flag

Table 27-11. INTC: Interrupt Request Sources (continued)

Offset	Vector	Source	Module	Description
0x0430	67	ETPU_MCRAB[MGEA,MGEB, ILFA,ILFB,SCMMISF]	eTPUA	eTPU Engine A and B Global Exception
0x0440	68	ETPU_CISR_A[CIS0]		eTPU Engine A Channel 0 Interrupt Status
0x0450	69	ETPU_CISR_A[CIS1]		eTPU Engine A Channel 1 Interrupt Status
0x0460	70	ETPU_CISR_A[CIS2]		eTPU Engine A Channel 2 Interrupt Status
0x0470	71	ETPU_CISR_A[CIS3]		eTPU Engine A Channel 3 Interrupt Status
0x0480	72	ETPU_CISR_A[CIS4]		eTPU Engine A Channel 4 Interrupt Status
0x0490	73	ETPU_CISR_A[CIS5]		eTPU Engine A Channel 5 Interrupt Status
0x04A0	74	ETPU_CISR_A[CIS6]		eTPU Engine A Channel 6 Interrupt Status
0x04B0	75	ETPU_CISR_A[CIS7]		eTPU Engine A Channel 7 Interrupt Status
0x04C0	76	ETPU_CISR_A[CIS8]		eTPU Engine A Channel 8 Interrupt Status
0x04D0	77	ETPU_CISR_A[CIS9]		eTPU Engine A Channel 9 Interrupt Status
0x04E0	78	ETPU_CISR_A[CIS10]		eTPU Engine A Channel 10 Interrupt Status
0x04F0	79	ETPU_CISR_A[CIS11]		eTPU Engine A Channel 11 Interrupt Status
0x0500	80	ETPU_CISR_A[CIS12]		eTPU Engine A Channel 12 Interrupt Status
0x0510	81	ETPU_CISR_A[CIS13]		eTPU Engine A Channel 13 Interrupt Status
0x0520	82	ETPU_CISR_A[CIS14]		eTPU Engine A Channel 14 Interrupt Status
0x0530	83	ETPU_CISR_A[CIS15]		eTPU Engine A Channel 15 Interrupt Status
0x0540	84	ETPU_CISR_A[CIS16]		eTPU Engine A Channel 16 Interrupt Status
0x0550	85	ETPU_CISR_A[CIS17]		eTPU Engine A Channel 17 Interrupt Status
0x0560	86	ETPU_CISR_A[CIS18]		eTPU Engine A Channel 18 Interrupt Status
0x0570	87	ETPU_CISR_A[CIS19]		eTPU Engine A Channel 19 Interrupt Status
0x0580	88	ETPU_CISR_A[CIS20]		eTPU Engine A Channel 20 Interrupt Status
0x0590	89	ETPU_CISR_A[CIS21]		eTPU Engine A Channel 21 Interrupt Status
0x05A0	90	ETPU_CISR_A[CIS22]		eTPU Engine A Channel 22 Interrupt Status
0x05B0	91	ETPU_CISR_A[CIS23]		eTPU Engine A Channel 23 Interrupt Status
0x05C0	92	ETPU_CISR_A[CIS24]		eTPU Engine A Channel 24 Interrupt Status
0x05D0	93	ETPU_CISR_A[CIS25]		eTPU Engine A Channel 25 Interrupt Status
0x05E0	94	ETPU_CISR_A[CIS26]		eTPU Engine A Channel 26 Interrupt Status
0x05F0	95	ETPU_CISR_A[CIS27]		eTPU Engine A Channel 27 Interrupt Status
0x0600	96	ETPU_CISR_A[CIS28]		eTPU Engine A Channel 28 Interrupt Status
0x0610	97	ETPU_CISR_A[CIS29]		eTPU Engine A Channel 29 Interrupt Status
0x0620	98	ETPU_CISR_A[CIS30]		eTPU Engine A Channel 30 Interrupt Status
0x0630	99	ETPU_CISR_A[CIS31]		eTPU Engine A Channel 31 Interrupt Status

Table 27-11. INTC: Interrupt Request Sources (continued)

Offset	Vector	Source	Module	Description
0x0640	100	EQADCA_FISR*x*[TORF] EQADCA_FISR*x*[RFOF] EQADCA_FISR*x*[CFUF]	eQADC_A	eQADC combined overrun interrupt requests from all of the FIFOs: Trigger Overrun, Receive FIFO Overflow, and command FIFO Underflow
0x0650	101	EQADCA_FISR0[NCF]		eQADC command FIFO 0 Non-Coherency Flag
0x0660	102	EQADCA_FISR0[PF]		eQADC command FIFO 0 Pause Flag
0x0670	103	EQADCA_FISR0[EOQF]		eQADC command FIFO 0 command queue End of Queue Flag
0x0680	104	EQADCA_FISR0[CFFF]		eQADC Command FIFO 0 Fill Flag
0x0690	105	EQADCA_FISR0[RFDF]		eQADC Receive FIFO 0 Drain Flag
0x06A0	106	EQADCA_FISR1[NCF]		eQADC command FIFO 1 Non-Coherency Flag
0x06B0	107	EQADCA_FISR1[PF]		eQADC command FIFO 1 Pause Flag
0x06C0	108	EQADCA_FISR1[EOQF]		eQADC command FIFO 1 command queue End of Queue Flag
0x06D0	109	EQADCA_FISR1[CFFF]		eQADC Command FIFO 1 Fill Flag
0x06E0	110	EQADCA_FISR1[RFDF]		eQADC Receive FIFO 1 Drain Flag
0x06F0	111	EQADCA_FISR2[NCF]		eQADC command FIFO 2 Non-Coherency Flag
0x0700	112	EQADCA_FISR2[PF]		eQADC command FIFO 2 Pause Flag
0x0710	113	EQADCA_FISR2[EOQF]		eQADC command FIFO 2 command queue End of Queue Flag
0x0720	114	EQADCA_FISR2[CFFF]		eQADC Command FIFO 2 Fill Flag
0x0730	115	EQADCA_FISR2[RFDF]		eQADC Receive FIFO 2 Drain Flag
0x0740	116	EQADCA_FISR3[NCF]		eQADC command FIFO 3 Non-Coherency Flag
0x0750	117	EQADCA_FISR3[PF]		eQADC command FIFO 3 Pause Flag
0x0760	118	EQADCA_FISR3[EOQF]		eQADC command FIFO 3 command queue End of Queue Flag
0x0770	119	EQADCA_FISR3[CFFF]		eQADC Command FIFO 3 Fill Flag
0x0780	120	EQADCA_FISR3[RFDF]		eQADC Receive FIFO 3 Drain Flag
0x0790	121	EQADCA_FISR4[NCF]		eQADC command FIFO 4 Non-Coherency Flag
0x07A0	122	EQADCA_FISR4[PF]		eQADC command FIFO 4 Pause Flag
0x07B0	123	EQADCA_FISR4[EOQF]		eQADC command FIFO 4 command queue End of Queue Flag
0x07C0	124	EQADCA_FISR4[CFFF]		eQADC Command FIFO 4 Fill Flag
0x07D0	125	EQADCA_FISR4[RFDF]		eQADC Receive FIFO 4 Drain Flag
0x07E0	126	EQADCA_FISR5[NCF]		eQADC command FIFO 5 Non-Coherency Flag
0x07F0	127	EQADCA_FISR5[PF]		eQADC command FIFO 5 Pause Flag
0x0800	128	EQADCA_FISR5[EOQF]		eQADC command FIFO 5 command queue End of Queue Flag
0x0810	129	EQADCA_FISR5[CFFF]		eQADC Command FIFO 5 Fill Flag
0x0820	130	EQADCA_FISR5[RFDF]		eQADC Receive FIFO 5 Drain Flag

Table 27-11. INTC: Interrupt Request Sources (continued)

Offset	Vector	Source	Module	Description
0x0830	131	DSPIB_SR[TFUF] DSPIB_SR[RFOF] DSPIB_SR[SPEF] DSPIB_SR[DPEF]	DSPI_BCD	DSPIB combined overrun & parity error interrupt requests: Transmit FIFO Underflow/Receive FIFO Overflow SPI Parity Error/DSI Parity Error
0x0840	132	DSPIB_SR[EOQF]		DSPIB transmit FIFO End of Queue Flag
0x0850	133	DSPIB_SR[TFFF]		DSPIB Transmit FIFO Fill Flag
0x0860	134	DSPIB_SR[TCF]|DSPIB_SR[DDIF]		DSPIB Transfer Complete/DSI Data Match Flag
0x0870	135	DSPIB_SR[RFDF]		DSPIB Receive FIFO Drain Flag
0x0880	136	DSPIC_SR[TFUF] DSPIC_SR[RFOF] DSPIC_SR[SPEF] DSPIC_SR[DPEF]		DSPIC combined overrun & parity error interrupt requests: Transmit FIFO Underflow/Receive FIFO Overflow SPI Parity Error/DSI Parity Error
0x0890	137	DSPIC_SR[EOQF]		DSPIC transmit FIFO End of Queue Flag
0x08A0	138	DSPIC_SR[TFFF]		DSPIC Transmit FIFO Fill Flag
0x08B0	139	DSPIC_SR[TCF]|DSPIC_SR[DDIF]		DSPIC Transfer Complete/DSI Data Match Flag
0x08C0	140	DSPIC_SR[RFDF]		DSPIC Receive FIFO Drain Flag
0x08D0	141	DSPID_SR[TFUF] DSPID_SR[RFOF] DSPID_SR[SPEF] DSPID_SR[DPEF]		DSPID combined overrun & parity error interrupt requests: Transmit FIFO Underflow/Receive FIFO Overflow SPI Parity Error/DSI Parity Error
0x08E0	142	DSPID_SR[EOQF]		DSPID transmit FIFO End of Queue Flag
0x08F0	143	DSPID_SR[TFFF]		DSPID Transmit FIFO Fill Flag
0x0900	144	DSPID_SR[TCF]|DSPID_SR[DDIF]		DSPID Transfer Complete/DSI Data Match Flag
0x0910	145	DSPID_SR[RFDF]		DSPID Receive FIFO Drain Flag
0x0920	146	ESCIA_SR[TDRE] ESCIA_SR[TC] ESCIA_SR[RDRF] ESCIA_SR[IDLE] ESCIA_SR[OR] ESCIA_SR[NF] ESCIA_SR[FE] ESCIA_SR[PF] ESCIA_SR[BERR] ESCIA_SR[RXRDY] ESCIA_SR[TXRDY] ESCIA_SR[LWAKE] ESCIA_SR[STO] ESCIA_SR[PBERR] ESCIA_SR[CERR] ESCIA_SR[CKERR] ESCIA_SR[FRC] ESCIA_SR[OVFL]	eSCI_A	Combined Interrupt Requests of ESCI Module A: Transmit Data Register Empty, Transmit Complete, Receive Data Register Full, Idle line, Overrun, Noise Flag, Framing Error Flag, and Parity Error Flag interrupt requests, SCI Status Register 2 Bit Error interrupt request, LIN Status Register 1 Receive Data Ready, Transmit Data Ready, Received LIN Wakeup Signal, Slave TimeOut, Physical Bus Error, CRC Error, Checksum Error, Frame Complete interrupts requests, and LIN Status Register 2 Receive Register Overflow
0x0930	147	Master0 Snoop ipi_int	PCU	PCU_IR0[OIF] | PCU_IR0[EIF]
0x0940	148	Master1 Snoop ipi_int		PCU_IR1[OIF] | PCU_IR1[EIF]

Table 27-11. INTC: Interrupt Request Sources (continued)

Offset	Vector	Source	Module	Description
0x0950	149	ESCIB_SR[TDRE] ESCIB_SR[TC] ESCIB_SR[RDRF] ESCIB_SR[IDLE] ESCIB_SR[OR] ESCIB_SR[NF] ESCIB_SR[FE] ESCIB_SR[PF] ESCIB_SR[BERR] ESCIB_SR[RXRDY] ESCIB_SR[TXRDY] ESCIB_SR[LWAKE] ESCIB_SR[STO] ESCIB_SR[PBERR] ESCIB_SR[CERR] ESCIB_SR[CKERR] ESCIB_SR[FRC] ESCIB_SR[OVFL]	eSCI_B	Combined Interrupt Requests of ESCI Module B: Transmit Data Register Empty, Transmit Complete, Receive Data Register Full, Idle line, Overrun, Noise Flag, Framing Error Flag, and Parity Error Flag interrupt requests, SCI Status Register 2 Bit Error interrupt request, LIN Status Register 1 Receive Data Ready, Transmit Data Ready, Received LIN Wakeup Signal, Slave TimeOut, Physical Bus Error, CRC Error, Checksum Error, Frame Complete interrupts requests, and LIN Status Register 2 Receive Register Overflow
0x0960	150	LRNEIF | DRNEIF		Flexray
0x0970	151	LRCEIF | DRCEIF		Flexray
0x0980	152	CANA_ESR[BOFF_INT]	FlexCAN_A	FLEXCAN_A Bus off Interrupt
0x0990	153	CANA_ESR[ERR_INT]		FLEXCAN_A Error Interrupt
0x09A0	154	Reserved for Wakeup		Reserved for Wakeup
0x09B0	155	CANA_IFRL[BUF0]		FLEXCAN_A Buffer 0 Interrupt
0x09C0	156	CANA_IFRL[BUF1]		FLEXCAN_A Buffer 1 Interrupt
0x09D0	157	CANA_IFRL[BUF2]		FLEXCAN_A Buffer 2 Interrupt
0x09E0	158	CANA_IFRL[BUF3]		FLEXCAN_A Buffer 3 Interrupt
0x09F0	159	CANA_IFRL[BUF4]		FLEXCAN_A Buffer 4 Interrupt
0x0A00	160	CANA_IFRL[BUF5]		FLEXCAN_A Buffer 5 Interrupt
0x0A10	161	CANA_IFRL[BUF6]		FLEXCAN_A Buffer 6 Interrupt
0x0A20	162	CANA_IFRL[BUF7]		FLEXCAN_A Buffer 7 Interrupt
0x0A30	163	CANA_IFRL[BUF8]		FLEXCAN_A Buffer 8 Interrupt
0x0A40	164	CANA_IFRL[BUF9]		FLEXCAN_A Buffer 9 Interrupt
0x0A50	165	CANA_IFRL[BUF10]		FLEXCAN_A Buffer 10 Interrupt
0x0A60	166	CANA_IFRL[BUF11]		FLEXCAN_A Buffer 11 Interrupt
0x0A70	167	CANA_IFRL[BUF12]		FLEXCAN_A Buffer 12 Interrupt
0x0A80	168	CANA_IFRL[BUF13]		FLEXCAN_A Buffer 13 Interrupt
0x0A90	169	CANA_IFRL[BUF14]		FLEXCAN_A Buffer 14 Interrupt
0x0AA0	170	CANA_IFRL[BUF15]		FLEXCAN_A Buffer 15 Interrupt
0x0AB0	171	CANA_IFRL[BUF31:BUF16]		FLEXCAN_A Buffers 31 - 16 Interrupts
0x0AC0	172	CANA_IFRH[BUF63:BUF32]		FLEXCAN_A Buffers 63 - 32 Interrupts

Table 27-11. INTC: Interrupt Request Sources (continued)

Offset	Vector	Source	Module	Description
0x0AD0	173	CANC_ESR[BOFF_INT]	FlexCAN_C	FLEXCAN_C Bus off Interrupt
0x0AE0	174	CANC_ESR[ERR_INT]		FLEXCAN_C Error Interrupt
0x0AF0	175	Reserved for Wakeup		Reserved for Wakeup
0x0B00	176	CANC_IFRL[BUF0]		FLEXCAN_C Buffer 0 Interrupt
0x0B10	177	CANC_IFRL[BUF1]		FLEXCAN_C Buffer 1 Interrupt
0x0B20	178	CANC_IFRL[BUF2]		FLEXCAN_C Buffer 2 Interrupt
0x0B30	179	CANC_IFRL[BUF3]		FLEXCAN_C Buffer 3 Interrupt
0x0B40	180	CANC_IFRL[BUF4]		FLEXCAN_C Buffer 4 Interrupt
0x0B50	181	CANC_IFRL[BUF5]		FLEXCAN_C Buffer 5 Interrupt
0x0B60	182	CANC_IFRL[BUF6]		FLEXCAN_C Buffer 6 Interrupt
0x0B70	183	CANC_IFRL[BUF7]		FLEXCAN_C Buffer 7 Interrupt
0x0B80	184	CANC_IFRL[BUF8]		FLEXCAN_C Buffer 8 Interrupt
0x0B90	185	CANC_IFRL[BUF9]		FLEXCAN_C Buffer 9 Interrupt
0x0BA0	186	CANC_IFRL[BUF10]		FLEXCAN_C Buffer 10 Interrupt
0x0BB0	187	CANC_IFRL[BUF11]		FLEXCAN_C Buffer 11 Interrupt
0x0BC0	188	CANC_IFRL[BUF12]		FLEXCAN_C Buffer 12 Interrupt
0x0BD0	189	CANC_IFRL[BUF13]		FLEXCAN_C Buffer 13 Interrupt
0x0BE0	190	CANC_IFRL[BUF14]		FLEXCAN_C Buffer 14 Interrupt
0x0BF0	191	CANC_IFRL[BUF15]		FLEXCAN_C Buffer 15 Interrupt
0x0C00	192	CANC_IFRL[BUF31:BUF16]		FLEXCAN_C Buffers 31 - 16 Interrupts
0x0C10	193	CANC_IFRH[BUF63:BUF32]		FLEXCAN_C Buffers 63 - 32 Interrupts
0x0C20	194	Reserved	Reserved for FEC	Reserved FEC Transmit Frame flag
0x0C30	195	Reserved		Reserved FEC Receive Frame flag
0x0C40	196	Reserved		Reserved Combined Interrupt Requests of the FEC Ethernet Interrupt Event Register: Heartbeat Error, Babbling Receive Error, Babbling Transmit Error, Graceful Stop Complete, Transmit Buffer, Receive Buffer, Media Independent Interface, Ethernet Bus Error, Late Collision, Collision Retry Limit, and Transmit FIFO Underrun
0x0C50	197	DECFILTER_MSR_A[IDF]	DEC_A	Decimation A Input (Fill)
0x0C60	198	DECFILTER_MSR_A[ODF||SDF]		Decimation A Output/Integ (Drain/Integ)
0x0C70	199	DECFILTER A ERRORS		Decimation A Error
0x0C80	200	STM_CIR0[CIF]	STM	System Timer Channel 0 Interrupt
0x0C90	201	STM_CIR[1:3][CIF]		System Timer Channels 1,2,3 Interrupt

Table 27-11. INTC: Interrupt Request Sources (continued)

Offset	Vector	Source	Module	Description
0x0CA0	202	EMIOS_GFR[F16]	eMIOS_A	eMIOS_A channel 16 Flag
0x0CB0	203	EMIOS_GFR[F17]		eMIOS_A channel 17 Flag
0x0CC0	204	EMIOS_GFR[F18]		eMIOS_A channel 18 Flag
0x0CD0	205	EMIOS_GFR[F19]		eMIOS_A channel 19 Flag
0x0CE0	206	EMIOS_GFR[F20]		eMIOS_A channel 20 Flag
0x0CF0	207	EMIOS_GFR[F21]		eMIOS_A channel 21 Flag
0x0D00	208	EMIOS_GFR[F22]		eMIOS_A channel 22 Flag
0x0D10	209	EMIOS_GFR[F23]		eMIOS_A channel 23 Flag

Table 27-11. INTC: Interrupt Request Sources (continued)

Offset	Vector	Source	Module	Description
0x0D20	210	EDMAA_ERRH[ERR63:ERR32]	eDMA2_A	eDMA channel Error flags 32 - 63
0x0D30	211	EDMAA_IRQRH[INT32]		eDMA channel Interrupt 32
0x0D40	212	EDMAA_IRQRH[INT33]		eDMA channel Interrupt 33
0x0D50	213	EDMAA_IRQRH[INT34]		eDMA channel Interrupt 34
0x0D60	214	EDMAA_IRQRH[INT35]		eDMA channel Interrupt 35
0x0D70	215	EDMAA_IRQRH[INT36]		eDMA channel Interrupt 36
0x0D80	216	EDMAA_IRQRH[INT37]		eDMA channel Interrupt 37
0x0D90	217	EDMAA_IRQRH[INT38]		eDMA channel Interrupt 38
0x0DA0	218	EDMAA_IRQRH[INT39]		eDMA channel Interrupt 39
0x0DB0	219	EDMAA_IRQRH[INT40]		eDMA channel Interrupt 40
0x0DC0	220	EDMAA_IRQRH[INT41]		eDMA channel Interrupt 41
0x0DD0	221	EDMAA_IRQRH[INT42]		eDMA channel Interrupt 42
0x0DE0	222	EDMAA_IRQRH[INT43]		eDMA channel Interrupt 43
0x0DF0	223	EDMAA_IRQRH[INT44]		eDMA channel Interrupt 44
0x0E00	224	EDMAA_IRQRH[INT45]		eDMA channel Interrupt 45
0x0E10	225	EDMAA_IRQRH[INT46]		eDMA channel Interrupt 46
0x0E20	226	EDMAA_IRQRH[INT47]		eDMA channel Interrupt 47
0x0E30	227	EDMAA_IRQRH[INT48]		eDMA channel Interrupt 48
0x0E40	228	EDMAA_IRQRH[INT49]		eDMA channel Interrupt 49
0x0E50	229	EDMAA_IRQRH[INT50]		eDMA channel Interrupt 50
0x0E60	230	EDMAA_IRQRH[INT51]		eDMA channel Interrupt 51
0x0E70	231	EDMAA_IRQRH[INT52]		eDMA channel Interrupt 52
0x0E80	232	EDMAA_IRQRH[INT53]		eDMA channel Interrupt 53
0x0E90	233	EDMAA_IRQRH[INT54]		eDMA channel Interrupt 54
0x0EA0	234	EDMAA_IRQRH[INT55]		eDMA channel Interrupt 55
0x0EB0	235	EDMAA_IRQRH[INT56]		eDMA channel Interrupt 56
0x0EC0	236	EDMAA_IRQRH[INT57]		eDMA channel Interrupt 57
0x0ED0	237	EDMAA_IRQRH[INT58]		eDMA channel Interrupt 58
0x0EE0	238	EDMAA_IRQRH[INT59]		eDMA channel Interrupt 59
0x0EF0	239	EDMAA_IRQRH[INT60]		eDMA channel Interrupt 60
0x0F00	240	EDMAA_IRQRH[INT61]		eDMA channel Interrupt 61
0x0F10	241	EDMAA_IRQRH[INT62]		eDMA channel Interrupt 62
0x0F20	242	EDMAA_IRQRH[INT63]		eDMA channel Interrupt 63

Table 27-11. INTC: Interrupt Request Sources (continued)

Offset	Vector	Source	Module	Description
0x0F30	243	ETPU_CISR_B[CIS0]	eTPU_B	eTPU Engine B Channel 0 Interrupt Status
0x0F40	244	ETPU_CISR_B[CIS1]		eTPU Engine B Channel 1 Interrupt Status
0x0F50	245	ETPU_CISR_B[CIS2]		eTPU Engine B Channel 2 Interrupt Status
0x0F60	246	ETPU_CISR_B[CIS3]		eTPU Engine B Channel 3 Interrupt Status
0x0F70	247	ETPU_CISR_B[CIS4]		eTPU Engine B Channel 4 Interrupt Status
0x0F80	248	ETPU_CISR_B[CIS5]		eTPU Engine B Channel 5 Interrupt Status
0x0F90	249	ETPU_CISR_B[CIS6]		eTPU Engine B Channel 6 Interrupt Status
0x0FA0	250	ETPU_CISR_B[CIS7]		eTPU Engine B Channel 7 Interrupt Status
0x0FB0	251	ETPU_CISR_B[CIS8]		eTPU Engine B Channel 8 Interrupt Status
0x0FC0	252	ETPU_CISR_B[CIS9]		eTPU Engine B Channel 9 Interrupt Status
0x0fd0	253	ETPU_CISR_B[CIS10]		eTPU Engine B Channel 10 Interrupt Status
0x0fe0	254	ETPU_CISR_B[CIS11]		eTPU Engine B Channel 11 Interrupt Status
0x0ff0	255	ETPU_CISR_B[CIS12]		eTPU Engine B Channel 12 Interrupt Status
0x1000	256	ETPU_CISR_B[CIS13]		eTPU Engine B Channel 13 Interrupt Status
0x1010	257	ETPU_CISR_B[CIS14]		eTPU Engine B Channel 14 Interrupt Status
0x1020	258	ETPU_CISR_B[CIS15]		eTPU Engine B Channel 15 Interrupt Status
0x1030	259	ETPU_CISR_B[CIS16]		eTPU Engine B Channel 16 Interrupt Status
0x1040	260	ETPU_CISR_B[CIS17]		eTPU Engine B Channel 17 Interrupt Status
0x1050	261	ETPU_CISR_B[CIS18]		eTPU Engine B Channel 18 Interrupt Status
0x1060	262	ETPU_CISR_B[CIS19]		eTPU Engine B Channel 19 Interrupt Status
0x1070	263	ETPU_CISR_B[CIS20]		eTPU Engine B Channel 20 Interrupt Status
0x1080	264	ETPU_CISR_B[CIS21]		eTPU Engine B Channel 21 Interrupt Status
0x1090	265	ETPU_CISR_B[CIS22]		eTPU Engine B Channel 22 Interrupt Status
0x10A0	266	ETPU_CISR_B[CIS23]		eTPU Engine B Channel 23 Interrupt Status
0x10B0	267	ETPU_CISR_B[CIS24]		eTPU Engine B Channel 24 Interrupt Status
0x10C0	268	ETPU_CISR_B[CIS25]		eTPU Engine B Channel 25 Interrupt Status
0x10D0	269	ETPU_CISR_B[CIS26]		eTPU Engine B Channel 26 Interrupt Status
0x10E0	270	ETPU_CISR_B[CIS27]		eTPU Engine B Channel 27 Interrupt Status
0x10F0	271	ETPU_CISR_B[CIS28]		eTPU Engine B Channel 28 Interrupt Status
0x1100	272	ETPU_CISR_B[CIS29]		eTPU Engine B Channel 29 Interrupt Status
0x1110	273	ETPU_CISR_B[CIS30]		eTPU Engine B Channel 30 Interrupt Status
0x1120	274	ETPU_CISR_B[CIS31]		eTPU Engine B Channel 31 Interrupt Status
0x1130	275	DSPIA_SR[TFUF] DSPIA_SR[RFOF] DSPIA_SR[SPEF] DSPIA_SR[DPEF]	DSPI_A	DSPIA combined overrun & parity error interrupt requests: Transmit FIFO Underflow/Receive FIFO Overflow SPI Parity Error/DSI Parity Error
0x1140	276	DSPIA_SR[EOQF]		DSPIA transmit FIFO End of Queue Flag
0x1150	277	DSPIA_SR[TFFF]		DSPIA Transmit FIFO Fill Flag
0x1160	278	DSPIA_SR[TCF]|DSPIA_SR[DDIF]		DSPIA Transfer Complete/DSI Data Match Flag
0x1170	279	DSPIA_SR[RFDF]		DSPIA Receive FIFO Drain Flag

Table 27-11. INTC: Interrupt Request Sources (continued)

Offset	Vector	Source	Module	Description
0x1180	280	CANB_ESR[BOFF_INT]	FlexCAN_B	FLEXCAN_B Bus off Interrupt
0x1190	281	CANB_ESR[ERR_INT]		FLEXCAN_B Error Interrupt
0x11A0	282	Reserved for Wakeup		Reserved for Wakeup
0x11B0	283	CANB_IFRL[BUF0]		FLEXCAN_B Buffer 0 Interrupt
0x11C0	284	CANB_IFRL[BUF1]		FLEXCAN_B Buffer 1 Interrupt
0x11D0	285	CANB_IFRL[BUF2]		FLEXCAN_B Buffer 2 Interrupt
0x11E0	286	CANB_IFRL[BUF3]		FLEXCAN_B Buffer 3 Interrupt
0x11F0	287	CANB_IFRL[BUF4]		FLEXCAN_B Buffer 4 Interrupt
0x1200	288	CANB_IFRL[BUF5]		FLEXCAN_B Buffer 5 Interrupt
0x1210	289	CANB_IFRL[BUF6]		FLEXCAN_B Buffer 6 Interrupt
0x1220	290	CANB_IFRL[BUF7]		FLEXCAN_B Buffer 7 Interrupt
0x1230	291	CANB_IFRL[BUF8]		FLEXCAN_B Buffer 8 Interrupt
0x1240	292	CANB_IFRL[BUF9]		FLEXCAN_B Buffer 9 Interrupt
0x1250	293	CANB_IFRL[BUF10]		FLEXCAN_B Buffer 10 Interrupt
0x1260	294	CANB_IFRL[BUF11]		FLEXCAN_B Buffer 11 Interrupt
0x1270	295	CANB_IFRL[BUF12]		FLEXCAN_B Buffer 12 Interrupt
0x1280	296	CANB_IFRL[BUF13]		FLEXCAN_B Buffer 13 Interrupt
0x1290	297	CANB_IFRL[BUF14]		FLEXCAN_B Buffer 14 Interrupt
0x12A0	298	CANB_IFRL[BUF15]		FLEXCAN_B Buffer 15 Interrupt
0x12B0	299	CANB_IFRL[BUF31:BUF16]		FLEXCAN_B Buffers 31 - 16 Interrupts
0x12C0	300	CANB_IFRH[BUF63:BUF32]		FLEXCAN_B Buffers 63 - 32 Interrupts
0x12D0	301	PIT[0]	PIT_RTI	PIT Timer 0 Interrupt
0x12E0	302	PIT[1]		PIT Timer 1 Interrupt
0x12F0	303	PIT[2]		PIT Timer 2Interrupt
0x1300	304	PIT[3]		PIT Timer 3 Interrupt
0x1310	305	RTI		PIT RIT Interrupt
0x1320	306	PMC		PMC LVI Interrupt
0x1330	307	ECSM_ESR[R1BC]|ECSM_ESR[F1BC]	ECSM	Internal SRAM 1 bit Correction or Flash 1 bit Correction

Table 27-11. INTC: Interrupt Request Sources (continued)

Offset	Vector	Source	Module	Description
0x1340	308	CAND_ESR[BOFF_INT]	FlexCAN_D	FLEXCAN_D bus off
0x1350	309	CAND_ESR[ERR_INT]		FLEXCAN_D error
0x1360	310	Reserved for Wakeup		Reserved for Wakeup
0x1370	311	CAND_IFRL[BUF0]		FLEXCAN_D buffer 0
0x1380	312	CAND_IFRL[BUF1]		FLEXCAN_D buffer 1
0x1390	313	CAND_IFRL[BUF2]		FLEXCAN_D buffer 2
0x13A0	314	CAND_IFRL[BUF3]		FLEXCAN_D buffer 3
0x13B0	315	CAND_IFRL[BUF4]		FLEXCAN_D buffer 4
0x13C0	316	CAND_IFRL[BUF5]		FLEXCAN_D buffer 5
0x13D0	317	CAND_IFRL[BUF6]		FLEXCAN_D buffer 6
0x13E0	318	CAND_IFRL[BUF7]		FLEXCAN_D buffer 7
0x13F0	319	CAND_IFRL[BUF8]		FLEXCAN_D buffer 8
0x1400	320	CAND_IFRL[BUF9]		FLEXCAN_D buffer 9
0x1410	321	CAND_IFRL[BUF10]		FLEXCAN_D buffer 10
0x1420	322	CAND_IFRL[BUF11]		FLEXCAN_D buffer 11
0x1430	323	CAND_IFRL[BUF12]		FLEXCAN_D buffer 12
0x1440	324	CAND_IFRL[BUF13]		FLEXCAN_D buffer 13
0x1450	325	CAND_IFRL[BUF14]		FLEXCAN_D buffer 14
0x1460	326	CAND_IFRL[BUF15]		FLEXCAN_D buffer 15
0x1470	327	CAND_IFRL[BUF31:BUF16]		FLEXCAN_D buffer 16-31
0x1480	328	CAND_IFRH[BUF63:BUF32]		FLEXCAN_D buffer 32-63

Table 27-11. INTC: Interrupt Request Sources (continued)

Offset	Vector	Source	Module	Description
0x1490	329	Reserved	Reserved for FlexCAN E	Reserved
0x14A0	330	Reserved		Reserved
0x14B0	331	Reserved		Reserved
0x14C0	332	Reserved		Reserved
0x14D0	333	Reserved		Reserved
0x14E0	334	Reserved		Reserved
0x14F0	335	Reserved		Reserved
0x1500	336	Reserved		Reserved
0x1510	337	Reserved		Reserved
0x1520	338	Reserved		Reserved
0x1530	339	Reserved		Reserved
0x1540	340	Reserved		Reserved
0x1550	341	Reserved		Reserved
0x1560	342	Reserved		Reserved
0x1570	343	Reserved		Reserved
0x1580	344	Reserved		Reserved
0x1590	345	Reserved		Reserved
0x15A0	346	Reserved		Reserved
0x15B0	347	Reserved		Reserved
0x15C0	348	Reserved		Reserved
0x15D0	349	Reserved		Reserved
0x15E0	350	FNEAIF	FlexRay	Flexray
0x15F0	351	FNEBIF		Flexray
0x1600	352	WUPIF		Flexray
0x1610	353	PRIF		Flexray
0x1620	354	CHIF		Flexray
0x1630	355	TBIF		Flexray
0x1640	356	RBIF		Flexray
0x1650	357	MIF		Flexray
0x1660	358	Reserved	Reaction Channels	Reserved
0x1670	359	Reserved		Reserved
0x1680	360	Reserved		Reserved
0x1690	361	Reserved		Reserved
0x16A0	362	Reserved		Reserved
0x16B0	363	Reserved		Reserved
0x16C0	364	Reserved		Reserved
0x16D0	365	Reserved		Reserved
0x16E0	366	DECFILTER_MSR_B[IDF]	DEC_B	Decimation B Input (Fill)
0x16F0	367	DECFILTER_MSR_B[ODF‖SDF]		Decimation B Output/Integ (Drain/Integ)
0x1700	368	DECFILTER B ERRORS		Decimation B Error

Table 27-11. INTC: Interrupt Request Sources (continued)

Offset	Vector	Source	Module	Description
0x1710	369	ETPU_MCR_C[MGEC, ILFC, SCMMISF]		eTPU Engine C Global Exceptions
0x1720	370	ETPU_CISR_C[CIS1]	eTPU_C	eTPU Engine C Channel 0 Interrupt Status
0x1730	371	ETPU_CISR_C[CIS2]		eTPU Engine C Channel 1 Interrupt Status
0x1740	372	ETPU_CISR_C[CIS3]		eTPU Engine C Channel 2 Interrupt Status
0x1750	373	ETPU_CISR_C[CIS4]		eTPU Engine C Channel 3 Interrupt Status
0x1760	374	ETPU_CISR_C[CIS5]		eTPU Engine C Channel 4 Interrupt Status
0x1770	375	ETPU_CISR_C[CIS6]		eTPU Engine C Channel 5 Interrupt Status
0x1780	376	ETPU_CISR_C[CIS7]		eTPU Engine C Channel 6 Interrupt Status
0x1790	377	ETPU_CISR_C[CIS8]		eTPU Engine C Channel 7 Interrupt Status
0x17A0	378	ETPU_CISR_C[CIS9]		eTPU Engine C Channel 8 Interrupt Status
0x17B0	379	ETPU_CISR_C[CIS10]		eTPU Engine C Channel 9 Interrupt Status
0x17C0	380	ETPU_CISR_C[CIS11]		eTPU Engine C Channel 10 Interrupt Status
0x17D0	381	ETPU_CISR_C[CIS12]		eTPU Engine C Channel 11 Interrupt Status
0x17E0	382	ETPU_CISR_C[CIS13]		eTPU Engine C Channel 12 Interrupt Status
0x17F0	383	ETPU_CISR_C[CIS14]		eTPU Engine C Channel 13 Interrupt Status
0x1800	384	ETPU_CISR_C[CIS15]		eTPU Engine C Channel 14 Interrupt Status
0x1810	385	ETPU_CISR_C[CIS16]		eTPU Engine C Channel 15 Interrupt Status
0x1820	386	ETPU_CISR_C[CIS17]		eTPU Engine C Channel 16 Interrupt Status
0x1830	387	ETPU_CISR_C[CIS18]		eTPU Engine C Channel 17 Interrupt Status
0x1840	388	ETPU_CISR_C[CIS19]		eTPU Engine C Channel 18 Interrupt Status
0x1850	389	ETPU_CISR_C[CIS20]		eTPU Engine C Channel 19 Interrupt Status
0x1860	390	ETPU_CISR_C[CIS21]		eTPU Engine C Channel 20 Interrupt Status
0x1870	391	ETPU_CISR_C[CIS22]		eTPU Engine C Channel 21 Interrupt Status
0x1880	392	ETPU_CISR_C[CIS23]		eTPU Engine C Channel 22 Interrupt Status
0x1890	393	ETPU_CISR_C[CIS24]		eTPU Engine C Channel 23 Interrupt Status

Table 27-11. INTC: Interrupt Request Sources (continued)

Offset	Vector	Source	Module	Description
0x18A0	394	EQADCB_FISR*x*[TORF] EQADCB_FISR*x*[RFOF] EQADCB_FISR*x*[CFUF]	eQADC_B	eQADC combined overrun interrupt requests from all of the FIFOs: Trigger Overrun, Receive FIFO Overflow, and command FIFO Underflow
0x18B0	395	EQADCB_FISR0[NCF]		eQADC command FIFO 0 Non-Coherency Flag
0x18C0	396	EQADCB_FISR0[PF]		eQADC command FIFO 0 Pause Flag
0x18D0	397	EQADCB_FISR0[EOQF]		eQADC command FIFO 0 command queue End of Queue Flag
0x18E0	398	EQADCB_FISR0[CFFF]		eQADC Command FIFO 0 Fill Flag
0x18F0	399	EQADCB_FISR0[RFDF]		eQADC Receive FIFO 0 Drain Flag
0x1900	400	EQADCB_FISR1[NCF]		eQADC command FIFO 1 Non-Coherency Flag
0x1910	401	EQADCB_FISR1[PF]		eQADC command FIFO 1 Pause Flag
0x1920	402	EQADCB_FISR1[EOQF]		eQADC command FIFO 1 command queue End of Queue Flag
0x1930	403	EQADCB_FISR1[CFFF]		eQADC Command FIFO 1 Fill Flag
0x1940	404	EQADCB_FISR1[RFDF]		eQADC Receive FIFO 1 Drain Flag
0x1950	405	EQADCB_FISR2[NCF]		eQADC command FIFO 2 Non-Coherency Flag
0x1960	406	EQADCB_FISR2[PF]		eQADC command FIFO 2 Pause Flag
0x1970	407	EQADCB_FISR2[EOQF]		eQADC command FIFO 2 command queue End of Queue Flag
0x1980	408	EQADCB_FISR2[CFFF]		eQADC Command FIFO 2 Fill Flag
0x1990	409	EQADCB_FISR2[RFDF]		eQADC Receive FIFO 2 Drain Flag
0x19A0	410	EQADCB_FISR3[NCF]		eQADC command FIFO 3 Non-Coherency Flag
0x19B0	411	EQADCB_FISR3[PF]		eQADC command FIFO 3 Pause Flag
0x19C0	412	EQADCB_FISR3[EOQF]		eQADC command FIFO 3 command queue End of Queue Flag
0x19D0	413	EQADCB_FISR3[CFFF]		eQADC Command FIFO 3 Fill Flag
0x19E0	414	EQADCB_FISR3[RFDF]		eQADC Receive FIFO 3 Drain Flag
0x19F0	415	EQADCB_FISR4[NCF]		eQADC command FIFO 4 Non-Coherency Flag
0x1A00	416	EQADCB_FISR4[PF]		eQADC command FIFO 4 Pause Flag
0x1A10	417	EQADCB_FISR4[EOQF]		eQADC command FIFO 4 command queue End of Queue Flag
0x1A20	418	EQADCB_FISR4[CFFF]		eQADC Command FIFO 4 Fill Flag
0x1A30	419	EQADCB_FISR4[RFDF]		eQADC Receive FIFO 4 Drain Flag
0x1A40	420	EQADCB_FISR5[NCF]		eQADC command FIFO 5 Non-Coherency Flag
0x1A50	421	EQADCB_FISR5[PF]		eQADC command FIFO 5 Pause Flag
0x1A60	422	EQADCB_FISR5[EOQF]		eQADC command FIFO 5 command queue End of Queue Flag
0x1A70	423	EQADCB_FISR5[CFFF]		eQADC Command FIFO 5 Fill Flag
0x1A80	424	EQADCB_FISR5[RFDF]		eQADC Receive FIFO 5 Drain Flag

Table 27-11. INTC: Interrupt Request Sources (continued)

Offset	Vector	Source	Module	Description
0x1A90	425	EDMAB_ERRH[ERR31:ERR0]	eDMA2_B	eDMA channel Error flags 0 - 31
0x1AA0	426	EDMAB_IRQRH[INT1]		eDMA channel Interrupt 0
0x1AB0	427	EDMAB_IRQRH[INT2]		eDMA channel Interrupt 1
0x1AC0	428	EDMAB_IRQRH[INT3]		eDMA channel Interrupt 2
0x1AD0	429	EDMAB_IRQRH[INT4]		eDMA channel Interrupt 3
0x1AE0	430	EDMAB_IRQRH[INT5]		eDMA channel Interrupt 4
0x1AF0	431	EDMAB_IRQRH[INT6]		eDMA channel Interrupt 5
0x1B00	432	EDMAB_IRQRH[INT7]		eDMA channel Interrupt 6
0x1B10	433	EDMAB_IRQRH[INT8]		eDMA channel Interrupt 7
0x1B20	434	EDMAB_IRQRH[INT9]		eDMA channel Interrupt 8
0x1B30	435	EDMAB_IRQRH[INT10]		eDMA channel Interrupt 9
0x1B40	436	EDMAB_IRQRH[INT11]		eDMA channel Interrupt 10
0x1B50	437	EDMAB_IRQRH[INT12]		eDMA channel Interrupt 11
0x1B60	438	EDMAB_IRQRH[INT13]		eDMA channel Interrupt 12
0x1B70	439	EDMAB_IRQRH[INT14]		eDMA channel Interrupt 13
0x1B80	440	EDMAB_IRQRH[INT15]		eDMA channel Interrupt 14
0x1B90	441	EDMAB_IRQRH[INT16]		eDMA channel Interrupt 15
0x1BA0	442	EDMAB_IRQRH[INT17]		eDMA channel Interrupt 16
0x1BB0	443	EDMAB_IRQRH[INT18]		eDMA channel Interrupt 17
0x1BC0	444	EDMAB_IRQRH[INT19]		eDMA channel Interrupt 18
0x1BD0	445	EDMAB_IRQRH[INT20]		eDMA channel Interrupt 19
0x1BE0	446	EDMAB_IRQRH[INT21]		eDMA channel Interrupt 20
0x1BF0	447	EDMAB_IRQRH[INT22]		eDMA channel Interrupt 21
0x1C00	448	EDMAB_IRQRH[INT23]		eDMA channel Interrupt 22
0x1C10	449	EDMAB_IRQRH[INT24]		eDMA channel Interrupt 23
0x1C20	450	EDMAB_IRQRH[INT25]		eDMA channel Interrupt 24
0x1C30	451	EDMAB_IRQRH[INT26]		eDMA channel Interrupt 25
0x1C40	452	EDMAB_IRQRH[INT27]		eDMA channel Interrupt 26
0x1C50	453	EDMAB_IRQRH[INT28]		eDMA channel Interrupt 27
0x1C60	454	EDMAB_IRQRH[INT29]		eDMA channel Interrupt 28
0x1C70	455	EDMAB_IRQRH[INT30]		eDMA channel Interrupt 29
0x1C80	456	EDMAB_IRQRH[INT31]		eDMA channel Interrupt 30
0x1C90	457	EDMAB_IRQRH[INT32]		eDMA channel Interrupt 31
0x1CA0	458	Unassigned		Unassigned

Table 27-11. INTC: Interrupt Request Sources (continued)

Offset	Vector	Source	Module	Description
0x1CB0	459	EMIOS_GFR[F24]	eMIOS_A	eMIOS_A channel 24 Flag
0x1CC0	460	EMIOS_GFR[F25]		eMIOS_A channel 25 Flag
0x1CD0	461	EMIOS_GFR[F26]		eMIOS_A channel 26 Flag
0x1CE0	462	EMIOS_GFR[F27]		eMIOS_A channel 27 Flag
0x1CF0	463	EMIOS_GFR[F28]		eMIOS_A channel 28 Flag
0x1D00	464	EMIOS_GFR[F29]		eMIOS_A channel 29 Flag
0x1D10	465	EMIOS_GFR[F30]		eMIOS_A channel 30 Flag
0x1D20	466	EMIOS_GFR[F31]		eMIOS_A channel 31 Flag
0x1D30	467	DECFILTER_MSR_C[IDF]	DEC_C	Decimation C Input (Fill)
0x1D40	468	DECFILTER_MSR_C[ODF‖SDF]		Decimation C Output/Integ (Drain/Integ)
0x1D50	469	DECFILTER C ERRORS		Decimation C Error
0x1D60	470	DECFILTER_MSR_D[IDF]	DEC_D	Decimation D Input (Fill)
0x1D70	471	DECFILTER_MSR_D[ODF‖SDF]		Decimation D Output/Integ (Drain/Integ)
0x1D80	472	DECFILTER D ERRORS		Decimation D Error
0x1D90	473	ESCIC_SR[TDRE] ESCIC_SR[TC] ESCIC_SR[RDRF] ESCIC_SR[IDLE] ESCIC_SR[OR] ESCIC_SR[NF] ESCIC_SR[FE] ESCIC_SR[PF] ESCIC_SR[BERR] ESCIC_SR[RXRDY] ESCIC_SR[TXRDY] ESCIC_SR[LWAKE] ESCIC_SR[STO] ESCIC_SR[PBERR] ESCIC_SR[CERR] ESCIC_SR[CKERR] ESCIC_SR[FRC] ESCIC_SR[OVFL]	eSCI_C	Combined Interrupt Requests of ESCI Module C: Transmit Data Register Empty, Transmit Complete, Receive Data Register Full, Idle line, Overrun, Noise Flag, Framing Error Flag, and Parity Error Flag interrupt requests, SCI Status Register 2 Bit Error interrupt request, LIN Status Register 1 Receive Data Ready, Transmit Data Ready, Received LIN Wakeup Signal, Slave TimeOut, Physical Bus Error, CRC Error, Checksum Error, Frame Complete interrupts requests, and LIN Status Register 2 Receive Register Overflow
0x1DA0	474	Reserved		Reserved
0x1DB0	475	Reserved		Reserved
0x1DC0	476	DECFILTER_MSR_E[IDF‖ODF‖ERRORS‖SDF]	DEC_E	DecFil E Input/Output/Err/Intg (Fill/Drain/Error/Intg)
0x1DD0	477	DECFILTER_MSR_F[IDF‖ODF‖ERRORS‖SDF]	DEC_F	DecFil F Input/Output/Err/Intg (Fill/Drain/Error/Intg)
0x1DE0	478	DECFILTER_MSR_G[IDF‖ODF‖ERRORS‖SDF]	DEC_G	DecFil G Input/Output/Err/Intg (Fill/Drain/Error/Intg)
0x1DF0	479	DECFILTER_MSR_H[IDF‖ODF‖ERRORS‖SDF]	DEC_H	DecFil H Input/Output/Err/Intg (Fill/Drain/Error/Intg)
0x1E00	480	DECFILTER_MSR_I[IDF‖ODF‖ERRORS‖SDF]	DEC_I	DecFil I Input/Output/Err/Intg (Fill/Drain/Error/Intg)
0x1E10	481	DECFILTER_MSR_J[IDF‖ODF‖ERRORS‖SDF]	DEC_J	DecFil J Input/Output/Err/Intg (Fill/Drain/Error/Intg)

Table 27-11. INTC: Interrupt Request Sources (continued)

Offset	Vector	Source	Module	Description
0x1E20	482	DECFILTER_MSR_K[IDF||ODF||ERRORS||SDF]	DEC_K	DecFil K Input/Output/Err/Intg (Fill/Drain/Error/Intg)
0x1E30	483	DECFILTER_MSR_L[IDF||ODF||ERRORS||SDF]	DEC_L	DecFil L Input/Output/Err/Intg (Fill/Drain/Error/Intg)
0x1E40	484	Reserved		Reserved for DEC_M
0x1E50	485	Reserved		Reserved for DEC_N
0x1E60	486	Reserved		Reserved for DEC_O
0x1E70	487	Reserved		Reserved for DEC_P
0x1E80	488	EDMAB_ERRH[ERR63:ERR32]	eDMA2_B	eDMA channel Error flags 32 - 63
0x1E90	489	EDMAB_IRQRH[INT32:39]		eDMA channel Interrupts 32 - 39
0x1EA0	490	EDMAB_IRQRH[INT40:47]		eDMA channel Interrupts 40 - 47
0x1EB0	491	EDMAB_IRQRH[INT48:55]		eDMA channel Interrupts 48 - 55
0x1EC0	492	EDMAB_IRQRH[INT56:63]		eDMA channel Interrupts 56 - 63
0x1ED0	493	ETPU_CISR_C[CIS25]	**eTPU_C**	eTPU Engine C Channel 24 Interrupt Status
0x1EE0	494	ETPU_CISR_C[CIS26]		eTPU Engine C Channel 25 Interrupt Status
0x1EF0	495	ETPU_CISR_C[CIS27]		eTPU Engine C Channel 26 Interrupt Status
0x1F00	496	ETPU_CISR_C[CIS28]		eTPU Engine C Channel 27 Interrupt Status
0x1F10	497	ETPU_CISR_C[CIS29]		eTPU Engine C Channel 28 Interrupt Status
0x1F20	498	ETPU_CISR_C[CIS30]		eTPU Engine C Channel 29 Interrupt Status
0x1F30	499	ETPU_CISR_C[CIS31]		eTPU Engine C Channel 30 Interrupt Status
0x1F40	500	ETPU_CISR_C[CIS32]		eTPU Engine C Channel 31 Interrupt Status
0x1F50	501	SWTB_IR[TIF]	Watchdog B	Software Watchdog B Interrupt flag
0x1F60	502	SEMA4_CP0INE[0:15]	Semaphore	Core 0 requested semaphore has unlocked
0x1F70	503	SEMA4_CP1INE[0:15]		Core 1 requested semaphore has unlocked
0x1F80	504	Reserved	Semaphore	Reserved
0x1F90	505	Reserved		Reserved
0x1FA0	506	DSPIE_SR[TFUF] DSPIE_SR[RFOF] DSPIE_SR[SPEF] DSPIE_SR[DPEF]	DSPI_E	DSPIE combined overrun & parity error interrupt requests: Transmit FIFO Underflow/Receive FIFO Overflow SPI Parity Error/DSI Parity Error
0x1FB0	507	DSPIE_SR[EOQF]		DSPIE transmit FIFO End of Queue Flag
0x1FC0	508	DSPIE_SR[TFFF]		DSPIE Transmit FIFO Fill Flag
0x1FD0	509	DSPIE_SR[TCF]||DSPIE_SR[DDIF]		DSPIE Transfer Complete/DSI Data Match Flag
0x1FE0	510	DSPIE_SR[RFDF]		DSPIE Receive FIFO Drain Flag
0x1FF0	511	STCU_INTERRUPT[IFLAG]	STCU	STCU Done or Watchdog Timeout Flag

NOTE

The peripheral or software settable interrupt request asserts when the PRI*n* value in the interrupt priority select register (INTC_PSR*n*) is greater than the PRI*n* value in interrupt current priority register (INTC_CPR).

If an asserted peripheral or software settable interrupt request negates before the processor acknowledges its request, the interrupt request can reassert and remain asserted. If this occurs, the processor uses the INTC_PSR*n* value to locate the IRQ vector, and updates the PRI*n* value in the INTC_CPR with the PRI*n* value in INTC_PSR*n*.

Clearing the peripheral interrupt request enable bit for the peripheral initiating the request, or setting the IRQ mask bit has the same consequences as clearing its flag bit. Setting its enable bit or clearing its mask bit while its flag bit is asserted has the same effect on the INTC as an interrupt event setting the flag bit.

27.4.1.1 Peripheral Interrupt Requests

An interrupt event in a peripheral’s hardware sets a flag bit that resides in the peripheral. The interrupt request from the peripheral is driven by that flag bit.

The time from when the peripheral starts to drive its peripheral interrupt request to the INTC to the time that the INTC starts to drive the interrupt request to the processor is three clocks.

Interrupt requests from devices external to the MPC5676R are classified as peripheral interrupt requests in this reference manual. These type of external peripheral interrupts are handled by the SIU (see Section 27.4.1, “External Interrupt Request Sources”).

27.4.1.2 Software Settable Interrupt Requests

The software set/clear interrupt registers (INTC_SSCIR*x*) support the setting or clearing of software-settable interrupt requests. These registers contain eight independent sets of bits to set and clear a corresponding flag bit by software. With the exception of being set by software, this flag bit behaves the same as a flag bit set within a peripheral. This flag bit generates an interrupt request within the INTC just like a peripheral interrupt request.

An interrupt request is triggered by software by writing a 1 to a SET*n* bit in INTC_SSCIR0–INTC_SSCIR7. This write sets a CLR*n* flag bit that generates an interrupt request. The interrupt request is cleared by writing a 1 to the CLR*n* bit. Specific behavior includes the following:

- Writing a 1 to SET*n* leaves SET*n* unchanged at 0 but sets the flag bit (CLR*n* bit).
- Writing a 0 to SET*n* has no effect.
- Writing a 1 to CLR*n* clears the flag (CLR*n*) bit.
- Writing a 0 to CLR*n* has no effect.
- If a 1 is written to a pair of SET*n* and CLR*n* bits at the same time, the flag (CLR*n*) is set, regardless of whether CLR*n* was asserted before the write.

The time from the write to the SET*n* bit to the time that the INTC starts to drive the interrupt request to the processor is four clocks.

27.4.1.3 Unique Vector for Each Interrupt Request Source

Each peripheral and software settable interrupt request is assigned a hardwired unique 9-bit vector. Software settable interrupts 0–7 are assigned vectors 0–7 respectively. The peripheral interrupt requests are assigned vectors 8 to as high as needed to include all the peripheral interrupt requests.

27.4.2 Priority Management

The asserted interrupt requests are compared to each other based on their PRI*n* and PRC_SEL*n* values set in the INTC priority select registers (INTC_PSR0 –INTC_PSR511). The result of that comparison is compared to PRI in the associated current priority register (INTC_CPR_PRC0 or INTC_CPR_PRC1). The results of those comparisons are used to manage the priority of the ISR being executed by the associated processor. The associated LIFO also assists in managing that priority.

27.4.2.1 Current Priority and Preemption

The priority arbitrator, selector, encoder, and comparator submodules shown in Figure 27-1 are used to compare the priority of the asserted interrupt requests to the current priority. If the priority of any asserted peripheral or software settable interrupt request is higher than the current priority for a given processor, then the interrupt request to the processor is asserted. Also, a unique vector for the preempting peripheral or software settable interrupt request is generated for the associated INTC interrupt acknowledge register (INTC_IACKR_PRC0 or INTC_IACKR_PRC1) and, if in hardware vector mode, for the interrupt vector provided to the processor.

27.4.2.1.1 Priority Arbitrator Submodule

The priority arbitrator submodule for each processor compares all the priorities of all of the asserted interrupt requests assigned to that processor, both peripheral and software settable. The output of the priority arbitrator submodule is the highest of those priorities assigned to a given processor. Also, any interrupt requests which have this highest priority are output as asserted interrupt requests to the associated request selector submodule.

27.4.2.1.2 Request Selector Submodule

If only one interrupt request from the associated priority arbitrator submodule is asserted, then it is passed as asserted to the associated vector encoder submodule. If multiple interrupt requests from the associated priority arbitrator submodule are asserted, only the one with the lowest vector passes as asserted to the associated vector encoder submodule. The lower vector is chosen regardless of the time order of the assertions of the peripheral or software settable interrupt requests.

27.4.2.1.3 Vector Encoder Submodule

The vector encoder submodule generates the unique 9-bit vector for the asserted interrupt request from the request selector submodule for the associated processor.

27.4.2.1.4 Priority Comparator Submodule

The priority comparator submodule compares the highest priority output from the associated priority arbitrator submodule with PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1. If the priority comparator submodule detects that the highest priority is higher than the current priority, then it asserts the interrupt request to the associated processor. This interrupt request to the processor asserts whether this highest priority is raised above the value of PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1, or the PRI value in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 is lowered below this highest priority. This highest priority becomes the new priority which is written to PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 when the interrupt request to the processor is acknowledged. Interrupt requests whose PRI*n* in INTC_PSR*n*_*n* are 0 will not cause a preemption because their PRI*n* will not be higher than PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1.

Another function of the priority comparator subblock is to signal an update of the INTC_IACKR_PRC0 and INTC_IACKR_PRC1 with the vector number of the first interrupt that arrives that has a priority higher than the current priority. Once the vector number and priority are captured, they cannot be superseded by a higher priority interrupt until an update of the INTC_CPR_PRC0 or INTC_CPR_PRC1 occurs.

27.4.2.2 Last-In First-Out (LIFO)

The LIFO stores the preempted PRI values from the associated INTC_CPR_PRC0 or INTC_CPR_PRC1. Therefore, because these priorities are stacked within the INTC, if interrupts need to be enabled during the ISR, at the beginning of the interrupt exception handler the PRI value in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 does not need to be loaded from the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 and stored onto the context stack. Likewise, at the end of the interrupt exception handler, the priority does not need to be loaded from the context stack and stored into the associated INTC_CPR_PRC0 or INTC_CPR_PRC1.

The PRI value in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 is pushed onto the LIFO when the associated INTC_IACKR_PRC0 or INTC_IACKR_PRC1 is read in software vector mode or when the interrupt acknowledge signal from the associated processor is asserted in hardware vector mode. The priority is popped into PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 when the associated INTC_EOIR_PRC0 or INTC_EOIR_PRC1 is written.

Although the INTC supports 16 priorities, an ISR executing with PRI in the INTC_CPR_PRC0 or INTC_CPR_PRC1 equal to 15 will not be preempted. Therefore, the LIFO supports the stacking of 15 priorities. However, the LIFO is only 14 entries deep. An entry for a priority of 0 is not needed because of how pushing onto a full LIFO and popping an empty LIFO are treated. If the LIFO is pushed 15 or more times than it is popped, the first priorities pushed are overwritten. A priority of 0 would be an overwritten priority. However, the LIFO will pop 0s if it is popped more times than pushed. Therefore, although a priority of 0 was overwritten, it is regenerated with the popping of an empty LIFO.

The LIFO is not memory mapped.

27.4.3 Details on Handshaking with Processor

27.4.3.1 Software Vector Mode Handshaking

27.4.3.1.1 Acknowledging Interrupt Request to Processor

A timing diagram of the interrupt request and acknowledge handshaking in software vector mode, along with the handshake near the end of the interrupt exception handler, is shown in Figure 27-18. The INTC examines the peripheral and software settable interrupt requests. When it finds an asserted peripheral or software settable interrupt request with a higher priority than PRI in the associated INTC current priority register (INTC_CPR_PRC0 or INTC_CPR_PRC1), it asserts the interrupt request to the associated processor. The INTVEC field in the associated INTC current priority register (INTC_IACKR_PRC0 or INTC_IACKR_PRC1) is updated with the preempting interrupt request’s vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the processor is asserted. The handshaking process is described in Section 27.1.3.1, “Software Vector Mode.”

27.4.3.1.2 End of Interrupt Exception Handler

Before the interrupt exception handling completes, INTC end-of-interrupt register (INTC_EOIR_PRC0 or INTC_EOIR_PRC1) must be written. When written, the associated LIFO is popped so the preempted priority is restored into PRI of the associated INTC_CPR_PRC0 or INTC_CPR_PRC1. Before it is written, the peripheral or software settable flag bit must be cleared so that the peripheral or software settable interrupt request is negated.

NOTE

To ensure proper operation across all Power Architecture MCUs, execute an **mbar** or **msync** instruction between the access to clear the flag bit and the write to the INTC_EOIR_PRC*n*.

When returning from the preemption, the INTC does not search for the peripheral or software settable interrupt request whose ISR was preempted. Depending on how much the ISR progressed, that interrupt request may no longer be asserted. When PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1 is lowered to the priority of the preempted ISR, the interrupt request for the preempted ISR or other asserted peripherals or software settable interrupt requests at or below that priority will not cause a preemption. Instead, after the restoration of the preempted context, the processor returns to the next instruction address it was about to execute before it was preempted. This next instruction is part of the preempted ISR or the interrupt exception handler’s prolog or epilog.

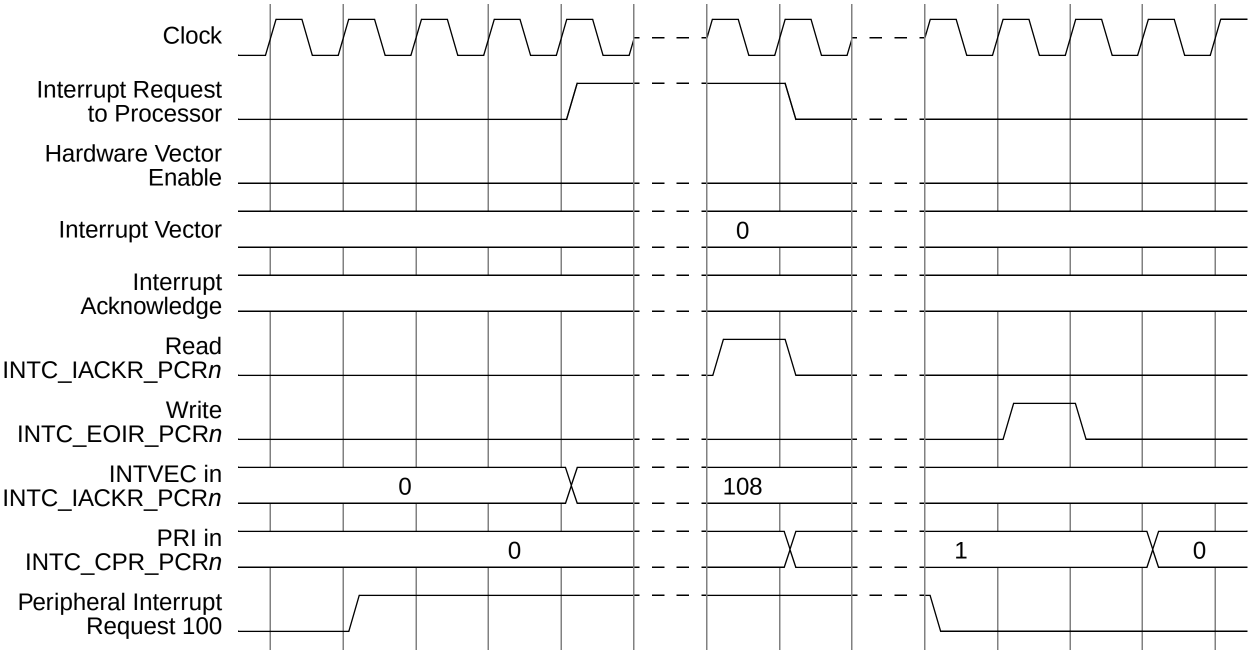


Figure 27-18. Software Vector Mode Handshaking Timing Diagram

27.4.3.2 Hardware Vector Mode Handshaking

A timing diagram of the interrupt request and acknowledge handshaking in hardware vector mode and handshaking near the end of the interrupt exception handler is shown in Figure 27-19. As in software vector mode, the INTC examines the peripheral and software settable interrupt requests and, when it finds one asserted with a higher priority than PRI in the associated INTC_CPR_PRC0 or INTC_CPR_PRC1, it asserts the interrupt request to the associated processor. The INTVEC field in the associated INTC_IACKR_PRC0 or INTC_IACKR_PRC1 is updated with the preempting peripheral or software settable interrupt request's vector when the interrupt request to the processor is asserted. The INTVEC field retains that value until the next time the interrupt request to the associated processor is asserted. In addition, the value of the interrupt vector to the associated processor also matches the value of the INTVEC field in the associated INTC_IACKR_PRC0 or INTC_IACKR_PRC1. The rest of the handshaking process is described in Section 27.1.3.2, "Hardware Vector Mode."

The handshaking near the end of the interrupt exception handler, that is written to the associated INTC_EOIR_PRC0 or INTC_EOIR_PRC1, is the same as in software vector mode (see Section 27.4.3.1.2, "End of Interrupt Exception Handler").

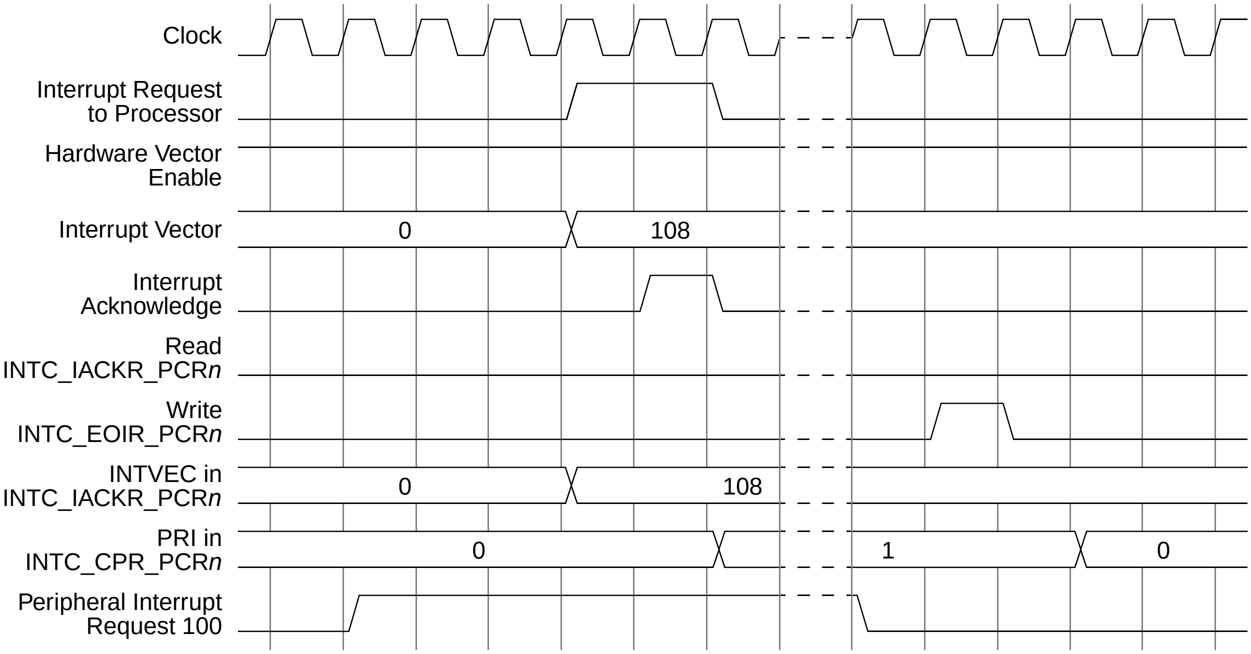


Figure 27-19. Hardware Vector Mode Handshaking Timing Diagram

27.5 Initialization/Application Information

27.5.1 Initialization Flow

After exiting reset, all of the PRIn and PRC_SELn fields in the INTC priority select registers (INTC_PSR0–INTC_PSR511) are cleared (set to 0), and PRI in both INTC_CPR_PRC0 and INTC_CPR_PRC1 is set to 0xF (0b1111). These reset values prevent the INTC from asserting interrupt requests to the processors. The enable or mask bits in the peripherals are reset such that the peripheral interrupt requests are negated.

An initialization sequence for allowing the peripheral and software settable interrupt requests to cause an interrupt request to the processor is:

```
interrupt_request_initialization:
configure VTES_PRC0,VTES_PRC1,HVEN_PRC0 and HVEN_PRC1 in INTC_MCR
configure VTBA_PRCn in INTC_IACKR_PRCn
raise the PRIn fields and set the PRC_SELx fields to the desired processor in INTC_PSRn_n
set the enable bits or clear the mask bits for the peripheral interrupt requests
lower PRI in INTC_CPR_PRCn to zero
enable processor(s) recognition of interrupts
```

27.5.2 Interrupt Exception Handler

These example interrupt exception handlers use Power Architecture assembly code.

27.5.2.1 Software Vector Mode

```
interrupt_exception_handler:
code to create stack frame, save working register, and save SRR0 and SRR1

lis      r3,INTC_IACKR_PRCn@ha      # form adjusted upper half of INTC_IACKR_PRCn address
lwz      r3,INTC_IACKR_PRCn@l(r3)   # load INTC_IACKR_PRCn, which clears request to processor
lwz      r3,0x0(r3)                 # load address of ISR from vector table
wrteei   1                          # enable processor recognition of interrupts

code to save rest of context required by e500 EABI

mtlr     r3                         # move address of ISR into link register
blrl                                # branch to ISR; link register updated with epilog
                                    # address

epilog:
code to restore most of context required by e500 EABI

# Popping the LIFO after the restoration of most of the context and the disabling of processor
# recognition of interrupts eases the calculation of the maximum stack depth at the cost of
# postponing the servicing of the next interrupt request.
mbar                                # ensure store to clear flag bit has completed
lis      r3,INTC_EOIR_PRCn@ha       # form adjusted upper half of INTC_EOIR address
li       r4,0x0                     # form 0 to write to INTC_EOIR_PRCn
wrteei   0                          # disable processor recognition of interrupts
stw      r4,INTC_EOIR_PRCn@l(r3)    # store to INTC_EOIR_PRCn, informing INTC to lower priority

code to restore SRR0 and SRR1, restore working registers, and delete stack frame

rfi

vector_table_base_address:
address of ISR for interrupt with vector 0
address of ISR for interrupt with vector 1
         .
         .
         .
address of ISR for interrupt with vector 510
address of ISR for interrupt with vector 511

ISRx:
code to service the interrupt event
code to clear flag bit which drives interrupt request to INTC

blr                                 # return to epilog
```

27.5.2.2 Hardware Vector Mode

This interrupt exception handler is useful with processor and system bus implementations which support a hardware vector. This example assumes that each `interrupt_exception_handlerx` only has space for four instructions, and therefore a branch to `interrupt_exception_handler_continuedx` is needed.

```
interrupt_exception_handlerx:
b        interrupt_exception_handler_continuedx# 4 instructions available, branch to continue

interrupt_exception_handler_continuedx:
code to create stack frame, save working register, and save SRR0 and SRR1

wrteei   1                          # enable processor recognition of interrupts

code to save rest of context required by e500 EABI

bl       ISRx                       # branch to ISR for interrupt with vector x


epilog:
code to restore most of context required by e500 EABI

# Popping the LIFO after the restoration of most of the context and the disabling of processor
# recognition of interrupts eases the calculation of the maximum stack depth at the cost of
# postponing the servicing of the next interrupt request.
mbar                                # ensure store to clear flag bit has completed
lis      r3,INTC_EOIR_PRCn@ha       # form adjusted upper half of INTC_EOIR_PRCn address
li       r4,0x0                     # form 0 to write to INTC_EOIR_PRCn
wrteei   0                          # disable processor recognition of interrupts
stw      r4,INTC_EOIR_PRCn@l(r3)    # store to INTC_EOIR_PRCn, informing INTC to lower priority

code to restore SRR0 and SRR1, restore working registers, and delete stack frame

rfi


ISRx:
code to service the interrupt event
code to clear flag bit which drives interrupt request to INTC

blr                                 # branch to epilog
```

27.5.3 ISR, RTOS, and Task Hierarchy

The RTOS and all of the tasks under its control typically execute with PRI in INTC current priority register (INTC_CPR_PRC0 or INTC_CPR_PRC1) having a value of 0. The RTOS executes the tasks according to whatever priority scheme it may have, but that priority scheme is independent and has a lower priority of execution than the priority scheme of the INTC. In other words, the ISRs execute above INTC_CPR_PRC*n* priority 0 and outside the control of the RTOS, the RTOS executes at INTC_CPR_PRC*n* priority 0, and while the tasks execute at different priorities under the control of the RTOS, they also execute at INTC_CPR_PRC*n* priority 0.

If a task shares a resource with an ISR and the PCP is being used to manage that shared resource, then the task's priority can be elevated in the INTC_CPR_PRC*n* while the shared resource is being accessed.

An ISR whose PRI*n* in INTC priority select registers (INTC_PSR0–INTC_PSR511) has a value of 0 does not cause an interrupt request to the selected processor, even if its peripheral or software settable interrupt request is asserted. For a peripheral interrupt request, not setting its enable bit or disabling the mask bit causes it to remain negated, which consequently also does not cause an interrupt request to the processor. Since the ISRs are outside the control of the RTOS, this ISR does not run unless called by another ISR or the interrupt exception handler, perhaps after executing another ISR.

27.5.4 Order of Execution

An ISR with a higher priority can preempt an ISR with a lower priority, regardless of the unique vectors associated with each of their peripheral or software settable interrupt requests. However, if multiple peripheral or software settable interrupt requests are asserted, more than one has the highest priority, and that priority is high enough to cause preemption, the INTC selects the one with the lowest unique vector regardless of the order in time that they asserted. However, the ability to meet deadlines with this scheduling scheme is no less than if the ISRs execute in the time order that their peripheral or software settable interrupt requests asserted.

The example in Table 27-12 shows the order of execution of both ISRs with different priorities and the same priority.

Table 27-12. Order of ISR Execution Example

Step#	Step Description	Code Executing at End of Step						PRI in INTC_CPR at End of Step
		RTOS	ISR108[1]	ISR208	ISR308	ISR408	Interrupt Exception Handler	
1	RTOS at priority 0 is executing.	X						0
2	Peripheral interrupt request 100 at priority 1 asserts. Interrupt taken.		X					1
3	Peripheral interrupt request 400 at priority 4 is asserts. Interrupt taken.					X		4
4	Peripheral interrupt request 300 at priority 3 is asserts.					X		4
5	Peripheral interrupt request 200 at priority 3 is asserts.					X		4
6	ISR408 completes. Interrupt exception handler writes to INTC_EOIR_PRC*n*.						X	1
7	Interrupt taken. ISR208 starts to execute, even though peripheral interrupt request 300 asserted first.			X				3
8	ISR208 completes. Interrupt exception handler writes to INTC_EOIR_PRC*n*.						X	1
9	Interrupt taken. ISR308 starts to execute.				X			3

Table 27-12. Order of ISR Execution Example (continued)

Step#	Step Description	Code Executing at End of Step						PRI in INTC_CPR at End of Step
		RTOS	ISR108[1]	ISR208	ISR308	ISR408	Interrupt Exception Handler	
10	ISR308 completes. Interrupt exception handler writes to INTC_EOIR_PRC*n*.						X	1
11	ISR108 completes. Interrupt exception handler writes to INTC_EOIR_PRC*n*.						X	0
12	RTOS continues execution.	X						0

[1] ISR108 executes for peripheral interrupt request 100 because the first eight ISRs are for software settable interrupt requests.

27.5.5 Priority Ceiling Protocol

27.5.5.1 Elevating Priority

The PRI field in INTC current priority register (INTC_CPR_PRC0 or INTC_CPR_PRC1) is elevated in the OSEK PCP to the ceiling of all of the priorities of the ISRs that share a resource. This protocol allows coherent accesses of the ISRs to that shared resource.

For example, ISR1 has a priority of 1, ISR2 has a priority of 2, and ISR3 has a priority of 3. They all share the same resource. Before ISR1 or ISR2 can access that resource, they must raise the PRI value in INTC_CPR_PRC*n* to 3, the ceiling of all of the ISR priorities. After they release the resource, the PRI value in INTC_CPR_PRC*n* can be lowered. If they do not raise their priority, ISR2 can preempt ISR1, and ISR3 can preempt ISR1 or ISR2, possibly corrupting the shared resource. Another possible failure mechanism is deadlock if the higher priority ISR needs the lower priority ISR to release the resource before it can continue, but the lower priority ISR cannot release the resource until the higher priority ISR completes and execution returns to the lower priority ISR.

Using the PCP instead of disabling processor recognition of all interrupts eliminates the time when accessing a shared resource that all higher priority interrupts are blocked. For example, while ISR3 cannot preempt ISR1 while it is accessing the shared resource, all of the ISRs with a priority higher than 3 can preempt ISR1.

27.5.5.2 Ensuring Coherency

27.5.5.2.1 Interrupt with Blocked Priority

In systems where resources are shared among two or more tasks, ensuring resource coherency is a fundamental requirement.

As an example of non-coherent accesses to a shared resource, consider the following scenario: ISR1 and ISR2 both share a resource. ISR1 has a lower priority than ISR2. ISR1 is executing, and it writes to the INTC_CPR_PRC0. The instruction following this store is a store to a value in a shared coherent data block.

Either just before or at the same time as the first store, the INTC asserts the interrupt request to the processor because the peripheral interrupt request for ISR2 has asserted.

As the processor is responding to the interrupt request from the INTC, and as it is aborting transactions and flushing its pipeline, it is possible that both of these stores will be executed. ISR2 thereby thinks that it can access the data block coherently, but the data block has been corrupted.

When the Current Priority Register’s (CPR’s) PRI field in the INTC is updated by software, external interrupts from the INTC to the Core can be disabled so that an external interrupt will never be taken in the window of time when a system resource (PRI field) is being changed. Disabling interrupts and re-enabling them after the PRI elevation, as well as providing additional core clocks of delay after interrupts have been enabled, provides enough delay for the ISR that has elevated its PRI to be interrupted by a pending ISR before the ISR elevating PRI accesses a shared memory block.

To ensure coherent access to the shared data block, modifications to PRI in INTC_CPR_PRCx can be made by services managing shared resources with the following code sequence:

- disable processor recognition of interrupts
- PRI modification (guarded, cache inhibited)
- enable processor recognition of interrupts
- delay 5 core clocks
- coherent data access

The delay of 5 clocks minimum can be achieved by the following series of instructions:

```
or r1, r1, r1
or r1, r1, r1
or r1, r1, r1
or r1, r1, r1
or r1, r1, r1
```

The or r1, r1, r1 is not dual issued because the destination of one “or” instruction is the source of the next, hence they can not be executed in parallel.

27.5.5.2.2 Raised Priority Preserved

Figure 27-20 shows the detailed timing diagram for a scenario where a priority that is raised in one ISR is preserved after execution of a pre-empting ISR. Table 27-13 explains the events. The example is for software vector mode, but except for the method of retrieving the vector and acknowledging the interrupt request to the processor, hardware vector mode is identical.

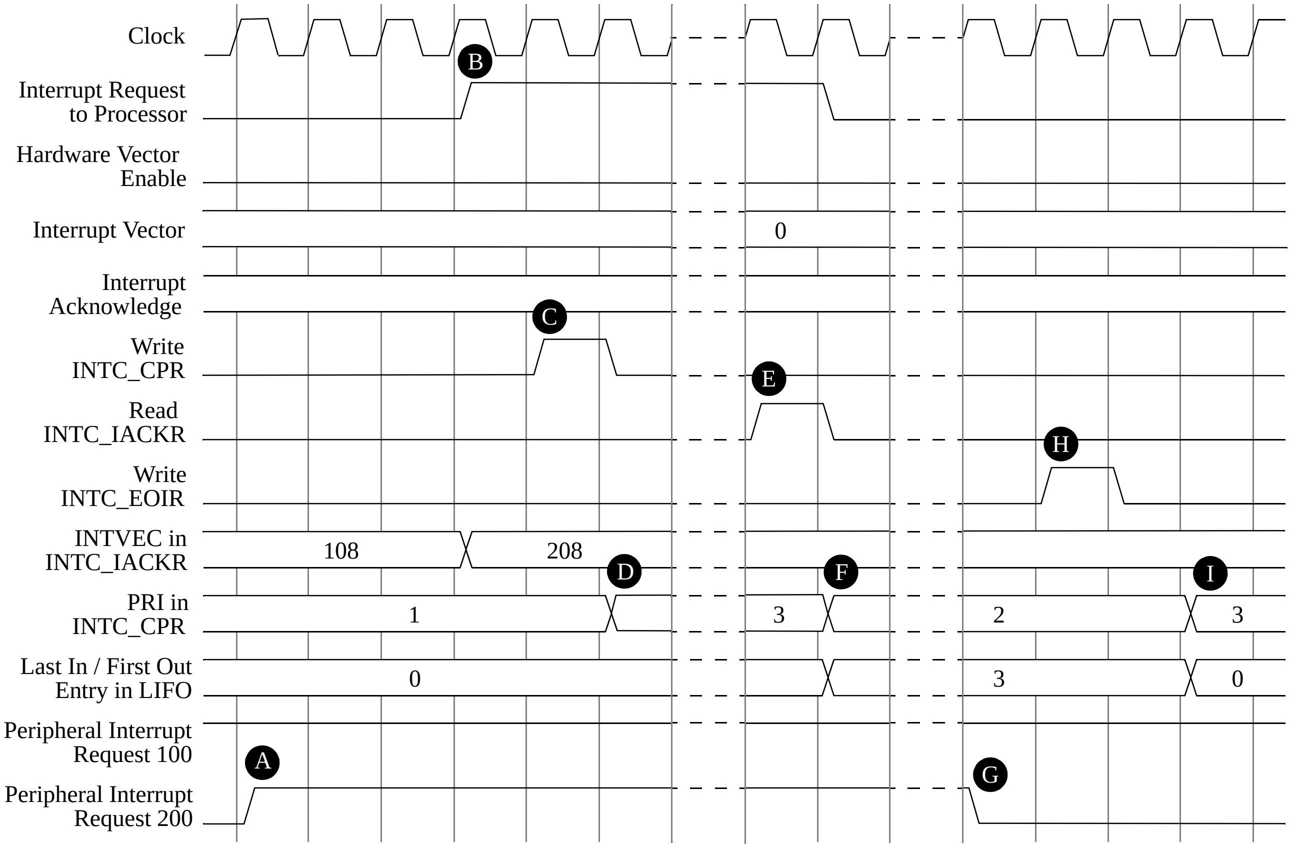


Figure 27-20. Raised Priority Preserved Timing Diagram

Table 27-13. Raised Priority Preserved Events

Event	Description
A	Peripheral interrupt request 200 asserts during execution of ISR108 running at priority 1.
B	Interrupt request to processor asserts. INTVEC in INTC_IACKR updates with vector for that peripheral interrupt request.
C	ISR108 writes to INTC_CPR to raise priority to 3 before accessing shared coherent data block.
D	PRI in INTC_CPR now at 3, reflecting the write. This write, just before accessing data block, is the last instruction the processor executes before being interrupted.
E	Interrupt exception handler prolog acknowledges interrupt by reading INTC_IACKR.
F	PRI of 3 pushed onto LIFO. PRI in INTC_CPR updates to 2, the priority of ISR208.
G	ISR208 clears its flag bit, deasserting its peripheral interrupt request.
H	Interrupt exception handler epilog writes to INTC_EOIR.
I	LIFO pops 3, restoring the raised priority onto PRI in INTC_CPR. Next value to pop from LIFO is the priority from before peripheral interrupt request 100 interrupted. ISR108 now can access data block coherently after interrupt exception handler executes rfi instruction.

27.5.6 Selecting Priorities According to Request Rates and Deadlines

The selection of the priorities for the ISRs can be made using rate monotonic scheduling (RMS) or a superset of it, deadline monotonic scheduling (DMS). In RMS, the ISRs which have higher request rates have higher priorities. In DMS, if the deadline is before the next time the ISR is requested, then the ISR is assigned a priority according to the time from the request for the ISR to the deadline, not from the time of the request for the ISR to the next request for it.

For example, ISR1 executes every 100 µs, ISR2 executes every 200 µs, and ISR3 executes every 300 µs. ISR1 has a higher priority than ISR2 which has a higher priority than ISR3; however, if ISR3 has a deadline of 150 µs, then it has a higher priority than ISR2.

The INTC has 16 priorities, which can be significantly fewer than the number of ISRs. In this case, group the ISRs with other ISRs that have similar deadlines. For example, when a priority is allocated for every time the request rate doubles, ISRs with request rates around 1 ms would share a priority, ISRs with request rates around 500 µs would share a priority, ISRs with request rates around 250 µs would share a priority, etc. With this approach, a range of ISR request rates of 2^{16} could be covered, regardless of the number of ISRs.

Reducing the number of priorities reduces the processor’s ability to meet its deadlines. However, it also allows easier management of ISRs with similar deadlines that share a resource. They do not need to use the PCP to access the shared resource.

27.5.7 Software Settable Interrupt Requests

The software settable interrupt requests can be used in two ways. They can be used to schedule a lower priority portion of an ISR and for processors to interrupt other processors in a multiple processor system.

27.5.7.1 Scheduling a Lower Priority Portion of an ISR

A portion of an ISR needs to be executed at the PRI*n* value in INTC priority select registers (INTC_PSR0–INTC_PSR511), which becomes the PRI value in INTC current priority register (INTC_CPR_PRC0 or INTC_CPR_PRC1) with the interrupt acknowledge. The ISR, however, can have a portion of it which does not need to be executed at this higher priority. Therefore, executing this later portion that does not need to be executed at this higher priority can prevent the execution of ISRs that do not have a higher priority than the earlier portion of the ISR but do have a higher priority than what the later portion of the ISR needs. This preemptive scheduling inefficiency reduces the processor’s ability to meet its deadlines.

One option is for the ISR to complete the earlier higher priority portion, but then schedule through the RTOS a task to execute the later lower priority portion. However, some RTOSs can require a large amount of time for an ISR to schedule a task. Therefore, a second option is for the ISR, after completing the higher priority portion, to set a SET*n* bit in INTC software set/clear interrupt registers (INTC_SSCIR0–INTC_SSCIR7). Writing a 1 to SET*n* causes a software settable interrupt request. This software settable interrupt request usually has a lower PRI*n* value in the INTC_PSR*n*, and therefore does not cause preemptive scheduling inefficiencies.

After generating a software settable interrupt request, the higher priority ISR completes. The lower priority ISR is scheduled according to its priority. Execution of the higher priority ISR is not resumed after the completion of the lower priority ISR.

27.5.7.2 Scheduling an ISR on Another Processor

Since the SET*n* bits in the INTC_SSCIR*n* are memory mapped, processors in multiple-processor systems can schedule ISRs on the other processors. One application is that one processor wants to command another processor to perform a piece of work and the initiating processor does not need to use the results of that work. If the initiating processor is concerned that the processor executing the software settable ISR has not completed the work before asking it to again execute the ISR, it can check if the corresponding CLR*n* bit in INTC_SSCIR*n* is asserted before again writing a 1 to the SET*n* bit.

Another application is the sharing of a block of data. For example, a first processor has completed accessing a block of data and wants a second processor to then access it. Furthermore, after the second processor has completed accessing the block of data, the first processor again wants to access it. The accesses to the block of data must be done coherently. The procedure is that the first processor writes a 1 to a SET*n* bit on the second processor. The second processor, after accessing the block of data, clears the corresponding CLR*n* bit and then writes 1 to a SET*n* bit on the first processor, informing it that it now can access the block of data.

27.5.8 Lowering Priority Within an ISR

In implementations without the software-settable interrupt requests in the INTC software set/clear interrupt registers (INTC_SSCIR0–INTC_SSCIR7), one way — besides scheduling a task through an RTOS — to prevent preemptive scheduling inefficiencies with an ISR whose work spans multiple priorities is to lower the current priority (see Section 27.5.7.1, “Scheduling a Lower Priority Portion of an ISR”). However, the INTC has a LIFO whose depth is determined by the number of priorities.

NOTE

> Lowering the PRI value in either INTC_CPR_PRC0 or INTC_CPR_PRC1 within an ISR to below the ISR’s corresponding PRI value in INTC_PSR0–INTC_PSR511 allows more preemptions than the LIFO depth can support.

Therefore, through its use of the LIFO, the INTC does not support lowering the current priority within an ISR as a way to avoid preemptive scheduling inefficiencies.

27.5.9 Negating an Interrupt Request Outside of its ISR

27.5.9.1 Negating an Interrupt Request as a Side Effect of an ISR

Some peripherals have flag bits that can be cleared as a side effect of servicing a peripheral interrupt request. For example, reading a specific register can clear the flag bits and consequentially their corresponding interrupt requests too. This clearing as a side effect of servicing a peripheral interrupt request can cause the negation of other peripheral interrupt requests besides the peripheral interrupt request whose ISR presently is executing. This negating of a peripheral interrupt request outside of its ISR can be a desired effect.

27.5.9.2 Negating Multiple Interrupt Requests in One ISR

An ISR can clear other flag bits besides its own. One reason that an ISR clears multiple flag bits is because it serviced those flag bits, and therefore the ISRs for these flag bits do not need to be executed.

27.5.9.3 Proper Setting of Interrupt Request Priority

Whether an interrupt request negates outside its own ISR due to the side effect of an ISR execution or the intentional clearing a flag bit, the priorities of the peripheral or software settable interrupt requests for these other flag bits must be selected properly. Their PRI*n* values in INTC priority select registers (INTC_PSR0–INTC_PSR511) must be selected to be at or lower than the priority of the ISR that cleared their flag bits. Otherwise, those flag bits can cause the interrupt request to the processor to assert. Furthermore, the clearing of these other flag bits also has the same timing relationship to the writing to INTC end-of-interrupt register (INTC_EOIR_PRC*n*) as the clearing of the flag bit that caused the present ISR to be executed (see Section 27.4.3.1.2, “End of Interrupt Exception Handler”).

A flag bit whose enable bit or mask bit is negating its peripheral interrupt request can be cleared at any time, regardless of the peripheral interrupt request’s PRI*n* value in INTC_PSR*x*.

27.5.10 Examining LIFO Contents

Normally you do not need to know the contents of the LIFO, or even how deep the LIFO is nested. Although the LIFO contents are not memory mapped, you can read the contents by popping the LIFO and reading the PRI field in the INTC current priority register (INTC_CPR_PRC0 or INTC_CPR_PRC1). Disabling processor recognition of interrupts while examining the LIFO contents provides a coherent view of the preempted priorities.

The code sequence is:

```
pop_lifo:
store to INTC_EOIR_PRCn
load INTC_CPR_PRCn, examine PRI, and store onto stack
if PRI is not zero or value when interrupts were enabled, branch to pop_lifo
```

When you are finished examining the LIFO contents, you can restore it in software vector mode using the following code sequence:

```
push_lifo:
load stacked PRI value and store to INTC_CPR_PRCn
load INTC_IACKR_PRCn
if stacked PRI values are not depleted, branch to push_lifo
```

NOTE

Reading the INTC_IACKR_PRC*n* acknowledges the interrupt request to the processor and updates the INTC_CPR_PRC*n*[PRI] with the priority of the preempting interrupt request. If the processor recognition of interrupts is disabled during the LIFO restoration, interrupt requests to the processor can go undetected. However, since the peripheral or software settable interrupt requests are not cleared, the peripheral interrupt request to the processor re-asserts when INTC_CPR_PRC*n*[PRI] is lower than the priorities of those peripheral or software settable interrupt requests.

Chapter 28
Memory Protection Unit (MPU)

28.1 Introduction

The memory protection unit (MPU) provides hardware access control for all memory references generated in a device. Using pre-programmed region descriptors that define memory spaces and their associated access rights, the MPU concurrently monitors all system bus transactions and evaluates the appropriateness of each transfer. Memory references with sufficient access control rights are allowed to complete, but references that are not mapped to any region descriptor or have insufficient rights are terminated with a protection error response.

The MPU implements a set of program-visible region descriptors that monitor all system bus addresses. The result is a hardware structure with a two-dimensional connection matrix, where the region descriptors represent one dimension and the individual system bus addresses and attributes are the second dimension.

28.1.1 Block Diagram

A simplified block diagram illustrates how the MPU block is connected to the four XBAR MPU ports and the shared port splitter (see Figure 28-1 and Table 28-1).

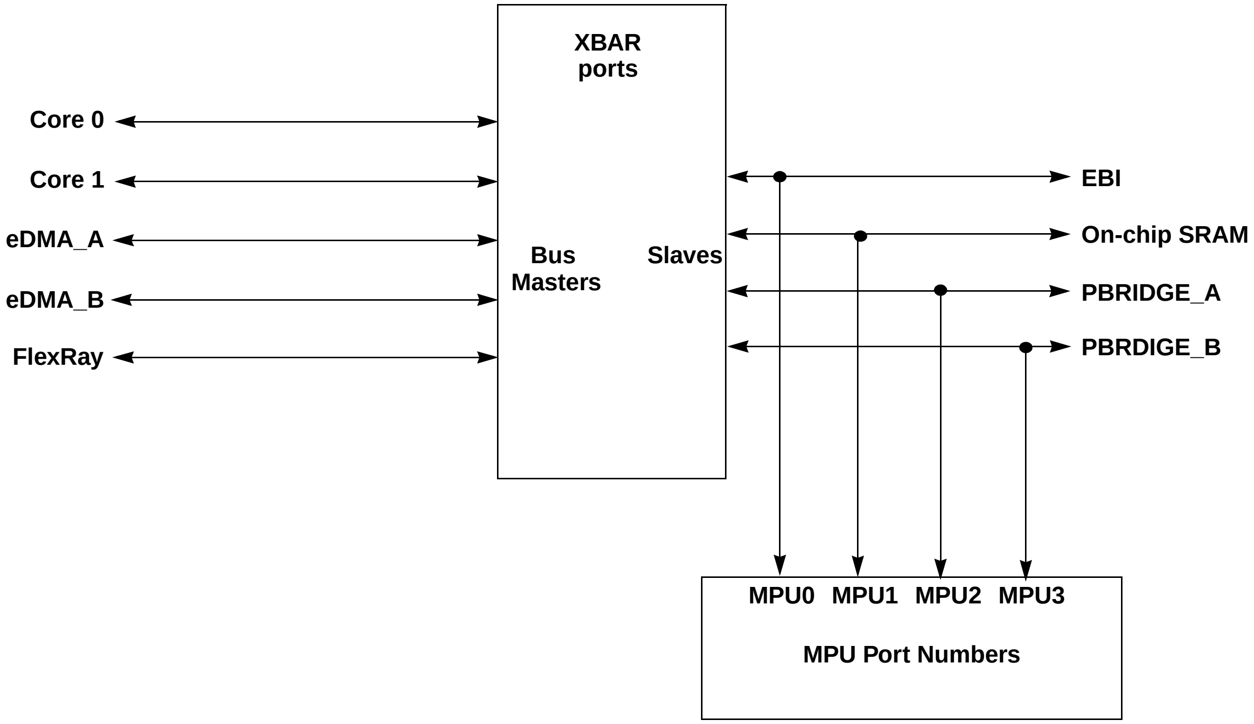


Figure 28-1. MPU Connections to XBAR Slaves

Table 28-1 enumerates the MPU Ports that are attached to slave modules. The Master IDs of all bus master modules are also shown, as their values are required to configure certain MPU registers described in this chapter.

Table 28-1. XBAR Switch Ports

Module	Bus Master ID	MPU Port
Core 0	0	—
Core 1	1	—
eDMA_A	4	—
eDMA_B	5	—
FlexRay	6	—
EBI (development bus)	—	0
On-chip SRAM	—	1
Peripheral bridge A (PBRIDGE_A)	—	2
Peripheral bridge B (PBRIDGE_B)	—	3

28.1.2 Features

The MPU has these major features:

- Support for 16 memory region descriptors, each 128 bits in size
    - Specification of start and end addresses provide granularity for region sizes from 32 bytes to 4 GB
    - MPU is invalid at reset, thus no access restrictions are enforced
    - Two types of access control definitions: processor core bus master (e200z7) supports the traditional {read, write, execute} permissions with independent definitions for supervisor and user mode accesses; the remaining non-core bus masters (eDMA_A, eDMA_B, FlexRay) support {read, write} attributes
    - Automatic hardware maintenance of the region descriptor valid bit removes issues associated with maintaining a coherent image of the descriptor
    - Alternate memory view of the access control word for each descriptor provides an efficient mechanism to dynamically alter the access rights of a descriptor only
    - For overlapping region descriptors, priority is given to permission granting over access denying as this approach provides more flexibility to system software
- Support for four AHB slave port connections
    - PBRIDGE_A, PBRIDGE_B, EBI (development bus), general purpose SRAM
    - MPU hardware monitors every AHB slave port access using the pre-programmed memory region descriptors
    - An access protection error is detected if a memory reference does not hit in any memory region or the reference is flagged as illegal in all memory regions where it does hit; in the event of an access error, the AHB reference is terminated with an error response and the MPU inhibits the bus cycle being sent to the targeted slave device
    - 64-bit error registers, one for each AHB slave port, capture the last faulting address, attributes, and detail information

28.2 Memory Map and Registers

This section provides a detailed description of all MPU registers.

28.2.1 Module Memory Map

The MPU memory map is shown in Table 28-2. The address of each register is given as an offset to the MPU base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed.

The MPU registers can be referenced using 32-bit (word) accesses only. Attempted references using different access sizes, to undefined (reserved) addresses, or with a non-supported access type (for example, a write to a read-only register or a read of a write-only register) generate an error termination.

Table 28-2. MPU Memory Map

Offset from MPU_BASE (0xFFF1_0000)	Register	Bits	Access	Reset Value	Section/Page
0x0000	MPU_CESR — MPU control/error status register	32	R/W	0x0081_4200	28.2.2.1/28-6
0x0004–0x000F	Reserved				
0x0010	MPU_EAR0 — MPU error address register, MPU port 0	32	R	—[1]	28.2.2.2/28-7
0x0014	MPU_EDR0 — MPU error detail register, MPU port 0	32	R	—[1]	28.2.2.3/28-7
0x0018	MPU_EAR1 — MPU error address register, MPU port 1	32	R	—[1]	28.2.2.2/28-7
0x001C	MPU_EDR1 — MPU error detail register, MPU port 1	32	R	—[1]	28.2.2.3/28-7
0x0020	MPU_EAR2 — MPU error address register, MPU port 2	32	R	—[1]	28.2.2.2/28-7
0x0024	MPU_EDR2 — MPU error detail register, MPU port 2	32	R	—[1]	28.2.2.3/28-7
0x0028	MPU_EAR3— MPU error address register, MPU port 3	32	RO	—[1]	28.2.2.2/28-7
0x002C	MPU_EDR3—MPU error detail register, MPU port 3	32	RO	—[1]	28.2.2.3/28-7
0x0030–0x03FF	Reserved				
0x0400	MPU_RGD0 — MPU region descriptor 0	32	R/W	—[1]	28.2.2.4/28-8
0x0410	MPU_RGD1 — MPU region descriptor 1	32	R/W	—[1]	28.2.2.4/28-8
0x0420	MPU_RGD2 — MPU region descriptor 2	32	R/W	—[1]	28.2.2.4/28-8
0x0430	MPU_RGD3 — MPU region descriptor 3	32	R/W	—[1]	28.2.2.4/28-8
0x0440	MPU_RGD4 — MPU region descriptor 4	32	R/W	—[1]	28.2.2.4/28-8
0x0450	MPU_RGD5 — MPU region descriptor 5	32	R/W	—[1]	28.2.2.4/28-8
0x0460	MPU_RGD6 — MPU region descriptor 6	32	R/W	—[1]	28.2.2.4/28-8
0x0470	MPU_RGD7 — MPU region descriptor 7	32	R/W	—[1]	28.2.2.4/28-8
0x0480	MPU_RGD8 — MPU region descriptor 8	32	R/W	—[1]	28.2.2.4/28-8

Table 28-2. MPU Memory Map (continued)

Offset from MPU_BASE (0xFFF1_0000)	Register	Bits	Access	Reset Value	Section/Page
0x0490	MPU_RGD9 — MPU region descriptor 9	32	R/W	—[1]	28.2.2.4/28-8
0x04A0	MPU_RGD10 — MPU region descriptor 10	32	R/W	—[1]	28.2.2.4/28-8
0x04B0	MPU_RGD11 — MPU region descriptor 11	32	R/W	—[1]	28.2.2.4/28-8
0x04C0	MPU_RGD12 — MPU region descriptor 12	32	R/W	—[1]	28.2.2.4/28-8
0x04D0	MPU_RGD13 — MPU region descriptor 13	32	R/W	—[1]	28.2.2.4/28-8
0x04E0	MPU_RGD14 — MPU region descriptor 14	32	R/W	—[1]	28.2.2.4/28-8
0x04F0	MPU_RGD15 — MPU region descriptor 15	32	R/W	—[1]	28.2.2.4/28-8
0x00500–0x07FF	Reserved				
0x0800	MPU_RGDAAC0 — MPU RGD alternate access control 0	32	W	—[1]	28.2.2.5/28-13
0x0804	MPU_RGDAAC1 — MPU RGD alternate access control 1	32	W	—[1]	28.2.2.5/28-13
0x0808	MPU_RGDAAC2 — MPU RGD alternate access control 2	32	W	—[1]	28.2.2.5/28-13
0x080C	MPU_RGDAAC3 — MPU RGD alternate access control 3	32	W	—[1]	28.2.2.5/28-13
0x0810	MPU_RGDAAC4 — MPU RGD alternate access control 4	32	W	—[1]	28.2.2.5/28-13
0x0814	MPU_RGDAAC5 — MPU RGD alternate access control 5	32	W	—[1]	28.2.2.5/28-13
0x0818	MPU_RGDAAC6 — MPU RGD alternate access control 6	32	W	—[1]	28.2.2.5/28-13
0x081C	MPU_RGDAAC7 — MPU RGD alternate access control 7	32	W	—[1]	28.2.2.5/28-13
0x0820	MPU_RGDAAC8 — MPU RGD alternate access control 8	32	W	—[1]	28.2.2.5/28-13
0x0824	MPU_RGDAAC9 — MPU RGD alternate access control 9	32	W	—[1]	28.2.2.5/28-13
0x0828	MPU_RGDAAC10 — MPU RGD alternate access control 10	32	W	—[1]	28.2.2.5/28-13
0x082C	MPU_RGDAAC11 — MPU RGD alternate access control 11	32	W	—[1]	28.2.2.5/28-13
0x0830	MPU_RGDAAC12 — MPU RGD alternate access control 12	32	W	—[1]	28.2.2.5/28-13
0x0834	MPU_RGDAAC13 — MPU RGD alternate access control 13	32	W	—[1]	28.2.2.5/28-13
0x0838	MPU_RGDAAC14 — MPU RGD alternate access control 14	32	W	—[1]	28.2.2.5/28-13

Table 28-2. MPU Memory Map (continued)

Offset from MPU_BASE (0xFFF1_0000)	Register	Bits	Access	Reset Value	Section/Page
0x083C	MPU_RGDAAC15 — MPU RGD alternate access control 15	32	W	—[1]	28.2.2.5/28-13
0x0840–0x08FF	Reserved				

[1] See register definition.

28.2.2 Register Descriptions

This section lists the MPU registers in address order and describes the registers and their bit fields.

28.2.2.1 MPU Control/Error Status Register (MPU_CESR)

The MPU_CESR provides one byte of error status and three bytes of configuration information. A global MPU enable/disable bit is also included in this register.

Offset: MPU_BASE+0x0000

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SPERR[1]								1	0	0	0	HRL				NSP				NRGD				0	0	0	0	0	0	0	VLD
W	(w1c)																															
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0

Figure 28-2. MPU Control/Error Status Register (MPU_CESR)

[1] Each SPERR bit can be cleared by writing a one to the bit location.

Table 28-3. MPU_CESR Bit Field Descriptions

Field	Description
0–7 SPERR	MPU port *n* Error (where the MPU port number matches the bit number (see Figure 28-1)). Each bit in this read-only field represents a flag maintained by the MPU for signaling the presence of a captured error contained in the MPU_EAR*n* and MPU_EDR*n* registers. The individual bit is set when the hardware detects an error and records the faulting address and attributes. It is cleared when the corresponding bit is written to a logical one. If another error is captured at the exact same cycle as a write of a logical one, this flag remains set. A find-first-one instruction (or equivalent) can be used to detect the presence of a captured error. 0 The corresponding MPU_EAR*n*/MPU_EDR*n* registers do not contain an unread captured error 1 The corresponding MPU_EAR*n*/MPU_EDR*n* registers do contain an unread captured error **Note:** Bit 0 indicates an EBI protection error, bit 1 indicates an SRAM protection error, bit 2 indicates a peripheral bridge B protection error, and bit 3 indicates a peripheral bridge A protection error.
8–11	Reserved
12–15 HRL	Hardware Revision Level. This 4-bit read-only field specifies the MPU's hardware and definition revision level. It can be read by software to determine the functional definition of the module. This field reads as 0 on this device.
16–19 NSP	Number of MPU ports. This 4-bit read-only field specifies the number of slave ports connected to the MPU. This field reads as 0b0100 on this device.

Table 28-3. MPU_CESR Bit Field Descriptions (continued)

Field	Description
20–23 NRGD	Number of Region Descriptors. This 4-bit read-only field specifies the number of region descriptors implemented in the MPU. The defined encodings include: 0000 8 region descriptors 0010 16 region descriptors This field reads as 0b0010 on this device.
24–30	Reserved
31 VLD	Valid. This bit provides a global enable/disable for the MPU. 0 The MPU is disabled 1 The MPU is enabled While the MPU is disabled, all accesses from all bus masters are allowed.

28.2.2.2 MPU Error Address Register, MPU Port 0 to 3 (MPU_EAR*n*)

When the MPU detects an access error on MPU port *n*, the 32-bit reference address is captured in this read-only register and the corresponding bit in the MPU_CESR[SPERR] field set. Additional information about the faulting access is captured in the corresponding MPU_EDR*n* register at the same time.

Offset: MPU_BASE + 0x0010 (MPU_EAR0)
MPU_BASE + 0x0018 (MPU_EAR1)
MPU_BASE + 0x0020 (MPU_EAR2)
MPU_BASE + 0x0028 (MPU_EAR3)

Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EADDR																															
W																																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 28-3. MPU Error Address Register, MPU Port *n* (MPU_EAR*n*)

Table 28-4. MPU_EAR Bit Field Descriptions

Field	Description
0–31 EADDR	Error Address. This read-only field is the reference address from MPU port *n* that generated the access error.

28.2.2.3 MPU Error Detail Register, MPU Port 0 to 3 (MPU_EDR*n*)

When the MPU detects an access error on MPU port *n*, 32 bits of error detail are captured in this read-only register and the corresponding bit in the MPU_CESR[SPERR] field set. Information on the faulting address is captured in the corresponding MPU_EAR*n* register at the same time. A read of the MPU_EDR*n* register clears the corresponding bit in the MPU_CESR[SPERR] field.

Offset: MPU_BASE + 0x00014 (MPU_EDR0)
MPU_BASE + 0x001C (MPU_EDR1)
MPU_BASE + 0x0024 (MPU_EDR2)
MPU_BASE + 0x002C (MPU_EDR3)

Access: User read only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	EACD																EPID								EMN				EATTR			ERW
W																																
Reset	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–

Figure 28-4. MPU Error Detail Register, MPU Port *n* (MPU_EDR*n*)

Table 28-5. MPU_EDR Bit Field Descriptions

Field	Description
0–15 EACD	Error Access Control Detail. This 16-bit read-only field implements one bit per region descriptor and is an indication of the region descriptor hit logically-ANDed with the access error indication. The MPU performs a reference-by-reference evaluation to determine the presence/absence of an access error. When an error is detected, the hit-qualified access control vector is captured in this field. If the MPU_EDR*n* register contains a captured error and the EACD field is all zeroes, this signals an access that did not hit in any region descriptor. All non-zero EACD values signal references that hit in a region descriptor(s), but failed due to a protection error as defined by the specific set bits.
16–23 EPID	Error Process Identification. This 8-bit read-only field records the process identifier of the faulting reference. The process identifier is typically driven by processor cores only; for other bus masters, this field is cleared.
24–27 EMN	Error Master Number. This 4-bit read-only field records the logical master number of the faulting reference. This field is used to determine the bus master that generated the access error.
28–30 EATTR	Error Attributes. This 3-bit read-only field records attribute information about the faulting reference. The supported encodings are defined as: 000 User mode, instruction access 001 User mode, data access 010 Supervisor mode, instruction access 011 Supervisor mode, data access All other encodings are reserved. For non-core bus masters, the access attribute information is typically wired to supervisor, data (0b011).
31 ERW	Error Read/Write. This 1-bit read-only field signals the access type (read, write) of the faulting reference. 0 Read 1 Write

28.2.2.4 MPU Region Descriptor *n* (MPU_RGD*n*)

Each 128-bit (16 byte) region descriptor specifies a given memory space and the access attributes associated with that space. The descriptor definition is fundamental to the operation of the MPU.

The region descriptors are organized sequentially in the MPU's programming model and each of the four 32-bit words are detailed in the subsequent sections.

28.2.2.4.1 MPU Region Descriptor *n*, Word 0 (MPU_RGD*n*.Word0)

The first word of the MPU region descriptor defines the 0-modulo-32 byte start address of the memory region. Writes to this word clear the region descriptor’s valid bit.

Offset: MPU_BASE + 0x400 + (16**n*) + 0x0 (MPU_RGD*n*.Word0) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SRTADDR																											0	0	0	0	0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-5. MPU Region Descriptor, Word 0 Register (MPU_RGD*n*.Word0)

Table 28-6. MPU_RGD Word 0 Description

Field	Description
0–26 SRTADDR	Start Address. This field defines the most significant bits of the 0-modulo-32 byte start address of the memory region.
27–31	Reserved

28.2.2.4.2 MPU Region Descriptor *n*, Word 1 (MPU_RGD*n*.Word1)

The second word of the MPU region descriptor defines the 31-modulo-32 byte end address of the memory region. Writes to this word clear the region descriptor’s valid bit.

Offset: MPU_BASE + 0x400 + (16**n*) + 0x4 (MPU_RGD*n*.Word1) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	ENDADDR																											1	1	1	1	1
W																																
Reset (n=0)	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Reset (n>0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1

Figure 28-6. MPU Region Descriptor, Word 1 Register (MPU_RGD*n*.Word1)

Table 28-7. MPU_RGD Word 1 Description

Field	Description
0–26 ENDADDR	End Address. This field defines the most significant bits of the 31-modulo-32 byte end address of the memory region. There are no hardware checks to verify that ENDADDR ≥ SRTADDR; the software must properly load these region descriptor fields.
27–31	Reserved

28.2.2.4.3 MPU Region Descriptor *n*, Word 2 (MPU_RGD*n*.Word2)

The third word of the MPU region descriptor defines the access control rights of the memory region. The access control privileges are dependent on two broad classifications of bus masters. Bus masters 0–3 are typically reserved for processor cores. The corresponding access control is a 6-bit field defining separate privilege rights for user and supervisor mode accesses as well as the optional inclusion of a process identification field within the definition. Bus masters 4–7 are typically reserved for data movement engines and their capabilities are limited to separate read and write permissions. For these fields, the bus master number refers to the logical master number defined as the AHB `hmaster[3:0]`signal.

For the processor privilege rights, there are three flags associated with this function: {read, write, execute}. In this context, these flags follow the traditional definition:

- Read (r) permission refers to the ability to access the referenced memory address using an operand (data) fetch.
- Write (w) permission refers to the ability to update the referenced memory address using a store (data) instruction.
- Execute (x) permission refers to the ability to read the referenced memory address using an instruction fetch.

The evaluation logic defines the processor access type based on multiple AHB signals: read or write as specified by the `hwrite` signal and the low-order two bits of `hprot[1:0]`, which identify a data reference versus an instruction fetch and the operating mode (supervisor, user) of the requesting processor.

For non-processor data movement engines (bus masters 4–7), the evaluation logic simply uses `hwrite` to determine if the access is a read or write. The `hprot[1:0]` signal is ignored for these masters.

Writes to this word clear the region descriptor’s valid bit. Because it is also expected that system software may adjust only the access controls within a region descriptor (MPU_RGD*n*.Word2) as different tasks execute, an alternate programming view of this 32-bit entity is provided. If only the access controls are being updated, this operation should be performed by writing to MPU_RGDAAC*n* (alternate access control *n*) as stores to these locations do not affect the descriptor’s valid bit.

Offset: MPU_BASE + 0x400 + (16*n) + 0x8 (MPU_RGDn.Word2) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	M6RE	M6WE	M5RE	M5WE	M4RE	M4WE	0	0	0	0	0	0	0	0
W																
Reset (n=0)	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1
Reset (n>0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	M1PE	M1SM		M1UM			M0PE	M0SM		M0UM		
W								r	w	x				r	w	x
Reset (n=0)	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
Reset (n>0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: Refer to Table 28-1to see the Master ID assignments.

Figure 28-7. MPU Region Descriptor, Word 2 Register (MPU_RGD*n*.Word2)

Table 28-8. MPU_RGD Word 2 Description

Field	Description
	Note: For future code compatibility, do not change the value of reserved bits from their reset value
0–1	Reserved
2 4 6 M*n*RE	Bus Master ID *n* Read Enable. If set, this flag allows bus master ID *n* to perform read operations. If cleared, any attempted read by bus master ID *n* terminates with an access error and the read is not performed. **Note:** See Table 28-1 for the MPU Master ID list.
3 5 7 M*n*WE	Bus Master ID *n* Write Enable. If set, this flag allows bus master ID *n* to perform write operations. If cleared, any attempted write by bus master ID *n* terminates with an access error and the write is not performed. **Note:** See Table 28-1 for the MPU Master ID list.
8–25	Reserved
M1PE	Bus Master ID 1Process Identifier Enable. If set, this flag specifies that the process identifier and mask defined in MPU_RGD*n*.Word3 are to be included in the region hit evaluation. If cleared, the region hit evaluation does not include the process identifier. **Note:** See Table 28-1 for the MPU Master ID list.
M1SM	Bus Master ID 1Supervisor Mode Access Control. This 2-bit field defines the access controls for bus master ID 1 when operating in supervisor mode. The M1SM field is defined as: 00 `r, w, x` = read, write and execute allowed 01 `r, -, x` = read and execute allowed, but no write 10 `r, w, -` = read and write allowed, but no execute 11 Same access controls as that defined by M1UM for user mode **Note:** See Table 28-1 for the MPU Master ID list.
M1UM	Bus Master ID 1User Mode Access Control. This 3-bit field defines the access controls for bus master ID 1when operating in user mode. The M1UM field consists of three independent bits, enabling read, write, and execute permissions: `{r,w,x}`. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed. **Note:** See Table 28-1 for the MPU Master ID list.
26 M0PE	Bus Master ID 0 Process Identifier Enable. If set, this flag specifies that the process identifier and mask defined in MPU_RGD*n*.Word3 are to be included in the region hit evaluation. If cleared, the region hit evaluation does not include the process identifier. **Note:** See Table 28-1 for the MPU Master ID list.
27–28 M0SM	Bus Master ID 0 Supervisor Mode Access Control. This 2-bit field defines the access controls for bus master ID 0 when operating in supervisor mode. The M0SM field is defined as: 00 `r, w, x` = read, write and execute allowed 01 `r, -, x` = read and execute allowed, but no write 10 `r, w, -` = read and write allowed, but no execute 11 Same access controls as that defined by M0UM for user mode **Note:** See Table 28-1 for the MPU Master ID list.
29–31 M0UM	Bus Master ID 0 User Mode Access Control. This 3-bit field defines the access controls for bus master ID 0 when operating in user mode. The M0UM field consists of three independent bits, enabling read, write, and execute permissions: `{r, w, x}`. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed. **Note:** See Table 28-1 for the MPU Master ID list.

28.2.2.4.4 MPU Region Descriptor *n*, Word 3 (MPU_RGD*n*.Word3)

The fourth word of the MPU region descriptor contains the optional process identifier and mask, plus the region descriptor's valid bit.

Because the region descriptor is a 128-bit entity, there are potential coherency issues as this structure is being updated because multiple writes are required to update the entire descriptor. Accordingly, the MPU hardware assists in the operation of the descriptor valid bit to prevent incoherent region descriptors from generating spurious access errors. In particular, it is expected that a complete update of a region descriptor is typically done with sequential writes to MPU_RGD*n*.Word0, then MPU_RGD*n*.Word1, ..., and MPU_RGD*n*.Word3. The MPU hardware automatically clears the valid bit on any writes to words {0,1,2} of the descriptor. Writes to this word set/clear the valid bit in a normal manner.

Because it is also expected that system software may adjust the access controls within a region descriptor (MPU_RGDn.Word2) only as different tasks execute, an alternate programming view of this 32-bit entity is provided. If only the access controls are being updated, this operation must be performed by writing to MPU_RGDAAC*n* (alternate access control *n*) as stores to these locations do not affect the descriptor's valid bit.

Offset: MPU_BASE + 0x400 + (16**n*) + 0xc (MPU_RGD*n*.Word3)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	PID								PIDMASK								0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	VLD
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-8. MPU Region Descriptor, Word 3 Register (MPU_RGD*n*.Word3)

Table 28-9. MPU_RGD Word 3 Description

Field	Description
0–7 PID	Process Identifier. This 8-bit field specifies that the optional process identifier is to be included in the determination of whether the current access hits in the region descriptor. This field is combined with the PIDMASK and included in the region hit determination if MPU_RGD*n*.Word2[MxPE] is set.
8–15 PIDMASK	Process Identifier Mask. This 8-bit field provides a masking capability so that multiple process identifiers can be included as part of the region hit determination. If a bit in the PIDMASK is set, the corresponding bit of the PID is ignored in the comparison. This field is combined with the PID and included in the region hit determination if MPU_RGD*n*.Word2[MxPE] is set. For more information on the handling of the PID and PIDMASK, see Section 28.3.1.1, "Access Evaluation—Hit Determination".
16–30	Reserved
31 VLD	Valid. This bit signals the region descriptor is valid. Any write to MPU_RGD*n*.Word{0,1,2} clears this bit, but a write to MPU_RGD*n*.Word3 sets or clears this bit depending on bit 31 of the write operand. 0 Region descriptor is invalid 1 Region descriptor is valid

28.2.2.5 MPU Region Descriptor Alternate Access Control *n* (MPU_RGDAAC*n*)

As noted in Section 28.2.2.4.3, “MPU Region Descriptor n, Word 2 (MPU_RGDn.Word2)”, it is expected that because system software may adjust the access controls within a region descriptor (MPU_RGD*n*.Word2) only as different tasks execute, an alternate programming view of this 32-bit entity is desired. If only the access controls are being updated, this operation should be performed by writing to MPU_RGDAAC*n* (alternate access control *n*) as stores to these locations do not affect the descriptor’s valid bit.

The memory address therefore provides an alternate location for updating MPU_RGD*n*.Word2.

Address: MPU_BASE + 0x800 + (4**n*) (MPU_RGDAAC*n*) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	M6RE	M6WE	M5RE	M5WE	M4RE	M4WE	0	0	0	0	0	0	0	0
W																
Reset (n=0)	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	1
Reset (n>0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	M1PE	M1SM		M1UM			M0PE	M0SM		M0UM		
W								r	w	x				r	w	x
Reset (n=0)	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
Reset (n>0)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-9. MPU RGD Alternate Access Control *n* (MPU_RGDAAC*n*)

Because the MPU_RGDAAC*n* register is another memory mapping for MPU_RGD*n*.Word2, the field definitions shown in Table 28-10 are identical to those presented in Table 28-8.

Table 28-10. MPU_RGDAAC Bit Field Descriptions

Field	Description
	Note: For future code compatibility, do not change the value of reserved bits from their reset value
0–1	Reserved
2 4 6 M*n*RE	Bus Master ID *n* Read Enable. If set, this flag allows bus master ID *n* to perform read operations. If cleared, any attempted read by bus master ID *n* terminates with an access error and the read is not performed. **Note:** See Table 28-1 for the MPU Master ID list.
3 5 7 M*n*WE	Bus Master *n* Write Enable. If set, this flag allows bus master *n* to perform write operations. If cleared, any attempted write by bus master *n* terminates with an access error and the write is not performed. **Note:** See Table 28-1 for the MPU Master ID list.
8–25	Reserved

Table 28-10. MPU_RGDAAC Bit Field Descriptions (continued)

Field	Description
M1PE	Bus Master ID 1Process Identifier Enable. If set, this flag specifies that the process identifier and mask defined in MPU_RGD*n*.Word3 are to be included in the region hit evaluation. If cleared, the region hit evaluation does not include the process identifier. **Note:** See Table 28-1 for the MPU Master ID list.
M1SM	Bus Master ID 1Supervisor Mode Access Control. This 2-bit field defines the access controls for bus master ID 1 when operating in supervisor mode. The M1SM field is defined as: 00 `r, w, x` = read, write and execute allowed 01 `r, -, x` = read and execute allowed, but no write 10 `r, w, -` = read and write allowed, but no execute 11 Same access controls as that defined by M1UM for user mode **Note:** See Table 28-1 for the MPU Master ID list.
M1UM	Bus Master ID 1User Mode Access Control. This 3-bit field defines the access controls for bus master ID 1when operating in user mode. The M1UM field consists of three independent bits, enabling read, write, and execute permissions: `{r,w,x}`. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed. **Note:** See Table 28-1 for the MPU Master ID list.
26 M0PE	Bus Master 0 Process Identifier Enable. If set, this flag specifies that the process identifier and mask (defined in MPU_RGDn.Word3) are to be included in the region hit evaluation. If cleared, then the region hit evaluation does not include the process identifier.
27–28 M0SM	Bus Master 0 Supervisor Mode Access Control. This 2-bit field defines the access controls for bus master 0 when operating in supervisor mode. The M0SM field is defined as: 00 `r, w, x` = read, write and execute allowed 01 `r, -, x` = read and execute allowed, but no write 10 `r, w, -` = read and write allowed, but no execute 11 Same access controls as that defined by M0UM for user mode **Note:** See Table 28-1 for the MPU Master ID list.
29–31 M0UM	Bus Master 0 User Mode Access Control. This 3-bit field defines the access controls for bus master 0 when operating in user mode. The M0UM field consists of three independent bits, enabling read, write, and execute permissions: `{r, w, x}`. If set, the bit allows the given access type to occur; if cleared, an attempted access of that mode may be terminated with an access error (if not allowed by any other descriptor) and the access not performed. **Note:** See Table 28-1 for the MPU Master ID list.

28.3 Functional Description

In this section, the functional operation of the MPU is detailed. In particular, subsequent sections discuss the operation of the access evaluation macro as well as the handling of error-terminated AHB bus cycles.

28.3.1 Access Evaluation

As discussed, the basic operation of the MPU is performed in the access evaluation macro, a hardware structure replicated in the two-dimensional connection matrix. The access evaluation macro inputs the AHB system bus address and the contents of a region descriptor (RGD*n*) and performs two major functions: region hit determination and detection of an access protection violation.

28.3.1.1 Access Evaluation—Hit Determination

To determine if the current AHB reference hits in the given region, two magnitude comparators are used with the region's start and end addresses. There are no hardware checks to verify that the region end address is greater than or equal to the region start address. The software must properly load appropriate values into these fields of the region descriptor.

In addition to the comparison of the AHB reference address versus the region descriptor's start and end addresses, the optional process identifier is examined against the region descriptor's PID and PIDMASK fields. For AHB bus masters that do not output a process identifier, the MPU forces the PID term to be asserted.

28.3.1.2 Access Evaluation—Privilege Violation Determination

While the access evaluation macro is making the region hit determination, the logic is also evaluating if the current access is allowed by the permissions defined in the region descriptor. Using the AHB supervisor/user mode signals, a set of permissions is generated from the appropriate fields in the region descriptor. The protection violation logic evaluates the access against the effective permissions.

The access evaluation macro then uses the hit and permission signals to determine if the current access is allowed and the MPU_EDR*n* (error detail register) is updated in the event of an error.

28.3.2 AHB Error Terminations

For each AHB slave port being monitored, the MPU tests any access for permission violations as above. If a violation occurs, the MPU terminates the bus cycle and reports a protection error for three conditions:

1. If the access does not hit in any region descriptor, a protection error is reported.
2. If the access hits in a single region descriptor and that region signals a protection violation, a protection error is reported.
3. If the access hits in multiple (overlapping) regions and all regions signal protection violations, then a protection error is reported.

The third condition reflects that priority is given to permission granting over access denying for overlapping regions as this approach provides more flexibility to system software in region descriptor assignments. For an example of the use of overlapping region descriptors, see Section 28.5, "Application Information".

When the MPU causes a termination error to occur, the effect on the system depends on the bus master requesting the access. If the error was caused by a core access, a machine check is taken. If the error was caused by an eDMA access, an eDMA source or destination error occurs in the eDMA controller, which can be enabled to provide an interrupt request through the INTC. If the error was caused by a FlexRay access, a controller host interface (CHI) illegal system memory access error occurs in the FlexRay controller, which can be enabled to provide an interrupt request to the INTC.

28.4 Initialization Information

The reset state of MPU_CESR[VLD] disables the entire module. While the MPU is disabled, all accesses from all bus masters are allowed. This state also minimizes the power dissipation of the MPU. The power dissipation of each access evaluation macro is minimized when the associated region descriptor is marked as invalid or when MPU_CESR[VLD] = 0.

Typically the appropriate number of region descriptors (MPU_RGD*n*) are loaded at system startup, including the setting of the MPU_RGD*n*.Word3[VLD] bits, before MPU_CESR[VLD] is set, enabling the module. This approach allows all the loaded region descriptors to be enabled simultaneously. Once the MPU is enabled, if a memory reference does not hit in any region descriptor, the attempted access is terminated with an error.

28.5 Application Information

In an application's system, interfacing with the MPU can generally be classified into the following activities:

1. Creation of a new memory region requires loading the appropriate region descriptor into an available register location. When a new descriptor is loaded into a RGD*n*, it would typically be performed using four 32-bit word writes. As discussed in Section 28.2.2.4.4, "MPU Region Descriptor n, Word 3 (MPU_RGDn.Word3)", the hardware assists in the maintenance of the valid bit, so if this approach is followed, there are no coherency issues associated with the multi-cycle descriptor writes. Deletion/removal of an existing memory region is performed by clearing MPU_RGD*n*.Word3[VLD].
2. If only the access rights for an existing region descriptor need to change, a 32-bit write to the alternate version of the access control word (MPU_RGDAAC*n*) would typically be performed. Writes to the region descriptor using this alternate access control location do not affect the valid bit, so there are, by definition, no coherency issues involved with the update. The access rights associated with the memory region switch instantaneously to the new value as the peripheral write completes.
3. If the region's start and end addresses are to be changed, this would typically be performed by writing a minimum of three words of the region descriptor: MPU_RGD*n*.Word{0,1,3}, where the writes to Word0 and Word1 redefine the start and end addresses respectively and the write to Word3 re-enables the region descriptor valid bit. In many situations, all four words of the region descriptor would be rewritten.
4. Typically, references to the MPU's programming model would be restricted to supervisor mode accesses from a specific processor(s), so a region descriptor would be specifically allocated for this purpose with attempted accesses from other masters or while in user mode terminated with an error.
5. When the MPU detects an access error, the current AHB bus cycle is terminated with an error response and information on the faulting reference captured in the MPU_EAR*n* and MPU_EDR*n* registers. The error-terminated AHB bus cycle typically initiates some type of error response in the originating bus master. For example, a processor core may respond with a bus error exception, while a data movement bus master may respond with an error interrupt. In any event, the processor

can retrieve the captured error address and detail information simply be reading the MPU_E{A,D}R*n* registers. Information on which error registers contain captured fault data is signaled by MPU_CESR[SPERR].

6. Finally, consider the use of overlapping region descriptors. Application of overlapping regions can reduce the number of descriptors required for a given set of access controls. In the overlapping memory space, the protection rights of the corresponding region descriptors are logically summed together (the boolean OR operator). In the following example of a dual-core system, there are four bus masters: the two processors (CP0, CP1) and two DMA engines (eDMA, a traditional data movement engine transferring data between RAM and peripherals, and FlexRay, a second engine transferring data to/from the RAM only). Consider the following region descriptor assignments:

<table>
<tr><th>Region Description</th><th colspan="2">RGDn</th><th>CP0</th><th>CP1</th><th>eDMA</th><th>FlexRay</th><th></th></tr>
<tr><td>CP0 Code</td><td colspan="2">0</td><td>rwx</td><td>r--</td><td>--</td><td>--</td><td rowspan="2">Flash</td></tr>
<tr><td>CP1 Code</td><td colspan="2">1</td><td>r--</td><td>rwx</td><td>--</td><td>--</td></tr>
<tr><td>CP0 Data & Stack</td><td colspan="2">2</td><td>rw-</td><td>---</td><td>--</td><td>--</td><td rowspan="5">RAM</td></tr>
<tr><td>CP0 -> CP1 Shared Data</td><td></td><td rowspan="2">3</td><td rowspan="2">r--</td><td rowspan="2">r--</td><td rowspan="2">--</td><td rowspan="2">--</td></tr>
<tr><td>CP1 -> CP0 Shared Data</td><td></td></tr>
<tr><td>CP0 Data & Stack</td><td colspan="2">4</td><td>---</td><td>rw-</td><td>--</td><td>--</td></tr>
<tr><td>Shared DMA Data</td><td colspan="2">5</td><td>rw-</td><td>rw-</td><td>rw</td><td>rw</td></tr>
<tr><td>MPU</td><td colspan="2">6</td><td>rw-</td><td>rw-</td><td>--</td><td>--</td><td rowspan="2">Peripheral</td></tr>
<tr><td>Peripherals</td><td colspan="2">7</td><td>rw-</td><td>rw-</td><td>rw</td><td>--</td></tr>
</table>

Figure 28-10. Overlapping Region Descriptor Example

In this example, there are eight descriptors used to span nine regions in the three main spaces of the system memory map (flash, RAM, and peripheral space). Each region indicates the specific permissions for each of the four bus masters and this definition provides an appropriate set of shared, private and executable memory spaces.

Of particular interest are the two overlapping spaces: region descriptors 2 and 3, and 3 and 4.

The space defined by RGD2 with no overlap is a private data and stack area that provides read/write access to CP0 only. The overlapping space between RGD2 and RGD3 defines a shared data space for passing data from CP0 to CP1 and the access controls are defined by the logical OR of the two region descriptors. Thus, CP0 has (`rw- | r--`) = (`rw-`) permissions, while CP1 has (`--- | r--`) = (`r--`) permission in this space. Both DMA engines are excluded from this shared processor data region. The overlapping spaces between RGD3 and RGD4 defines another shared data space, this one for passing data from CP1 to CP0. For this overlapping space, CP0 has (`r-- | ---`) = (`r--`) permission, while CP1 has (`rw- | r--`) = (`rw-`) permission. The non-overlapped space of RGD4 defines a private data and stack area for CP1 only.

The space defined by RGD5 is a shared data region, accessible by all four bus masters. Finally, the slave peripheral space mapped onto the peripheral bus is partitioned into two regions: one (RGD6) containing the MPU's programming model accessible only to the two processor cores, and the remaining peripheral region (RGD7) accessible to both processors and the traditional eDMA master.

This example is intended to show one possible application of the capabilities of the memory protection unit in a typical system.

Chapter 29
Periodic Interrupt Timer (PIT_RTI)

29.1 Introduction

29.1.1 Overview

This section describes the function of the Periodic Interrupt Timer block (PIT_RTI). The PIT is an array of timers that can be used to generate interrupts. It also provides a dedicated Real Time Interrupt Timer (RTI), which runs on a separate clock and can be used for system wakeup from low power mode.

29.1.2 Block Diagram

A block diagram of the PIT_RTI module is shown in Figure 29-1.

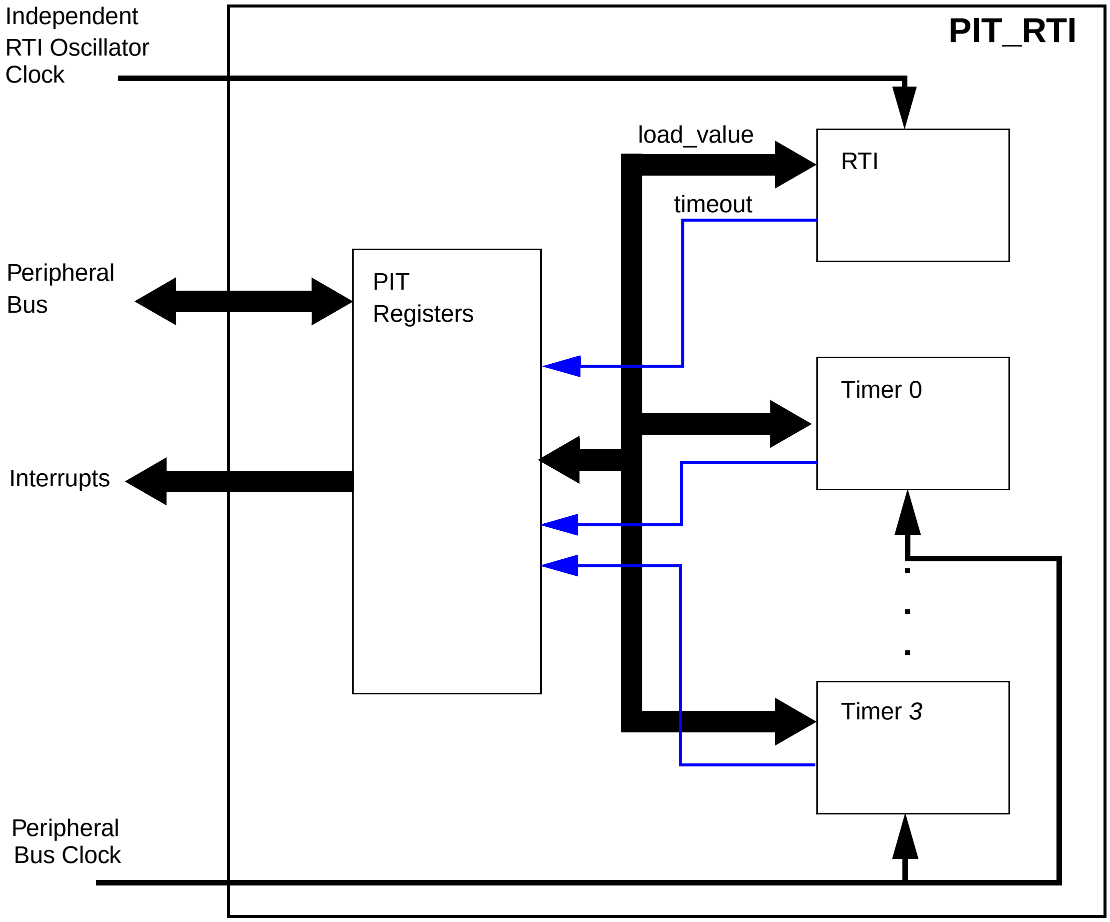


Figure 29-1. Block diagram of PIT_RTI

29.1.3 Features

The main features of this block are:

- Timers can be configured to generate interrupts
- All interrupts are maskable
- Independent timeout periods for each timer and RTI
- RTI can be used to generate a CPU wake-up interrupt
- RTI clock source is the crystal oscillator, no pre-scalars are used
- PIT timer clock source is the peripheral clock, no pre-scalars are used

29.2 Signal Description

The PIT module has no external pins.

29.3 Memory Map and Register Description

This section provides a detailed description of all registers accessible in the PIT_RTI module.

29.3.1 Memory Map

Table 29-1 provides an overview off all PIT_RTI registers.

Table 29-1. PIT_RTI Memory Map

Base Address Offset (Base = 0xC3FF0000	Register	Bits	Access	Reset Value	Section/Page
0x000	PIT_MCR—PIT Module Control Register	32	R/W	0x0000_0000	29.3.2.1/4
0x004—0x0EC	Reserved				
0x0F4	PIT_RTI_CVAL—RTI current value register	32	R	0x0000_0000	29.3.2.3/5
0x0F8	PIT_RTI_TCTRL—RTI timer control register	32	R/W	0x0000_0000	29.3.2.4/6
0x0FC	PIT_RTI_TFLAG—RTI timer flag register	32	R/W	0x0000_0000	29.3.2.5/6
0x100	PIT_CH0_LDVAL—Channel 0 load value register	32	R/W	0x0000_0000	29.3.2.2/5
0x104	PIT_CH0_CVAL—Channel 0 current value register	32	R	0x0000_0000	29.3.2.3/5
0x108	PIT_CH0_TCTRL—Channel 0 timer control register	32	R/W	0x0000_0000	29.3.2.4/6
0x10C	PIT_CH0_TFLAG—Channel 0 timer channel flag register	32	R/W	0x0000_0000	29.3.2.5/6
0x110	PIT_CH1_LDVAL—Channel 1 load value register	32	R/W	0x0000_0000	29.3.2.2/5
0x114	PIT_CH1_CVAL—Channel 1 current value register	32	R	0x0000_0000	29.3.2.3/5
0x118	PIT_CH1_TCTRL—Channel 1 timer control register	32	R/W	0x0000_0000	29.3.2.4/6
0x11C	PIT_CH1_TFLAG—Channel 1 timer channel flag register	32	R/W	0x0000_0000	29.3.2.5/6
0x120	PIT_CH2_LDVAL—Channel 2 load value register	32	R/W	0x0000_0000	29.3.2.2/5
0x124	PIT_CH2_CVAL—Channel 2 current value register	32	R	0x0000_0000	29.3.2.3/5
0x128	PIT_CH2_TCTRL—Channel 2 timer control register	32	R/W	0x0000_0000	29.3.2.4/6
0x12C	PIT_CH2_TFLAG—Channel 2 timer channel flag register	32	R/W	0x0000_0000	29.3.2.5/6

Table 29-1. PIT_RTI Memory Map (continued)

Base Address Offset (Base = 0xC3FF0000	Register	Bits	Access	Reset Value	Section/Page
0x130	PIT_CH3_LDVAL—Channel 3 load value register	32	R/W	0x0000_0000	29.3.2.2/5
0x134	PIT_CH3_CVAL—Channel 3 current value register	32	R	0x0000_0000	29.3.2.3/5
0x138	PIT_CH3_TCTRL—Channel 3 timer control register	32	R/W	0x0000_0000	29.3.2.4/6
0x13C	PIT_CH3_TFLAG—Channel 3 timer channel flag register	32	R/W	0x0000_0000	29.3.2.5/6

29.3.2 Register Descriptions

This section describes all PIT_RTI registers and their individual bits.

29.3.2.1 PIT Module Control Register (PIT_MCR)

This register provides a mechanism for enabling and disabling timers, and setting whether timers continue to count in debug mode.

Offset 0x000 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	MDIS_RTI	MDIS	FRZ
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-2. PIT Module Control Register (PIT_MCR)

Table 29-2. PITMCR Field Descriptions

Field	Description
0–2	Reserved
29 MDIS_RTI	Module Disable - RTI section. This is used to disable the RTI timer. This bit should be enabled before any RTI setup is done. 0 Clock for RTI is enabled 1 Clock for RTI disabled

Field	Description
30 MDIS	Module Disable - (PIT section). This is used to disable the standard timers. The RTI timer is not affected by this bit. This bit should be enabled before any other setup is done. 0 Clock for PIT Timers is enabled 1 Clock for PIT Timers is disabled
31 FRZ	Freeze. Allows the timers to be stopped when the device enters debug mode. 0 = Timers continue to run in debug mode. 1 = Timers are stopped in debug mode.

29.3.2.2 Timer Load Value Register (PIT_RTI_LDVAL, PIT_CH*n*_LDVAL)

This register selects the timeout period for the timer interrupts.

Offset: channel_base + 0x00
RTI base = 0x0F0
CH0 base = 0x100
CH1 base = 0x110
CH2 base = 0x120
CH3 base = 0x130

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TSV[0:31]																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-3. Timer Load Value Register (PIT_RTI_LDVAL, PIT_CH*n*_LDVAL)

Table 29-3. PIT_RTI_LDVAL, PIT_CH*n*_LDVAL Field Descriptions

Field	Description
0–31 TSV	Time Start Value Bits. These bits set the timer start value. The timer counts down until it reaches 0, then it generates an interrupt and loads this register value again. Writing a new value to this register does not restart the timer, instead the value is loaded once the timer expires. To abort the current cycle and start a timer period with the new value, the timer must be disabled and enabled again (see Figure 29-8).

29.3.2.3 Current Timer Value Register (PIT_RTI_CVAL, PIT_CH*n*_CVAL)

This register indicates the current timer position.

Offset: channel_base + 0x04

Access: User read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TVL[31:0]																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-4. Current Timer Value Register (PIT_RTI_CVAL, PIT_CH*n*_CVAL)

Table 29-4. PIT_RTI_CVAL, PIT_CH*n*_CVAL Field Descriptions

Field	Description
0–31 TVL	Current Timer Value. These bits represent the current timer value. Note that the timer uses a downcounter. NOTE: The timer values are frozen in Debug mode if the FRZ bit is set in the PIT Module Control Register (see Figure 29-2)

29.3.2.4 Timer Control Register (PIT_RTI_TCTRL, PIT_CH*n*_TCTRL)

This register contains the control bits for each timer.

Offset channel_base + 0x08
RTI base = 0x0F0
CH0 base = 0x100
CH1 base = 0x110
CH2 base = 0x120
CH3 base = 0x130

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIE	TEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-5. Timer Control Register (PIT_RTI_TCTRL, PIT_CH*n*_TCTRL)

Table 29-5. PIT_RTI_TCTRL, PIT_CH*n*_TCTRL Field Descriptions

Field	Description
0–29	Reserved
30 TIE	Timer Interrupt Enable Bit. 0 Interrupt requests from Timer x are disabled 1 Interrupt is requested whenever TIF is set When an interrupt is pending (TIF set), enabling the interrupt immediately causes an interrupt event. To avoid this, the associated TIF flag must be cleared first.
31 TEN	Timer Enable Bit. 0 Timer is disabled 1 Timer is active

29.3.2.5 Timer Flag Register (PIT_RTI_TFLG, PIT_CH*n*_TFLG)

This register holds the PIT interrupt flag for each timer.

Offset channel_base + 0x0C
RTI base = 0x0F0
CH0 base = 0x100
CH1 base = 0x110
CH2 base = 0x120
CH3 base = 0x130

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIF
W																w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-6. Timer Flag Register (PIT_RTI_TFLG, PIT_CH*n*_TFLG)

Table 29-6. PIT_RTI_TFLG, PIT_CH*n*_TFLG Field Descriptions

Field	Description
0–30	Reserved
31 TIF	Time Interrupt Flag. TIF is set to 1 at the end of the timer period.This flag can be cleared only by writing it with a 1. Writing a 0 has no effect. If enabled (TIE = 1), TIF causes an interrupt request. 0 Time-out has not yet occurred 1 Time-out has occurred

29.4 Functional Description

29.4.1 General

This section gives detailed information on the internal operation of the module. Each timer can be used to generate a unique interrupt vector.

29.4.1.1 Timers

Once enabled, the timers can be configured to generate interrupts at periodic intervals. The timer loads its start value, as specified in the LDVAL register, then counts down until the count reaches 0. Then the value in the LDVAL register is loaded again and the process repeats. Each time the timer reaches 0, an interrupt is generated if enabled, and the interrupt flag is set.

All interrupts can be enabled or masked (by setting the TIE bits in the TCTRL registers). A new interrupt can be generated only after the previous one is cleared.

If desired, the current counter value of the timer can be read via the CVAL registers.

The counter period can be restarted, by first disabling, then enabling the timer with the TEN bit (see Figure 29-7).

The counter period of a running timer can be modified, by first disabling the timer, setting a new load value and then enabling the timer again (see Figure 29-8). In the case of the RTI, because of the different clock domains (system clock / oscillator clock), a delay must be respected between setting the new value and re-enabling the RTI.

It is also possible to change the counter period without restarting the timer by writing the LDVAL register with the new load value. This value is loaded after the next trigger (counter reaches 0) event (see Figure 29-9).

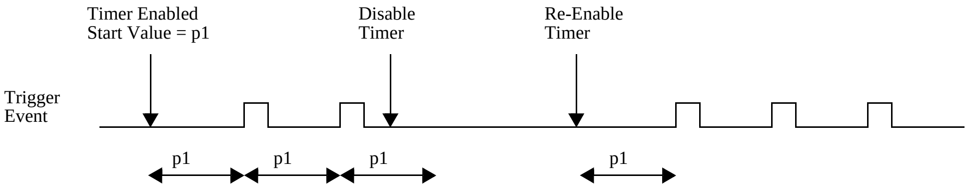


Figure 29-7. Stopping and Starting a Timer

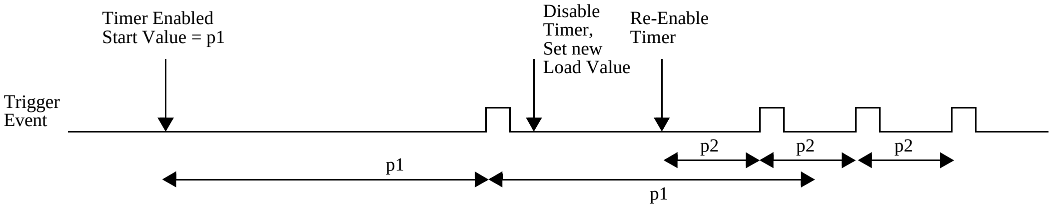


Figure 29-8. Modifying Running Timer Period

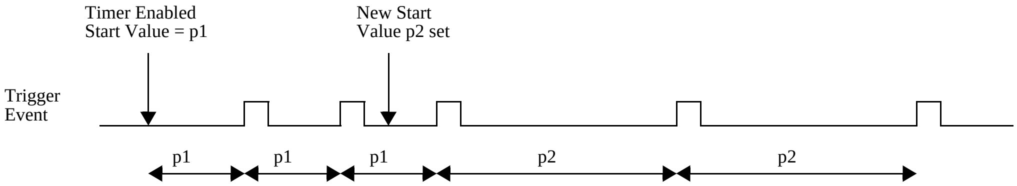


Figure 29-9. Dynamically Setting a New Load Value

29.4.1.2 Debug Mode

In debug mode the timers may be configured to stop when the debugger halts the device. This is intended to aid software development, allowing the developer to halt the processor, investigate the current state of the system (e.g. the timer values) and then continue the operation.

29.4.2 Interrupts

All of the timers support interrupt generation. Refer to Section 27.4.1, “External Interrupt Request Sources”, for related vector addresses and priorities.

Timer interrupts can be disabled by setting the timer PIT_CH*n*_TCTRL[TIE] bit to zero. The PIT_CH*n*_TFLG[TIF] bit is set to 1 when a timeout occurs on the associated timer, and is cleared by writing a 1 to that bit.

29.5 Initialization and Application Information

29.5.1 Example Configuration

In the example configuration:

- the PIT clock has a frequency of 50 MHz
- timer 1 shall create an interrupt every 5.12 ms

First the PIT module is activated by writing a 0 to the MDIS bit in the PITCTRL register.

The 50 MHz clock frequency equates to a clock period of 20ns. Timer 1 needs to trigger every 5.12 ms/20 ns = 256000 cycles. The value for the LDVAL register trigger would be calculated as (period / clock period) -1.

This means that PIT_PIT_CH1_LDVAL with 0003E7FF hex.

The interrupt for Timer 1 is enabled by setting PIT_CH1_TCTRL[TIE] bit. The timer is started by writing a 1 to the PIT_CH1_TCTRL[TEN] bit.

The following example pseudo-code matches the described setup:

```
// turn on PIT
PIT_MCR = 0x00;

// RTI
PIT_RTI_LDVAL = 0x004C4B3F; // setup RTI for 5000000 cycles
PIT_RTI_TCTRL[TIE] = 1; // let RTI generate interrupts
PIT_RTI_TCTRL[TEN] = 1; // start RTI

// Timer 1
PIT_CH1_LDVAL = 0x0003E7FF; // setup timer 1 for 256000 cycles
PIT_CH1_TCTRL[TIE] = 1; // enable Timer 1 interrupts
PIT_CH1_TCTRL[TEN] = 1; // start timer 1
```

29.5.2 Low Power Mode – Using the RTI for System Wakeup

This section describes the use of the low power mode, both with and without use of the RTI timer for wakeup.

29.5.2.1 Low Power Mode Without RTI Wakeup

The SIU_HALT register may be used to place certain peripherals selectively in low power mode. See Section 3.2.1.30, "Halt Register (SIU_HLT)", for more information about the SIU_HLT register. The general process for setting low power modes is:

1. Disable interrupts for any peripherals that are to be placed in low power mode and ensure there are no pending interrupts.
2. Using the bit assignment in the SIU_HLT register, set the bits for any modules where low power operation is desired. Note that the CPU (SIU_HLT[0]) bit should not be selected.
3. The SIU_HLTACK register indicates when a selected peripheral clock has stopped.
4. To re-enable the clock to any peripheral, clear the corresponding bit in the SIU_HLT register and re-enable the interrupts for the desired peripheral. The SIU_HLTACK updates accordingly when the peripheral clock has been re-enabled.

NOTE

Most of the peripherals have a MDIS (module disable) bit in the module control register that can be set to disable the module clock, reducing power consumption. In most cases the peripheral registers are still readable and writeable. The SIU_HALT register performs the same function on multiple modules with a single 32-bit write, however using the SIU_HALT function also disables R/W function on the peripheral registers for additional power saving.

29.5.2.2 Low Power Mode With RTI Wakeup

The RTI can be used in conjunction with the SIU_HALT register to enter and exit from low power mode. See Section 3.2.1.30, "Halt Register (SIU_HLT)", for more information about the SIU_HALT register. The general procedure for entering and exiting low power mode is:

1. Configure the interrupt handler for the RTI interrupt. The interrupt handler code can vary, depending on what behavior is required for the application upon exiting low power mode. More information on configuring and using the interrupts is found in Section 27.5, "Initialization/Application Information", of the Interrupt Controller chapter.
2. Configure the RTI for the desired timeout period as described in Section 29.5.1, "Example Configuration".
3. Disable interrupts for all peripherals to be placed in low power mode and ensure there are no pending interrupts.
4. Write the desired bits to the SIU_HLT register, selecting the modules that are to have the low power mode enabled. Note that if the CPU bit is also set for lowest power operation, an RTI interrupt is required to exit low power mode.
5. The CPU executes the 'msync', 'isync', and 'wait' instructions and the device enters low power mode. Note that the 'msync' and 'isync' instructions ensure that all current core operations complete before entering low power mode.

6. At the programmed RTI time interval, the RTI timer triggers an interrupt that is serviced by the interrupt controller. (The interrupt controller is not put in low power mode by the SIU_HALT register.) This interrupt also re-enables the CPU clock so that full CPU operation is restored.
7. In the RTI interrupt handler, the SIU_HALT register may be modified to restore operation as desired. In some cases where periodic operation is preferred, the interrupt handler may perform a set of tasks, and then write the SIU_HALT register mask and re-enter the low power mode by executing the ‘msync’, ‘isync’, and ‘wait’ instructions again. The next RTI timeout repeats the process.

NOTE

The RTI is a convenient mechanism for waking up the CPU in a controlled state once placed in low power mode. However, the CPU will also exit low power mode under any of the following conditions:

- Any external interrupt from interrupt controller
- Critical interrupt
- NMI event
- Core watchdog timeout
- Core fixed interval timeout
- Core decrementer timeout
- Various debug events

Chapter 30
Peripheral Bridge (PBRIDGE)

30.1 Introduction

The PBRIDGE acts as an interface between the system bus and lower bandwidth peripherals.

30.1.1 Terminology

Table 30-1. Terms and Acronyms

Terms	Description
AHB 2.v6	AMBA AHB-lite version 2.0 with v6 extensions
AMBA AHB-Lite Interface	A standard AHB-lite bus interface
Pipeline	Act of initiating a bus cycle while another bus cycle is in progress. Thus, the bus can have multiple bus cycles pending at one time.
Slave	A bus slave is a device that responds to a bus transaction, but never initiates a cycle on the bus.
Transaction	A bus transaction consists of an address transfer (address phase) and one or more data transfer(s) (data phase).

30.1.2 Block Diagram

A simplified block diagram of the PBRIDGE illustrates the functionality and interdependence of major blocks (see Figure 30-1).

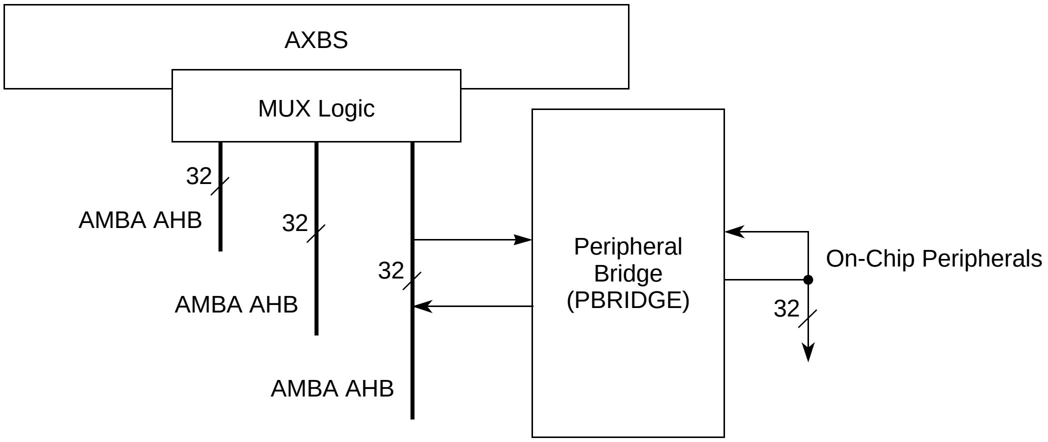


Figure 30-1. PBRIDGE Block Diagram

30.1.3 Features

The PBRIDGE has these major features:

- PBRIDGE supports the slave interface signals. This interface is meant for slave peripherals only.
- PBRIDGE supports 32-bit peripherals. (Byte, halfword, and word reads and write are supported to each.)
- Read and write accesses of 32 bits or less require two clocks, provided they do not cross a 32-bit boundary.
  - Read and write accesses that cross a 32-bit boundary are not supported.
- The peripherals connected to the PBRIDGE may be configured in groups to run at less than the system clock frequency.
- If a peripheral’s clock is disabled, PBRIDGE will generate a bus termination error if an access to that peripheral is attempted.

30.1.4 Modes of Operation

The PBRIDGE has only one operating mode.

30.2 External Signal Description

The PBRIDGE has no external signals.

30.3 Memory Map and Registers

The PBRIDGE does not contain any user-programmable registers.

30.4 Functional Description

The PBRIDGE serves as an interface between an AHB 2.v6 system bus and the peripheral interface bus. It functions as a protocol translator.

Accesses that fall within the address space of the PBRIDGE are decoded to provide individual module selects for peripheral devices on the peripheral bus interface.

See the peripherals section for a description of which peripherals are allocated to which 16 KB memory space in the PBRIDGE address map.

30.4.1 Read Cycles

Two-clock read accesses are possible with the PBRIDGE when the reference size is 32 bits or smaller. This module does not support any type of misaligned read access crossing a 32-bit boundary.

30.4.2 Write Cycles

Two-clock write accesses are possible with the PBRIDGE when the reference size is 32 bits or smaller. This module does not support any type of misaligned write access crossing a 32-bit boundary.

Chapter 31
Self-Test Control Unit (STCU)

31.1 Introduction

This chapter describes the configuration and operation of the Self-Test Control Unit (STCU). It begins with a general overview and purpose of the STCU and its operation. The remainder of the chapter describes the programming model and the intended use for the STCU. It also includes examples that the designer can follow to better understand how to configure the STCU for normal operation and how to stop it while it is running.

31.1.1 Glossary and Acronyms

Table 31-1. Acronyms

Term	Description
BIST	Built in Self Test
LBIST	Logic Built-in Self Test
TCU	Test Control Unit
PADI	Pad Interface
STCU	Self-Test Control Unit
FSM	Finite State Machine
MISR	Multiple Input Shift Register
CRC	Cyclic Redundancy Code
SoC	System-on-Chip
PLL	Phase Locked Loop

31.1.2 Features

The STCU contains the following features:

- Supports software to test the CPU Cores during normal operation.
- Allows software to run LBIST individually.
- Does not allow programming until unlocked by a specific key sequence.
- Able to stop in mid-sequence of the test by writing to the STCU_ABORT bit.

31.1.3 Modes of Operation

The primary operational modes are described in this section.

31.1.3.1 Normal Mode

Normal Mode is the main mode for the STCU. In normal mode, the STCU issues commands so that the LBIST engines run. When the LBIST engine finishes, the STCU latches the MISR value and interrupts the other core.

In Normal Mode, the CPU is able to abort the LBIST run at any point and allows for the core under test to be brought back to its normal mode for normal operation.

After any run in the STCU, the STCU must receive a soft reset command so that all state machines are in their initial state.

31.1.4 Block Diagram

The Block Diagram shows a high-level view of the STCU. The Functional Description of the STCU is provided in Section 31.4, “Functional Description.” The STCU interacts with the clock generation logic, the core under test, platform, LBIST state machines, and interrupt controller. Each of these connections is explained below.

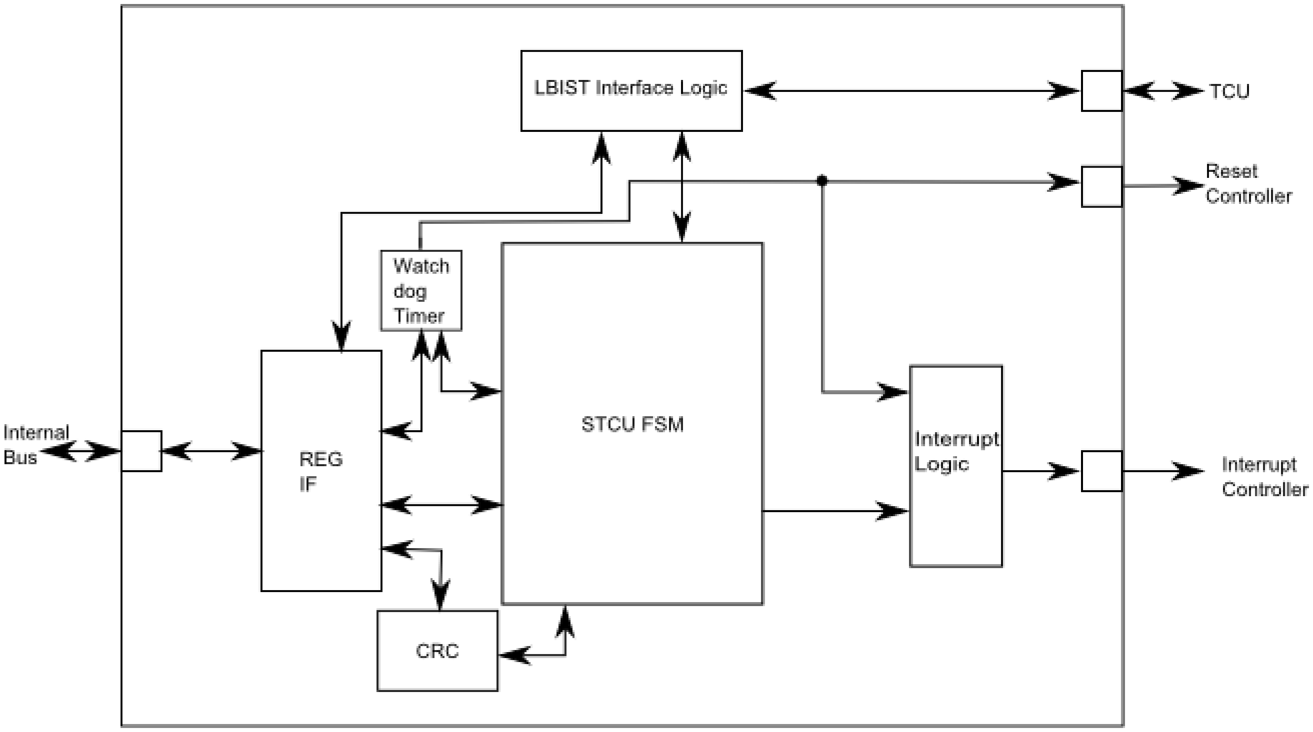


Figure 31-1. STCU Block Diagram

REG IF: Register Interface interacts with the internal bus to program the registers that are defined in the Memory Map Section.

Watchdog Timer: The Watchdog Timer provides a method of aborting LBIST execution after a programmable number of system clocks has elapsed. After the STCU starts, the watchdog timer starts counting from the value loaded in the TIMEOUT register. When the watchdog timer reaches zero the first time, the STCU issues an interrupt and reloads the watchdog timer with the initial TIMEOUT value. If the SoC does not react to the interrupt before the watchdog timer expires, the STCU requests a reset. When the STCU finishes testing, it resets the watchdog timer with the TIMEOUT value.

CRC: The Cyclic Redundancy Code uses the CRC-8-Dallas/Maxim polynomial ($x^8+x^5+x^4+x^1$). The CRC-8 engine monitors the outputs of the LBIST interface logic to make sure that the correct output sequence is sent to the LBIST State Machine.

STCU FSM: The STCU FSM coordinates the LBIST sequences and reports the status of the LBIST and watches for the watchdog expiring. After the STCU FSM finishes, it generates an interrupt to the Interrupt Controller to allow software to check the status of the STCU.

LBIST Interface Logic: The LBIST Interface Logic controls the LBIST and Clock Control outputs during an STCU test.

Interrupt Logic: The Interrupt Logic monitors the watchdog expired flag and the STCU Done flag. When an interrupt condition occurs, the STCU asserts its interrupt so that software can check the STCU.

31.2 External Signal Description

There are no external signals for the STCU.

31.3 STCU Memory Map

31.3.1 Memory Map & Register Descriptions

This section provides details of the STCU memory map. It also provides detailed descriptions for each of the registers.

Table 31-2. STCU memory map

Absolute address (hex) + 0xC3FF_4000	Register name	Width (in bits)	Access	Reset value	Section
0000_0000	(STCU_CTRL)	32	R/W	0000_0000h	31.3.2
0000_0004	(STCU_ENABLE)	32	R/W	0000_0000h	31.3.3
0000_0008	(STCU_STAT)	32	R	0000_20FFh	31.3.4
0000_000C	(STCU_WDGT)	32	R/W	FFFF_FFFFh	31.3.5
0000_0014	(STCU_KEY)	32	W (always reads zero)	0000_0000h	31.3.6
0000_0018	(STCU_LBIST_CTRL)	32	R/W	7000_0400h	31.3.7
0000_001C	(STCU_LBIST_PC_START)	32	R/W	0000_0000h	31.3.8

Table 31-2. STCU memory map

Absolute address (hex) + 0xC3FF_4000	Register name	Width (in bits)	Access	Reset value	Section
0000_0020	(STCU_LBIST_PC_END)	32	R/W	0000_0000h	31.3.9
0000_0024	(STCU_LBIST_PRPGH)	32	R/W	0000_0000h	31.3.10
0000_0028	(STCU_LBIST_PRPGL)	32	R/W	0000_0000h	31.3.11
0000_002C	(STCU_LBIST_ENABLE)	32	R/W	0000_0000h	31.3.12
0000_0030	(STCU_LBIST_STATUS)	32	R/W	0000_0000h	31.3.13
0000_0050	(STCU_INTERRUPT)	32	R/W	0000_0000h	31.3.14
0000_0054	(STCU_CURRENT_WDGT)	32	R	0000_000Xh	31.3.15
0000_0080	(STCU_LBIST_MISRH0)	32	R	0000_000Xh	31.3.16
0000_0084	(STCU_LBIST_MISRL0)	32	R	0000_000Xh	31.3.17
0000_0088	(STCU_LBIST_MISRH1)	32	R	0000_000Xh	31.3.18
0000_008C	(STCU_LBIST_MISRL1)	32	R	0000_000Xh	31.3.19

31.3.2 STCU Control Register (STCU_CTRL)

The STCU Control Register can be used to enable/disable LBIST and its self-test capabilities. It also provides software a method to see if there were any error injections in the last self-test run.

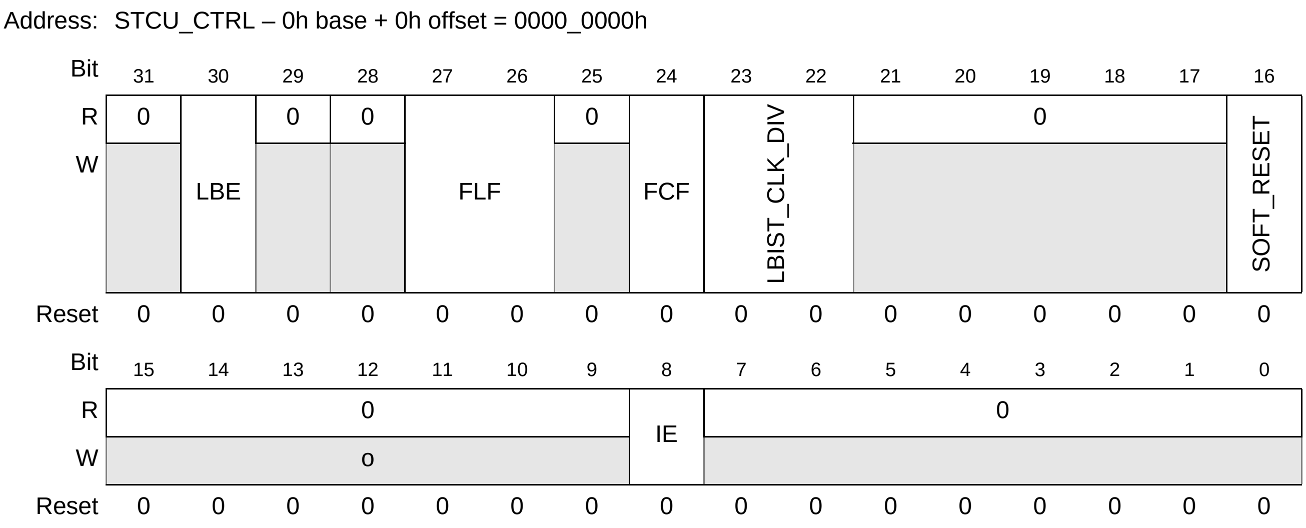


Figure 31-2. STCU Control Register (STCU_CTRL)

Table 31-3. STCU_CTRL field descriptions

Field	Description
31 Reserved	This read-only bit is reserved and always has the value zero.
30 LBE	Logical BIST Enabled When the STCU_ENABLE[STCU_START] bit is set, LBIST runs. LBE cannot be set at the same time as the STCU_ENABLE[STCU_START] bit. 0 LBIST does not run when STCU_ENABLE[STCU_START] bit is set. 1 LBIST runs when STCU_ENABLE[STCU_START] is set.
29 Reserved	This read-only bit is reserved and always has the value zero.
28 Reserved	This read-only bit is reserved and always has the value zero.
27–26 FLF	Force LBIST Failure The FLF selects what kind of LBIST Failure to put into the LBIST generation. The first puts a corrupted Pseudo-Random Start value into the LBIST register. This results in an unexpected MISR value. The second option uses the correct Pseudo-Random Start Value and corrupt the MISR value. 00 Do not force an LBIST error 01 Reserved 10 Corrupt the Pseudo-Random Start Value (STCU_LBIST_PRPGH and STCU_LBIST_PRPGL). 11 Corrupt the MISR Value.
25 Reserved	This read-only bit is reserved and always has the value zero.
24 FCF	Force CRC Failure 0 Do not force a CRC error 1 Force a CRC error. This corrupts the input values.
23–22 LBIST_CLK_DIV	LBIST Clock Divider Information 00 Use the current speed of system clock 01 Divide system clock by 2 10 Divide system clock by 4 11 Divide system clock by 8
21–17 Reserved	This read-only bit-field is reserved and always has the value zero.

Field	Description
16 SOFT_RESET	When this bit is asserted, the STCU resets its internal state machines and the following memory mapped registers/bit-fields. • STCU_ENABLE[STCU_START] • STCU_STAT[STCUR,LBISTR, DNE, LDNE, ABORT, INIT_SEQ, WDE, CRC_RESULT] • STCU_LBIST_STATUS The SOFT_RESET bit can be asserted directly by the CPU when it writes to the CTRL register, but it is also asserted when global soft reset is requested at the MCU level. Since soft reset is synchronous and has to follow a request/acknowledge procedure across clock domains, it may take some time to fully propagate its effect. The SOFT_RESET bit remains asserted while reset is pending, and is automatically negated when reset completes. Therefore, software can poll this bit to know when the soft reset has completed. Soft reset cannot be applied while clocks are shut down. 0 No reset requested. 1 Resets the registers mentioned in the field description.
15–9 Reserved	This read-only bit-field is reserved and always has the value zero.
8 IE	Interrupt Enable The Interrupt Enable bit allows interrupts to be sent to the Interrupt Controller. Internal interrupts are generated and stored without regard to this bit. 0 Disable asynchronous interrupts to the system. System software must poll the INTERRUPT[IFLAG] bit-field to know if an interrupt has occurred. 1 Enable asynchronous interrupts to the system.
7–0 Reserved	This read-only bit-field is reserved and always has the value zero.

31.3.3 STCU Enable Register (STCU_ENABLE)

The STCU Enable register starts the STCU sequence. By writing a 1 to the STCU_START bit, the STCU Finite State Machine runs.

If the SoC needs to abort the STCU while it is running, the SoC can write a 1 to the STCU_ABORT bit. When the abort is issued, the STCU stops LBIST by shifting out the appropriate information to the LBIST State Machine. After issuing the STCU_ABORT, the SoC should monitor the STCU_STATUS[ABORT] flag to see when the ABORT has finished. After issuing an abort, the SoC should also issue a soft-reset to return the STCU to its initial state.

Address: STCU_ENABLE – 0h base + 4h offset = 0000_0004h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0															0
W																STCU_ABORT
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0															0
W																STCU_START
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-3. STCU Enable Register (STCU_ENABLE)

Table 31-4. STCU_ENABLE field descriptions

Field	Description
31–17 Reserved	This read-only bit-field is reserved and always has the value zero.
16 STCU_ABORT	Abort the STCU engine Aborts the STCU's current operation. This also stops the Watchdog Timer. 0 Do not abort the STCU current operation. 1 Abort the STCU current operation

Field	Description
15–1 Reserved	This read-only bit-field is reserved and always has the value zero.
0 STCU_START	Start the STCU engine When set by software, this starts the STCU engine. 0 Do not start the STCU engine. 1 Start the STCU engine.

31.3.4 STCU Status Register (STCU_STAT)

The STCU Status Register gives the status of the different STCU functions and the results during the last STCU run. The STCU status register gives an overview of why the STCU failed on the last run. If the STCU causes a reset, the only bits that are reset are STCUR and LOCK. If another part of the SoC causes a reset, all bits are reset to 0.

Address: STCU_STAT – 0h base + 8h offset = 0000_0008h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0					STCUR	LBISTR	0	0	DNE	LDNE	0	0			0
W																
Reset	0	0	0	0	0	0	0	0	0	?	?	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	ABORT	LOCK	INIT_SEQ		WDE		0	CRC_RESULT							
W																
Reset	0	?	1	?	?	?	?	0	1	1	1	1	1	1	1	1

* Notes:

- WDE bit-field: DNE, LDNE, ABORT, INIT_SEQ, and WDE can only be cleared by a software reset. A hard reset does not change the values of these registers.

Figure 31-4. STCU Status Register (STCU_STAT)

Table 31-5. STCU_STAT field descriptions

Field	Description
31–27 Reserved	This read-only bit-field is reserved and always has the value zero.
26 STCUR	Indicates if the STCU is currently running. 0 The STCU is not currently running 1 The STCU is currently running
25 LBISTR	Indicates LBIST run status. 0 The LBIST portion is not running 1 The LBIST portion is running.
24 Reserved	This read-only bit is reserved and always has the value zero.
23 Reserved	This read-only bit is reserved and always has the value zero.
22 DNE	Indicates if the STCU finished running on the last test. 0 The STCU did not finish running during the last run. 1 The STCU did finish running during the last run.
21 LDNE	Indicates if the LBIST portion of the STCU finished. 0 The LBIST section did not finish on the run. 1 The LBIST section finished on the run.
20 Reserved	This read-only bit is reserved and always has the value zero.
19–17 Reserved	This read-only bit-field is reserved and always has the value zero.
16 Reserved	This read-only bit is reserved and always has the value zero.
15 Reserved	This read-only bit is reserved and always has the value zero.
14 ABORT	The ABORT sequence has finished. The ABORT sequence has completed. This bit is set after STCU_ENABLE[STCU_ABORT] is set or a soft reset has occurred. 0 The ABORT command has not finished. 1 The ABORT command has finished.
13 LOCK	Indicates if the Registers are locked for writes. 0 The STCU is unlocked and all registers are able to be written. 1 The STCU is locked and the only write allowed is to the STCU_UNLOCK register.
12–11 INIT_SEQ	00 The STCU did not run before the last reset. 01 The STCU was started by the IPS-interface 10 The JTAG Interfaces started the STCU. (Future Use) 11 Flash Fuse Loader started the STCU. (Future Use)

Field	Description
10–9 WDE	The Watch Dog Timer Expired on the last run. 00 The Watchdog Timer did not expire on the last run. 01 The Watchdog Timer expired on the last run. Clearing the interrupt stops the watchdog timer. 10 The test finished before the watchdog expired. Waiting for the Interrupt to be cleared. Clearing the interrupt stops the watchdog timer. 11 The Watchdog Timer expired twice and caused a system reset.
8 Reserved	This read-only bit is reserved and always has the value zero.
7–0 CRC_RESULT	Cyclic Redundancy Code Result The CRC result from the internal CRC engine on the state machine and other status signals. The value is determined during simulation and is reported. Writing a 1 to CF clears this field.

31.3.5 STCU WATCHDOG TIMER (STCU_WDGT)

The STCU WATCHDOG TIMER register has two functions. The first function is to make sure that the STCU does not go into an endless cycle without a way for the system to be notified and to stop the STCU. The second function is for the user to see how long the test took by reading the STCU_CURRENT_WDGT value. The user writes the number of clocks that the STCU is supposed to run until it signals that the watchdog timer has expired. After expiring the first time, the initial value is loaded back into the timer to count down again. After counting down a second time, a reset request is sent to the SoC.

Address: STCU_WEGT - 0h base + Ch offset = 0000_000Ch

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	WATCHDOG_TIMER																															
W																																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 31-5. STCU WATCHDOG TIMER (STCU_WDGT)

Table 31-6. STCU_WDGT field descriptions

Field	Description
31–0 WATCHDOG_TIMER	Watchdog Timer When the STCU is not running, writes are allowed to this register to set the watchdog timer. When the STCU is running, only reads are allowed and this register shows how many clocks are left until the watchdog expires.

31.3.6 STCU UNLOCK KEY (STCU_KEY)

The STCU UNLOCK KEY is used to unlock the STCU and allows for writes to occur. The STCU stays unlocked for 8192 cycles before locking again. This register is used to prevent runaway code from starting the STCU.

Address: STCU_KEY - 0h base + 14h offset - 0000_0014h

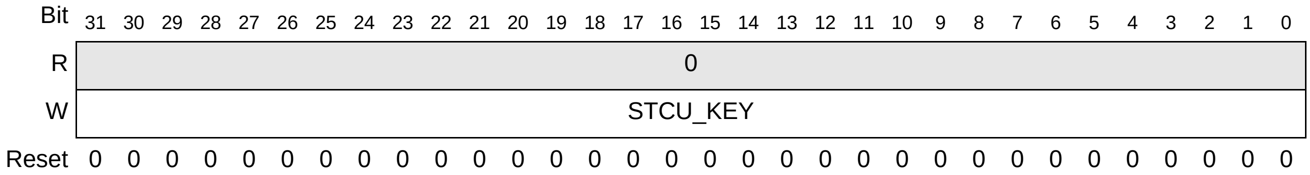


Figure 31-6. STCU UNLOCK KEY (STCU_KEY)

Table 31-7. STCU_KEY field descriptions

Field	Description
31–0 STCU_KEY	Value of the STCU KEY to write the value to. 0x953FA404 This is the first key that has to be written to start the unlock process. 0xF7EBA9E4 This is the second key that has to be written to unlock the STCU.

31.3.7 Logical BIST Control Register (STCU_LBIST_CTRL)

The Logical BIST Control Register controls the LBIST control logic. The control logic includes how long the scan enable is on, how long the scan enable is off, and the shift speed.

Address: STCU_LBIST_CTRL – 0h base + 18h offset = 0000_0018h

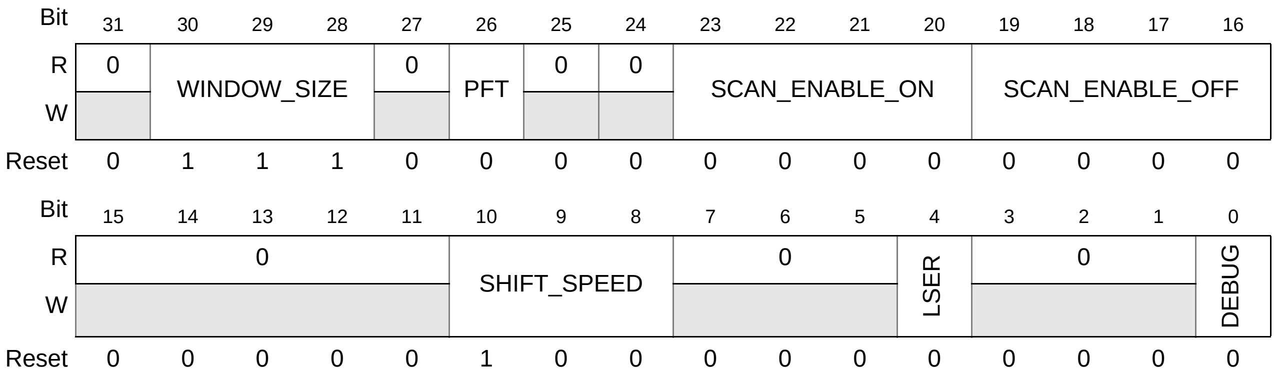


Figure 31-7. Logical BIST Control Register (STCU_LBIST_CTRL)

Table 31-8. STCU_LBIST_CTRL field descriptions

Field	Description
31 Reserved	This read-only bit is reserved and always has the value zero.
30–28 WINDOW_SIZE	The Window Size for the LBIST reads 000 Illegal Value 001 Controller waits 1 shift cycle for capture to finish. 010 Controller waits 2 shift cycle for capture to finish. 011 Controller waits 3 shift cycle for capture to finish. 100 Controller waits 4 shift cycle for capture to finish. 101 Controller waits 5 shift cycle for capture to finish. 110 Controller waits 6 shift cycle for capture to finish. 111 Controller waits 7 shift cycle for capture to finish.
27 Reserved	This read-only bit is reserved and always has the value zero.
26 PFT	Past Flush Test By default, the controller starts by applying flush test patterns, and stops when the pattern matches a hard-coded value (default value 32). In order to start the sequence beyond this hard-coded breakpoint, it is necessary to set this bit. 0 Use Flush Test Patterns 1 Do not use Flush Test Patterns.
25 Reserved	This read-only bit is reserved and always has the value zero.
24 Reserved	This read-only bit is reserved and always has the value zero.
23–20 SCAN_ENABLE_ON	Tells the controller how many clock cycles to delay the capture clock's rising edge after the scan enable signal is deasserted. For most applications, this is a zero, but if a speed path is expected, delaying the capture clock is a way of diagnosing a speed path. 0000 0 delay cycles. The Capture Clock occurs on the next rising edge. 0001 1 delay cycle. The capture clock is not enabled for 1 cycle. The Capture Clock happens 2 rising edges later. 0010 2 delay cycles. 0011 3 delay cycles. 0100 4 delay cycles. 0101 5 delay cycles. 0110 6 delay cycles. 0111 7 delay cycles. 1000 8 delay cycles 1001 9 delay cycles 1010 10 delay cycles 1011 11 delay cycles 1100 12 delay cycles 1101 13 delay cycles 1110 14 delay cycles 1111 15 delay cycles

Field	Description
19–16 SCAN_ENABLE_OFF	The SCAN_ENABLE_OFF values tell the LBIST controller how many clocks to mask after the capture clock and turning the scan_enable block on. The capture clock cycle occurs, then on the negative edge the scan_enable signal asserts. 0000 0 delay cycles. There are no clocks masked after the capture clock. 0001 1 delay cycle. There is 1 clock masked after the capture clock and scan_enable asserting. 0010 2 delay cycles. 0011 3 delay cycles. 0100 4 delay cycles. 0101 5 delay cycles. 0110 6 delay cycles. 0111 7 delay cycles. 1000 8 delay cycles. 1001 9 delay cycles. 1010 10 delay cycles. 1011 11 delay cycles. 1100 12 delay cycles. 1101 13 delay cycles. 1110 14 delay cycles. 1111 15 delay cycles.
15–11 Reserved	This read-only bit-field is reserved and always has the value zero.
10–8 SHIFT_SPEED	000 Shift at full (bist_clk) rate. 001 Shift at 1/2 bist_clk rate. 010 Shift at 1/3 bist_clk rate. 011 Shift at 1/4 bist_clk rate. 100 Shift at 1/5 bist_clk rate. 101 Shift at 1/6 bist_clk rate. 110 Shift at 1/7 bist_clk rate. 111 Shift at 1/8 bist_clk rate.
7–5 Reserved	This read-only bit-field is reserved and always has the value zero.
4 LSER	0 The LBIST engines run in parallel. 1 The LBIST engines run in serial.
3–1 Reserved	This read-only bit-field is reserved and always has the value zero.
0 DEBUG	Sets up the LBIST engines and only runs 256 clock cycles to see that the LBIST engine was configured correctly and ran. This is different from the Diagnostics Mode by not stopping after each pattern and using clock cycles. 0 The LBIST runs normally. 1 The LBIST runs a debug cycle.

31.3.8 LBIST Pattern Counter Start Register (STCU_LBIST_PC_START)

The LBIST_PC_START register contains the start value for the patterns that all LBIST engines run. One pattern means that the LBIST engine scans in a pattern, releases scan enable and scans out the resulting logic.

Address: STCU_LBIST_PC_START - 0h base + 1Ch offset = 0000_001Ch

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PC_START																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-8. LBIST Pattern Counter Start Register (STCU_LBIST_PC_START)

Table 31-9. STCU_LBIST_PC_START field descriptions

Field	Description
31–0 PC_START	The LBIST Pattern Counter Initial Value.

31.3.9 LBIST Pattern End Counter Register (STCU_LBIST_PC_END)

The LBIST_PC_END register contains the end value for the patterns that all LBIST engines run. One pattern means that the LBIST engine scans in a pattern, releases scan enable, and scans out the resulting logic.

Address: STCU_LBIST_PC_END - 0h base + 20 offset = 0000_0020h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	PC_END																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-9. LBIST Pattern End Counter Register (STCU_LBIST_PC_END)

Table 31-10. STCU_LBIST_PC_END field descriptions

Field	Description
31–0 PC_END	The LBIST Pattern Counter Final Value.

31.3.10 LBIST Pseudo-Random Number (STCU_LBIST_PRPGH)

The LBIST_PRPGH register is the most significant start value that goes into the Pseudo-Random Number Generator in the LBIST engine. Changing this value causes the LBIST engine to generate different scan patterns.

Address: STCU_LBIST_PRPGH - 0h base + 24th offset = 0000_0024h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	LBIST_PRPGH																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-10. LBIST Pseudo-Random Number (STCU_LBIST_PRPGH)

Table 31-11. STCU_LBIST_PRPGH field descriptions

Field	Description
31–0 LBIST_PRPGH	The LBIST Pseudo-Random Number Generator Initial Value.

31.3.11 LBIST Pseudo-Random Number (STCU_LBIST_PRPGL)

The LBIST_PRPGL register is the least-significant start value that goes into the Pseudo-Random Number Generator in the LBIST engine. Changing this value causes the LBIST engine to generate different scan patterns.

Address: STCU_LBIST_PRPGL - 0h base + 28h offset = 0000_0028h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	LBIST_PRPGL																															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-11. LBIST Pseudo-Random Number (STCU_LBIST_PRPGL)

Table 31-12. STCU_LBIST_PRPGL field descriptions

Field	Description
31–0 LBIST_PRPGL	The LBIST Pseudo-Random Number Generator Initial Value.

31.3.12 LBIST ENABLE REGISTER (STCU_LBIST_ENABLE)

Address: STCU_LBIST_ENABLE - 0h base + 2Ch offset = 0000_002Ch

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0																														LBE1	LBE0
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-12. LBIST ENABLE REGISTER (STCU_LBIST_ENABLE)

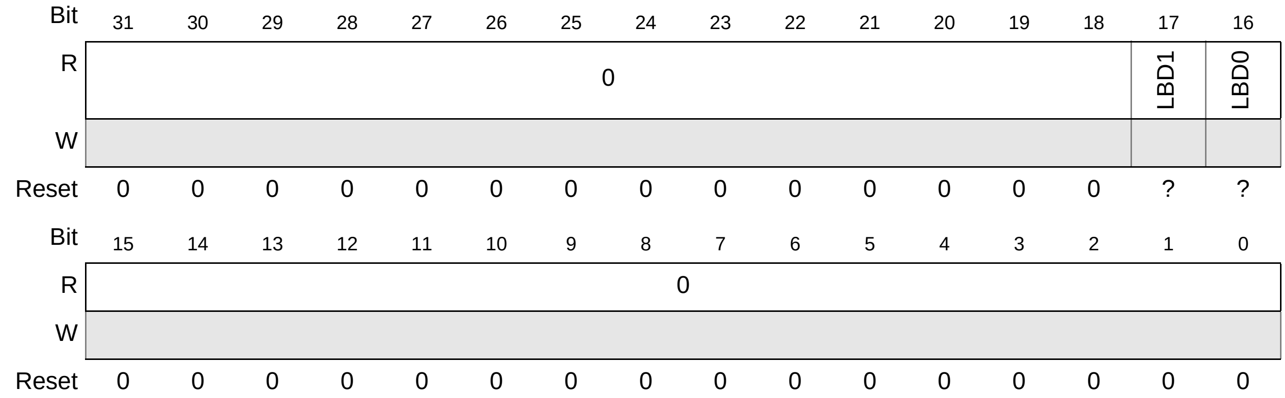
Table 31-13. STCU_LBIST_ENABLE field descriptions

Field	Description
31–2 Reserved	This read-only bit-field is reserved and always has the value zero.
1 LBE1	Logical BIST Enable 1 0 Do not enable LBIST engine 1. 1 Run LBIST engine 1 when LBE is set and STCU_ENABLE is set.
0 LBE0	Logical BIST Enable 0 0 Do not enable LBIST engine 0. 1 Run LBIST engine 0 when LBE is set and STCU_ENABLE is set.

31.3.13 LBIST STATUS Register (STCU_LBIST_STATUS)

The LBIST STATUS registers indicate which LBIST engine finished in the last run. It does not indicate whether the LBIST engines passed or failed. It is up to the process to compare the MISR values stored in the registers to expected values.

Address: STCU_LBIST_STATUS – 0h base + 30h offset = 0000_0030h



Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0														LBD1	LBD0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	?	?

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

* Notes:

- LBD0 bit-field:

Figure 31-13. LBIST STATUS Register (STCU_LBIST_STATUS)

Table 31-14. STCU_LBIST_STATUS field descriptions

Field	Description
31–18 Reserved	This read-only bit-field is reserved and always has the value zero.
17 LBD1	LBIST Engine 1 Done 0 LBIST Engine 1 has not finished. Writing a 1 to this field clears this bit and clears the value of LBIST_MISRH1 and LBIST_MISRL1. 1 LBIST Engine 1 has finished.
16 LBD0	LBIST Engine 0 Done 0 LBIST Engine 0 has not finished. Writing a 1 to this field clears this bit and clears the value of LBIST_MISRH0 and LBIST_MISRL0. 1 LBIST Engine 0 has finished.
15–0 Reserved	This read-only bit-field is reserved and always has the value zero.

31.3.14 STCU Interrupt Enable/Status Register (STCU_INTERRUPT)

The STCU Interrupt Enable/Status Register controls interrupt generation.

Address: STCU_INTERRUPT – 0h base + 50h offset = 0000_0050h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0															
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0															IFLAG
W																w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-14. STCU Interrupt Enable/Status Register (STCU_INTERRUPT)

Table 31-15. STCU_INTERRUPT field descriptions

Field	Description
31–1 Reserved	This read-only bit-field is reserved and always has the value zero.
0 IFLAG	Interrupt Acknowledge The Interrupt Flag can be read by the system software to see if an interrupt has occurred. This is useful if the system software implements a polling technique and not interrupt-driven software code. Writing a 1 to this bit clears the interrupt. Please note that the source of the interrupt (STCU Done, Watchdog Timeout) must be cleared for the interrupt to be cleared. 0 No interrupt is pending. 1 An interrupt is pending. Writing a 1 to this bit clears this bit.

31.3.15 STCU Current Watchdog Timer (STCU_CURRENT_WDGT)

The STCU Current Watchdog Timer stores the value of the Watchdog Timer from the last run. When the STCU DONE flag asserts, the current watchdog value is stored in this register to be read by the CPU for informational purposes.

This register can only be reset by a SOFT RESET command.

Address: STCU_CURRENT_WDGT - 0h base + 54h offset = 0000_0054h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	WATCHDOG_TIMER																															
W																																
Reset	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*

* Notes:

- x = Undefined at reset.

Figure 31-15. STCU Current Watchdog Timer (STCU_CURRENT_WDGT)

Table 31-16. STCU_CURRENT_WDGT field descriptions

Field	Description
31–0 WATCHDOG_TIMER	The current value in the watchdog timer.

31.3.16 LBIST 0 MISRH Register (STCU_LBIST_MISRH0)

LBIST_MISRH0 is the most significant 32 bits of the MISR register from the LBIST portion controlled by LBE0.

Address: STCU_LBIST_MISRH0 - 0h base + 80h offset = 0000_0080h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	LBIST_MISRH0																															
W																																
Reset	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*

* Notes:

- x = Undefined at reset.

Figure 31-16. LBIST 0 MISRH Register (STCU_LBIST_MISRH0)

Table 31-17. STCU_LBIST_MISRH0 field descriptions

Field	Description
31–0 LBIST_MISRH0	The LBIST_MISRH0 contains the most significant word of the MISR value for LBIST Engine 0. This is a read-only register and contains the MISR value from the last run of the LBIST. It is the responsibility of the User software to determine whether the MISR value is correct by comparing this register value to the expected value provided by Freescale.

31.3.17 LBIST 0 MISRL Register (STCU_LBIST_MISRL0)

The LBILBIST_MISRL0 is the least significant 32 bits of the MISR register from the LBIST portion controlled by LBE0.

Address: STCU_LBIST_MISRL0 - 0h base + 84h offset = 0000_0084h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	LBIST_MISRL0																															
W																																
Reset	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*

* Notes:

* x = Undefined at reset.

Figure 31-17. LBIST 0 MISRL Register (STCU_LBIST_MISRL0)

Table 31-18. STCU_LBIST_MISRL0 field descriptions

Field	Description
31–0 LBIST_MISRL0	The LBIST_MISRL0 contains the least significant word of the MISR value for LBIST Engine 0. This is a read-only register and contains the MISR value from the last run of the LBIST. It is the responsibility of the User software to determine whether the MISR value is correct by comparing this register value to the expected value provided by Freescale.

31.3.18 LBIST 1 MISRH Register (STCU_LBIST_MISRH1)

The LBIST_MISRH1 is the most significant 32 bits of the MISR register from the LBIST portion controlled by LBE1.

Address: STCU_LBIST_MISRH1 - 0h base + 88h offset = 0000_0088h

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	LBIST_MISRH1																															
W																																
Reset	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*

* Notes:

- x = Undefined at reset

Figure 31-18. LBIST 1 MISRH Register (STCU_LBIST_MISRH1)

Table 31-19. STCU_LBIST_MISRH1 field descriptions

Field	Description
31–0 LBIST_MISRH1	The LBIST_MISRH1 contains the most significant word of the MISR value for LBIST Engine 1. This is a read-only register and contains the MISR value from the last run of the LBIST. It is the responsibility of the User software to determine whether the MISR value is correct by comparing this register value to the expected value provided by Freescale.

31.3.19 LBIST 1 MISRL Register (STCU_LBIST_MISRL1)

The LBIST_MISRL1 is the least significant 32 bits of the MISR register from the LBIST portion controlled by LBE1.

Address: STCU_LBIST_MISRL1 - 0h base + 8Ch offset = 0000_008Ch

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	LBIST_MISRL1																															
W																																
Reset	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*	x*

* Notes:

- x = Undefined at reset.

Figure 31-19. LBIST 1 MISRL Register (STCU_LBIST_MISRL1)

Table 31-20. STCU_LBIST_MISRL1 field descriptions

Field	Description
31–0 LBIST_MISRL1	The LBIST_MISRL1 contains the least significant word of the MISR value for LBIST Engine 1. This is a read-only register and contains the MISR value from the last run of the LBIST. It is the responsibility of the User software to determine whether the MISR value is correct by comparing this register value to the expected value provided by Freescale.

31.4 Functional Description

This section describes the operation of the STCU, beginning with the hardware and software initialization sequence, then the software interface for determining failures.

Following the software initialization and operation sections are sections providing a detailed description of the STCU's function.

The Self-Test Control Unit (STCU) provides an interface for the user to run Logical BIST (LBIST) in the field. The user programs the STCU for the desired function and runs the enabled algorithms. After the STCU runs the required tests, it stores the results of the Built-in Self Tests, interrupts the system, and waits for the system to access the results. After interrupting the system, the STCU is expecting the system to reset, but the system does not have to do a full-system reset, just a reset on the portion of the SoC that was tested. The STCU still retains the test results when its interrupt signal is asserted.

31.4.1 Initialization Sequence

This section describes which registers are reset due to hardware reset, which are reset by STCU_CTRL[SOFT_RESET], and what locations the user must initialize prior to enabling the STCU.

31.4.1.1 Hardware Controlled Initialization

In the STCU, registers and control logic are reset by hardware (system reset). A system reset deasserts internal control signals and resets general configuration bits.

By setting the STCU_CTRL[SOFT_RESET] bit, the configuration control registers such as LBIST_PRPG and LBIST_PC_START are not reset, but the entire data path is reset. If STCU_ENABLE[STCU_ABORT] is asserted, the STCU should also be given a soft reset to purge any state machine data.

31.4.1.2 User Initialization (Prior to Asserting STCU_ENABLE[STCU_START])

The user needs to initialize portions of the STCU prior to setting the STCU_ENABLE[STCU_START] bit. The exact values depend on the particular application. The first two items and the last item in the sequence are important, but in between, the order is not important.

STCU registers requiring initialization are defined below.

Table 31-21. User Initialization (Before Asserting STCU_ENABLE[STCU_START])

Description
Write KEY1 to STCU_KEY.
Write KEY2 to STCU_KEY.
Initialize STCU_CTRL.
Write STCU_WDGT.
Initialize STCU_LBIST_CTRL.
Initialize STCU_LBIST_PC_START.
Initialize STCU_LBIST_PC_END.
Initialize STCU_LBIST_PRPGH.
Initialize STCU_LBIST_PRPGL.
Initialize STCU_LBIST_ENABLE.
Set STCU_ENABLE[STCU_START].

31.4.2 STCU Operation

This section describes the different STCU Operations during normal operation mode.

31.4.2.1 LBIST Functional Description

The Logic BIST section of the STCU performs a pseudo-random scan test of selected portions of the SoC. The system documentation provides more detail on which sections of the SoC the STCU runs LBIST.

The user sets up the STCU to run the LBIST. Using the system documentation, different Pseudo-Random Pattern Generators (PRPG), Pattern Count Start Values, and Pattern Count End Values can be loaded into the STCU to produce different patterns, coverage numbers, and MISR values. After the user has set up the LBIST control and starts the LBIST engine, the STCU starts to control the LBIST. The STCU sets up the LBIST’s registers and waits for the LBIST to finish. After the LBIST finishes, the STCU stores the MISR value, sets its interrupt and waits for the SoC to read the value of the MISR and compare it to known values.

31.4.3 Detailed Description

31.4.3.1 System Accesses to the STCU

The System can read from the STCU at any time. Reads do not corrupt the values of the registers and provide information from the last run of the STCU. The System must unlock the STCU to write to it. For the System to unlock the STCU, the system must write KEY1, then KEY2 to the STCU_KEY register. After unlocking the STCU, the STCU locks itself again after 8,192 clocks, when the System starts the STCU running, or when the System resets.

31.4.3.2 Setting up the STCU to run

After unlocking the STCU, the system can enable any LBIST engines to run. The system must write a 1 to the desired BIST engines. After enabling the individual BIST engines, it must enable the overall desired function bits in the STCU_CTRL register. For LBIST, STCU_CTRL[LBE] must be set.

31.4.3.2.1 LBIST Options

The first LBIST option is to run the LBIST engines in parallel or serial. This ability is controlled by LBIST_CTRL[LSER]. The second option is to run an LBIST debug cycle. During an LBIST debug cycle, the STCU writes the registers to LBIST engine, run several scan cycles, and then stop the LBIST. After stopping the LBIST, the STCU reads the status and see that it did start correctly. To consult more LBIST options, please look at the LBIST_CTRL register and see the possible configurations.

31.4.3.2.2 Forcing Failures

The System can set up the STCU to inject errors into the LBIST engines. The possible locations to force errors are given in the STCU_CTRL register.

31.4.4 Reset Description

Reset has to be handled carefully with the STCU. Since the STCU is highly destructive, a reset of the affected logic must be performed. If the SoC is designed to reset itself completely, the STCU has a mechanism so that the status values from the previous run are saved.

When the system issues a reset, the following sequence is followed to determine whether data needs to be saved. If the corresponding ENABLE bit and the corresponding DONE FLAG are set, then the LBIST_MISR register is stored. If the system wants these registers to reset, the system must clear the DONE and ENABLE signals. Please note writing a 1 to the LBIST_DONE registers clears the respective LBIST_MISR registers.

31.5 Application Information

The STCU as stated throughout this document is a very powerful block. In most systems, the STCU is run on one core as the other core is executing. This way, the other cores can stop the STCU or respond to a key-on scenario where the STCU needs to stop immediately and have the system go back to normal operations. The following timing diagrams show how the STCU reacts to different scenarios.

31.5.1 Normal Operation

Normal operation consists of the SoC writing to the STCU, starting the STCU, and waiting for the engines to finish. When the engines finish, the STCU sends an interrupt to the system. The following diagram shows the normal flow for the STCU and LBIST.

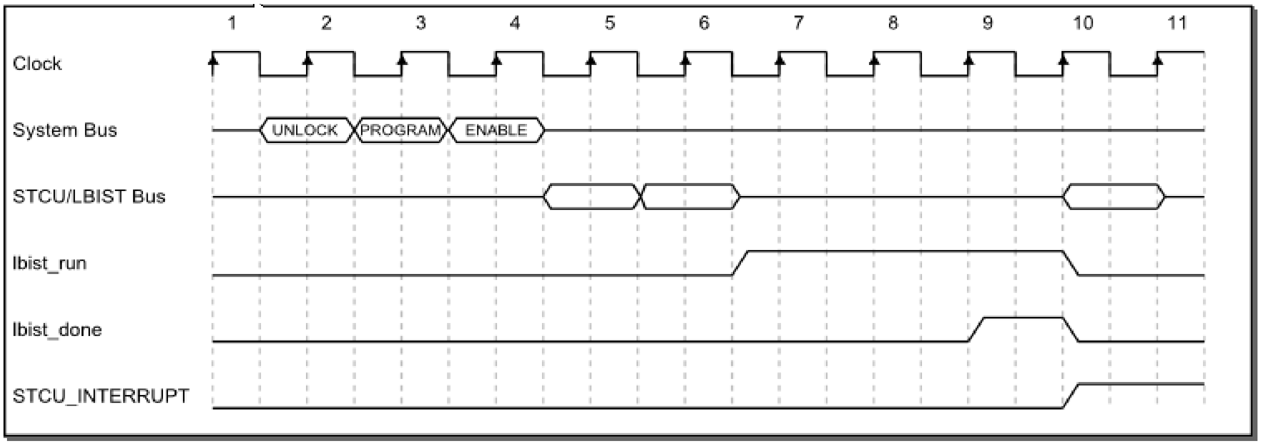


Figure 31-20. STCU Normal Operation

The above figure shows the system programming the STCU, the STCU programming the LBIST controller, and the LBIST running and finishing.

31.5.1.1 STCU Setup

The first two accesses are used to unlock the STCU. After unlocking the STCU, the SoC can setup the STCU to control the LBIST engines. The following sequence can be done in different orders, except for the last write.

1. Write to the STCU_CTRL register. LBE needs to be set, FLF set to 00, FCF set to be 0, and LBIST_CLK_DIV can be any four-bit number.
2. Write to the STCU_WDGT register. The value should be a number so that the tests can finish with some overhead. If the value is too small, the STCU stops and sends an interrupt to the system.
3. Write to the STCU_LBIST_CTRL register. For this register, the most common value to write is 0. If there is a need to change up the scan enable clocking, and shift speeds of the bist_clk, it is done here.
4. Write to the STCU_LBIST_PC_START and STCU_LBIST_PC_END register. The pattern counter register can be thought of as the following for-loop.

```
for (lcv = PC_START; lcv < PC_END; lcv++) {
   run_one_pattern();
}
```

5. Write to the STCU_LBIST_PRPGH and STCU_LBIST_PRPGL registers.
6. Write to the STCU_LBIST_ENABLE register. Select which LBIST engines to run.
7. Write to the STCU_INTERRUPT register. Set the IE bit. Write to the IACK bit to make sure that the Interrupt flag is cleared.

8. Write to the STCU_ENABLE register.

31.5.1.2 LBIST Control Accesses

After the STCU releases the LBIST controller from its reset state, it programs the LBIST through its serial interface. The STCU takes the values loaded into the LBIST registers (STCU_LBIST_CTRL, STCU_LBIST_PC, STCU_LBIST_PRPGH, STCU_LBIST_PRPGL) and places them into the LBIST controller. The basic access is to select the register, then send the data to the register. After setting up the LBIST controller, the STCU starts the LBIST running.

```
# of clocks per access = (9 + MISR_LENGTH) * (LBIST_CLK_DIV + 1)
```

At this point, the STCU waits for LBIST to signal it is done.

31.5.1.3 LBIST Completion

After the LBIST completes, the LBIST controller sets the STCU_STATUS[LBD0] or STCU_STATUS[LBD1] bit depending on which LBIST is active. At this point, the STCU latches the MISR value into the respective register. After storing the value, the STCU asserts its interrupt and waits for the SoC to read its status.

31.5.2 Abort due to Reset

One way to abort the STCU is to reset the SoC. When System Reset is asserted, the xbounding logic releases the inputs first to the xbounded core, then releases the outputs one cycle later.

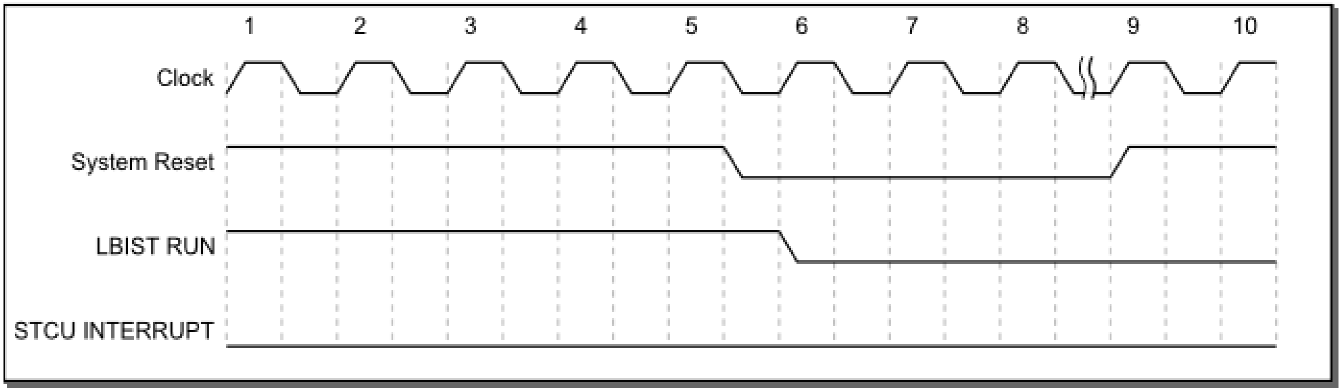


Figure 31-21. STCU Abort Due to Reset

31.5.3 Abort due to Kill

The SoC can stop the LBIST operation by writing to the STCU and writing the respective LBEn signal to a 0. By writing a 0 to the LBIST enable register, the STCU stops the LBIST controller and asserts its interrupt. Also, the ABORT bit in the STCU_STAT register is set to 1.

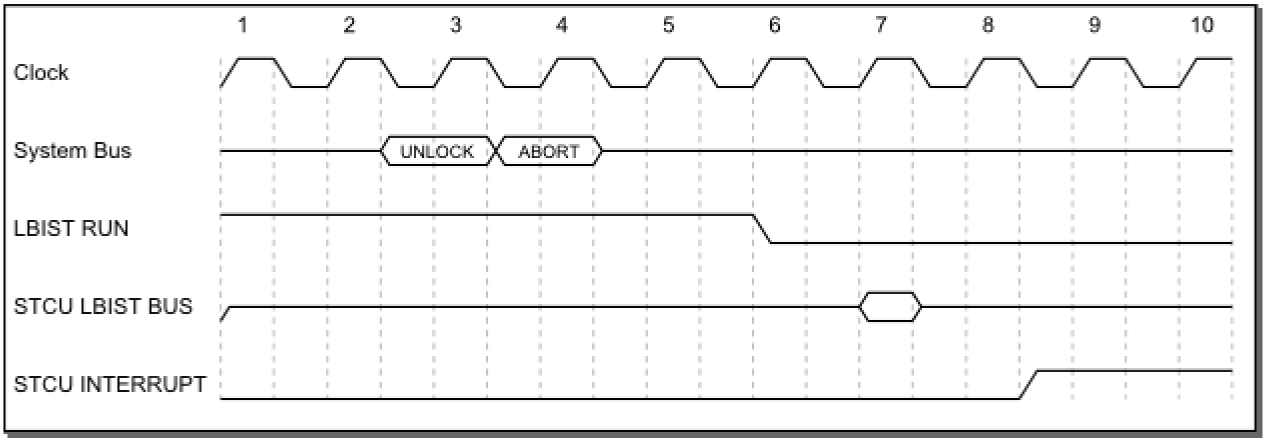


Figure 31-22. STCU Abort Due to Kill

31.5.4 Watchdog Timer

The Watchdog Timer may be programmed to force the termination of LBIST execution in the event it does not complete within a specified number of clock cycles. If the LBIST does not finish within the given number of System Clock cycles, the STCU issues an interrupt. The Watchdog Timer then resets to its initial value and runs until the Watchdog Error has been cleared or the watchdog timer reaches zero again. If the watchdog timer reaches zero without the interrupt being cleared, the STCU then requests that the system reset itself.

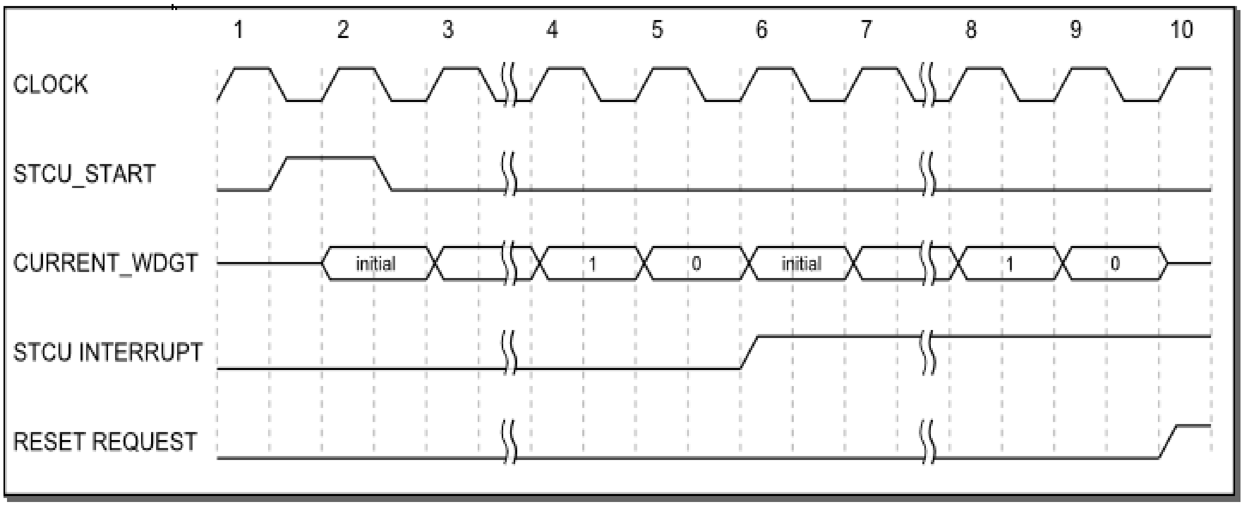


Figure 31-23. STCU Reset on Unserviced Interrupt

Chapter 32
Semaphores

32.1 Introduction

In a dual processor chip, semaphores are used to let each processor know who has control of common memory. Before a core can update or read memory coherently, it has to check the semaphore to see if the other core is not already updating the memory. If the semaphore is clear, it can write common memory, but if it is set, it has to wait for the other core to finish and clear the semaphore.

The semaphores module provides the hardware support needed in multi-core systems for implementing semaphores and provide a simple mechanism to achieve lock/unlock operations via a single write access. This approach eliminates architecture-specific implementations like atomic (indivisible) read-modify-write instructions or reservation mechanisms. The result is an architecture-neutral solution that provides hardware-enforced gates as well as other useful system functions related to the gating mechanisms.

32.1.1 Block Diagram

Figure 32-1 is a simplified block diagram of the semaphores module that illustrates the functionality and interdependence of major blocks. In the diagram, the register blocks named gate0, gate1, ..., gate 15 include the finite state machines implementing the semaphore gates plus the interrupt notification logic.

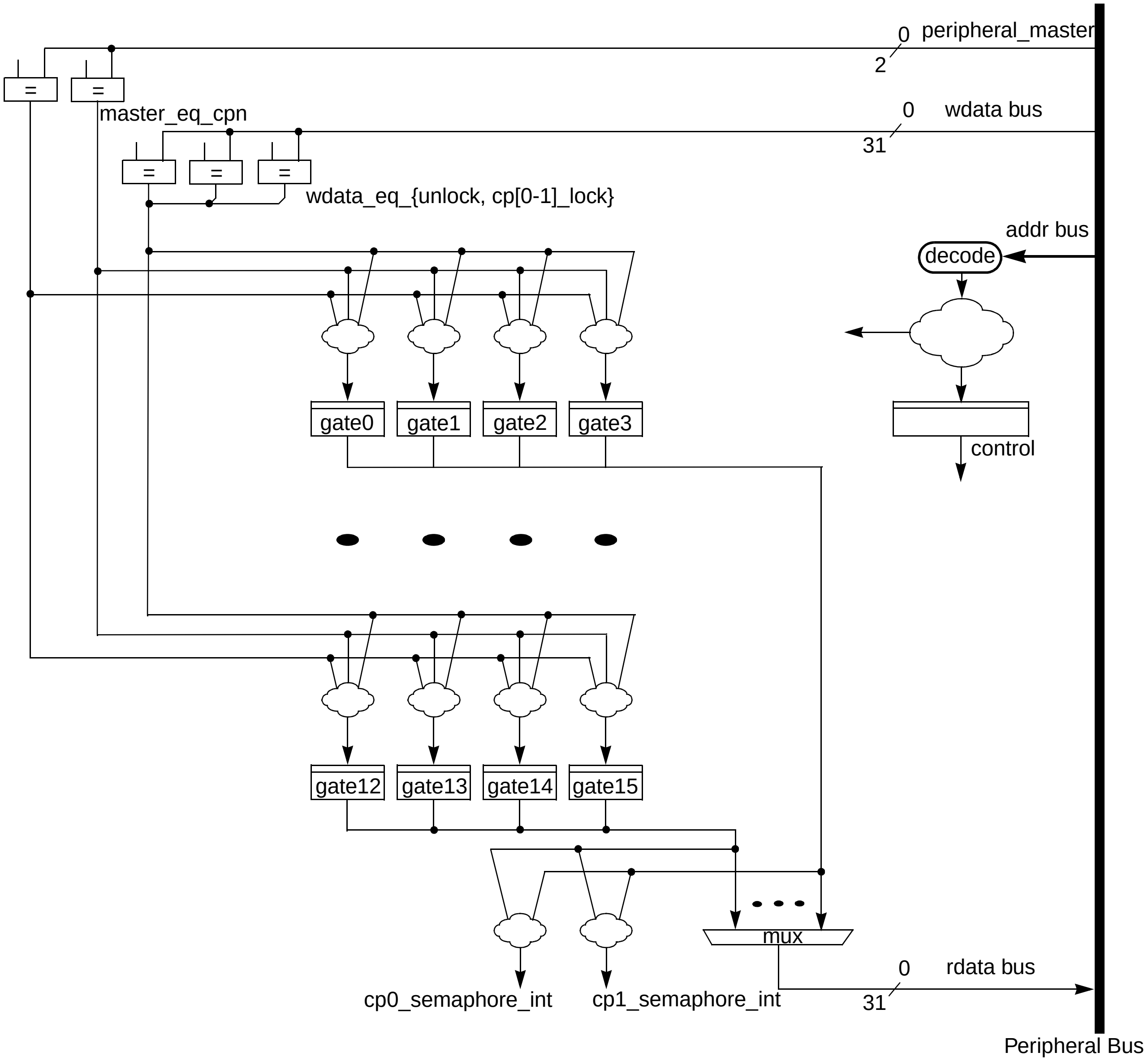


Figure 32-1. Semaphores Block Diagram

32.1.2 Features

The semaphores module implements hardware-enforced semaphores as a peripheral device and has these major features:

- Support for 16 hardware-enforced gates in a dual-processor configuration
  - Each hardware gate appears as a three-state, 2-bit state machine, with all 16 gates mapped as an array of bytes
    - Three-state implementation

      if gate = 0b00, then state = unlocked

      if gate = 0b01, then state = locked by core 0 (master ID = 0)

      if gate = 0b10, then state = locked by core 1 (master ID = 1)
    - Uses the bus master ID number as a reference attribute plus the specified data patterns to validate all write operations
    - After it is locked, the gate must be unlocked by a write of zeroes from the locking processor
  - Optionally enabled interrupt notification after a failed lock write provides a mechanism to indicate the gate is unlocked
  - Secure reset mechanisms are supported to clear the contents of individual semaphore gates or notification logic, and clear_all capability

32.1.3 Modes of Operation

The semaphores module does not support any special modes of operation.

32.2 Signal Description

The semaphores module does not include any external signals.

32.3 Memory Map and Registers

This section provides a detailed description of all semaphores registers.

32.3.1 Module Memory Map

The semaphores programming model map is shown in Table 32-1. The address of each register is given as an offset to the semaphore base address. Registers are listed in address order, identified by complete name and mnemonic, and list the type of accesses allowed.

Table 32-1. Semaphores Memory Map

Offset from SEMA4_BASE (0xFFF2_4000)	Register	Access	Reset Value	Section/ Page
0x0000	SEMA4_Gate00—Semaphores gate 0	R/W	0x00	32.3.2.1/32-5
0x0001	SEMA4_Gate01—Semaphores gate 1	R/W	0x00	32.3.2.1/32-5
0x0002	SEMA4_Gate02—Semaphores gate 2	R/W	0x00	32.3.2.1/32-5
0x0003	SEMA4_Gate03—Semaphores gate 3	R/W	0x00	32.3.2.1/32-5
0x0004	SEMA4_Gate04—Semaphores gate 4	R/W	0x00	32.3.2.1/32-5
0x0005	SEMA4_Gate05—Semaphores gate 5	R/W	0x00	32.3.2.1/32-5
0x0006	SEMA4_Gate06—Semaphores gate 6	R/W	0x00	32.3.2.1/32-5
0x0007	SEMA4_Gate07—Semaphores gate 7	R/W	0x00	32.3.2.1/32-5
0x0008	SEMA4_Gate08—Semaphores gate 8	R/W	0x00	32.3.2.1/32-5
0x0009	SEMA4_Gate09—Semaphores gate 9	R/W	0x00	32.3.2.1/32-5
0x000A	SEMA4_Gate10—Semaphores gate 10	R/W	0x00	32.3.2.1/32-5
0x000B	SEMA4_Gate11—Semaphores gate 11	R/W	0x00	32.3.2.1/32-5
0x000C	SEMA4_Gate12—Semaphores gate 12	R/W	0x00	32.3.2.1/32-5
0x000D	SEMA4_Gate13—Semaphores gate 13	R/W	0x00	32.3.2.1/32-5
0x000E	SEMA4_Gate14—Semaphores gate 14	R/W	0x00	32.3.2.1/32-5
0x000F	SEMA4_Gate15—Semaphores gate 15	R/W	0x00	32.3.2.1/32-5
0x0010–0x003F	Reserved			
00x040	SEMA4_CP0INE—Semaphores CP0 IRQ notification enable	R/W	0x0000	32.3.2.2/32-6
0x0042–0x0047	Reserved			
0x0048	SEMA4_CP1INE—Semaphores CP1 IRQ notification enable	R/W	0x0000	32.3.2.2/32-6
0x004A–0x07F	Reserved			
0x0080	SEMA4_CP0NTF—Semaphores CP0 IRQ notification	R	0x0000	32.3.2.3/32-7
0x008 2–00x087	Reserved			
0x0088	SEMA4_CP1NTF—Semaphores CP1 IRQ notification	R	0x0000	32.3.2.2/32-6
0x008A–0x00FF	Reserved			
0x0100	SEMA4_RSTGT—Semaphores reset gate	R/W	0x0000	32.3.2.4/32-8
0x0102	Reserved			

Table 32-1. Semaphores Memory Map (continued)

Offset from SEMA4_BASE (0xFFF2_4000)	Register	Access	Reset Value	Section/ Page
0x0104	SEMA4_RSTNTF—Semaphores reset IRQ notification	R/W	0x00000	32.3.2.5/32-10
0x0106–0x3FFF	Reserved			

32.3.2 Register Descriptions

This section lists the semaphores registers in address order and describes the registers and their bit fields.

32.3.2.1 Semaphores Gate *n* Register (SEMA4_GATE*n*)

Each semaphore gate is implemented in a 2-bit finite state machine, right-justified in a byte data structure. The hardware uses the bus master number in conjunction with the data patterns to validate all attempted write operations. Only processor bus masters can modify the gate registers. After it is locked, a gate must be opened (unlocked) by the locking processor core.

Multiple gate values can be read in a single access, but only a single gate at a time can be updated via a write operation. 16- and 32-bit writes to multiple gates are allowed, but the write data operand must update the state of a single gate only. A byte write data value of 0x03 is defined as no operation and does not affect the state of the corresponding gate register. Attempts to write multiple gates in a single-aligned access with a size larger than an 8-bit (byte) reference generate an error termination and do not allow any gate state changes.

Offset: SEMA4_BASE + *n* (*n* = 0, 1, 2,..., 15)

Access: User read/write

	0	1	2	3	4	5	6	7
R	0	0	0	0	0	0	GTFSM	
W								
Reset	0	0	0	0	0	0	0	0

Figure 32-2. SEMA4 Gate *n* Register (SEMA4_GATE*n*)

Table 32-2. SEMA4_GATE*n* Field Descriptions

Field	Description
GTFSM	Gate Finite State Machine. The hardware gate is maintained in a three-state implementation, defined as: 00 The gate is unlocked (free). 01 The gate has been locked by core 0. 10 The gate has been locked by core 1. 11 This state encoding is never used and therefore reserved. Attempted writes of 0x03 are treated as no operation and do not affect the gate state machine. **Note:** The state of the gate reflects the last processor that locked it, which can be useful during system debug.

32.3.2.2 Semaphores Processor *n* IRQ Notification Enable (SEMA4_CPnINE)

The application of a hardware semaphore module provides an opportunity for implementation of helpful system-level features. An example is an optional mechanism to generate a processor interrupt after a failed lock attempt. Traditional software gate functions execute a spin-wait loop in an effort to obtain and lock the referenced gate. With this module, the processor that fails in the lock attempt could continue with other tasks and allow a properly-enabled notification interrupt to return its execution to the original lock function.

The optional notification interrupt function consists of two registers for each processor: an interrupt notification enable register (SEMA4_CP*n*INE) and the interrupt request register (SEMA4_CP*n*NTF). To support implementations with more than 16 gates, these registers can be referenced with aligned 16- or 32-bit accesses. For the SEMA4_CP*n*INE registers, unimplemented bits read as zeroes and writes are ignored.

Offset: SEMA4_BASE + 0x0040 (SEMA4_CP0INE)
SEMA4_BASE + 0x0048 (SEMA4_CP1INE)

Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R W	INE0	INE1	INE2	INE3	INE4	INE5	INE6	INE7	INE8	INE9	INE10	INE11	INE12	INE13	INE14	INE15
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 32-3. Semaphores Processor *n* IRQ Notification Enable (SEMA4_CPnINE)

Table 32-3. SEMA4_CPnNTF Field Descriptions

Field	Description
INE*n*	Interrupt Request Notification Enable *n*. This field is a bitmap to enable the generation of an interrupt notification from a failed attempt to lock gate *n*. 0 The generation of the notification interrupt is disabled. 1 The generation of the notification interrupt is enabled.

32.3.2.3 Semaphores Processor *n* IRQ Notification (SEMA4_CPnNTF)

The notification interrupt is generated via a unique finite state machine, one per hardware gate. This machine operates in the following manner:

- When an attempted lock fails, the FSM enters a first state where it waits until the gate is unlocked.
- After it is unlocked, the FSM enters a second state where it generates an interrupt request to the failed lock processor.
- When the failed lock processor succeeds in locking the gate, the IRQ is automatically negated and the FSM returns to the idle state. However, if the other processor locks the gate again, the FSM returns to the first state, negates the interrupt request, and waits for the gate to be unlocked again.

The notification interrupt request is implemented in a 3-bit, five-state machine, where two specific states are encoded and program-visible as SEMA4_CP0NTF[GN*n*] and SEMA4_CP1NTF[GN*n*].

Offset: SEMA4_BASE + 0x0080 (SEMA4_CP0NTF)
SEMA4_BASE + 0x0088 (SEMA4_CP1NTF)

Access: User read-only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	GN0	GN1	GN2	GN3	GN4	GN5	GN6	GN7	GN8	GN9	GN10	GN11	GN12	GN13	GN14	GN15
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 32-4. Semaphores Processor *n* IRQ Notification (SEMA4_CPnNTF)

Table 32-4. SEMA4_CPnNTF Field Descriptions

Field	Description
GN*n*	Gate *n* Notification. This read-only field is a bitmap of the interrupt request notification from a failed attempt to lock gate *n*. 0 No notification interrupt generated. 1 Notification interrupt generated.

32.3.2.4 Semaphores (Secure) Reset Gate *n* (SEMA4_RSTGT)

Although the intent of the hardware gate implementation specifies a protocol where the locking processor must unlock the gate, it is recognized that system operation may require a reset function to re-initialize the state of any gate(s) without requiring a system-level reset.

To support this special gate reset requirement, the semaphores module implements a secure reset mechanism which allows a hardware gate (or all the gates) to be initialized by following a specific dual-write access pattern. Using a technique similar to that required for the servicing of a software watchdog timer, the secure gate reset requires two consecutive writes with predefined data patterns from the same processor to force the clearing of the specified gate(s). The required access pattern is:

1. A processor performs a 16-bit write to the SEMA4_RSTGT memory location. The most significant byte (SEMA4_RSTGT[RSTGDP]) must be 0xE2; the least significant byte is a “don’t care” for this reference.
2. The same processor then performs a second 16-bit write to the SEMA4_RSTGT location. For this write, the upper byte (SEMA4_RSTGT[RSTGDP]) is the logical complement of the first data pattern (0x1D) and the lower byte (SEMA4_RSTGT[RSTGTN]) specifies the gate(s) to be reset. This gate field can specify a single gate be cleared or that all gates are cleared.
3. Reads of the SEMA4_RSTGT location return information on the 2-bit state machine (SEMA4_RSTGT[RSTGSM]) which implements this function, the bus master performing the reset (SEMA4_RSTGT[RSTGMS]) and the gate number(s) last cleared (SEMA4_RSTGT[RSTGTN]). Reads of the SEMA4_RSTGT register do not affect the secure reset finite state machine in any manner.

Offset: SEMA4_BASE + 0x0100 (SEMA4_RSTGT) Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	RSTGSM		0	RSTGMS			RSTGTN							
W	RSTGDP															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 32-5. Semaphores (Secure) Reset Gate *n* (SEMA4_RSTGT)

Table 32-5. SEMA4_RSTGT Field Descriptions

Field	Description
RSTGSM	Reset Gate Finite State Machine. The reset state machine is maintained in a 2-bit, three-state implementation, defined as: 00 Idle, waiting for the first data pattern write. 01 Waiting for the second data pattern write. 10 The 2-write sequence has completed. Generate the specified gate reset(s). After the reset is performed, this machine returns to the idle (waiting for first data pattern write) state. 11 This state encoding is never used and therefore reserved. Reads of the SEMA4_RSTGT register return the encoded state machine value. Note the RSTGSM = 0b10 state is valid for a single machine cycle only, so it is impossible for a read to return this value.
RSTGMS	Reset Gate Bus Master. This 3-bit read-only field records the Master ID of the bus master performing the gate reset function. The reset function requires that the two consecutive writes to this register be initiated by the same bus master to succeed. This field is updated each time a write to this register occurs. **Master** — **Master ID** core 0 — 0 core 1 — 1 — — 2 — — 3 eDMA_A — 4 eDMA_B — 5 FlexRay — 6 — — 7
RSTGTN	Reset Gate Number. This 8-bit field specifies the specific hardware gate to be reset. This field is updated by the second write. If RSTGTN < 64, then reset the single gate defined by RSTGTN, else reset all the gates. The corresponding secure IRQ notification state machine(s) are also reset.
RSTGDP	Reset Gate Data Pattern. This write-only field is accessed with the specified data patterns on the two consecutive writes to enable the gate reset mechanism. For the first write, RSTGDP = 0xe2 while the second write requires RSTGDP = 0x1d.

32.3.2.5 Semaphores (Secure) Reset IRQ Notification (SEMA4_RSTNTF)

As with the case of the secure reset function and the hardware gates, it is recognized that system operation may require a reset function to re-initialize the state of the IRQ notification logic without requiring a system-level reset.

To support this special notification reset requirement, the semaphores module implements a secure reset mechanism which allows an IRQ notification (or all the notifications) to be initialized by following a specific dual-write access pattern. When successful, the specified IRQ notification state machine(s) are reset. Using a technique similar to that required for the servicing of a software watchdog timer, the secure reset mechanism requires two consecutive writes with predefined data patterns from the same processor to force the clearing of the IRQ notification(s). The required access pattern is:

1. A processor performs a 16-bit write to the SEMA4_RSTNTF memory location. The most significant byte (SEMA4_RSTNTF[RSTNDP]) must be 0x47; the least significant byte is a “don’t care” for this reference.
2. The same processor performs a second 16-bit write to the SEMA4_RSTNTF location. For this write, the upper byte (SEMA4_RSTNTF[RSTNDP]) is the logical complement of the first data pattern (0xb8) and the lower byte (SEMA4_RSTNTF[RSTNTN]) specifies the notification(s) to be reset. This field can specify a single notification be cleared or that all notifications are cleared.
3. Reads of the SEMA4_RSTNTF location return information on the 2-bit state machine (SEMA4_RSTNTF[RSTNSM]) that implements this function, the bus master performing the reset (SEMA4_RSTNTF[RSTNMS]) and the notification number(s) last cleared (SEMA4_RSTNTF[RSTNTN]). Reads of the SEMA4_RSTNTF register do not affect the secure reset finite state machine in any manner.

Offset: SEMA4_BASE + 0x0104 (SEMA4_RSTNTF) Access: User read/write

<table>
<tr><th></th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th></tr>
<tr><td>R</td><td>0</td><td>0</td><td colspan="2">RSTNSM</td><td>0</td><td colspan="3">RSTNMS</td><td colspan="8" rowspan="2">RSTNTN</td></tr>
<tr><td>W</td><td colspan="8">RSTNDP</td></tr>
<tr><td>Reset</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

Figure 32-6. Semaphores (Secure) Reset IRQ Notification (SEMA4_RSTNTF)

Table 32-6. SEMA4_SEMA4_RSTNTF Field Descriptions

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>RSTNSM</td><td>Reset Notification Finite State Machine. The reset state machine is maintained in a 2-bit, three-state implementation, defined as:
00 Idle, waiting for the first data pattern write.
01 Waiting for the second data pattern write.
10 The two-write sequence has completed. Generate the specified notification reset(s). After the reset is performed, this machine returns to the idle (waiting for first data pattern write) state.
11 This state encoding is never used and therefore reserved.
Reads of the SEMA4_RSTNTF register return the encoded state machine value. Note the RSTNSM = 0b10 state is valid for a single machine cycle only, so it is impossible for a read to return this value.</td></tr>
<tr><td>RSTNMS</td><td>Reset Notification Bus Master. This 3-bit read-only field records the Master ID of the bus master performing the notification reset function. The reset function requires that the two consecutive writes to this register be initiated by the same bus master to succeed. This field is updated each time a write to this register occurs.
<table>
<tr><th>Master</th><th>Master ID</th></tr>
<tr><td>core 0</td><td>0</td></tr>
<tr><td>core 1</td><td>1</td></tr>
<tr><td>—</td><td>2</td></tr>
<tr><td>—</td><td>3</td></tr>
<tr><td>eDMA_A</td><td>4</td></tr>
<tr><td>eDMA_B</td><td>5</td></tr>
<tr><td>FlexRay</td><td>6</td></tr>
<tr><td>—</td><td>7</td></tr>
</table></td></tr>
<tr><td>RSTNTN</td><td>Reset Notification Number. This 8-bit field specifies the specific IRQ notification state machine to be reset. This field is updated by the second write.
If RSTNTN < 64, then reset the single IRQ notification machine defined by RSTNTN, else reset all the notifications.</td></tr>
<tr><td>RSTNDP</td><td>Reset Notification Data Pattern. This write-only field is accessed with the specified data patterns on the two consecutive writes to enable the notification reset mechanism. For the first write, RSTNDP = 0x47 while the second write requires RSTNDP = 0xb8.</td></tr>
</table>

32.4 Functional Description

Multi-processor systems require a function that can be used to safely and easily provide a locking mechanism that is then used by system software to control access to shared data structures, shared hardware resources, and etc. These gating mechanisms are used by the software to serialize (and synchronize) writes to shared data and/or resources to prevent race conditions and preserve memory coherency between processes and processors.

For example, if processor X enters a section of code where shared data values are to be updated or read coherently, it must first acquire a semaphore. This locks, or closes, a software gate. After the gate has been locked, a properly architected software system does not allow other processes (or processors) to execute the same code segment or modify the shared data structure protected by the gate, that is, other processes/processors are locked out. Many software implementations include a spin-wait loop within the lock function until the locking of the gate is accomplished. After the lock has been obtained, processor X continues execution and updates the data values protected by the particular lock. After the updates are complete, processor X unlocks (or opens) the software gate, allowing other processes/processors access to the updated data values.

There are three important rules that must be followed for a correctly implemented system solution:

- All writes to shared data values or shared hardware resources must be protected by a gate variable.
- After a processor locks a gate, accesses to the shared data or resources by other processes/processors must be blocked. This is enforced by software conventions.
- The processor that locks a particular gate is the only processor that can unlock, or open, that gate.

Information in the hardware gate identifying the locking processor can be useful for system-level debugging.

The Hennessy/Patterson text on computer architecture offers this description of software gating:

> “One of the major requirements of a shared-memory architecture multiprocessor is being able to coordinate processes that are working on a common task. Typically, a programmer will use *lock variables* to synchronize the processes.
>
> The difficulty for the architect of a multiprocessor is to provide a mechanism to decide which processor gets the lock and to provide the operation that locks a variable. Arbitration is easy for shared-bus multiprocessors, since the bus is the only path to memory. The processor that gets the bus locks out all the other processors from memory. If the CPU and bus provide an atomic swap operation, programmers can create locks with the proper semantics. The adjective *atomic* is key, for it means that a processor can both read a location **and** set it to the locked value in the same bus operation, preventing any other processor from reading or writing memory.” [Hennessy/Patterson, *Computer Architecture: A Quantitative Approach*, ppg. 471-472]

The classic text continues with a description of the steps required to lock/unlock a variable using an atomic swap instruction.

> “Assume that 0 means unlocked and 1 means locked. A processor first reads the lock variable to test its state. A processor keeps reading and testing until the value indicates that the lock is unlocked. The processor then races against all other processes that were similarly “spin waiting” to see who can lock the variable first. All processes use a swap instruction that reads the old value

> and stores a 1 into the lock variable. The single winner will see the 0, and the losers will see a 1 that was placed there by the winner. (The losers will continue to set the variable to the locked value, but that doesn't matter.) The winning processor executes the code after the lock and then stores a 0 into the lock when it exits, starting the race all over again. Testing the old value and then setting to a new value is why the atomic swap instruction is called *test and set* in some instruction sets." [Hennessy/Patterson, *Computer Architecture: A Quantitative Approach*, ppg. 472-473]

The sole drawback to a hardware-based semaphore module is the limited number of semaphores versus the infinite number that can be supported with Power Architecture reservation instructions.

32.4.1 Semaphore Usage

Example 1: Inter-processor communication done with software interrupts and semaphores...

- Core 0 uses software interrupts to tell core 1 that new data is available, or core 1 does the same to tell core 0 that there is new data available for transmission.
- Because only eight software interrupts are available, the user may need RAM locations or general-purpose registers in the SIU to refine the meaning of the software interrupt.
- Messages are passed between cores in a defined section of system RAM.
- Before a core updates a message, it must check the associated semaphore to see if the other core is in the process of updating the same message. If the RAM not being updated, then the semaphore must first be locked, then the message can be updated. A software interrupt can be sent to the other core and the semaphore can be unlocked. If the RAM is being updated, the CPU must wait for the other core to unlock the semaphore before proceeding with update.
- Using the same memory location for bidirectional communication might be difficult, so two one-way message areas might work better.
  - For example, if both cores want to update the same location, then the following sequence may occur.

    1. Core 0 locks the semaphore, updates the memory, unlocks the semaphore, and generates a software interrupt to core 1.

    2. Before core 1 takes the software interrupt request, it finds the semaphore to be unlocked, so it writes new data to the memory.

    3. Core 1 software interrupt ISR reads the data sent to core 0, not the data sent from core 0, and performs an incorrect operation.
  - Semaphores do not prevent this situation from occurring.

Example 2: Coherent read done with semaphores...

- Core 1 wants to coherently read a section of shared memory.
- Core 1 should check that the semaphore for the shared memory is not currently set.
- Core 1 should set the semaphore for the shared memory to prevent core 0 from updating the shared memory.
- Core 1 reads the required data, then unlock the semaphore.

32.5 Initialization Information

The reset state of the semaphores module allows it to begin operation without the need for any further initialization. All the internal state machines are cleared by any reset event, allowing the module to immediately begin operation.

32.6 Application Information

In an operational multi-core system, most interactions involving the semaphores module involves reads and writes to the SEMA4_GATE*n* registers for implementation of the hardware-enforced software gate functions. Typical code segments for gate functions perform the following operations:

- To lock (close) a gate
  - The processor performs a byte write of logical_processor_number + 1 to gate[i]
  - The processor reads back gate[i] and checks for a value of logical_processor_number + 1

```
If the compare indicates the expected value
then the gate is locked; proceed with the protected code segment
else
lock operation failed;
repeat process beginning with byte write to gate[i] in spin-wait loop, or
proceed with another execution path and wait for failed lock interrupt notification
```

A simple C-language example of a gatelock function is shown in Example 32-1. This function follows the Hennessy/Patterson example.

Example 32-1. Sample Gatelock Function

```
#define UNLOCK      0
#define CP0_LOCK    1
#define CP2_LOCK    2

void gateLock (n)
int   n;                            /* gate number to lock */
{
  int   i;
  int   current_value;
  int   locked_value;

  i = processor_number();  /* obtain logical CPU number */

  if (i == 0)
    locked_value = CP0_LOCK;
  else
    locked_value = CP1_LOCK;

  /* read the current value of the gate and wait until the state == UNLOCK */
  do {
    current_value = gate[n];
  } while (current_value != UNLOCK);

  /* the current value of the gate == UNLOCK. attempt to lock the gate for this
```

```
     processor. spin-wait in this loop until gate ownership is obtained */
  do {
    gate[n] = locked_value;    /* write gate with processor_number + 1 */
    current_value = gate[n];  /* read gate to verify ownership was obtained */
  } while (current_value != locked_value);
}
```

- To unlock (open) a gate
  - After completing the protected code segment, the locking processor performs a byte write of zeroes to gate[i], unlocking (opening) the gate

In this example, a reference to processor_number() is used to retrieve this hardware configuration value. Typically, the logical processor numbers are defined by a hardwired input vector to the individual cores. The exact method for accessing the logical processor number varies by architecture. For Power Architecture cores, there is a processor ID register (PIR) which is SPR 286 and contains this value. A single instruction can be used to move the contents of the PIR into a general-purpose register: mfspr rx,286 where rx is the destination GPRn. Other architectures may support a specific instruction to move the contents of the logical processor number into a general-purpose register, e.g., **`rdcpn rx`** for a read CPU number instruction.

If the optional failed lock IRQ notification mechanisms are used, then accesses to the related registers (SEMA4_CP*n*INE, SEMA4_ CP*n*NTF) are required. There is no required negation of the failed lock write notification interrupt as the request is automatically negated by the semaphores module once the gate has been successfully locked by the failing processor.

Finally, in the event a system state requires a software-controlled reset of a gate or IRQ notification register(s), accesses to the secure reset control registers (SEMA4_RSTGT, SEMA4_RSTNTF) are required. For these situations, it is recommended that the appropriate IRQ notification enable(s) (SEMA4_CP*n*INE) bits be disabled before initiating the secure reset 2-write sequence to avoid any race conditions involving spurious notification interrupt requests.

32.7 DMA Requests

There are no DMA requests associated with the IPS_Semaphore block.

32.8 Interrupt Requests

The semaphore interrupt requests are connected to the interrupt controller as described in Chapter 27, “Interrupts and Interrupt Controller (INTC).”

Chapter 33
Software Watchdog Timers (SWT_A, SWT_B)

33.1 Introduction

This device has two independent Software Watchdog Timers, SWT_A and SWT_B. SWT_A is designed for use by software executing on Core 0, and SWT_B for software executing on Core 1. Both SWTs have identical functionality. The default reset states and interrupt signal routing are different for each SWT, and are described in detail in the following sections. Where descriptions are common to both SWT_A and SWT_B, the generic term “SWT” or “SWT_x” is used.

33.1.1 Overview

The Software Watchdog Timer (SWT) is a peripheral module that can prevent system lockup in situations such as software getting trapped in a loop or if a bus transaction fails to terminate. When enabled, the SWT require periodic execution of a watchdog servicing operation. The servicing operation resets the timer to a specified time-out period. If this servicing action does not occur before the timer expires the SWT generates an interrupt or hardware reset. The SWT can be configured to generate a reset or interrupt on an initial time-out; a reset is always generated on a second consecutive time-out.

The SWT interrupt is ‘ORed’ with the critical interrupt signal from the SIU and routed to the critical interrupt inputs of the CPU; see the SIU chapter for details.

The SWT includes an interrupt status bit so the ISR software can determine if the critical interrupt request came from the SWT or the external critical interrupt pin (WKPCFG_GPIO213).

The SWT can assert a reset when the watchdog timer expires. This reset will cause a system reset equivalent to assertion of the RESET pin. The SIU_RSR register in the SIU indicates that an SWT was the source of the reset.

33.1.2 Features

The SWT has the following features:

- 32-bit time-out register to set the time-out period
- Programmable selection of system or oscillator clock for timer operation
- Programmable selection of window mode or regular servicing
- Programmable selection of reset or interrupt on an initial time-out
- Programmable selection of fixed or keyed servicing
- Master access protection
- Hard and soft configuration lock bits

33.1.3 Modes of Operation

The SWT supports three device modes of operation: normal, debug and stop. When the SWT is enabled in normal mode, its counter runs continuously. In debug mode, operation of the counter is controlled by the FRZ bit in the SWT_MCR. If the FRZ bit is set, the counter is stopped in debug mode, otherwise it continues to run. This device does not have a stop mode.

33.2 External Signal Description

The SWT module does not have any external interface signals.

33.3 Memory Map and Register Definition

The SWT programming model has seven 32-bit registers. The programming model can only be accessed using 32-bit (word) accesses. References using a different size are invalid. Other types of invalid accesses include: writes to read only registers, incorrect values written to the service register when enabled, accesses to reserved addresses and accesses by masters without permission. If the RIA bit in the SWT_MCR is set then the SWT generates a system reset on an invalid access otherwise a bus error is generated. If either the HLK or SLK bits in the SWT_MCR are set then the SWT_MCR, SWT_TO, SWT_WN, SWT_SK registers are read only.

33.3.1 Memory Map

The SWT_A and SWT_B memory maps are shown in Table 33-2. The base address for each SWT is given in Table 33-2.

Table 33-1. Base address for SWT_B and SWT_A

SWT Base	Address
SWT_B_BASE	0xFFF3_4000
SWT_A_BASE	0xFFF3_8000

Table 33-2. SWT Memory Map

Offset from SWT_A_BASE or SWT_B_BASE	Register	Bits	Access	Reset Value	Section/Page
0x0000	SWT_MCR—SWT Module Control Register	32	R/W	0xFF00_010*[1]	33.3.2.1/3
0x0004	SWT_IR—SWT Interrupt Register	32	R/W	0x0000_0000	33.3.2.2/5
0x0008	SWT_TO—SWT Time-out Register	32	R/W	0x0005_FCD0	33.3.2.3/5
0x000C	SWT_WN—SWT Window Register	32	R/W	0x0000_0000	33.3.2.4/6
0x0010	SWT_SR—SWT Service Register	32	R/W	0x0000_0000	33.3.2.5/6
0x0014	SWT_CO—SWT Counter Output Register	32	R	0x0000_0000	33.3.2.6/7
0x0018	SWT_SK—SWT Service Key Register	32	R/W[2]	0x0000_0000	33.3.2.7/8
0x001C–0x3FFF	Reserved				

NOTES:

[1] Reset value is determined by SWT_MCR reset defined in Table 33-3.

[2] If neither HLK or SLK lock bit in the SWT_MCR is set, the SWT_SK can be initialized by software to a non-zero value.

33.3.2 Register Descriptions

The following sections detail the individual registers within the SWT programming model.

33.3.2.1 SWT Control Register (SWT_MCR)

The SWT_MCR contains fields for configuring and controlling the SWT. This register is read only if either the SWT_MCR[HLK] or SWT_MCR[SLK] bits are set. The SWT_MCR reset values for SWT_A and SWT_B differ, as shown in Table 33-3.

Table 33-3. SWT_MCR Reset Values

SWT Instance	SWT_MCR value
A	0xFF00010B[1]
B	0xFF00010A

NOTES:

[1] This value is present after a hardware reset and before the BAM executes. The BAM code may change it before application code executes.

Offset 0x0000 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	MAP 0	MAP 1	1	1	MAP 4	MAP 5	MAP 6	MAP 7	0	0	0	0	0	0	0	0
W			▨	▨					▨	▨	▨	▨	▨	▨	▨	▨
Reset	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	KEY	RIA	WND	ITR	HLK	SLK	CSL	0	FRZ	WEN
W	▨	▨	▨	▨	▨	▨								▨		
Reset	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	*[1]

Figure 33-1. SWT Control Register (SWT_MCR)

NOTES:

[1] Reset value is determined by SWT_MCR reset defined in Table 33-3.

Table 33-4. SWT_MCR Field Descriptions

Field	Description
0–1 4–7 MAP*n*	Master Access Protection for Master n where n represents the bus Master ID. A Nexus port shares its core's protection attribute. 0 = Access for the master is not enabled 1 = Access for the master is enabled Master IDs are listed in Table 11-1. Once set, MAP*n* bit is cleared only by other masters that are not disabled, or only after the reset.
2–3 8–21	Reserved
22 KEY	Keyed Service Mode. 0 = Fixed Service Sequence, the fixed sequence 0xA602, 0xB480 is used to service the watchdog 1 = Keyed Service Mode, two pseudorandom key values are used to service the watchdog
23 RIA	Reset on Invalid Access. 0 = Invalid access to the SWT generates a bus error 1 = Invalid access to the SWT causes a system reset if WEN=1 (watchdog enabled)
24 WND	Window Mode. 0 = Regular mode, service sequence can be done at any time 1 = Windowed mode, the service sequence is only valid when the down counter is less than the value in the SWT_WN register.
25 ITR	Interrupt Then Reset. 0 = Generate a reset on a time-out 1 = Generate an interrupt on an initial time-out, reset on a second consecutive time-out
26 HLK	Hard Lock. This bit is only cleared at reset. 0 = SWT_MCR, SWT_TO, SWT_WN and SWT_SK are read/write registers if SLK=0 1 = SWT_MCR, SWT_TO, SWT_WN and SWT_SK are read only registers
27 SLK	Soft Lock. This bit is cleared by writing the unlock sequence to the service register. 0 = SWT_MCR, SWT_TO, SWT_WN and SWT_SK are read/write registers if HLK=0 1 = SWT_MCR, SWT_TO, SWT_WN and SWT_SK are read only registers
28 CSL	Clock Selection. Selects the clock that drives the internal timer. 0 = System clock. 1 = Oscillator clock.
29	Reserved
30 FRZ	Debug Mode Control. Allows the watchdog timer to be stopped when the device enters debug mode. 0 = SWT counter continues to run in debug mode 1 = SWT counter is stopped in debug mode
31 WEN	Watchdog Enabled. 0 = SWT is disabled 1 = SWT is enabled

33.3.2.2 SWT Interrupt Register (SWT_IR)

The SWT_IR contains the time-out interrupt flag.

Offset 0x0004 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TIF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 33-2. SWT Interrupt Register (SWT_IR)

Table 33-5. SWT_IR Field Descriptions

Field	Description
0–30	Reserved
31 TIF	Time-out Interrupt Flag. The flag and interrupt are cleared by writing a 1 to this bit. Writing a 0 has no effect. 0 = No interrupt request. 1 = Interrupt request due to an initial time-out.

33.3.2.3 SWT Time-Out Register (SWT_TO)

The SWT Time-Out (SWT_TO) register contains the 32-bit time-out period. This register is read only if either the SWT_MCR[HLK] or SWT_MCR[SLK] bits are set.

Offset: 0x008 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	WTO																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	0	0	1	1	0	1	0	0	0	0

Figure 33-3. SWT Time-Out Register (SWT_TO)

Table 33-6. SWT_TO Register Field Descriptions

Field	Description
0–31 WTO	Watchdog time-out period in clock cycles. An internal 32-bit down counter is loaded with this value or 0x100 which ever is greater when the service sequence is written or when the SWT is enabled.

33.3.2.4 SWT Window Register (SWT_WN)

The SWT Window (SWT_WN) register contains the 32-bit window start value. This register is cleared on reset. This register is read only if either the SWT_MCR[HLK] or SWT_MCR[SLK] bits are set.

Offset 0x00C

Access: Read/Write

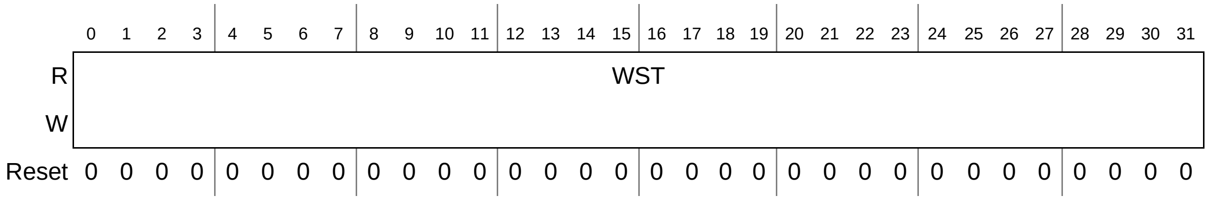


Figure 33-4. SWT Window Register (SWT_WN)

Table 33-7. SWT_WN Register Field Descriptions

Field	Description
0–31 WST	Window start value. When window mode is enabled, the service sequence can only be written when the internal down counter is less than this value.

33.3.2.5 SWT Service Register (SWT_SR)

The SWT Time-Out (SWT_SR) service register is the target for service operation writes used to reset the watchdog timer.

Offset 0x010

Access: Read/Write

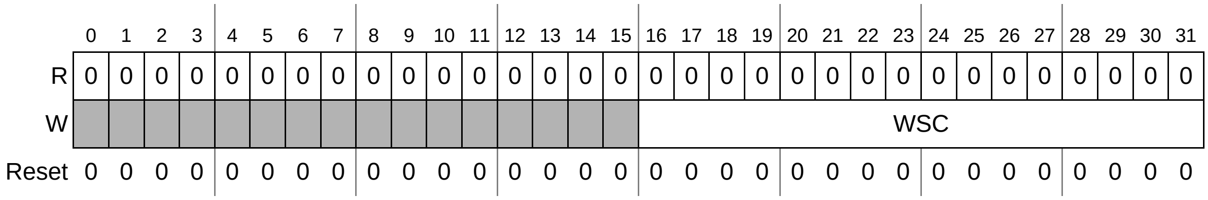


Figure 33-5. SWT Service Register (SWT_SR)

Table 33-8. SWT_SR Field Descriptions

Field	Description
0–15	Reserved
16–31 WSC	Watchdog Service Code.This field is used to service the watchdog and to clear the soft lock bit (SWT_MCR[SLK]). If the SWT_MCR[KEY] bit is set, two pseudorandom key values are written to service the watchdog, see section Section 33.4, “Functional Description”, for details. Otherwise, the sequence 0xA602 followed by 0xB480 is written to the WSC field. To clear the soft lock bit (SWT_MCR[SLK]), the value 0xC520 followed by 0xD928 is written to the WSC field.

33.3.2.6 SWT Counter Output Register (SWT_CO)

The SWT Counter Output (SWT_CO) register is a read only register that shows the value of the internal down counter when the SWT is disabled.

Offset 0x014 Access: Read Only

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CNT																															
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 33-6. SWT Counter Output Register (SWT_CO)

Table 33-9. SWT_CO Register Field Descriptions

Field	Description
0–31 CNT	Watchdog Count. When the watchdog is disabled (SWT_MCR[WEN]=0) this field shows the value of the internal down counter. When the watchdog is enabled the value of this field is 0x0000_0000. Values in this field can lag behind the internal counter value for up to six system plus eight counter clock cycles. Therefore, the value read from this field immediately after disabling the watchdog may be higher than the actual value of the internal counter.

33.3.2.7 SWT Service Key Register (SWT_SK)

The SWT Service Key (SWT_SK) register holds the previous (or initial) service key value. This register is read only if either the SWT_MCR[HLK] or SWT_MCR[SLK] bits are set.

Offset 0x018 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SK[1]															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

NOTES:

[1] If neither HLK or SLK lock bit in the SWT_MCR is set, the SWT_SK can be initialized by software to a non-zero value.

Figure 33-7. SWT Service Key Register (SWT_SK)

Table 33-10. SWT_SK Field Descriptions

Field	Description
0–15	Reserved
16–31 SK	Service Key. This field is the previous (or initial) service key value used in keyed service mode. If SWT_MCR[KEY] is set, the next key value to be written to the SWT_SR is (17*SK+3) mod 2^{16}.

33.4 Functional Description

The SWT is a 32-bit timer designed to enable the system to recover in situations such as software getting trapped in a loop or if a bus transaction fails to terminate. It includes a control register (SWT_MCR), an interrupt register (SWT_IR), a time-out register (SWT_TO), a window register (SWT_WN), a service register (SWT_SR), a counter output register (SWT_CO) and a service key register (SWT_SK).

The SWT_MCR includes bits to enable the timer, set configuration options and lock configuration of the module. The watchdog is enabled by setting the SWT_MCR[WEN] bit. The watchdog starts operation automatically after reset is released (WEN=1).

The SWT_TO register holds the watchdog time-out period in clock cycles unless the value is less than 0x100 in which case the time-out period is set to 0x100. This time-out period is loaded into an internal 32-bit down counter when the SWT is enabled and each time a valid service operation is performed. The SWT_MCR[CSL] bit selects which clock (system or oscillator) is used to drive the down counter.

The configuration of the SWT can be locked through use of either a soft lock or a hard lock. In either case, when locked the SWT_MCR, SWT_TO, SWT_WN and SWT_SK registers are read only. The hard lock is enabled by setting the SWT_MCR[HLK] bit which can only be cleared by a reset. The soft lock is enabled by setting the SWT_MCR[SLK] bit and is cleared by writing the unlock sequence to the service register. The unlock sequence is a write of 0xC520 followed by a write of 0xD928 to the SWT_SR[WSC] field. There is no timing requirement between the two writes. The unlock sequence logic ignores service sequence writes and recognizes the 0xC520, 0xD928 sequence regardless of previous writes. The unlock sequence can be written at any time and does not require the SWT_MCR[WEN] bit to be set.

When enabled, the SWT requires periodic execution of a servicing operation which consists of writing two values to the SWT_SR. Writing the proper sequence of values loads the internal down counter with the time-out period. There is no timing requirement between the two writes and the service sequence logic ignores unlock sequence writes. If the SWT_MCR[KEY] bit is zero, the fixed sequence 0xA602, 0xB480 is written to the SWT_SR[WSC] field to service the watchdog. If the SWT_MCR[KEY] bit is set, then two pseudorandom keys are written to the SWT_SR[WSC] field to service the watchdog. The key values are determined by the pseudorandom key generator defined in Figure 33-8. This algorithm will generate a sequence of 2^{16} different key values before repeating. The state of the key generator is held in the SWT_SK register. For example, if SWT_SK[SK] is 0x0100 then the service sequence keys are 0x1103, 0x2136. In this mode, each time a valid key is written to the SWT_SR register, the SWT_SK register is updated. So, after servicing the watchdog by writing 0x1103 and then 0x2136 to the SWT_SR[WSC] field, SWT_SK[SK] is 0x2136 and the next key sequence is 0x3499, 0x7E2C.

$$SK_{n+1} = (17 * SK_n + 3) \bmod 2^{16}$$

Figure 33-8. Pseudorandom Key Generator

Accesses to SWT registers occur with no peripheral bus wait states. (The peripheral bus bridge may add one or more system wait states.) However, due to synchronization logic in the SWT design, recognition of the service sequence or configuration changes may require up to three system plus seven counter clock cycles.

If window mode is enabled (SWT_MCR[WND] bit is set), the service sequence must be performed in the last part of the time-out period defined by the window register. The window is open when the down counter is less than the value in the SWT_WN register. Outside of this window, service sequence writes are invalid accesses and generate a bus error or reset depending on the value of the SWT_MCR[RIA] bit. For example, if the SWT_TO register is set to 5000 and SWT_WN register is set to 1000 then the service sequence must be performed in the last 20% of the time-out period. There is a short lag in the time it takes for the window to open due to synchronization logic in the watchdog design. This delay could be up to three system plus four counter clock cycles.

The interrupt then reset bit (SWT_MCR[ITR]) controls the action taken when a time-out occurs. If the SWT_MCR[ITR] bit is not set, a reset is generated immediately on a time-out. If the SWT_MCR[ITR] bit is set, an initial time-out causes the SWT to generate an interrupt and load the down counter with the time-out period. If the service sequence is not written before the second consecutive time-out, the SWT generates a system reset. The interrupt is indicated by the time-out interrupt flag (SWT_IR[TIF]). The interrupt request is cleared by writing a one to the SWT_IR[TIF] bit. Refer to Section 27.4.1, “External Interrupt Request Sources”, and Section 3.2.1.5, “DMA/Interrupt Request Enable Register (SIU_DIRER)”, for details on the enabling and routing of the SWT interrupt signals.

The SWT_CO register shows the value of the down counter when the watchdog is disabled. When the watchdog is enabled this register is cleared. The value shown in this register can lag behind the value in the internal counter for up to six system plus eight counter clock cycles.

The SWT_CO can be used during a software self test of the SWT. For example, the SWT can be enabled and not serviced for a fixed period of time less than the time-out value. Then the SWT can be disabled (SWT_MCR[WEN] cleared) and the value of the SWT_CO read to determine if the internal down counter is working properly.

Chapter 34
Shared Time Angle Counter Bus (STAC)

Content for this chapter to be added later.

Chapter 35
System RAM (SRAM)

35.1 Introduction

The MPC5676R includes 384KB on-chip general-purpose SRAM including 48KB of standby RAM. The first 48K of SRAM is powered by its own power supply pin during standby operation.

35.2 Features

The SRAM controller includes these features:

- Supports read/write accesses mapped to the SRAM memory from any master
- 48K- block powered by separate supply for standby operation
- Byte, halfword, word and doubleword addressable
- 7-bit ECC

35.3 Modes of Operation

35.3.1 Normal (Functional) Mode

Allows reads and writes of the SRAM memory arrays.

35.3.2 Standby Mode

Preserves contents of the standby portion of the memory when the 1.2 V (V_{DD}) power drops below the level of the standby power supply voltage. There are two possible supplies for standby: 1.0 V directly from the VSTBY pin and 2 – 5 volts (also on the VSTBY pin), which enables a standby regulator.

Updates to the standby portion of the SRAM are inhibited during system reset or during Standby Mode.

35.4 Block Diagram

The SRAM block diagram is shown in Figure 35-1.

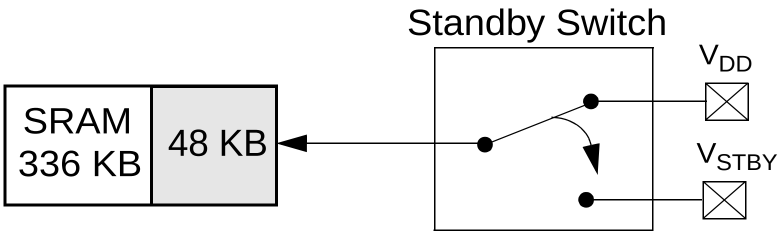


Figure 35-1. SRAM Block Diagram

35.5 External Signal Description

The external signal for SRAM is the V_{STBY} RAM power supply. If the standby feature of the SRAM is not used, tie the V_{STBY} pin to V_{SS}.

35.5.1 Register Memory Map

The SRAM occupies 384K of memory starting at the base address as shown in Table 1-2.

The internal SRAM has no registers. Registers for the SRAM ECC are located in the ECSM. See Chapter 22, “Error Correction Status Module (ECSM).

NOTE

> The ECSM module contains the register MUDCR that enables SRAM to be configured with an additional wait state. This is required when the CPU is configured to operate at its maximum frequency. See Section 18.4.3, Miscellaneous User-Defined Control Register (ECSM_MUDCR), for details.

35.6 Functional Description

ECC checks are performed during the read portion of an SRAM ECC read/write (R/W) operation, and ECC calculations are performed during the write portion of a read/write (R/W) operation. Because the ECC bits can contain random data after the device is powered on, you must initialize the SRAM by executing 32-bit write instructions to the entire SRAM. For more information, see Section 35.8, “Initialization and Application Information.

For software compatibility with other members of the MPC5500 and MPC5600 families, 64-bit writes can be used to initialize the ECC. This initializes the ECC bits of two 32-bit words at a time.

35.7 SRAM ECC Mechanism

The SRAM ECC detects the following conditions and produces the following results:

- Detects and corrects all 1-bit errors
- Detects and flags all 2-bit errors as non-correctable errors

SRAM does not detect all errors greater than 2 bits. Internal SRAM writes are done on byte boundaries:

- 1 byte (0:7 bits)
- 2 bytes (0:15 bits)
- 4 bytes or 1 word (0:31 bits)

If the entire 32 data bits are written to SRAM, no read operation is performed and the ECC is calculated across the 32 bits of data. The 7-bit ECC is appended to the data segment and written to SRAM. If the write operation is less than the entire 32-bit data width (1- or 2-byte segment), the following occurs:

1. The ECC mechanism checks the entire 32 bits of data for errors, detecting and either correcting or flagging errors.

2. The write data bytes (1- or 2-byte segment) are merged with the corrected 32 bits on the data bus.
3. The ECC is then calculated on the resulting 32 bits formed in the previous step.
4. The 7-bit ECC result is appended to the 32 bits from the data, and the 39-bit value is then written to SRAM.

35.7.1 Access Timing

The system bus is a two stage pipelined bus, which makes the timing of any access dependent on the access during the previous clock. Table 35-1 shows the wait states for accesses, current is the access being measured, **previous** is the RAM access during the previous clock.

Table 35-1. Number of wait states required for RAM operation

Current operation	Previous operation	Number of wait states
Read	Idle	0[1] / 1[2]
	Read	0[1]/ 1[2]
	32 or 64-bit write	0[1] / 1[2]
	8 or 16-bit write	1[1] / 2[2]
32 or 64-bit write	Idle	0
	Read	0
	32 or 64-bit write	0
	8 or 16-bit write	1
8 or 16-bit write	Idle	0
	Read	0
	32 or 64-bit write	0
	8 or 16-bit write	1

[1] Applies if additional SRAM read wait state in ECSM_MUDCR is disabled

[2] Applies if additional SRAM read wait state in ECSM_MUDCR is enabled

35.7.2 Reset Effects on SRAM Accesses

If a reset event asserts during a read or write operation to SRAM, the completion of that access depends on the cycle at which the reset occurs. Data read from or written to SRAM before the reset event occurred is retained, and no other address locations are accessed or changed.

If the system SRAM is cached, cache lines can retain indeterminate data that is not written to memory unless the region is set for write-through mode.

NOTE

Standby memory can contain the previous data values if a reset occurs while cache is running in copy back mode.

35.8 Initialization and Application Information

To use the SRAM, the ECC must check all bits that require initialization after power on. Use either a 32-bit or 64-bit cache-inhibited write to each SRAM location to initialize the SRAM array as part of the application initialization code. All writes must specify an even number of registers performed on 32-bit or 64-bit word-aligned boundaries respectively. If the write is not the entire 32 bits (8 or 16 bits), a read/modify/write operation is generated that checks the ECC value upon the read. See Section 35.7, "SRAM ECC Mechanism.

NOTE

You **must** initialize SRAM, even if the application does not use ECC reporting.

35.8.1 Example Code

To initialize SRAM correctly, use a store multiple word **(stmw)** instruction to implement 64-bit writes to all SRAM locations. The **stmw** instruction concatenates two 32-bit registers to implement a single 64-bit write. To ensure the writes are 64 bits, specify an even number of registers and write on 64-bit word-aligned boundaries.

The following example code illustrates the use of the **stmw** instruction to initialize the SRAM ECC bits.

```
init_RAM:
lis      r11,0x4000         # base address of the SRAM, 64-bit word aligned
ori      r11,r11,0          # not needed for this address but could be for others
li       r12,1536           # loop counter to get all of SRAM;
                            # 192*1024/4 bytes/32 GPRs =1536
mtctr    r12
init_ram_loop:
stmw     r0,0(r11)          # write all 32 GPRs to SRAM
addi     r11,r11,128        # inc the ram ptr; 32 GPRs * 4 bytes = 128
bdnz     init_ram_loop      # loop for 192K of SRAM
blr                         # done
```

Chapter 36
System Timer Module (STM)

36.1 Introduction

The System Timer Module (STM) is a 32-bit timer designed to support commonly required system and application software timing functions. The STM includes a 32-bit up counter and four 32-bit compare channels with a separate interrupt source for each channel. The counter is driven by the system clock divided by an 8-bit prescale value (1 to 256).

36.1.1 Modes of Operation

The STM supports two device modes of operation: normal and debug. When the STM is enabled in normal mode, its counter runs continuously. In debug mode, operation of the counter is controlled by the FRZ bit in the STM_CR register. If the FRZ bit is set, the counter is stopped in debug mode, otherwise it continues to run.

36.2 External Signal Description

The STM does not have any external interface signals.

36.3 Memory Map and Register Definition

The STM programming model has fourteen 32-bit registers. The STM registers can only be accessed using 32-bit (word) accesses. Attempted references using a different size or to a reserved address generates a bus error termination.

36.3.1 Memory Map

The STM memory map is shown in Table 36-1.

Table 36-1. STM memory map

Address offset	Register name	Register description	Size (bits)	Access	Location
0x0000	STM_CR	STM Control Register	32	R/W	on page 36-2
0x0004	STM_CNT	STM Count Register	32	R/W	on page 36-3
0x0008	Reserved		—	—	—
0x000C	Reserved		—	—	—
0x0010	STM_CCR0	STM Channel 0 Control Register	32	R/W	on page 36-4
0x0014	STM_CIR0	STM Channel 0 Interrupt Register	32	R/W	on page 36-4
0x0018	STM_CMP0	STM Channel 0 Compare Register	32	R/W	on page 36-5
0x001C	Reserved		—	—	—
0x0020	STM_CCR1	STM Channel 1 Control Register	32	R/W	on page 36-4

Table 36-1. STM memory map (continued)

Address offset	Register name	Register description	Size (bits)	Access	Location
0x0024	STM_CIR1	STM Channel 1 Interrupt Register	32	R/W	on page 36-4
0x0028	STM_CMP1	STM Channel 1 Compare Register	32	R/W	on page 36-5
0x002C	Reserved		—	—	—
0x0030	STM_CCR2	STM Channel 2 Control Register	32	R/W	on page 36-4
0x0034	STM_CIR2	STM Channel 2 Interrupt Register	32	R/W	on page 36-4
0x0038	STM_CMP2	STM Channel 2 Compare Register	32	R/W	on page 36-5
0x003C	Reserved		—	—	—
0x0040	STM_CCR3	STM Channel 3 Control Register	32	R/W	on page 36-4
0x0044	STM_CIR3	STM Channel 3 Interrupt Register	32	R/W	on page 36-4
0x0048	STM_CMP3	STM Channel 3 Compare Register	32	R/W	on page 36-5
0x004C – 0x3FFF	Reserved		—	—	—

36.3.2 Register Descriptions

The following sections detail the individual registers within the STM programming model.

36.3.2.1 STM Control Register (STM_CR)

The STM Control Register (STM_CR) includes the prescale value, freeze control and timer enable bits.

Offset 0x000 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CPS								0	0	0	0	0	0	FRZ	TEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-1. STM Control Register (STM_CR)

Table 36-2. STM_CR field description

Field	Description
CPS	Counter Prescaler Selects the clock divide value for the prescaler (1 – 256) 0x00 = No division and the divider logic is switched off, which results in power saving. 0x01 = Divide system clock by 2 ... 0xFF = Divide system clock by 256
FRZ	Freeze Allows the timer counter to be stopped when the device enters debug mode 0 = STM counter continues to run in debug mode. 1 = STM counter is stopped in debug mode.
TEN	Timer Counter Enabled 0 = Counter is disabled 1 = Counter is enabled

36.3.2.2 STM Count Register (STM_CNT)

The STM Count Register (STM_CNT) holds the timer count value.

Offset 0x004 Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CNT																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-2. STM Count Register (STM_CNT)

Table 36-3. STM_CNT field description

Field	Description
CNT	Timer count value used as the time base for all channels When enabled, the counter increments at the rate of the system clock divided by the prescale value. Also, you can write the required starting value of the counter into this register, and the counter starts the increment from the specified value.

36.3.2.3 STM Channel n Control Register (STM_CCRn)

The STM Channel n Control Register (STM_CCRn) has the enable bit for channel n of the timer.

Offset 0x10+0x10*n Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CEN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-3. STM Channel Control Register (STM_CCRn)

Table 36-4. STM_CCRn field description

Field	Description
CEN	Channel Enable 0 = The channel is disabled. 1 = The channel is enabled.

36.3.2.4 STM Channel n Interrupt Register (STM_CIRn)

The STM Channel n Interrupt Register (STM_CIRn) has the interrupt flag for channel n of the timer.

Offset 0x14+0x10*n Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CIF
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-4. STM Channel Interrupt Register (STM_CIRn)

Table 36-5. STM_CIRn field description

Field	Description
CIF	Channel Interrupt Flag The flag and interrupt are cleared by writing a ‘1’ to this bit. Writing a ‘0’ has no effect. 0 = No interrupt request 1 = Interrupt request due to a match on the channel

36.3.2.5 STM Channel Compare Register (STM_CMPn)

The STM channel compare register (STM_CMPn) holds the compare value for channel n.

Offset 0x18+0x10*n

Access: Read/Write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CMP																															
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 36-5. STM Channel Compare Register (STM_CMPn)

Table 36-6. STM_CMPn Register field description

Field	Description
CMP	Compare value for channel n If the STM_CCRn[CEN] bit is set and the STM_CMPn register matches the STM_CNT register, a channel interrupt request is generated and the STM_CIRn[CIF] bit is set.

36.4 Functional Description

The System Timer Module (STM) is a 32-bit timer designed to support commonly required system and application software timing functions. The STM includes a 32-bit up counter and four 32-bit compare channels with a separate interrupt source for each channel.

The STM has one 32-bit up counter (STM_CNT) that is used as the time base for all channels. When enabled, the counter increments at the system clock frequency divided by a prescale value. The STM_CR[CPS] field sets the divider to any value in the range from 1 to 256. The counter is enabled with the STM_CR[TEN] bit. When enabled in normal mode the counter continuously increments. When enabled in debug mode the counter operation is controlled by the STM_CR[FRZ] bit. When the STM_CR[FRZ] bit is set, the counter is stopped in debug mode, otherwise it continues to run in debug mode. The counter rolls over at 0xFFFF_FFFF to 0x0000_0000 with no restrictions at this boundary.

The STM has four identical compare channels. Each channel includes a channel control register (STM_CCRn), a channel interrupt register (STM_CIRn) and a channel compare register (STM_CMPn). The channel is enabled by setting the STM_CCRn[CEN] bit. When enabled, the channel will set the STM_CIR[CIF] bit and generate an interrupt request when the channel compare register matches the timer counter. The interrupt request is cleared by writing a ‘1’ to the STM_CIRn[CIF] bit. A write of ‘0’ to the STM_CIRn[CIF] bit has no effect.

Chapter 37
Temperature Sensor

37.1 Overview

MPC5676R MCUs include an on-board temperature sensor that monitors device temperature and produces a voltage directly proportional to the internal junction temperature. Internal junction temperature must be calculated by software based on the sampled temperature sensor voltage, sampled bandgap voltage and calibration parameter values stored in internal flash memory.

37.2 Detailed Description

The temperature sensor generates a voltage that increases linearly with temperature. Since the voltage is an amplified version of a ΔV_{BE} voltage it is proportional to absolute temperature. This voltage, $V_{TSENS}(T)$, is read by software using the on-board eQADC module and used with the bandgap voltage and constants stored in flash memory during factory test to calculate device junction temperature.

Five calibration parameters are stored in flash memory during factory test:

- T_{LOW} is the low temperature factory calibration temperature value.
- T_{HIGH} is the hot factory calibration temperature value.
- $V_{BG\_CODE}(T_{LOW})$ is the bandgap voltage at low calibration temperature (T_{LOW}) sampled by the eQADC and converted to a 14-bit value.
- $T_{TSENS\_CODE}(T_{LOW})$ is the temperature sensor voltage at low calibration temperature (T_{LOW}) sampled by the eQADC and converted to a 14-bit value.
- $T_{TSENS\_CODE}(T_{HIGH})$ is the temperature sensor voltage at high calibration temperature (T_{HIGH}) sampled by the eQADC and converted to a 14-bit value.

The calibration points are illustrated in Figure 37-1.

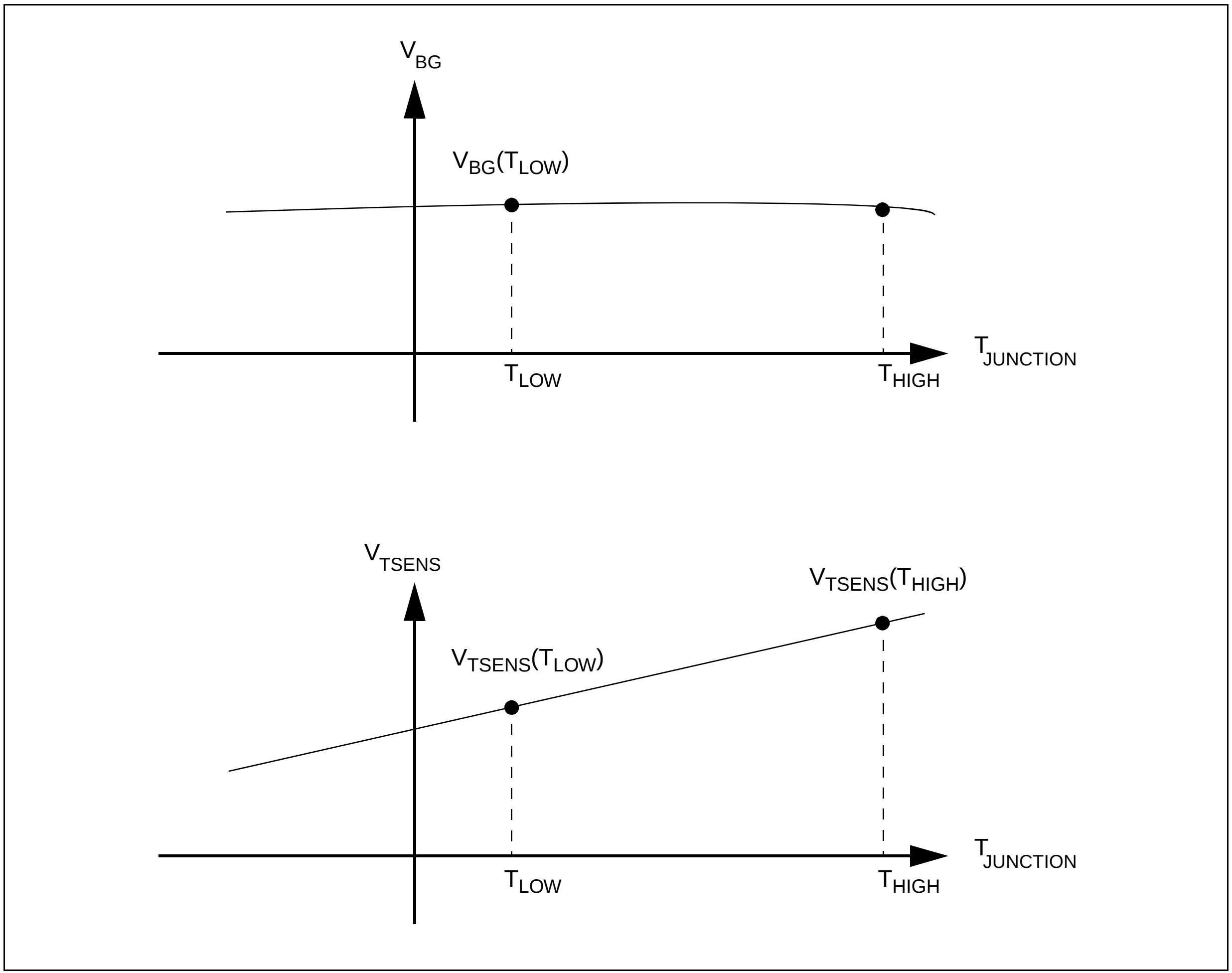


Figure 37-1. Calibration Points

37.3 Temperature formula

The temperature formula is shown in Figure 37-2.

$$T = T_{LOW} + \frac{T_{TSENS\_CODE}(T) \times \beta - T_{TSENS\_CODE}(T_{LOW})}{T_{TSENS\_CODE}(T_{HIGH}) - T_{TSENS\_CODE}(T_{LOW})} \times (T_{HIGH} - T_{LOW})$$

where:

$$T_{TSENS\_CODE}(T_{LOW}) = \frac{V_{TSENS}(T_{LOW})}{V_{ref0}} \times 2^{14}$$ (Stored in device flash during factory calibration)

$$T_{TSENS\_CODE}(T_{HIGH}) = \frac{V_{TSENS}(T_{HIGH})}{V_{ref0}} \times 2^{14}$$ (Stored in device flash during factory calibration)

$$V_{BG\_CODE}(T_{LOW}) = \frac{V_{BG}(T_{LOW})}{V_{ref0}} \times 2^{14}$$ (Stored in device flash during factory calibration)

$$V_{BG\_CODE}(T) = \frac{V_{BG}(T)}{V_{ref}} \times 2^{14}$$

$$T_{TSENS\_CODE}(T) = \frac{V_{TSENS}(T)}{V_{ref}} \times 2^{14}$$

$$\beta = \frac{V_{BG\_CODE}(T_{LOW})}{V_{BG\_CODE}(T)}$$

Notes:

- $V_{TSENS}(T)$ is the temperature sensor output sampled by the ADC
- $V_{BG}(T)$ is the bandgap voltage sampled by the ADC
- V_{ref} is the ADC reference voltage
- V_{ref0} is the ADC reference voltage during factory calibration
- T_{LOW} is the low temperature factory calibration temperature (stored in device flash)
- T_{HIGH} is the hot factory calibration temperature (stored in device flash)

Figure 37-2. Temperature formula

The following sections detail the values required and where to get them.

37.3.1 T_{LOW} and T_{HIGH}

T_{LOW} is the factory low calibration temperature; T_{HIGH} is the hot factory calibration temperature. These values are stored in shadow flash memory during factory calibration. See *Section 37.3.6.1, "Temperature Calculation Constants Register 0*", for details.

37.3.2 $T_{TSENS\_CODE}(T_{LOW})$ and $T_{TSENS\_CODE}(T_{HIGH})$

$T_{TSENS\_CODE}(T_{LOW})$ is the sampled output voltage of the temperature sensor during low temperature factory calibration. $T_{TSENS\_CODE}(T_{HIGH})$ is the sampled output voltage of the temperature sensor during hot temperature factory calibration. These values are stored in shadow flash memory during factory calibration. See *Section 37.3.6.1, "Temperature Calculation Constants Register 0*", for details.

37.3.3 $V_{BG\_CODE}(T_{LOW})$

$V_{BG\_CODE}(T_{LOW})$ is the value of the bandgap voltage sampled during low temperature factory calibration. This value is stored in shadow flash memory during factory calibration. See *Section 37.3.6.2, “Temperature Calculation Constants Register 1”*, for details.

37.3.4 Temperature sensor voltage ($V_{TENS}(T)$)

$V_{TENS}(T)$ is the output voltage of the device temperature sensor. Software must sample the voltage from eQADC_A channel 128 (ADC0 and ADC1).

37.3.5 Bandgap reference voltage ($V_{BG\_CODE}(T)$)

V_{BG} is the bandgap reference voltage. Software must sample the voltage from eQADC_A channel 45 (ADC0).

37.3.6 Registers

The calibration constants described previously, i.e., T_{LOW}, T_{HIGH}, $T_{SENS\_CODE}(T_{LOW})$, $T_{SENS\_CODE}(T_{HIGH})$ and $V_{BG\_CODE}(T_{LOW})$, are stored in device shadow flash memory during factory test. This section details the registers where the values reside.

37.3.6.1 Temperature Calculation Constants Register 0

This register contains the calibration temperatures and temperature sensor outputs measured during factory calibration:

- T_{HIGH}
- $T_{SENS\_CODE}(T_{HIGH})$
- T_{LOW}
- $T_{SENS\_CODE}(T_{LOW})$

Address: 0xFFFE_C000

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	THIGH		TSCV2													
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	TLOW		TSCV1													
W	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
RESET:	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x

▨ = Unimplemented or Reserved

Figure 37-3. Temperature Calculation Constants Register 0

The bit fields are described in Table 37-1.

Table 37-1. Temperature Calculation Constants Register 0 Field Descriptions

Field	Description
0–1 THIGH	The THIGH field contains a value indicating the hot factory calibration temperature (T_{HIGH}). The values are as follows: 00: T_{HIGH} = Reserved 01: T_{HIGH} = 125 °C 10: T_{HIGH} = 145 °C 11: T_{HIGH} = 150 °C
2–15 TSCV2	Temperature sensor output at hot factory calibration temperature (TSENS_CODE(T_{HIGH})). TSCV2 is the temperature sensor voltage sampled and converted by the eQADC during factory test with device at hot temperature (T_{HIGH}). This is the T_{SENS_CODE}(T_{HIGH}) parameter value referenced in the temperature calculation formula (see Figure 37-2).
16–17 TLOW	The TLOW field contains a code indicating the low factory calibration temperature (T_{LOW}). The values are as follows: 00: T_{LOW} = 25 °C 01: T_{LOW} = 40 °C 10: T_{LOW} = –45 °C 11: T_{LOW} = Reserved
18–31 TSCV1	Temperature sensor output at the low factory calibration temperature (TSENS_CODE(T_{LOW})). TSCV1 is the temperature sensor voltage sampled and converted by the eQADC during factory test with device at the low calibration temperature. This is the T_{SENS_CODE}(T_{LOW}) parameter value referenced in the temperature calculation formula (see Figure 37-2).

37.3.6.2 Temperature Calculation Constants Register 1

This register contains the V_{BG_CODE}(T_{LOW}) parameter value used in the temperature calculation.

Address: 0xFFFE_C004

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	Reserved		TSCV3													
W																
RESET:	0	0	x	x	x	x	x	x	x	x	x	x	x	x	x	x

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	Reserved															
W																
RESET:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

▨ = Unimplemented or Reserved

Figure 37-4. Temperature Calculation Constants Register 1

The bit fields are described in Table 37-2.

Table 37-2. Temperature Calculation Constants Register 1 Field Descriptions

Field	Description
0–1	Reserved
2–15 TSCV3	Bandgap voltage sampled and converted by ADC during factory test. This is the $V_{BG\_CODE}(T_{LOW})$ parameter value referenced in the temperature calculation formula (see Figure 37-2).
16–31	Reserved

Appendix A
Revision History

This appendix describes corrections to the *MPC5676R Reference Manual*. For convenience, the corrections are grouped by revision. Grammatical and formatting changes are not listed here unless the meaning of something changed.

A.1 Changes Between Revision 4 and Revision 5

Table A-1. Changes Between Revision 4 and Revision 5

Chapter	Description
Device Overview	Made minor editorial changes.
Signal Description	Made minor editorial changes.
Resets	Made minor editorial changes. Changed “core soft watchdog” to “core watchdog” in the “WTE” bit field description of the “RCHW Structure”.
Core	Made minor editorial changes. Changed “The SPR number for L1CSR0 ...” to “The SPR number for L1CSR1 ...” in the “L1CSR1” description.
Debug	Minor editorial changes.
Enhanced Modular Input/Output Subsystem (eMIOS200)	Updated the note in the “eMIOS[*n*]” section.
Enhanced Queued Analog-to-Digital Converter (eQADC)	Updated the note in the “Variable Gain Amplification (VGA) for Pre-gain” section.
Enhanced Serial Communications Interface (eSCI)	• Updated the “TXDIR” bit description of the “eSCI_CR2” register. • Replaced the following text with a note in the “Single Wire Mode” section: “The TXDIR bit (eSCI_CR2[1]) determines whether the TXD pin is going to be used as an input (TXDIR = 0) or an output (TXDIR = 1) in this mode of operation”.
Error Correction Status Module (ECSM)	Updated the register and bit field descriptions of the “Miscellaneous User-Defined Control Register (MUDCR)”.

A.2 Changes Between Revisions 3 and 4

Table A-2. Changes Between Revisions 3 and 4

Chapter	Description
Device Overview	Updated the “Calibration bus” value for “MPC5676R” in the “MPC5500/MPC5600 Family Comparison” table.
Signal Descriptions	Updated the descriptions of “EMIOS[16:23]_ETPUB[0:7]_GPIO[195:202]” to “EMIOS31_PCSC5_GPIO437” signal names in the “eMIOS Signals” table: Changed “Output” to “Input/Output”.
Resets	Removed “PLL” line from the “Reset Configuration Timing” diagram.
Clocking	Updated the “MPC5676R System Level Clock Diagram”: Updated “DIV” ”CLKOUT” and “Core0” “CLK Gate” connections.
Frequency Modulated Phase-Locked Loop (FMPLL)	Removed “MPC5676R System Level Clock Diagram”.
Debug	Updated the description of the “TM” bit field (1XX and XX1 values) and added a footnote to the “DC1 Field Description” table.
Enhanced Queued Analog-to-Digital Converter (eQADC)	• Updated the “Overview” section and the “Analog to Digital Conversion Sub-system” diagram. • Updated “DEST” description in the “ADC_ACR1-8 Field Descriptions” table. • Updated “Overview” section of “ADC Calibration Feature”. • Added a note to the “Variable Gain Amplification (VGA) for Pre-gain” section. • Removed the following sentence from “Features” section: Selectable common mode conversion range (0–5V; 0–2.5V; 0–1.25V). • Moved the following note from “ADC0/1_EMUX” to “ADC0/1_EN” in the “EQADC_CFxRw Field Descriptions” table: “Both ADC0 and ADC1 of an eQADC module pair must be enabled before calibrating or using either ADC0 or ADC1 of the pair. Failure to enable both ADC0 and ADC1 of the pair can result in inaccurate conversions.”
Enhanced Time Processing Unit (eTPU)	Reverted MPC5676R RM Rev 3 changes :
FlexCAN Module	Replaced all instances of “BCC” with “MBFEN”.
Temperature Sensor	• Updated the channel information in the “Bandgap reference voltage” section. • Updated the “01: T_{HIGH}” and “10: T_{LOW}” values in the “Temperature Calculation Constants Register 0 Field Descriptions” table.

A.3 Changes Between Revisions 2 and 3

Table A-3. Changes Between Revisions 2 and 3

Chapter	Description
Signal Descriptions	Changed the direction of the VDDSYN pin to I/O. Made the following changes to the “Signal Properties and Muxing Summary” table: • Added a footnote for ANA0-ANA7 functions. • Changed the signal name “WKPCFG_GPIO213” to “WKPCFG_NMI_GPIO213” and added a footnote for the signal name. • Added a footnote for MDO0-MDO15 and MSEO0 functions. Added footnotes to the “eQADC Signals” and “JTAG and Nexus Signals” tables.
System Integration Unit (SIU, SIU_B)	Updated the “IFEE_NMI8” description in the “SIU_IFEER Bit Field Descriptions” table. Updated the “eQADC Trigger Input” for “00000” and “00111” in the “eTSEL0ADV_B Bit Field Descriptions” table. Changed the order of the bit-fields of the SIU_DECFIL1, SIU_DECFIL2, and SIU_DECFIL3 register. Made minor editorial changes and corrections. Reordered the bits of the SIU_DECFIL4 and SIU_DECFIL5 registers. Added a paragraph to the “RSTVEC” descriptions in the “SIU_RSTVEC0/SIU_RSTVEC1 Field Descriptions” tables. Replaced “Core 0” with “Core 1” in “RSTVEC” description in the “SIU_RSTVEC1 Field Descriptions” table.
Boot Assist Module (BAM)	Added a paragraph to the “Download Protocol Execution” note.
Frequency Modulated Phase-Locked Loop (FMPLL)	**Previous errata err001111 integrated into the reference manual:** Updated the “LOLF” field description in the “SYNSR Bit Field Descriptions” table. Added a footnote to the “ESYNCR2 Bit Field Descriptions” table.
Power Management Controller (PMC)	Updated the SMPS and LDO voltage values in the features sections. Updated the nominal value of the “REGCTL” pin in the “PMC Signals” table. Updated the “PMC Internal 1.2V Voltage Regulator Selection” section. Updated the “Internal Regulator 1.2V LDO and SMPS” diagram: Changed “SI5656DC” to “SI3460”.
Core	Added a paragraph to the “Cache Coherency Unit (CCU)” section.
Debug	Added notes to the “Message Data Out (MDO[11:0] or [15:0])” and “Message Start/End Out (MSEO[1:0])” sections.
Decimation Filter	Made minor changes to the “Overview section: Changed two instances of “eQADC” to “eQADC_A”. Updated the “OSEL” descriptions in the “DECFILT_x_IB and DECFILT_x_IB Field Descriptions” tables.
Deserial Serial Peripheral Interface (DSPI)	**Previous errata err001147 integrated into the reference manual:** Added a note to the “Parallel chaining” section. **Previous errata err000575 integrated into the reference manual:** Added a warning to the “Continuous Selection Format” section. Added a note to the “Continuous selection format” section. Updated the bullets in the “Continuous serial communications clock” section. Added information about the following registers: • DSPI DSI Serialization Source Select Register (DSPI_SSR) • DSPI DSI Parallel Input Select Register 0 (DPSI_PISR0) • DSPI DSI Parallel Input Select Register 1 (DPSI_PISR1) • DSPI DSI Parallel Input Select Register 2 (DPSI_PISR2) • DSPI DSI Parallel Input Select Register 3 (DPSI_PISR3) • DSPI DSI Deserialized Data Interrupt Mask Register (DSPI_DIMR) DSPI DSI Deserialized Data Polarity Interrupt Register (DSPI_DPIR)

Table A-3. Changes Between Revisions 2 and 3 (continued)

Chapter	Description
Enhanced Modular Input/Output Subsystem (eMIOS200)	Updated the “eMIOS200 Block Diagram” and the “IPM Example” diagram. Updated the “External Signal Description”, “Output Disable Input — eMIOS200 Output Disable Input Signal”, and “eMIOS[n]” sections. Updated the “ETB”, GPREN, and SRV descriptions in the “EMIOS_CCR[*n*] Field Descriptions” table. Updated the “ODISSL” and “UCPREN” descriptions of the “EMIOS_CCR[*n*] Field Descriptions” table. Added the following to the “On the next system clock cycle after the A1...” sentence in the “Modulus Counter Buffered (MCB) Mode” section: “and the FLAG bit is set to '1'”. Changed “$1” to “0x00_0001” in the Modulus Counter Buffered (MCB) Mode and Center-Aligned Output PWM Buffered with Dead Time (OPWMCB) Mode. Removed a paragraph from the “Application Information” section.
Enhanced Queued Analog-to-Digital Converter (eQADC)	**Previous errata err000652 integrated into the reference manual:** • Added a footnote to the “Pin Mapping to Channel Mapping” table. • Updated the description of the “(b) 0010_1010 channel” description in the “Multiplexed and non-multiplexed channel assignments” table. Updated the “DSM” description in the “EQADC_MCR Field Descriptions” table. Updated the “DEST” description in the “ADC_ACR1-8 Field Descriptions” table. **Previous errata err002449 integrated into reference manual:** Added a note to the “ADC0/1_EMUX” description of the “EQADC_CFxRw Field Descriptions” table.
Enhanced Serial Communications Interface (eSCI)	Removed references to the LIN 2.1 specification.
Enhanced Time Processing Unit (eTPU)	Removed information about the ETPUWDSR_A and ETPUWDSR_B registers; removed the second list item from the “Watchdog” section and updated a bullet item in the section. Replaced “STAC signals” with “STAC Bus” in the “eTPU A/B Module Block Diagram”. Updated the “eTPU Channel Output Disable Signals” section. Updated the “High Level Memory Map” footnote. Updated the footnotes of the “Detailed Memory Map A/B and eTPU C” tables. Updated the “ETPUSCMOFFDATAR” section. Added a table to the “ETPUREDCR” section. Removed “Unimplemented or Reserved” from the following registers: “ETPUCDCR”, “ETPUMISCCMPR”, and “ETPUSCMOFFDATAR”. Removed figure footnotes from the following registers: ETPUMECR, ETPUDEIAR, ETPUDEIDPR, ETPUDEIPPR, ETPUDERAR, ETPUDERDR, ETPUDERSR, ETPUMESR, ETPUCEIAR, ETPUCEIDPR, ETPUCEIPPR, ETPUCERAR, ETPUCERDR, ETPUCERSR, and ETPUCEFR
Error Correction Status Module (ECSM)	Updated the “Features section”. Updated the “PCT” description in the “ECSM_PCT Field Descriptions” table. Changed the reset value of the “ECSM_IMC” register to “0x4847EC00”
Flash Memory Array and Control	Updated the “Shadow Block Memory Map” table. Added a footnote and corrected a reset value in the “Flash Configuration Register Memory Map” table. Updated the “FLASH_BIUCR3” section. Corrected the ordering of the “M0PFE” bit of the “Flash Bus Interface Configuration Register 3 (FLASH_BIUCR3)” register

Table A-3. Changes Between Revisions 2 and 3 (continued)

Chapter	Description
FlexCAN Module	Made editorial changes in the “Module Disable Mode” (“Modes of Operation” and “Modes of Operation Details” sections). Made editorial changes in the “Transmit Process” section. Added a note to the “Transmission Abort Mechanism” section. **Previous errata err002360 integrated into the reference manual:** Added “Precautions when using Global Mask and Individual Mask registers” and “CAN protocol related features.
FlexRay Module (FlexRay)	Updated the “Channel Assignment Description” table. **Previous errata err002421 integrated into the reference manual:** Updated the description of the “Application Transitions” sections.
Interrupts and Interrupt Controller (INTC)	Updated “Software Vector Mode” and “External Interrupt Request Sources” sections.
Memory Protection Unit (MPU)	Updated the reset value of the “MPU_CESR” register. Updated the reset values of the “M1SM” bit fields of “MPU_RGD0, Word 2” and “MPU_RGDAAC0” registers.
Software Watchdog Timers (SWT_A, SWT_B)	Updated the “Modes of Operation” section. Added footnotes to the “SWT Memory Map” and to the “SWT_SK” register. Removed the “STP” bit from “SWT_MCR”. Updated the “MAP*n*” description in the “SWT_MCR Field Descriptions” table. Added the following to the “Functional Description” section: Refer to Section 27.4.1, “External Interrupt Request Sources” and Section 3.2.1.5, “DMA/Interrupt Request Enable Register (SIU_DIRER)” for details on the enabling and routing of the SWT interrupt signals.

Table 37-3. Changes Added to MPC5676R RM for Rev. 2 Release

Date	Revision	Changes
9 Nov 2010	1	Initial Release
	2	**Added the following chapters:** Boot Assist Module (BAM), Clocking, Decimation Filters (DecFilt), Enhanced Direct Memory Access Controller (eDMA), Enhanced Queued Analog-to-Digital Converter (eQADC), Enhanced Serial Communications Interface, Enhanced Time Processing Unit (eTPU), Error Correction Status Module (ECSM), FlexRay Module (FlexRay), Interrupts & Interrupt Controller (INTC), Self Test Control Unit (STCU), System RAM (SRAM), System Timer Module (STM) **Updated the following chapters:** SIU, Resets, FMPLL, DSPI, eMIOS200, eQADC, ECSM, FlexRay, INTC, PIT_RTI, PBRIDGE, SWT, Device Overview, PMC, Core, Debug, ESCI, Flash Memory, FlexCan