

# MCF5441x Reference Manual

## Devices Supported:

MCF54410  
MCF54415  
MCF54416  
MCF54417  
MCF54418

Document Number: MCF54418RM  
Rev. 3  
9/2010



**How to Reach Us:****Home Page:**

[www.freescale.com](http://www.freescale.com)

**Web Support:**

<http://www.freescale.com/support>

**USA/Europe or Locations Not Listed:**

Freescale Semiconductor, Inc.  
Technical Information Center, EL516  
2100 East Elliot Road  
Tempe, Arizona 85284  
+1-800-521-6274 or +1-480-768-2130  
[www.freescale.com/support](http://www.freescale.com/support)

**Europe, Middle East, and Africa:**

Freescale Halbleiter Deutschland GmbH  
Technical Information Center  
Schatzbogen 7  
81829 Muenchen, Germany  
+44 1296 380 456 (English)  
+46 8 52200080 (English)  
+49 89 92103 559 (German)  
+33 1 69 35 48 48 (French)  
[www.freescale.com/support](http://www.freescale.com/support)

**Japan:**

Freescale Semiconductor Japan Ltd.  
Headquarters  
ARCO Tower 15F  
1-8-1, Shimo-Meguro, Meguro-ku,  
Tokyo 153-0064  
Japan  
0120 191014 or +81 3 5437 9125  
[support.japan@freescale.com](mailto:support.japan@freescale.com)

**Asia/Pacific:**

Freescale Semiconductor China Ltd.  
Exchange Building 23F  
No. 118 Jianguo Road  
Chaoyang District  
Beijing 100022  
China  
+86 10 5879 8000  
[support.asia@freescale.com](mailto:support.asia@freescale.com)

**For Literature Requests Only:**

Freescale Semiconductor Literature Distribution Center  
1-800-441-2447 or 303-675-2140  
Fax: 303-675-2150  
[LDCForFreescaleSemiconductor@hibbertgroup.com](mailto:LDCForFreescaleSemiconductor@hibbertgroup.com)

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

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor 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 Semiconductor 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 Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.



Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

MCF54418RM  
Rev. 3  
9/2010

About This Book .....	xxxv
Audience .....	xxxv
Suggested Reading .....	xxxv
General Information .....	xxxv
ColdFire Documentation .....	xxxvi
Conventions .....	xxxvi
Register Figure Conventions .....	xxxvii

## Chapter 1 Overview

1.1 Introduction.....	1-1
1.2 MCF5441x Family Comparison .....	1-2
1.3 Block Diagram .....	1-4
1.4 Operating Parameters .....	1-5
1.5 Packages .....	1-5
1.6 Chip Level Features .....	1-5
1.7 Module-by-Module Feature List .....	1-6
1.7.1 Version 4 ColdFire Variable-Length RISC Processor .....	1-6
1.7.2 On-chip Memories .....	1-6
1.7.3 Phase Locked Loop (PLL) and Crystal Oscillator .....	1-6
1.7.4 Power Management.....	1-7
1.7.5 Chip Configuration Module (CCM).....	1-7
1.7.6 Reset Controller.....	1-7
1.7.7 System Control Module .....	1-7
1.7.8 Crossbar Switch.....	1-7
1.7.9 Universal Serial Bus (USB) Host Controller.....	1-8
1.7.10 Universal Serial Bus (USB) 2.0 On-The-Go (OTG) Controller.....	1-8
1.7.11 DDR SDRAM Controller .....	1-8
1.7.12 FlexBus (External Interface) .....	1-9
1.7.13 Ethernet Assembly.....	1-9
1.7.14 Cryptography Acceleration Unit (CAU) .....	1-10
1.7.15 Random Number Generator .....	1-10
1.7.16 Secure Digital Host Controller (SDHC) .....	1-10
1.7.17 Subscriber Identity Module (SIM) .....	1-10
1.7.18 Synchronous Serial Interfaces (SSI) .....	1-10
1.7.19 FlexCAN Modules .....	1-11
1.7.20 Analog-Digital Converters (ADC).....	1-11
1.7.21 Digital-Analog Converters (DAC).....	1-11

1.7.22	NAND Flash Controller . . . . .	1-11
1.7.23	1-Wire Interface . . . . .	1-12
1.7.24	Robust Real Time Clock . . . . .	1-12
1.7.25	Programmable Interrupt Timers (PIT) . . . . .	1-12
1.7.26	DMA Timers. . . . .	1-13
1.7.27	DMA Serial Peripheral Interfaces (DSPI). . . . .	1-13
1.7.28	Motor Control Pulse Width Modulation (mcPWM) Module . . . . .	1-13
1.7.29	Universal Asynchronous Receiver Transmitters (UARTs). . . . .	1-13
1.7.30	I2C Modules. . . . .	1-14
1.7.31	Interrupt Controllers. . . . .	1-14
1.7.32	Edge Port Module . . . . .	1-14
1.7.33	DMA Controller . . . . .	1-14
1.7.34	Rapid GPIO Interface . . . . .	1-15
1.7.35	General Purpose I/O Interface. . . . .	1-15
1.7.36	System Debug Support . . . . .	1-15
1.7.37	JTAG Support . . . . .	1-15
1.8	Memory Map Overview. . . . .	1-16
1.8.1	Internal Peripheral Space . . . . .	1-17
1.9	Documentation . . . . .	1-19

## Chapter 2 Signal Descriptions

2.1	Introduction. . . . .	2-1
2.2	Signal Properties Summary . . . . .	2-1
2.3	Signal Primary Functions . . . . .	2-10
2.3.1	Reset Signals. . . . .	2-10
2.3.2	PLL and Clock Signals. . . . .	2-10
2.3.3	Mode Selection . . . . .	2-11
2.3.4	Enhanced Secure Digital Host Controller . . . . .	2-11
2.3.5	SmartCard Interface Ports. . . . .	2-11
2.3.6	FlexBus Signals. . . . .	2-12
2.3.7	SDRAM Controller Signals . . . . .	2-13
2.3.8	Serial Boot Facility Signals . . . . .	2-14
2.3.9	External Interrupt Signals . . . . .	2-14
2.3.10	DMA Signals . . . . .	2-14
2.3.11	Ethernet Controllers (MACNET0–1) Signals . . . . .	2-14
2.3.12	NAND Flash Controller Signals . . . . .	2-15
2.3.13	Analog-to-Digital Converter Signals. . . . .	2-16
2.3.14	Digital-to-Analog Converter Signals. . . . .	2-16
2.3.15	Rapid GPIO Signals. . . . .	2-16
2.3.16	1-Wire Signals . . . . .	2-16
2.3.17	PWM I/O Signals . . . . .	2-16

2.3.18	FlexCAN Signals . . . . .	2-17
2.3.19	I2C I/O Signals. . . . .	2-17
2.3.20	DMA Serial Peripheral Interface (DSPI) Signals . . . . .	2-18
2.3.21	Synchronous Serial Interface (SSI) Signals. . . . .	2-18
2.3.22	Universal Serial Bus (USB) Signals. . . . .	2-19
2.3.23	UART Module Signals . . . . .	2-19
2.3.24	DMA Timer Signals . . . . .	2-20
2.3.25	Debug Support Signals . . . . .	2-20
2.3.26	Test Signals. . . . .	2-22
2.3.27	Power and Ground Pins. . . . .	2-22
2.4	External Boot Mode . . . . .	2-22

## Chapter 3 ColdFire Core

3.1	Introduction. . . . .	3-1
3.1.1	Overview . . . . .	3-1
3.2	Memory Map/Register Description . . . . .	3-4
3.2.1	Data Registers (D0–D7) . . . . .	3-6
3.2.2	Address Registers (A0–A6) . . . . .	3-7
3.2.3	Supervisor/User Stack Pointers (A7 and OTHER_A7) . . . . .	3-7
3.2.4	Condition Code Register (CCR) . . . . .	3-8
3.2.5	Program Counter (PC) . . . . .	3-9
3.2.6	Cache Programming Model. . . . .	3-9
3.2.7	MMU Programming Model. . . . .	3-9
3.2.8	Vector Base Register (VBR) . . . . .	3-9
3.2.9	Status Register (SR) . . . . .	3-10
3.2.10	Memory Base Address Register (RAMBAR) . . . . .	3-11
3.3	Functional Description . . . . .	3-11
3.3.1	Version 4 ColdFire Microarchitecture . . . . .	3-11
3.3.2	Instruction Set Architecture (ISA_C) . . . . .	3-13
3.3.3	Exception Processing Overview . . . . .	3-14
3.3.4	Processor Exceptions . . . . .	3-17
3.3.5	Instruction Execution Timing . . . . .	3-26

## Chapter 4 Memory Management Unit (MMU)

4.1	Introduction. . . . .	4-1
4.1.1	Block Diagram . . . . .	4-1
4.1.2	Features. . . . .	4-2
4.2	Memory Map/Register Definition . . . . .	4-3
4.2.1	Address Space ID (ASID) . . . . .	4-4
4.2.2	MMU Base Address Register (MMUBAR) . . . . .	4-4

4.2.3	MMU Control Register (MMUCR) . . . . .	4-5
4.2.4	MMU Operation Register (MMUOR) . . . . .	4-6
4.2.5	MMU Status Register (MMUSR) . . . . .	4-7
4.2.6	MMU Fault, Test, or TLB Address Register (MMUAR) . . . . .	4-8
4.2.7	MMU Read/Write Tag Entry Registers (MMUTR) . . . . .	4-9
4.2.8	MMU Read/Write Data Entry Register (MMUDR) . . . . .	4-10
4.3	Functional Description . . . . .	4-11
4.3.1	Virtual Memory Management Architecture . . . . .	4-12
4.3.2	Debugging in a Virtual Environment . . . . .	4-16
4.3.3	Virtual Memory Architecture Processor Support . . . . .	4-16
4.3.4	Effective Address Attribute Determination. . . . .	4-18
4.3.5	MMU Functionality. . . . .	4-19
4.3.6	MMU TLB. . . . .	4-19
4.3.7	MMU Operation . . . . .	4-19
4.3.8	MMU Implementation. . . . .	4-21
4.3.9	MMU Instructions. . . . .	4-24

## Chapter 5 Enhanced Multiply-Accumulate Unit (EMAC)

5.1	Introduction. . . . .	5-1
5.1.1	Overview . . . . .	5-1
5.2	Memory Map/Register Definition . . . . .	5-3
5.2.1	MAC Status Register (MACSR). . . . .	5-3
5.2.2	Mask Register (MASK) . . . . .	5-5
5.2.3	Accumulator Registers (ACC0–3) . . . . .	5-7
5.2.4	Accumulator Extension Registers (ACCext01, ACCext23) . . . . .	5-7
5.3	Functional Description . . . . .	5-8
5.3.1	Fractional Operation Mode . . . . .	5-10
5.3.2	EMAC Instruction Set Summary . . . . .	5-12
5.3.3	EMAC Instruction Execution Times . . . . .	5-13
5.3.4	Data Representation . . . . .	5-14
5.3.5	MAC Opcodes . . . . .	5-14

## Chapter 6 Cache

6.1	Introduction. . . . .	6-1
6.1.1	Block Diagram . . . . .	6-1
6.1.2	Overview . . . . .	6-1
6.2	Cache Organization . . . . .	6-2
6.2.1	Cache Line States: Invalid, Valid-Unmodified, and Valid-Modified . . . . .	6-3
6.2.2	The Cache at Start-Up. . . . .	6-3
6.3	Memory Map/Register Definition . . . . .	6-5

6.3.1	Cache Control Register (CACR) . . . . .	6-5
6.3.2	Access Control Registers (ACR <sub>n</sub> ) . . . . .	6-8
6.4	Functional Description . . . . .	6-9
6.4.1	Caching Modes . . . . .	6-12
6.4.2	Cache Protocol . . . . .	6-14
6.4.3	Cache Coherency (Data Cache Only) . . . . .	6-16
6.4.4	Memory Accesses for Cache Maintenance . . . . .	6-16
6.4.5	Cache Locking . . . . .	6-17
6.4.6	Cache Management . . . . .	6-20
6.4.7	Cache Operation Summary . . . . .	6-22
6.4.8	CPUSHL Enhancements . . . . .	6-27
6.5	Initialization/Application Information . . . . .	6-28

## Chapter 7 Static RAM (SRAM)

7.1	Introduction . . . . .	7-1
7.1.1	Overview . . . . .	7-1
7.1.2	Features . . . . .	7-1
7.2	Memory Map/Register Description . . . . .	7-2
7.2.1	SRAM Base Address Register (RAMBAR) . . . . .	7-2
7.3	Initialization/Application Information . . . . .	7-4
7.3.1	SRAM Initialization Code . . . . .	7-4
7.3.2	Power Management . . . . .	7-5

## Chapter 8 Clock Module

8.1	Introduction . . . . .	8-1
8.1.1	Block Diagram . . . . .	8-3
8.1.2	Features . . . . .	8-3
8.1.3	Modes of Operation . . . . .	8-4
8.2	Memory Map/Register Definition . . . . .	8-6
8.2.1	PLL Control Register (PLL_CR) . . . . .	8-6
8.2.2	PLL Divider Register (PLL_DR) . . . . .	8-7
8.2.3	PLL Status Register (PLL_SR) . . . . .	8-9
8.3	Functional Description . . . . .	8-10
8.3.1	PLL Frequency Multiplication Factor Select . . . . .	8-10
8.3.2	PLL Frequency Synthesis . . . . .	8-11
8.3.3	Lock Conditions . . . . .	8-11
8.3.4	Loss-of-Lock Detection . . . . .	8-12
8.3.5	Loss-of-Clock Detection . . . . .	8-12
8.3.6	System Clock Modes . . . . .	8-13
8.3.7	Clock Operation During Reset . . . . .	8-13

## Chapter 9 Power Management

9.1	Introduction . . . . .	9-1
9.1.1	Features . . . . .	9-1
9.2	Memory Map/Register Definition . . . . .	9-1
9.2.1	Wake-up Control Register (WCR) . . . . .	9-2
9.2.2	Peripheral Power Management Set Registers (PPMSR0, PPMSR1) . . . . .	9-3
9.2.3	Peripheral Power Management Clear Register (PPMCR0, PPMCR1) . . . . .	9-4
9.2.4	Peripheral Power Management Registers (PPMHR{1,0}, PPMLR{1,0}) . . . . .	9-5
9.2.5	Low-Power Control Register (LPCR) . . . . .	9-9
9.3	Functional Description . . . . .	9-9
9.3.1	Peripheral Shut Down . . . . .	9-10
9.3.2	Limp mode . . . . .	9-10
9.3.3	Low-Power Modes . . . . .	9-10
9.3.4	Peripheral Behavior in Low-Power Modes . . . . .	9-11
9.3.5	Summary of Peripheral State During Low-power Modes . . . . .	9-19

## Chapter 10 Chip Configuration Module (CCM)

10.1	Introduction . . . . .	10-1
10.1.1	Block Diagram . . . . .	10-1
10.1.2	Features . . . . .	10-1
10.1.3	Modes of Operation . . . . .	10-1
10.2	External Signal Descriptions . . . . .	10-2
10.2.1	BOOTMOD[1:0] . . . . .	10-2
10.2.2	FB_AD[7:0] (Reset Configuration Override) . . . . .	10-2
10.3	Memory Map/Register Definition . . . . .	10-2
10.3.1	Chip Configuration Register (CCR) . . . . .	10-3
10.3.2	Reset Configuration Register (RCON) . . . . .	10-4
10.3.3	Chip Identification Register (CIR) . . . . .	10-5
10.3.4	Miscellaneous Control Register (MISCCR) . . . . .	10-5
10.3.5	Clock-Divider Register High (CDRH) . . . . .	10-7
10.3.6	Clock-Divider Register Low (CDRL) . . . . .	10-8
10.3.7	USB On-the-Go Controller Status Register (UOCSR) . . . . .	10-8
10.3.8	USB Host Controller Status Register (UHCSR) . . . . .	10-10
10.3.9	Miscellaneous Control Register 3 (MISCCR3) . . . . .	10-11
10.3.10	Miscellaneous Control Register 2 (MISCCR2) . . . . .	10-11
10.3.11	ADC Trigger Select Register (ADCTSR) . . . . .	10-13
10.3.12	DAC Trigger Select Register (DACTS) . . . . .	10-14
10.3.13	FlexBus/NAND Flash Arbiter Control Register (FNACR) . . . . .	10-15
10.4	Functional Description . . . . .	10-15
10.4.1	Reset Configuration . . . . .	10-15

10.4.2	Boot Configuration . . . . .	10-18
10.4.3	Low Power Configuration . . . . .	10-19

## Chapter 11 Serial Boot Facility (SBF)

11.1	Introduction . . . . .	11-1
11.1.1	Overview . . . . .	11-1
11.1.2	Features . . . . .	11-2
11.2	External Signal Description . . . . .	11-2
11.3	Memory Map/Register Definition . . . . .	11-2
11.3.1	Serial Boot Facility Status Register (SBFSR) . . . . .	11-3
11.3.2	Serial Boot Facility Control Register (SBFCR) . . . . .	11-3
11.4	Functional Description . . . . .	11-4
11.4.1	Serial Initialization and Shift Clock Frequency Adjustment . . . . .	11-4
11.4.2	Reset Configuration and Optional Boot Load . . . . .	11-5
11.4.3	Execution Transfer . . . . .	11-5
11.5	Initialization Information . . . . .	11-6
11.5.1	SPI Memory Initialization . . . . .	11-6
11.5.2	FAST_READ Feature Initialization . . . . .	11-7

## Chapter 12 Reset Controller Module

12.1	Introduction . . . . .	12-1
12.1.1	Block Diagram . . . . .	12-1
12.1.2	Features . . . . .	12-1
12.2	External Signal Description . . . . .	12-2
12.2.1	RESET . . . . .	12-2
12.2.2	RSTOUT . . . . .	12-2
12.3	Memory Map/Register Definition . . . . .	12-2
12.3.1	Reset Control Register (RCR) . . . . .	12-3
12.3.2	Reset Status Register (RSR) . . . . .	12-3
12.4	Functional Description . . . . .	12-4
12.4.1	Reset Sources . . . . .	12-4
12.4.2	Reset Control Flow . . . . .	12-5

## Chapter 13 System Control Module (SCM)

13.1	Introduction . . . . .	13-1
13.1.1	Overview . . . . .	13-1
13.1.2	Features . . . . .	13-1
13.2	Memory Map/Register Definition . . . . .	13-1

13.2.1	Core Watchdog Control Register (CWCR) . . . . .	13-2
13.2.2	Core Watchdog Service Register (CWSR) . . . . .	13-3
13.2.3	SCM Interrupt Status Register (SCMISR) . . . . .	13-4
13.2.4	Burst Configuration Register (BCR) . . . . .	13-5
13.2.5	Core Fault Address Register (CFADR) . . . . .	13-5
13.2.6	Core Fault Interrupt Enable Register (CFIER) . . . . .	13-6
13.2.7	Core Fault Location Register (CFLOC) . . . . .	13-6
13.2.8	Core Fault Attributes Register (CFATR) . . . . .	13-7
13.2.9	Core Fault Data Register (CFDTR) . . . . .	13-7
13.3	Functional Description . . . . .	13-8
13.3.1	Core Watchdog Timer . . . . .	13-8
13.3.2	Core Data Fault Recovery Registers . . . . .	13-9

## Chapter 14 Crossbar Switch (XBS)

14.1	Overview . . . . .	14-1
14.2	Features . . . . .	14-4
14.3	Modes of Operation . . . . .	14-4
14.4	Memory Map / Register Definition . . . . .	14-4
14.4.1	XBS Priority Registers (XBS_PRSn) . . . . .	14-5
14.4.2	XBS Control Registers (XBS_CRSn) . . . . .	14-7
14.5	Functional Description . . . . .	14-8
14.5.1	Arbitration . . . . .	14-8
14.6	Initialization/Application Information . . . . .	14-9

## Chapter 15 Pin-Multiplexing and Control

15.1	Introduction . . . . .	15-1
15.1.1	Overview . . . . .	15-2
15.1.2	Features . . . . .	15-3
15.2	External Signal Description . . . . .	15-3
15.3	Memory Map/Register Definition . . . . .	15-12
15.3.1	Port Output Data Registers (PODR_x) . . . . .	15-14
15.3.2	Port Data Direction Registers (PDDR_x) . . . . .	15-15
15.3.3	Port Pin Data/Set Data Registers (PPDSDR_x) . . . . .	15-15
15.3.4	Port Clear Output Data Registers (PCLRR_x) . . . . .	15-16
15.3.5	Pull Control Registers (PCR_x) . . . . .	15-17
15.3.6	Pin Assignment Registers (PAR_x) . . . . .	15-17
15.3.7	SDRAM Mode Select Control Registers (MSCR_SDRAMC) . . . . .	15-24
15.3.8	Slew Rate Control Registers (SRCR_x) . . . . .	15-25
15.3.9	UART RTS and CTS Polarity Control Register (URTS_POL & UCTS_POL) . . . . .	15-29
15.3.10	UART Transmitter & Receiver Wired-Or Mode Control Registers (UTXD_WOM &	

URXD_WOM)15-29	
15.4 Hysteresis Control Registers (HCR0–1) . . . . .	15-30
15.5 Functional Description . . . . .	15-31
15.5.1 Overview . . . . .	15-31
15.5.2 Port Digital I/O Timing . . . . .	15-31
15.6 Initialization/Application Information . . . . .	15-32

## Chapter 16 Rapid GPIO (RGPIO)

16.1 Introduction . . . . .	16-1
16.1.1 Overview . . . . .	16-1
16.1.2 Features . . . . .	16-2
16.1.3 Modes of Operation . . . . .	16-3
16.2 External Signal Description . . . . .	16-3
16.2.1 Overview . . . . .	16-3
16.2.2 Detailed Signal Descriptions . . . . .	16-3
16.3 Memory Map/Register Definition . . . . .	16-4
16.3.1 RGPIO Base Address Register (RGPIOBAR) . . . . .	16-5
16.3.2 RGPIO Data Direction (RGPIO_DIR) . . . . .	16-5
16.3.3 RGPIO Data (RGPIO_DATA) . . . . .	16-6
16.3.4 RGPIO Pin Enable (RGPIO_ENB) . . . . .	16-6
16.3.5 RGPIO Clear Data (RGPIO_CLR) . . . . .	16-7
16.3.6 RGPIO Set Data (RGPIO_SET) . . . . .	16-7
16.3.7 RGPIO Toggle Data (RGPIO_TOG) . . . . .	16-8
16.4 Functional Description . . . . .	16-8
16.5 Initialization Information . . . . .	16-8
16.6 Application Information . . . . .	16-9
16.6.1 Application 1: Simple Square-Wave Generation . . . . .	16-9
16.6.2 Application 2: 16-bit Message Transmission using SPI Protocol . . . . .	16-10

## Chapter 17 Interrupt Controller Modules

17.1 Introduction . . . . .	17-1
17.1.1 68 K/ColdFire Interrupt Architecture Overview . . . . .	17-1
17.2 Memory Map/Register Definition . . . . .	17-2
17.2.1 Interrupt Pending Registers (IPRH $n$ , IPRL $n$ ) . . . . .	17-4
17.2.2 Interrupt Mask Register (IMRH $n$ , IMRL $n$ ) . . . . .	17-5
17.2.3 Interrupt Force Registers (INTFRCH $n$ , INTFRCL $n$ ) . . . . .	17-7
17.2.4 Interrupt Configuration Register (ICONFIG) . . . . .	17-8
17.2.5 Set Interrupt Mask Register (SIMR $n$ ) . . . . .	17-8
17.2.6 Clear Interrupt Mask Register (CIMR $n$ ) . . . . .	17-9
17.2.7 Current Level Mask Register (CLMASK) . . . . .	17-10

17.2.8	Saved Level Mask Register (SLMASK) . . . . .	17-11
17.2.9	Interrupt Control Register (ICR0 $n$ , ICR1 $n$ , ICR2 $n$ ( $n = 00, 01, 02, \dots, 63$ )) . . . . .	17-12
17.2.10	Software and Level 1–7 IACK Registers (SWIACK $n$ , L1IACK $n$ –L7IACK $n$ ) . . . . .	17-19
17.3	Functional Description . . . . .	17-20
17.3.1	Interrupt Controller Theory of Operation . . . . .	17-20
17.3.2	Prioritization Between Interrupt Controllers . . . . .	17-22
17.3.3	Low-Power Wake-up Operation . . . . .	17-22
17.4	Initialization/Application Information . . . . .	17-23
17.4.1	Interrupt Service Routines . . . . .	17-23

## Chapter 18 Edge Port Module (EPORT)

18.1	Introduction . . . . .	18-1
18.2	Low-Power Mode Operation . . . . .	18-2
18.3	Signal Descriptions . . . . .	18-2
18.4	Memory Map/Register Definition . . . . .	18-2
18.4.1	EPORT Pin Assignment Register (EPPAR) . . . . .	18-3
18.4.2	Edge Port Interrupt Enable Register (EPIER) . . . . .	18-4
18.4.3	Edge Port Flag Register (EPFR) . . . . .	18-4

## Chapter 19 Enhanced Direct Memory Access (eDMA)

19.1	Overview . . . . .	19-1
19.1.1	Block Diagram . . . . .	19-1
19.1.2	Features . . . . .	19-2
19.2	Modes of Operation . . . . .	19-2
19.2.1	Normal Mode . . . . .	19-2
19.2.2	Debug Mode . . . . .	19-3
19.3	External Signal Description . . . . .	19-3
19.3.1	External Signal Timing . . . . .	19-3
19.4	Memory Map/Register Definition . . . . .	19-4
19.4.1	eDMA Control Register (EDMA_CR) . . . . .	19-5
19.4.2	eDMA Error Status Register (EDMA_ES) . . . . .	19-7
19.4.3	eDMA Enable Request Registers (EDMA_ERQH, EDMA_ERQL) . . . . .	19-10
19.4.4	eDMA Enable Error Interrupt Registers (EDMA_EEIH, EDMA_EEIL) . . . . .	19-13
19.4.5	eDMA Set Enable Request Register (EDMA_SERQ) . . . . .	19-14
19.4.6	eDMA Clear Enable Request Register (EDMA_CERQ) . . . . .	19-15
19.4.7	eDMA Set Enable Error Interrupt Register (EDMA_SEEI) . . . . .	19-16
19.4.8	eDMA Clear Enable Error Interrupt Register (EDMA_CEEI) . . . . .	19-16
19.4.9	eDMA Clear Interrupt Request Register (EDMA_CINT) . . . . .	19-17
19.4.10	eDMA Clear Error Register (EDMA_CERR) . . . . .	19-18
19.4.11	eDMA Set START Bit Register (EDMA_SSRT) . . . . .	19-18

19.4.12 eDMA Clear DONE Status Bit Register (EDMA_CDNE) . . . . .	19-19
19.4.13 eDMA Interrupt Request Registers (EDMA_INTH, EDMA_INTL) . . . . .	19-20
19.4.14 eDMA Error Registers (EDMA_ERRH, EDMA_ERRL) . . . . .	19-21
19.4.15 eDMA Hardware Request Status Registers (EDMA_HRSH, EDMA_HRSL) .	19-22
19.4.16 eDMA Channel n Priority Registers (DCHPRIn) . . . . .	19-23
19.4.17 Transfer Control Descriptors (TCDn) . . . . .	19-24
<b>19.5 Functional Description . . . . .</b>	<b>19-32</b>
19.5.1 eDMA Microarchitecture . . . . .	19-32
19.5.2 eDMA Basic Data Flow . . . . .	19-33
<b>19.6 Initialization/Application Information . . . . .</b>	<b>19-36</b>
19.6.1 eDMA Initialization . . . . .	19-36
19.6.2 DMA Programming Errors . . . . .	19-39
19.6.3 DMA Arbitration Mode Considerations . . . . .	19-40
19.6.4 DMA Transfer . . . . .	19-41
19.6.5 eDMA TCDn Status Monitoring . . . . .	19-44
19.6.6 Channel Linking . . . . .	19-46
19.6.7 Dynamic Programming . . . . .	19-47

## Chapter 20 FlexBus

<b>20.1 Introduction . . . . .</b>	<b>20-1</b>
20.1.1 Overview . . . . .	20-1
20.1.2 Features . . . . .	20-1
20.1.3 Modes of Operation . . . . .	20-2
<b>20.2 External Signals . . . . .</b>	<b>20-2</b>
20.2.1 Address and Data Buses (FB_ADn) . . . . .	20-2
20.2.2 Chip Selects (FB_CS[5:0]) . . . . .	20-3
20.2.3 Byte Enables/Byte Write Enables (FB_BE/BWE[3:0]) . . . . .	20-3
20.2.4 Output Enable (FB_OE) . . . . .	20-3
20.2.5 Read/Write (FB_R/W) . . . . .	20-3
20.2.6 Address Latch Enable (FB_ALE) . . . . .	20-3
20.2.7 Transfer Size (FB_TSIZ[1:0]) . . . . .	20-4
20.2.8 Transfer Burst (FB_TBST) . . . . .	20-4
20.2.9 Transfer Acknowledge (FB_TA) . . . . .	20-5
<b>20.3 Memory Map/Register Definition . . . . .</b>	<b>20-5</b>
20.3.1 Chip-Select Address Registers (CSAR0 – CSAR5) . . . . .	20-6
20.3.2 Chip-Select Mask Registers (CSMR0 – CSMR5) . . . . .	20-6
20.3.3 Chip-Select Control Registers (CSCR0 – CSCR5) . . . . .	20-7
<b>20.4 Functional Description . . . . .</b>	<b>20-10</b>
20.4.1 Chip-Select Operation . . . . .	20-10
20.4.2 Data Transfer Operation . . . . .	20-11
20.4.3 Data Byte Alignment and Physical Connections . . . . .	20-12

20.4.4	Address/Data Bus Multiplexing . . . . .	20-13
20.4.5	Bus Cycle Execution . . . . .	20-13
20.4.6	FlexBus Timing Examples . . . . .	20-15
20.4.7	Burst Cycles. . . . .	20-27
20.4.8	Misaligned Operands. . . . .	20-35
20.4.9	Extended Transfer Start. . . . .	20-35
20.4.10	Bus Errors . . . . .	20-36

## Chapter 21

### DDR1/2 SDRAM Memory Controller (DDRCMC)

21.1	Overview. . . . .	21-1
21.1.1	Block Diagram . . . . .	21-1
21.1.2	Features. . . . .	21-1
21.2	Modes of Operation . . . . .	21-2
21.2.1	DDR1 and DDR2 . . . . .	21-2
21.2.2	Low Power Modes . . . . .	21-2
21.3	Signal Description. . . . .	21-3
21.3.1	Detailed Signal Descriptions . . . . .	21-4
21.4	Memory Map/Register Definition . . . . .	21-6
21.4.1	DDR Control Register 0 (DDR_CR00). . . . .	21-9
21.4.2	DDR Control Register 1 (DDR_CR01). . . . .	21-9
21.4.3	DDR Control Register 2 (DDR_CR02). . . . .	21-10
21.4.4	DDR Control Register 3 (DDR_CR03). . . . .	21-11
21.4.5	DDR Control Register 4 (DDR_CR04). . . . .	21-12
21.4.6	DDR Control Register 5 (DDR_CR05). . . . .	21-12
21.4.7	DDR Control Register 6 (DDR_CR06). . . . .	21-13
21.4.8	DDR Control Register 7 (DDR_CR07). . . . .	21-14
21.4.9	DDR Control Register 8 (DDR_CR08). . . . .	21-15
21.4.10	DDR Control Register 9 (DDR_CR09). . . . .	21-15
21.4.11	DDR Control Register 10 (DDR_CR10). . . . .	21-16
21.4.12	DDR Control Register 11 (DDR_CR11). . . . .	21-17
21.4.13	DDR Control Register 12 (DDR_CR12). . . . .	21-18
21.4.14	DDR Control Register 13 (DDR_CR13). . . . .	21-19
21.4.15	DDR Control Register 14 (DDR_CR14). . . . .	21-20
21.4.16	DDR Control Register 15 (DDR_CR15). . . . .	21-21
21.4.17	DDR Control Register 16 (DDR_CR16). . . . .	21-21
21.4.18	DDR Control Register 17 (DDR_CR17). . . . .	21-22
21.4.19	DDR Control Register 18 (DDR_CR18). . . . .	21-23
21.4.20	DDR Control Register 19 (DDR_CR19). . . . .	21-24
21.4.21	DDR Control Register 20 (DDR_CR20). . . . .	21-24
21.4.22	DDR Control Register 21 (DDR_CR21). . . . .	21-25
21.4.23	DDR Control Register 22 (DDR_CR22). . . . .	21-26

21.4.24 DDR Control Register 23 (DDR_CR23).....	21-27
21.4.25 DDR Control Register 24 (DDR_CR24).....	21-28
21.4.26 DDR Control Register 25 (DDR_CR25).....	21-29
21.4.27 DDR Control Register 26 (DDR_CR26).....	21-30
21.4.28 DDR Control Register 27 (DDR_CR27).....	21-30
21.4.29 DDR Control Register 28 (DDR_CR28).....	21-31
21.4.30 DDR Control Register 29 (DDR_CR29).....	21-32
21.4.31 DDR Control Register 30 (DDR_CR30).....	21-33
21.4.32 DDR Control Register 31 (DDR_CR31).....	21-33
21.4.33 DDR Control Register 32 (DDR_CR32).....	21-34
21.4.34 DDR Control Register 33 (DDR_CR33).....	21-34
21.4.35 DDR Control Register 34 (DDR_CR34).....	21-35
21.4.36 DDR Control Register 35 (DDR_CR35).....	21-35
21.4.37 DDR Control Register 36 (DDR_CR36).....	21-36
21.4.38 DDR Control Register 37 (DDR_CR37).....	21-36
21.4.39 DDR Control Register 38 (DDR_CR38).....	21-36
21.4.40 DDR Control Register 39 (DDR_CR39).....	21-37
21.4.41 DDR Control Register 40 (DDR_CR40).....	21-37
21.4.42 DDR Control Register 41 (DDR_CR41).....	21-38
21.4.43 DDR Control Register 42 (DDR_CR42).....	21-38
21.4.44 DDR Control Register 43 (DDR_CR43).....	21-38
21.4.45 DDR Control Register 44 (DDR_CR44).....	21-39
21.4.46 DDR Control Register 45 (DDR_CR45).....	21-39
21.4.47 DDR Control Register 46 (DDR_CR46).....	21-39
21.4.48 DDR Control Register 47 (DDR_CR47).....	21-40
21.4.49 DDR Control Register 48 (DDR_CR48).....	21-40
21.4.50 DDR Control Register 49 (DDR_CR49).....	21-40
21.4.51 DDR Control Register 50 (DDR_CR50).....	21-40
21.4.52 DDR Control Register 51 (DDR_CR51).....	21-40
21.4.53 DDR Control Register 52 (DDR_CR52).....	21-40
21.4.54 DDR Control Register 53 (DDR_CR53).....	21-40
21.4.55 DDR Control Register 54 (DDR_CR54).....	21-40
21.4.56 DDR Control Register 55 (DDR_CR55).....	21-41
21.4.57 DDR Control Register 56 (DDR_CR56).....	21-41
21.4.58 DDR Control Register 57 (DDR_CR57).....	21-42
21.4.59 DDR Control Register 58 (DDR_CR58).....	21-42
21.4.60 DDR Control Register 59 (DDR_CR59).....	21-43
21.4.61 DDR Control Register 60 (DDR_CR60).....	21-43
21.4.62 DDR Control Register 61 (DDR_CR61).....	21-43
21.4.63 DDR Control Register 62 (DDR_CR62).....	21-44
21.4.64 DDR Control Register 63 (DDR_CR63).....	21-44
21.4.65 RCR Control Register (DDR_RCR).....	21-44

21.4.66 DDR I/O Pad Control Register (DDR_PADCR) . . . . .	21-44
<b>21.5 Functional Description . . . . .</b>	<b>21-45</b>
21.5.1 High-Level Memory Controller Blocks . . . . .	21-45
21.5.2 Address Mapping . . . . .	21-46
21.5.3 Write Data Queue . . . . .	21-47
21.5.4 DRAM Command Processing . . . . .	21-48
21.5.5 Latency . . . . .	21-48
21.5.6 Core Command Queue with Placement Logic. . . . .	21-49
21.5.7 Command Execution Order After Placement. . . . .	21-51
21.5.8 Low Power Modes . . . . .	21-52
21.5.9 Out-of-Range Address Checking. . . . .	21-55
<b>21.6 Initialization/Application Information . . . . .</b>	<b>21-56</b>

## Chapter 22

### NAND Flash Controller (NFC)

<b>22.1 Introduction. . . . .</b>	<b>22-1</b>
22.1.1 Block Diagram . . . . .	22-2
22.1.2 Features. . . . .	22-2
<b>22.2 External Signal Description . . . . .</b>	<b>22-3</b>
<b>22.3 Memory Map/Register Definition . . . . .</b>	<b>22-3</b>
22.3.1 Flash Command 1 Register (NFC_CMD1) . . . . .	22-4
22.3.2 Flash Command 2 Register (NFC_CMD2) . . . . .	22-5
22.3.3 Column Address Register (NFC_CAR) . . . . .	22-6
22.3.4 Row Address Register (NFC_RAR). . . . .	22-6
22.3.5 Flash Command Repeat (NFC_RPT) . . . . .	22-7
22.3.6 Row Address Increment (NFC_RAI) . . . . .	22-7
22.3.7 Flash Status 1 Register (NFC_SR1) . . . . .	22-8
22.3.8 Flash Status 2 Register (NFC_SR2) . . . . .	22-8
22.3.9 DMA1 Address Register (NFC_DMA1) . . . . .	22-9
22.3.10 DMA2 Address Register (NFC_DMA2) . . . . .	22-9
22.3.11 DMA Configuration Register (NFC_DMACFG) . . . . .	22-9
22.3.12 Cache Swap Register (NFC_SWAP). . . . .	22-10
22.3.13 Sector Size Register (NFC_SECSZ) . . . . .	22-11
22.3.14 Flash Configuration Register (NFC_CFG) . . . . .	22-11
22.3.15 Interrupt Status Register (NFC_ISR) . . . . .	22-13
<b>22.4 Functional Description . . . . .</b>	<b>22-14</b>
22.4.1 NFC Buffer Memory Space . . . . .	22-15
22.4.2 Error Corrector Status . . . . .	22-16
22.4.3 NFC Basic Commands . . . . .	22-17
22.4.4 NAND Flash Boot . . . . .	22-22
22.4.5 Fast Flash Configuration for EDO . . . . .	22-24
22.4.6 Organization of the Data in the NAND Flash. . . . .	22-25

22.4.7	Flash Command Code Description . . . . .	22-28
22.4.8	Interrupts . . . . .	22-29

## Chapter 23

### Universal Serial Bus Interface – Host Module

23.1	Introduction . . . . .	23-1
23.1.1	Block Diagram . . . . .	23-2
23.1.2	Overview . . . . .	23-2
23.1.3	Features . . . . .	23-2
23.1.4	Modes of Operation . . . . .	23-3
23.2	External Signal Description . . . . .	23-4
23.2.1	USB Host Control and Status Signals . . . . .	23-4
23.3	Memory Map/Register Definitions . . . . .	23-5
23.4	Functional Description . . . . .	23-6

## Chapter 24

### Universal Serial Bus Interface – On-The-Go Module

24.1	Introduction . . . . .	24-1
24.1.1	Overview . . . . .	24-1
24.1.2	Block Diagram . . . . .	24-2
24.1.3	Features . . . . .	24-3
24.1.4	Modes of Operation . . . . .	24-4
24.2	External Signal Description . . . . .	24-5
24.2.1	USB OTG Control and Status Signals . . . . .	24-5
24.3	Memory Map/Register Definition . . . . .	24-7
24.3.1	Module Identification Registers . . . . .	24-8
24.3.2	Device/Host Timer Registers . . . . .	24-12
24.3.3	Capability Registers . . . . .	24-13
24.3.4	Operational Registers . . . . .	24-17
24.4	Functional Description . . . . .	24-46
24.4.1	System Interface . . . . .	24-46
24.4.2	DMA Engine . . . . .	24-47
24.4.3	FIFO RAM Controller . . . . .	24-47
24.4.4	Physical Layer (PHY) Interface . . . . .	24-47
24.5	Initialization/Application Information . . . . .	24-48
24.5.1	Host Operation . . . . .	24-48
24.5.2	Device Data Structures . . . . .	24-49
24.5.3	Device Operation . . . . .	24-56
24.5.4	Servicing Interrupts . . . . .	24-74
24.5.5	Deviations from the EHCI Specifications . . . . .	24-75

## Chapter 25

### Enhanced Secure Digital Host Controller

25.1	Overview . . . . .	25-1
25.1.1	Block Diagram . . . . .	25-3
25.1.2	Features . . . . .	25-3
25.1.3	Data Transfer Modes . . . . .	25-4
25.2	External Signal Description . . . . .	25-4
25.3	Memory Map/Register Definition . . . . .	25-5
25.3.1	DMA System Address Register (DSADDR) . . . . .	25-6
25.3.2	Block Attributes Register (BLKATTR) . . . . .	25-7
25.3.3	Command Argument Register (CMDARG) . . . . .	25-8
25.3.4	Transfer Type Register (XFERTYP) . . . . .	25-8
25.3.5	Command Response 0–3 (CMDRSP0–3) . . . . .	25-11
25.3.6	Buffer Data Port Register (DATPORT) . . . . .	25-13
25.3.7	Present State Register (PRSSTAT) . . . . .	25-13
25.3.8	Protocol Control Register (PROCTL) . . . . .	25-17
25.3.9	System Control Register (SYSCTL) . . . . .	25-20
25.3.10	Interrupt Status Register (IRQSTAT) . . . . .	25-22
25.3.11	Interrupt Status Enable Register (IRQSTATEN) . . . . .	25-26
25.3.12	Interrupt Signal Enable Register (IRQSIGEN) . . . . .	25-29
25.3.13	Auto CMD12 Error Status Register (AUTOC12ERR) . . . . .	25-30
25.3.14	Host Controller Capabilities (HOSTCAPBLT) . . . . .	25-33
25.3.15	Watermark Level Register (WML) . . . . .	25-34
25.3.16	Force Event Register (FEVT) . . . . .	25-34
25.3.17	ADMA Error Status Register . . . . .	25-36
25.3.18	ADMA System Address Register . . . . .	25-37
25.3.19	Vendor Specific Register . . . . .	25-38
25.3.20	Host Controller Version Register (HOSTVER) . . . . .	25-39
25.4	Functional Description . . . . .	25-39
25.4.1	Data Buffer . . . . .	25-39
25.4.2	DMA Crossbar Switch Interface . . . . .	25-43
25.4.3	SD Protocol Unit . . . . .	25-48
25.4.4	Clock & Reset Manager . . . . .	25-49
25.4.5	Clock Generator . . . . .	25-50
25.4.6	SDIO Card Interrupt . . . . .	25-50
25.4.7	Card Insertion and Removal Detection . . . . .	25-52
25.4.8	Power Management and Wake-Up Events . . . . .	25-52
25.5	Initialization/Application Information . . . . .	25-53
25.5.1	Command Send and Response Receive Basic Operation . . . . .	25-53
25.5.2	Card Identification Mode . . . . .	25-54
25.5.3	Card Access . . . . .	25-58
25.5.4	Switch Function . . . . .	25-64

25.5.5	Commands for MMC/SD/SDIO . . . . .	25-68
25.5.6	Software Restrictions. . . . .	25-72

## Chapter 26

### Cryptographic Acceleration Unit (CAU)

26.1	Introduction. . . . .	26-1
26.1.1	Block Diagram . . . . .	26-1
26.1.2	Overview . . . . .	26-1
26.1.3	Features. . . . .	26-2
26.2	Memory Map/Register Definition . . . . .	26-2
26.2.1	CAU Status Register (CASR) . . . . .	26-3
26.2.2	CAU Accumulator (CAA) . . . . .	26-4
26.2.3	CAU General Purpose Registers (CAn) . . . . .	26-4
26.3	Functional Description . . . . .	26-4
26.3.1	Programming Model . . . . .	26-4
26.3.2	Coprocessor Instructions. . . . .	26-5
26.3.3	CAU Commands . . . . .	26-5
26.4	Application/Initialization Information . . . . .	26-11
26.4.1	Code Example . . . . .	26-11
26.4.2	Assembler Equate Values . . . . .	26-11

## Chapter 27

### Random Number Generator (RNG)

27.1	Introduction. . . . .	27-1
27.1.1	Block Diagram . . . . .	27-1
27.1.2	Overview . . . . .	27-1
27.1.3	Features. . . . .	27-2
27.2	Modes of Operation . . . . .	27-2
27.2.1	Self Test Mode. . . . .	27-2
27.2.2	Seed Generation Mode . . . . .	27-2
27.2.3	Random Number Generation Mode. . . . .	27-2
27.3	Memory Map/Register Definition . . . . .	27-3
27.3.1	RNG Version ID Register (RNGVER) . . . . .	27-3
27.3.2	RNG Command Register (RNGCMD) . . . . .	27-4
27.3.3	RNG Control Register (RNGCR) . . . . .	27-5
27.3.4	RNG Status Register (RNGSR) . . . . .	27-6
27.3.5	RNG Error Status Register (RNGESR) . . . . .	27-7
27.3.6	RNG Output FIFO (RNGOUT) . . . . .	27-8
27.3.7	RNG Entropy Register (RNGER) . . . . .	27-8
27.4	Functional Description . . . . .	27-9
27.4.1	Pseudorandom Number Generator (PRNG) . . . . .	27-9
27.4.2	True Random Number Generator (TRNG) . . . . .	27-9

27.4.3 RNG Interrupts . . . . .	27-9
27.5 Initialization/Application Information . . . . .	27-10
27.5.1 Manual Seeding . . . . .	27-10
27.5.2 Automatic Seeding . . . . .	27-10

## Chapter 28

### Subscriber Identification Module (SIM)

28.1 Introduction . . . . .	28-1
28.1.1 Block Diagram . . . . .	28-1
28.1.2 Features . . . . .	28-1
28.1.3 Modes of Operation . . . . .	28-2
28.2 External Signal Description . . . . .	28-2
28.3 Memory Map/Register Definition . . . . .	28-2
28.3.1 SIM Port Control Registers (SIM_PCR $n$ ) . . . . .	28-4
28.3.2 SIM Port 1 Setup Register (SIM_SETUP) . . . . .	28-5
28.3.3 SIM Port Detect Registers (SIM_DETECT $n$ ) . . . . .	28-6
28.3.4 SIM Port Transmit Buffer Registers (SIM_TBUFn) . . . . .	28-7
28.3.5 SIM Port Receive Buffer Registers (SIM_RBUFn) . . . . .	28-7
28.3.6 SIM Control Register (SIM_CR) . . . . .	28-8
28.3.7 SIM Clock Prescaler Register (SIM_PRE) . . . . .	28-10
28.3.8 SIM Receive Threshold Register (SIM_RTHR) . . . . .	28-11
28.3.9 SIM Enable Register (SIM_EN) . . . . .	28-11
28.3.10 SIM Transmit Status Register (SIM_TSР) . . . . .	28-12
28.3.11 SIM Receive Status Register (SIM_RSR) . . . . .	28-13
28.3.12 SIM Interrupt Mask Register (SIM_IMR) . . . . .	28-15
28.3.13 SIM Data Format Register (SIM_FORMAT) . . . . .	28-17
28.3.14 SIM Transmit Threshold Register (SIM_TTHR) . . . . .	28-17
28.3.15 SIM Transmit Guard Control Register (SIM_TGCR) . . . . .	28-18
28.3.16 SIM Open Drain Configuration Control Register (SIM_ODCR) . . . . .	28-19
28.3.17 SIM Reset Control Register (SIM_RCR) . . . . .	28-19
28.3.18 SIM Character Wait Time Register (SIM_CWTR) . . . . .	28-20
28.3.19 SIM General Purpose Counter Register (SIM_GPCNT) . . . . .	28-21
28.3.20 SIM Divisor Register (SIM_DIV) . . . . .	28-21
28.3.21 SIM Block Wait Time Low Register (SIM_BWTL) . . . . .	28-22
28.3.22 SIM Block Guard Time Register (SIM_BGT) . . . . .	28-22
28.3.23 SIM Block Wait Time High Register (SIM_BWTH) . . . . .	28-23
28.3.24 SIM Transmit FIFO Status Register (SIM_TFSR) . . . . .	28-23
28.3.25 SIM Receive FIFO Counter Register (SIM_RFCR) . . . . .	28-24
28.3.26 SIM Receive FIFO Write Pointer Register (SIM_RFWP) . . . . .	28-24
28.3.27 SIM Receive FIFO Read Pointer Register (SIM_RFRP) . . . . .	28-24
28.4 Functional Description . . . . .	28-25
28.4.1 SIM Clock Generator . . . . .	28-25

28.4.2	SIM Transmitter . . . . .	28-26
28.4.3	SIM Receiver . . . . .	28-30
28.4.4	SIM Port Control . . . . .	28-36
28.4.5	SIM General Purpose Counter . . . . .	28-37
28.4.6	SIM LRC Block . . . . .	28-38
28.4.7	SIM CRC Block . . . . .	28-38
28.4.8	Module Interrupts . . . . .	28-40
28.5	Initialization/Application Information . . . . .	28-40
28.5.1	Configuring SIM for Operation . . . . .	28-41
28.5.2	Using SIM Receiver . . . . .	28-44
28.5.3	Using SIM Transmitter . . . . .	28-48
28.5.4	Suggested Programming Model . . . . .	28-50

## Chapter 29

### Analog Digital Converter (ADC)

29.1	Introduction . . . . .	29-1
29.1.1	Features . . . . .	29-1
29.2	Block Diagram . . . . .	29-2
29.3	External Signal Description . . . . .	29-2
29.4	Memory Map/Register Definition . . . . .	29-2
29.4.1	ADC Control Register 1 (ADC_CR1) . . . . .	29-3
29.4.2	ADC Control Register 2 (ADC_CR2) . . . . .	29-6
29.4.3	ADC Zero Crossing Control Register (ADC_ZCCR) . . . . .	29-7
29.4.4	Channel List n Registers (ADC_LST1–2) . . . . .	29-7
29.4.5	Sample Disable Register (ADC_SDIS) . . . . .	29-9
29.4.6	Status Register (ADC_SR) . . . . .	29-9
29.4.7	Limit Status (ADC_LSR) Register . . . . .	29-11
29.4.8	Zero Crossing Status Register (ADC_ZCSR) . . . . .	29-12
29.4.9	ADC Result Registers (ADC_RSLT0–7) . . . . .	29-13
29.4.10	Low and High Limit Registers (ADC_LLMT0–7 & ADCHLMT0–7) . . . . .	29-13
29.4.11	Offset Registers (ADC_OFS0–7) . . . . .	29-14
29.4.12	Power Control Register (ADC_PWR) . . . . .	29-15
29.4.13	Calibration Register (ADC_CAL) . . . . .	29-18
29.4.14	Power Control Register 2 (ADC_PWR2) . . . . .	29-19
29.4.15	Conversion Divisor Register (ADC_DIV) . . . . .	29-19
29.4.16	Auto-Standby Divisor Register (ADC_ASDIV) . . . . .	29-20
29.5	Functional Description . . . . .	29-21
29.5.1	Scan Modes . . . . .	29-22
29.5.2	Input Mux Function . . . . .	29-24
29.5.3	ADC Sample Conversion Modes of Operation . . . . .	29-26
29.5.4	ADC Data Processing . . . . .	29-29
29.5.5	Sequential vs. Parallel Sampling . . . . .	29-30

29.5.6 Scan Sequencing . . . . .	29-31
29.5.7 Power Management . . . . .	29-31
29.5.8 Power Management Details . . . . .	29-33
29.5.9 Clock Operation . . . . .	29-34
29.5.10 Voltage Reference Pins $V_{REFH}$ & $V_{REFLO}$ . . . . .	29-35
29.5.11 Supply Pins $V_{DDA}$ and $V_{SSA}$ . . . . .	29-36
29.5.12 Interrupt Operation . . . . .	29-36

## Chapter 30 Digital-to-Analog Converter (DAC)

30.1 Introduction . . . . .	30-1
30.1.1 Features . . . . .	30-1
30.1.2 Block Diagram . . . . .	30-2
30.2 External Signal Descriptions . . . . .	30-2
30.3 Memory Map/Register Definition . . . . .	30-3
30.3.1 DAC Control Register ( $DACn\_CR$ ) . . . . .	30-3
30.3.2 Buffered Data Register ( $DACn\_DATA$ ) . . . . .	30-5
30.3.3 Step Size Register ( $DACn\_STEP$ ) . . . . .	30-5
30.3.4 Minimum Value Register ( $DACn\_MIN$ ) . . . . .	30-6
30.3.5 Maximum Value Register ( $DACn\_MAX$ ) . . . . .	30-6
30.3.6 Status Register ( $DACn\_SR$ ) . . . . .	30-7
30.3.7 Filter Count Register ( $DACn\_FILTCNT$ ) . . . . .	30-7
30.4 Functional Description . . . . .	30-8
30.4.1 Normal Mode . . . . .	30-8
30.4.2 Automatic Mode . . . . .	30-8
30.4.3 Sources of Waveform Distortion . . . . .	30-10
30.5 Initialization/Application Information . . . . .	30-11

## Chapter 31 10/100Mbps Ethernet MAC-NET Core

31.1 Introduction . . . . .	31-1
31.1.1 Overview . . . . .	31-1
31.1.2 Features . . . . .	31-2
31.1.3 Block Diagram . . . . .	31-4
31.2 External Signal Description . . . . .	31-4
31.3 Memory Map/Register Definition . . . . .	31-5
31.3.1 Interrupt Event Register ( $ENETn\_EIR$ ) . . . . .	31-12
31.3.2 Interrupt Mask Register ( $ENETn\_EIMR$ ) . . . . .	31-13
31.3.3 Receive Descriptor Active Registers ( $ENETn\_RDAR$ ) . . . . .	31-14
31.3.4 Transmit Descriptor Active Registers ( $ENETn\_TDAR$ ) . . . . .	31-15
31.3.5 Ethernet Control Register ( $ENETn\_ECR$ ) . . . . .	31-16
31.3.6 MII Management Frame Register ( $ENETn\_MMFR$ ) . . . . .	31-17

31.3.7	MII Speed Control Register (ENET <sub>n</sub> _MSCR) . . . . .	31-18
31.3.8	MIB Control Register (ENET <sub>n</sub> _MIBC) . . . . .	31-19
31.3.9	Receive Control Register (ENET <sub>n</sub> _RCR) . . . . .	31-20
31.3.10	Transmit Control Register (ENET <sub>n</sub> _TCR) . . . . .	31-21
31.3.11	Physical Address Lower Register (ENET <sub>n</sub> _PALR) . . . . .	31-22
31.3.12	Physical Address Upper Register (ENET <sub>n</sub> _PAUR) . . . . .	31-23
31.3.13	Opcode/Pause Duration Register (ENET <sub>n</sub> _OPD) . . . . .	31-23
31.3.14	Descriptor Individual Upper Address Register (ENET <sub>n</sub> _IAUR) . . . . .	31-24
31.3.15	Descriptor Individual Lower Address Register (ENET <sub>n</sub> _IALR) . . . . .	31-24
31.3.16	Descriptor Group Upper Address Register (ENET <sub>n</sub> _GAUR) . . . . .	31-25
31.3.17	Descriptor Group Lower Address Register (ENET <sub>n</sub> _GALR) . . . . .	31-25
31.3.18	Transmit FIFO Watermark Register (ENET <sub>n</sub> _TFWR) . . . . .	31-26
31.3.19	Receive Descriptor Ring Start Register (ENET <sub>n</sub> _RDSR) . . . . .	31-26
31.3.20	Transmit Buffer Descriptor Ring Start Register (ENET <sub>n</sub> _TDSR) . . . . .	31-27
31.3.21	Maximum Receive Buffer Size Register (ENET <sub>n</sub> _MRBR) . . . . .	31-27
31.3.22	Receive FIFO Section Full Threshold (ENET <sub>n</sub> _RSFL) . . . . .	31-28
31.3.23	Receive FIFO Section Empty Threshold (ENET <sub>n</sub> _RSEM) . . . . .	31-29
31.3.24	Receive FIFO Almost Empty Threshold (ENET <sub>n</sub> _RAEM) . . . . .	31-29
31.3.25	Receive FIFO Almost Full Threshold (ENET <sub>n</sub> _RAFL) . . . . .	31-29
31.3.26	Transmit FIFO Section Empty Threshold (ENET <sub>n</sub> _TSEM) . . . . .	31-30
31.3.27	Transmit FIFO Almost Empty Threshold (ENET <sub>n</sub> _TAEM) . . . . .	31-30
31.3.28	Transmit FIFO Almost Full Threshold (ENET <sub>n</sub> _TAFL) . . . . .	31-31
31.3.29	Transmit Inter-Packet Gap (ENET <sub>n</sub> _TIPG) . . . . .	31-31
31.3.30	Frame Truncation Length (ENET <sub>n</sub> _FTRL) . . . . .	31-32
31.3.31	Transmit Accelerator Function Configuration (ENET <sub>n</sub> _TACC) . . . . .	31-32
31.3.32	Receive Accelerator Function Configuration (ENET <sub>n</sub> _RACC) . . . . .	31-33
31.3.33	Timer Control Register (ENET <sub>n</sub> _ATCR) . . . . .	31-34
31.3.34	Timer Value Register (ENET <sub>n</sub> _ATVR) . . . . .	31-35
31.3.35	Timer Offset Register (ENET <sub>n</sub> _ATOFF) . . . . .	31-36
31.3.36	Timer Period Register (ENET <sub>n</sub> _ATPER) . . . . .	31-36
31.3.37	Timer Correction Register (ENET <sub>n</sub> _ATCOR) . . . . .	31-36
31.3.38	Time-Stamping Clock Period Register (ENET <sub>n</sub> _ATINC) . . . . .	31-37
31.3.39	Timestamp of Last Transmitted Frame (ENET <sub>n</sub> _ATSTMP) . . . . .	31-37
31.3.40	Supplemental MAC Address Lower Registers (ENET <sub>n</sub> _SMACLx) . . . . .	31-38
31.3.41	Supplemental MAC Address Upper Registers (ENET <sub>n</sub> _SMACUX) . . . . .	31-38
31.4	Functional Description . . . . .	31-39
31.4.1	Ethernet MAC Frame Formats. . . . .	31-39
31.4.2	IP and Higher Layers Frame Format. . . . .	31-41
31.4.3	IEEE 1588 Message Formats . . . . .	31-45
31.4.4	MAC Receive. . . . .	31-49
31.4.5	MAC Transmit . . . . .	31-54
31.4.6	Full Duplex Flow Control Operation. . . . .	31-56

31.4.7	Magic Packet Detection . . . . .	31-58
31.4.8	IP Accelerator Functions . . . . .	31-59
31.4.9	Resets and Stop Controls . . . . .	31-62
31.4.10	IEEE 1588 Functions . . . . .	31-64
31.4.11	FIFO Thresholds . . . . .	31-68
31.4.12	Loopback Options . . . . .	31-70
31.4.13	Legacy Buffer Descriptors . . . . .	31-71
31.4.14	Enhanced Buffer Descriptors . . . . .	31-72
31.4.15	Client FIFO Application Interface . . . . .	31-78
31.4.16	FIFO Protection . . . . .	31-81
31.4.17	PHY Management Interface . . . . .	31-82
31.4.18	MII Interface . . . . .	31-84

## Chapter 32

### Ethernet Switch

32.1	Introduction . . . . .	32-1
32.1.1	Block Diagram . . . . .	32-2
32.1.2	Features . . . . .	32-3
32.2	Modes of Operation . . . . .	32-3
32.2.1	Passthrough Mode . . . . .	32-4
32.2.2	Switch Mode . . . . .	32-4
32.3	Memory Map/Register Definition . . . . .	32-6
32.3.1	Revision Register (ESW_REV) . . . . .	32-9
32.3.2	Scratch Register (ESW_SCR) . . . . .	32-9
32.3.3	Port Enable Register (ESW_PER) . . . . .	32-9
32.3.4	VLAN Verify (ESW_VLANV) . . . . .	32-10
32.3.5	Default Broadcast Resolution (ESW_DBCR) . . . . .	32-10
32.3.6	Default Multicast Resolution (ESW_DMCR) . . . . .	32-11
32.3.7	Blocking and Learning Enable (ESW_BKLR) . . . . .	32-11
32.3.8	Bridge Management Port Configuration (ESW_BMPC) . . . . .	32-12
32.3.9	Mode Configuration Register (ESW_MODE) . . . . .	32-13
32.3.10	VLAN Input Manipulation Select (ESW_VIMSEL) . . . . .	32-14
32.3.11	VLAN Output Manipulation Select (ESW_VOMSEL) . . . . .	32-14
32.3.12	VLAN input manipulation enable (ESW_VIMEN) . . . . .	32-15
32.3.13	VLAN Tag ID (ESW_VID) . . . . .	32-15
32.3.14	Mirror control register (ESW_MCR) . . . . .	32-16
32.3.15	Egress Port Definitions (ESW_EGMAP) . . . . .	32-17
32.3.16	Ingress Port Definitions (ESW_INGMAP) . . . . .	32-17
32.3.17	Ingress and Egress MAC Address Registers . . . . .	32-18
32.3.18	Mirror Count Value (ESW_MCVAL) . . . . .	32-18
32.3.19	Memory Manager Status Register (ESW_MMSR) . . . . .	32-19
32.3.20	Low Memory Threshold (ESW_LMT) . . . . .	32-20

32.3.21 Lowest Number of Free Cells (ESW_LFC) . . . . .	32-21
32.3.22 Port Congestion Status (ESW_PCSR) . . . . .	32-21
32.3.23 Input/Output Interface Status Register (ESW_IOSR) . . . . .	32-21
32.3.24 Queue Weights (ESW_QWT) . . . . .	32-22
32.3.25 Port 0 Backpressure Congestion Threshold (ESW_P0BCT) . . . . .	32-23
32.3.26 Port 0 Forced Forwarding Enable (ESW_FFEN) . . . . .	32-23
32.3.27 Port Snooping Registers (ESW_PSNP1–8) . . . . .	32-24
32.3.28 IP snooping registers (ESW_IPSNP1–8) . . . . .	32-25
32.3.29 Port 0–2 VLAN Priority Resolution Map (ESW_PnVRES) . . . . .	32-25
32.3.30 IPv4/v6 Priority Resolution Table (ESW_IPRES) . . . . .	32-26
32.3.31 Port $n$ Priority Resolution Configuration (ESW_PnRES) . . . . .	32-27
32.3.32 Port $n$ VLAN ID (ESW_PnID) . . . . .	32-28
32.3.33 VLAN Domain Resolution 0–31 (ESW_VRES0–31) . . . . .	32-28
32.3.34 Statistics Registers . . . . .	32-29
32.3.35 Port Statistics Registers . . . . .	32-29
32.3.36 Interrupt Status Register (ESW_ISR) . . . . .	32-29
32.3.37 Interrupt Mask Register (ESW_IMR) . . . . .	32-31
32.3.38 Receive Descriptor Ring Pointer (ESW_RDSR) . . . . .	32-31
32.3.39 Transmit Descriptor Ring Pointer (ESW_TDSR) . . . . .	32-32
32.3.40 Maximum receive buffer size (ESW_MRBR) . . . . .	32-32
32.3.41 Receive Descriptor Active Register (ESW_RDAR) . . . . .	32-33
32.3.42 Transmit Descriptor Active Register (ESW_TDAR) . . . . .	32-33
32.3.43 Learning Record Registers (ESW_LREC0 and ESW_LREC1) . . . . .	32-34
32.3.44 Learning Data Status Register (ESW_LSR) . . . . .	32-35
32.3.45 Look-up Memory Table . . . . .	32-35
<b>32.4 Functional Description . . . . .</b>	<b>32-36</b>
32.4.1 VLAN Input Processing Function . . . . .	32-36
32.4.2 IP Snooping . . . . .	32-37
32.4.3 TCP/UDP Port Number Snooping . . . . .	32-38
32.4.4 VLAN Output Processing Function . . . . .	32-38
32.4.5 Frame Classification and Priority Resolution . . . . .	32-39
32.4.6 Input Port Selection . . . . .	32-42
32.4.7 Layer 2 Look-Up Engine . . . . .	32-42
32.4.8 Layer 2 Lookup Tasks Overview . . . . .	32-44
32.4.9 Frame-Forwarding Tasks . . . . .	32-47
32.4.10 Output Frame Queuing . . . . .	32-52
32.4.11 Reset and Stop Functions . . . . .	32-54

## Chapter 33 FlexCAN

<b>33.1 Introduction . . . . .</b>	<b>33-1</b>
33.1.1 Block Diagram . . . . .	33-1

33.1.2	Features . . . . .	33-3
33.1.3	Modes of Operation . . . . .	33-4
33.2	External Signal Description . . . . .	33-5
33.3	Memory Map/Register Definition . . . . .	33-5
33.3.1	FlexCAN Configuration Register (CANMCR $n$ ) . . . . .	33-7
33.3.2	FlexCAN Control Register (CANCTRL $n$ ) . . . . .	33-9
33.3.3	FlexCAN Free Running Timer Register (TIMER $n$ ) . . . . .	33-12
33.3.4	Rx Mask Registers (RXGMASK $n$ , RX14MASK $n$ , RX15MASK $n$ ) . . . . .	33-13
33.3.5	FlexCAN Error Counter Register (ERRCNT $n$ ) . . . . .	33-14
33.3.6	FlexCAN Error and Status Register (ERRSTAT $n$ ) . . . . .	33-16
33.3.7	Interrupt Mask Register (IMASK $n$ ) . . . . .	33-18
33.3.8	Interrupt Flag Register (IFLAG $n$ ) . . . . .	33-18
33.3.9	Message Buffer Structure . . . . .	33-19
33.3.10	Rx FIFO Structure . . . . .	33-23
33.3.11	Rx Individual Masking Registers (RXIMR0–15) . . . . .	33-25
33.3.12	Functional Overview . . . . .	33-26
33.3.13	Transmit Process . . . . .	33-26
33.3.14	Arbitration Process . . . . .	33-26
33.3.15	Receive Process . . . . .	33-27
33.3.16	Matching Process . . . . .	33-29
33.3.17	Message Buffer Managing . . . . .	33-30
33.3.18	Rx FIFO . . . . .	33-32
33.3.19	CAN Protocol Related Frames . . . . .	33-33
33.3.20	Time Stamp . . . . .	33-34
33.3.21	Protocol Timing . . . . .	33-34
33.3.22	Arbitration and Matching Timing . . . . .	33-36
33.4	Initialization/Application Information . . . . .	33-37
33.4.1	Interrupts . . . . .	33-38
33.4.2	Mask Misalignment for Rx FIFO . . . . .	33-38

## Chapter 34 Motor Control Pulse-Width Modulator (mcPWM)

34.1	Introduction . . . . .	34-1
34.1.1	Overview . . . . .	34-1
34.1.2	Block Diagram . . . . .	34-2
34.1.3	Features . . . . .	34-2
34.1.4	Modes of Operation . . . . .	34-3
34.2	External Signal Descriptions . . . . .	34-4
34.3	Memory Map/Register Description . . . . .	34-4
34.3.1	Counter Register (PWM_SM $n$ CNT) . . . . .	34-7
34.3.2	Initial Count Register (PWM_SM $n$ INIT) . . . . .	34-7
34.3.3	Control Register 2 (PWM_SM $n$ CR2) . . . . .	34-7

34.3.4	Control Register 1 (PWM_SMnCR1) . . . . .	34-9
34.3.5	Value Registers (PWM_SMnVAL0–5) . . . . .	34-10
34.3.6	Output Control Register (PWM_SMnOCR) . . . . .	34-11
34.3.7	Status Register (PWM_SMnSR) . . . . .	34-13
34.3.8	Interrupt Enable Register (PWM_SMnIER) . . . . .	34-14
34.3.9	DMA Enable Register (PWM_SMnDMAEN) . . . . .	34-14
34.3.10	Output Trigger Control Register (PWM_SMnOTCR) . . . . .	34-15
34.3.11	Fault Disable Mapping Register (PWM_SMnDISMAP) . . . . .	34-16
34.3.12	Deadtime Count Registers (PWM_SMnDTCNT $m$ ) . . . . .	34-17
34.3.13	Capture Control Registers (PWM_SMnCCR $x$ ) . . . . .	34-17
34.3.14	Capture Compare Registers (PWM_SMnCCMP $x$ ) . . . . .	34-19
34.3.15	Capture Value Registers (PWM_SMnCVAL $m$ ) . . . . .	34-20
34.3.16	Capture Value Cycle Registers (PWM_SMnCCYC $m$ ) . . . . .	34-21
34.3.17	Output Enable Register (PWM_OUTEN) . . . . .	34-21
34.3.18	Mask Register (PWM_MASK) . . . . .	34-22
34.3.19	Software-Controlled Output Register (PWM_SWCOUT) . . . . .	34-23
34.3.20	Deadtime Source Select Register (PWM_DTSS) . . . . .	34-23
34.3.21	Master Control Register (PWM_MCR) . . . . .	34-24
34.3.22	Fault Control Register (PWM_FCR) . . . . .	34-25
34.3.23	Fault Status Register (PWM_FSR) . . . . .	34-26
34.3.24	Fault Filter Register (PWM_FFILT) . . . . .	34-27
34.4	Functional Description . . . . .	34-28
34.4.1	Center-Aligned PWMs . . . . .	34-28
34.4.2	Edge-Aligned PWMs . . . . .	34-29
34.4.3	Phase-Shifted PWMs . . . . .	34-29
34.4.4	Double-Switching PWMs . . . . .	34-31
34.4.5	ADC Triggering . . . . .	34-32
34.4.6	Enhanced Capture Capabilities (E-Capture) . . . . .	34-33
34.4.7	Synchronous Switching of Multiple Outputs . . . . .	34-34
34.4.8	Functional Details . . . . .	34-36
34.4.9	PWM Generator Loading . . . . .	34-52
34.5	Initialization/Application Information . . . . .	34-54
34.5.1	Interrupt Requests . . . . .	34-55
34.5.2	DMA Requests . . . . .	34-56

## Chapter 35 Synchronous Serial Interface (SSI)

35.1	Introduction . . . . .	35-1
35.1.1	Overview . . . . .	35-2
35.1.2	Features . . . . .	35-3
35.1.3	Modes of Operation . . . . .	35-3
35.2	External Signal Description . . . . .	35-5

35.2.1	SSI_CLKIN — SSI Clock Input . . . . .	35-5
35.2.2	SSIn_BCLK — Serial Bit Clock . . . . .	35-5
35.2.3	SSIn_MCLK — Serial Master Clock . . . . .	35-5
35.2.4	SSIn_FS — Serial Frame Sync . . . . .	35-5
35.2.5	SSIn_RXD — Serial Receive Data . . . . .	35-5
35.2.6	SSIn_TXD — Serial Transmit Data . . . . .	35-6
35.3	Memory Map/Register Definition . . . . .	35-7
35.3.1	SSI Transmit Data Registers 0 and 1 (SSIn_TX0/1) . . . . .	35-8
35.3.2	SSI Transmit FIFO 0 and 1 Registers . . . . .	35-9
35.3.3	SSI Transmit Shift Register (TXSR) . . . . .	35-9
35.3.4	SSI Receive Data Registers 0 and 1 (SSIn_RX0/1) . . . . .	35-11
35.3.5	SSI Receive FIFO 0 and 1 Registers. . . . .	35-11
35.3.6	SSI Receive Shift Register (RXSR) . . . . .	35-11
35.3.7	SSI Control Register (SSIn_CR) . . . . .	35-13
35.3.8	SSI Interrupt Status Register (SSIn_ISR) . . . . .	35-15
35.3.9	SSI Interrupt Enable Register (SSIn_IER) . . . . .	35-21
35.3.10	SSI Transmit Configuration Register (SSIn_TCR) . . . . .	35-23
35.3.11	SSI Receive Configuration Register (SSIn_RCR) . . . . .	35-24
35.3.12	SSI Clock Control Register (SSIn_CCR) . . . . .	35-25
35.3.13	SSI FIFO Control/Status Register (SSIn_FCSR) . . . . .	35-27
35.3.14	SSI AC97 Control Register (SSIn_ACR) . . . . .	35-33
35.3.15	SSI AC97 Command Address Register (SSIn_ACADD) . . . . .	35-34
35.3.16	SSI AC97 Command Data Register (SSIn_ACDAT) . . . . .	35-34
35.3.17	SSI AC97 Tag Register (SSIn_ATAG) . . . . .	35-35
35.3.18	SSI Transmit Time Slot Mask Register (SSIn_TMASK) . . . . .	35-35
35.3.19	SSI Receive Time Slot Mask Register (SSIn_RMASK) . . . . .	35-36
35.3.20	SSI AC97 Channel Status Register (SSI_ACCSR) . . . . .	35-36
35.3.20	<i>n</i> . . . . .	35-36
35.3.21	SSI AC97 Channel Enable Register (SSI_ACCEN) . . . . .	35-37
35.3.21	<i>n</i> . . . . .	35-37
35.3.22	SSI AC97 Channel Disable Register (SSI_ACCDIS) . . . . .	35-37
35.3.22	<i>n</i> . . . . .	35-37
35.4	Functional Description . . . . .	35-38
35.4.1	Detailed Operating Mode Descriptions . . . . .	35-38
35.4.2	SSI Clocking . . . . .	35-50
35.4.3	External Frame and Clock Operation . . . . .	35-53
35.4.4	Supported Data Alignment Formats . . . . .	35-53
35.4.5	Receive Interrupt Enable Bit Description . . . . .	35-55
35.4.6	Transmit Interrupt Enable Bit Description . . . . .	35-55
35.4.7	Internal Frame and Clock Shutdown . . . . .	35-56
35.5	Initialization/Application Information . . . . .	35-57

## Chapter 36 1-Wire Module

36.1	Overview . . . . .	36-1
36.1.1	Block Diagram . . . . .	36-1
36.1.2	Features . . . . .	36-1
36.1.3	Modes of Operation . . . . .	36-2
36.2	External Signals . . . . .	36-2
36.3	Memory Map/Register Definition . . . . .	36-2
36.3.1	Control Register (OW_CR) . . . . .	36-3
36.3.2	Time Divider Register (OW_DIV) . . . . .	36-4
36.3.3	Reset Register (OW_RST) . . . . .	36-4
36.3.4	Command Register (OW_CMD) . . . . .	36-5
36.3.5	Transmit/Receive Register (OW_TXRX) . . . . .	36-5
36.3.6	Interrupt Register (OW_ISR) . . . . .	36-6
36.3.7	Interrupt Enable Register (OW_IER) . . . . .	36-7
36.4	Functional Description . . . . .	36-8
36.4.1	Normal Operating Modes . . . . .	36-8
36.4.2	Low Power Mode . . . . .	36-10
36.4.3	Clocks . . . . .	36-10
36.4.4	Interrupts . . . . .	36-11

## Chapter 37 Robust Real Time Clock

37.1	Introduction . . . . .	37-1
37.2	Overview . . . . .	37-1
37.2.1	Block Diagram . . . . .	37-2
37.2.2	Features . . . . .	37-2
37.3	External Signal Description . . . . .	37-3
37.4	Memory Map/Register Definition . . . . .	37-3
37.4.1	RTC Year & Month Counter Register (RTC_YEARMON) . . . . .	37-4
37.4.2	RTC Day & Day-of-Week Counters Register (RTC_DAYS) . . . . .	37-5
37.4.3	RTC Hour and Minute Counter Register (RTC_HOURMIN) . . . . .	37-6
37.4.4	RTC Second Counter Register (RTC_SECONDS) . . . . .	37-7
37.4.5	RTC Year & Month Alarm Register (RTC_ALM_YRMON) . . . . .	37-8
37.4.6	RTC Days Alarm Register (RTC_ALM_DAYS) . . . . .	37-9
37.4.7	RTC Hours and Minutes Alarm Register (RTC_ALM_HM) . . . . .	37-10
37.4.8	RTC Seconds Alarm Register (RTC_ALM_SEC) . . . . .	37-11
37.4.9	RTC Control Register (RTC_CR) . . . . .	37-12
37.4.10	RTC Status Register (RTC_SR) . . . . .	37-14
37.4.11	RTC Interrupt Status Register (RTC_ISR) . . . . .	37-15
37.4.12	RTC Interrupt Enable Register (RTC_IER) . . . . .	37-16
37.4.13	RTC Countdown Timer Register (RTC_COUNT_DN) . . . . .	37-17

37.4.14 RTC Configuration Data Register (RTC_CFG_DATA) . . . . .	37-18
37.4.15 RTC Daylight Saving Time Hour Register (RTC_DST_HOUR) . . . . .	37-19
37.4.16 RTC Daylight Saving Time Month Register (RTC_DST_MON) . . . . .	37-20
37.4.17 RTC Daylight Saving Time Day Register (RTC_DST_DAY) . . . . .	37-20
37.4.18 RTC Compensation Register (RTC_COMPEN) . . . . .	37-21
37.4.19 RTC Up-Counter High Register (RTC_UP_CNTRH) . . . . .	37-21
37.4.20 RTC Up-Counter Low Register (RTC_UP_CNTRL) . . . . .	37-22
37.5 Functional Description . . . . .	37-23
37.5.1 Basic Data Flow . . . . .	37-23
37.6 Initialization/Application Information . . . . .	37-25
37.6.1 Compensation . . . . .	37-25
37.6.2 Write Protection . . . . .	37-27

## Chapter 38 Programmable Interrupt Timers (PIT0–PIT3)

38.1 Introduction . . . . .	38-1
38.1.1 Overview . . . . .	38-1
38.1.2 Block Diagram . . . . .	38-1
38.1.3 Low-Power Mode Operation . . . . .	38-1
38.2 Memory Map/Register Definition . . . . .	38-2
38.2.1 PIT Control and Status Register (PCSR $n$ ) . . . . .	38-3
38.2.2 PIT Modulus Register (PMR $n$ ) . . . . .	38-5
38.2.3 PIT Count Register (PCNTR $n$ ) . . . . .	38-5
38.3 Functional Description . . . . .	38-6
38.3.1 Set-and-Forget Timer Operation . . . . .	38-6
38.3.2 Free-Running Timer Operation . . . . .	38-6
38.3.3 Timeout Specifications . . . . .	38-7
38.3.4 Interrupt Operation . . . . .	38-7

## Chapter 39 DMA Timers (DTIM0–DTIM3)

39.1 Introduction . . . . .	39-1
39.1.1 Overview . . . . .	39-1
39.1.2 Features . . . . .	39-2
39.2 Memory Map/Register Definition . . . . .	39-3
39.2.1 DMA Timer Mode Registers (DTMR $n$ ) . . . . .	39-4
39.2.2 DMA Timer Extended Mode Registers (DTXMR $n$ ) . . . . .	39-5
39.2.3 DMA Timer Event Registers (DTER $n$ ) . . . . .	39-6
39.2.4 DMA Timer Reference Registers (DTRR $n$ ) . . . . .	39-7
39.2.5 DMA Timer Capture Registers (DTCR $n$ ) . . . . .	39-8
39.2.6 DMA Timer Counters (DTCN $n$ ) . . . . .	39-8
39.3 Functional Description . . . . .	39-9

39.3.1	Prescaler . . . . .	39-9
39.3.2	Capture Mode . . . . .	39-9
39.3.3	Reference Compare . . . . .	39-9
39.3.4	Output Mode . . . . .	39-10
39.3.5	Programmable Delay Mode . . . . .	39-10
39.3.6	IEEE 1588 Support . . . . .	39-11
39.4	Initialization/Application Information . . . . .	39-12
39.4.1	Code Example . . . . .	39-12
39.4.2	Calculating Time-Out Values . . . . .	39-13

## Chapter 40

### DMA Serial Peripheral Interface (DSPI)

40.1	Introduction . . . . .	40-1
40.1.1	Block Diagram . . . . .	40-1
40.1.2	Overview . . . . .	40-1
40.1.3	Features . . . . .	40-2
40.1.4	Modes of Operation . . . . .	40-3
40.2	External Signal Description . . . . .	40-4
40.2.1	Signal Overview . . . . .	40-4
40.2.2	Peripheral Chip Select/Slave Select (DSPI <sub>x</sub> _PCS0/SS) . . . . .	40-4
40.2.3	Peripheral Chip Selects 1–3 (DSPI <sub>x</sub> _PCS[1:3]) . . . . .	40-4
40.2.4	Serial Input (DSPI <sub>x</sub> _SIN) . . . . .	40-4
40.2.5	Serial Output (DSPI <sub>x</sub> _SOUT) . . . . .	40-4
40.2.6	Serial Clock (DSPI <sub>x</sub> _SCK) . . . . .	40-5
40.3	Memory Map/Register Definition . . . . .	40-5
40.3.1	DSPI Module Configuration Register (DSPI_MCR) . . . . .	40-6
40.3.2	DSPI Transfer Count Register (DSPI <sub>x</sub> _TCR) . . . . .	40-9
40.3.3	DSPI Clock and Transfer Attributes Registers 0–7 (DSPI <sub>x</sub> _CTAR <sub>n</sub> ) . . . . .	40-9
40.3.4	DSPI Status Register (DSPI <sub>x</sub> _SR) . . . . .	40-14
40.3.5	DSPI DMA/Interrupt Request Select and Enable Register (DSPI <sub>x</sub> _RSER) . . . . .	40-16
40.3.6	DSPI Push Transmit FIFO Register (DSPI <sub>x</sub> _PUSHR) . . . . .	40-17
40.3.7	DSPI Pop Receive FIFO Register (DSPI <sub>x</sub> _POPR) . . . . .	40-19
40.3.8	DSPI Transmit FIFO Registers 0–15 (DSPI <sub>x</sub> _TXFR <sub>n</sub> ) . . . . .	40-19
40.3.9	DSPI Receive FIFO Registers 0–15 (DSPI <sub>x</sub> _RXFR <sub>n</sub> ) . . . . .	40-20
40.4	Functional Description . . . . .	40-21
40.4.1	Start and Stop of DSPI Transfers . . . . .	40-21
40.4.2	Serial Peripheral Interface (SPI) Configuration . . . . .	40-22
40.4.3	DSPI Baud Rate and Clock Delay Generation . . . . .	40-25
40.4.4	Transfer Formats . . . . .	40-27
40.4.5	Continuous Serial Communications Clock . . . . .	40-33
40.4.6	Interrupts/DMA Requests . . . . .	40-34
40.4.7	Power Saving Features . . . . .	40-36

40.5 Initialization/Application Information . . . . .	40-37
40.5.1 How to Change Queues . . . . .	40-37
40.5.2 Switching Master and Slave Mode . . . . .	40-37
40.5.3 Baud Rate Settings . . . . .	40-37
40.5.4 Delay Settings . . . . .	40-38
40.5.5 Calculation of FIFO Pointer Addresses . . . . .	40-39

## Chapter 41 UART Modules

41.1 Introduction . . . . .	41-1
41.1.1 Overview . . . . .	41-1
41.1.2 Features . . . . .	41-2
41.2 External Signal Description . . . . .	41-3
41.3 Memory Map/Register Definition . . . . .	41-3
41.3.1 UART Mode Registers 1 (UMR1 $n$ ) . . . . .	41-5
41.3.2 UART Mode Register 2 (UMR2 $n$ ) . . . . .	41-6
41.3.3 UART Status Registers (USR $n$ ) . . . . .	41-8
41.3.4 UART Clock Select Registers (UCSR $n$ ) . . . . .	41-9
41.3.5 UART Command Registers (UCR $n$ ) . . . . .	41-10
41.3.6 UART Receive Buffers (URB $n$ ) . . . . .	41-12
41.3.7 UART Transmit Buffers (UTB $n$ ) . . . . .	41-13
41.3.8 UART Input Port Change Registers (UIPCR $n$ ) . . . . .	41-14
41.3.9 UART Auxiliary Control Register (UACR $n$ ) . . . . .	41-14
41.3.10 UART Interrupt Status/Mask Registers (UISR $n$ /UIMR $n$ ) . . . . .	41-15
41.3.11 UART Baud Rate Generator Registers (UBG1 $n$ /UBG2 $n$ ) . . . . .	41-16
41.3.12 UART Input Port Register (UIP $n$ ) . . . . .	41-17
41.3.13 UART Output Port Command Registers (UOP1 $n$ /UOP0 $n$ ) . . . . .	41-18
41.4 Functional Description . . . . .	41-18
41.4.1 Transmitter/Receiver Clock Source . . . . .	41-18
41.4.2 Transmitter and Receiver Operating Modes . . . . .	41-20
41.4.3 Looping Modes . . . . .	41-24
41.4.4 Multidrop Mode . . . . .	41-25
41.4.5 Single-Wire Mode with Polarity Control . . . . .	41-27
41.4.6 Bus Operation . . . . .	41-28
41.5 Initialization/Application Information . . . . .	41-28
41.5.1 Interrupt and DMA Request Initialization . . . . .	41-29
41.5.2 UART Module Initialization Sequence . . . . .	41-30

## Chapter 42 I<sup>2</sup>C Interface

42.1 Introduction . . . . .	42-1
42.1.1 Block Diagram . . . . .	42-2

42.1.2	Overview . . . . .	42-2
42.1.3	Features. . . . .	42-3
<b>42.2</b>	<b>Memory Map/Register Definition . . . . .</b>	<b>42-4</b>
42.2.1	I <sup>2</sup> C Address Register (I2ADR <sub>n</sub> ) . . . . .	42-5
42.2.2	I <sup>2</sup> C Frequency Divider Register (I2FDR <sub>n</sub> ) . . . . .	42-5
42.2.3	I <sup>2</sup> C Control Register (I2CR <sub>n</sub> ) . . . . .	42-6
42.2.4	I <sup>2</sup> C Status Register (I2SR <sub>n</sub> ) . . . . .	42-7
42.2.5	I <sup>2</sup> C Data I/O Register (I2DR <sub>n</sub> ) . . . . .	42-8
<b>42.3</b>	<b>Functional Description . . . . .</b>	<b>42-9</b>
42.3.1	START Signal . . . . .	42-9
42.3.2	Slave Address Transmission . . . . .	42-10
42.3.3	Data Transfer. . . . .	42-10
42.3.4	Acknowledge . . . . .	42-11
42.3.5	STOP Signal . . . . .	42-11
42.3.6	Repeated START . . . . .	42-11
42.3.7	Clock Synchronization and Arbitration. . . . .	42-13
42.3.8	Handshaking and Clock Stretching . . . . .	42-14
<b>42.4</b>	<b>Initialization/Application Information . . . . .</b>	<b>42-14</b>
42.4.1	Initialization Sequence. . . . .	42-14
42.4.2	Generation of START . . . . .	42-14
42.4.3	Post-Transfer Software Response. . . . .	42-15
42.4.4	Generation of STOP . . . . .	42-15
42.4.5	Generation of Repeated START . . . . .	42-16
42.4.6	Slave Mode . . . . .	42-16
42.4.7	Arbitration Lost. . . . .	42-16

## Chapter 43 Debug Module

<b>43.1</b>	<b>Introduction. . . . .</b>	<b>43-1</b>
43.1.1	Block Diagram . . . . .	43-1
43.1.2	Overview . . . . .	43-1
<b>43.2</b>	<b>Signal Descriptions. . . . .</b>	<b>43-3</b>
43.2.1	Processor Status/Debug Data (PSTDDATA[7:0]) . . . . .	43-4
<b>43.3</b>	<b>Memory Map/Register Definition . . . . .</b>	<b>43-6</b>
43.3.1	Shared Debug Resources . . . . .	43-8
43.3.2	Configuration/Status Register (CSR) . . . . .	43-8
43.3.3	Extended Configuration/Status Register (XCSR) . . . . .	43-11
43.3.4	Configuration/Status Register 2 (CSR2) . . . . .	43-13
43.3.5	BDM Address Attribute Register (BAAR). . . . .	43-15
43.3.6	Address Attribute Trigger Registers (AATR, AATR1) . . . . .	43-16
43.3.7	Trigger Definition Register (TDR) . . . . .	43-18
43.3.8	Program Counter Breakpoint/Mask Registers (PBR0–3, PBMR) . . . . .	43-21

43.3.9	PC Breakpoint ASID Control Register (PBAC) . . . . .	43-22
43.3.10	Address Breakpoint Registers (ABLR/ABLR1, ABHR/ABHR1) . . . . .	43-23
43.3.11	Data Breakpoint and Mask Registers (DBR/DBR1, DBMR/DBMR1) . . . . .	43-24
43.3.12	PC Breakpoint ASID Register (PBASID) . . . . .	43-25
43.3.13	Extended Trigger Definition Register (XTDR) . . . . .	43-26
43.3.14	PST Trace Buffer Longwords (TBLW0–23) . . . . .	43-30
43.3.15	Most Recently Sampled PC (PCRS) . . . . .	43-31
43.4	Functional Description . . . . .	43-31
43.4.1	Background Debug Mode (BDM) . . . . .	43-31
43.4.2	Real-Time Debug Support . . . . .	43-55
43.4.3	Concurrent BDM and Processor Operation . . . . .	43-57
43.4.4	Real-Time Trace Support . . . . .	43-58
43.4.5	Processor Status, Debug Data Definition . . . . .	43-67
43.4.6	Freescale-Recommended BDM Pinout . . . . .	43-73

## Chapter 44

### IEEE 1149.1 Test Access Port (JTAG)

44.1	Introduction . . . . .	44-1
44.1.1	Block Diagram . . . . .	44-1
44.1.2	Features . . . . .	44-2
44.1.3	Modes of Operation . . . . .	44-2
44.2	External Signal Description . . . . .	44-2
44.2.1	JTAG Enable (JTAG_EN) . . . . .	44-2
44.2.2	Test Clock Input (TCLK) . . . . .	44-3
44.2.3	Test Mode Select/Breakpoint (TMS/ $\overline{BKPT}$ ) . . . . .	44-3
44.2.4	Test Data Input/Development Serial Input (TDI/DSI) . . . . .	44-3
44.2.5	Test Reset/Development Serial Clock (TRST/DSCLK) . . . . .	44-4
44.2.6	Test Data Output/Development Serial Output (TDO/DSO) . . . . .	44-4
44.3	Memory Map/Register Definition . . . . .	44-4
44.3.1	Instruction Shift Register (IR) . . . . .	44-4
44.3.2	IDCODE Register . . . . .	44-5
44.3.3	Bypass Register . . . . .	44-5
44.3.4	TEST_CTRL Register . . . . .	44-5
44.3.5	Boundary Scan Register . . . . .	44-6
44.4	Functional Description . . . . .	44-6
44.4.1	JTAG Module . . . . .	44-6
44.4.2	TAP Controller . . . . .	44-6
44.4.3	JTAG Instructions . . . . .	44-7
44.5	Initialization/Application Information . . . . .	44-10
44.5.1	Restrictions . . . . .	44-10
44.5.2	Nonscan Chain Operation . . . . .	44-10

## About This Book

The primary objective of this reference manual is to define the processor for software and hardware developers. The information in this book is subject to change without notice, as described in the disclaimers on the title page. As with any technical documentation, the reader must use the most recent version of the documentation.

To locate any published errata or updates for this document, refer to the world-wide web at <http://www.freescale.com/coldfire>.

Portions of Chapter 23, “Universal Serial Bus Interface – Host Module,” and Chapter 24, “Universal Serial Bus Interface – On-The-Go Module,” relating to the EHCI specification are Copyright © Intel Corporation 1999-2001. The EHCI specification is provided “As Is” with no warranties whatsoever, including any warranty of merchantability, non-infringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification or sample. Intel disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in the EHCI specification. Intel may make changes to the EHCI specifications at any time, without notice.

## Audience

This manual is intended for system software and hardware developers and applications programmers who want to develop products with this ColdFire processor. It is assumed that the reader understands operating systems, microprocessor system design, basic principles of software and hardware, and basic details of the ColdFire® architecture.

## Suggested Reading

This section lists additional reading that provides background for the information in this manual as well as general information about ColdFire architecture.

## General Information

Useful information about the ColdFire architecture and computer architecture in general:

- *ColdFire Programmers Reference Manual* (MCF5200PRM/AD)
- *Using Microprocessors and Microcomputers: The Motorola Family*, William C. Wray, Ross Bannatyne, Joseph D. Greenfield
- *Computer Architecture: A Quantitative Approach*, Second Edition, by John L. Hennessy and David A. Patterson.
- *Computer Organization and Design: The Hardware/Software Interface*, Second Edition, David A. Patterson and John L. Hennessy.

## ColdFire Documentation

ColdFire documentation is available from the sources listed on the back cover of this manual, as well as our web site, <http://www.freescale.com/coldfire>.

- Reference manuals — These books provide details about individual ColdFire implementations and are intended to be used in conjunction with the *ColdFire Programmers Reference Manual*.
- Data sheets — Data sheets provide specific data regarding pin-out diagrams, bus timing, signal behavior, and AC, DC, and thermal characteristics, as well as other design considerations.
- Product briefs — Each device has a product brief that provides an overview of its features. This document is roughly equivalent to the overview (Chapter 1) of an device's reference manual.
- Application notes — These short documents address specific design issues useful to programmers and engineers working with Freescale Semiconductor processors.

Additional literature is published as new processors become available. For a current list of ColdFire documentation, refer to <http://www.freescale.com/coldfire>.

## Conventions

This document uses the following notational conventions:

cleared/set	When a bit takes the value zero, it is said to be cleared; when it takes a value of one, it is said to be set.
MNEMONICS	In text, instruction mnemonics are shown in uppercase.
mnemonics	In code and tables, instruction mnemonics are shown in lowercase.
<i>italics</i>	Italics indicate variable command parameters.
	Book titles in text are set in italics.
0x0	Prefix to denote hexadecimal number
0b0	Prefix to denote binary number
REG[FIELD]	Abbreviations for registers are shown in uppercase. Specific bits, fields, or ranges appear in brackets. For example, RAMBAR[BA] identifies the base address field in the RAM base address register.
nibble	A 4-bit data unit
byte	An 8-bit data unit
word	A 16-bit data unit <sup>1</sup>
longword	A 32-bit data unit
x	In some contexts, such as signal encodings, x indicates a don't care.
n	Used to express an undefined numerical value
~	NOT logical operator
&	AND logical operator
	OR logical operator

<sup>1</sup>The only exceptions to this appear in the discussion of serial communication modules that support variable-length data transmission units. To simplify the discussion these units are referred to as words regardless of length.

---

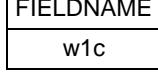
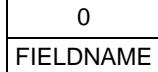
	Field concatenation operator
OVERBAR	An overbar indicates that a signal is active-low.

## Register Figure Conventions

This document uses the following conventions for the register reset values:

- Undefined at reset.
- u Unaffected by reset.
- [*signal\_name*] Reset value is determined by the polarity of the indicated signal.

The following register fields are used:

R 	Indicates a reserved bit field in a memory-mapped register. These bits are always read as zeros.
R 	Indicates a reserved bit field in a memory-mapped register. These bits are always read as ones.
R 	Indicates a read/write bit.
R 	Indicates a read-only bit field in a memory-mapped register.
R 	Indicates a write-only bit field in a memory-mapped register.
R 	Write 1 to clear: indicates that writing a 1 to this bit field clears it.
R 	Indicates a self-clearing bit.



# Chapter 1 Overview

## 1.1 Introduction

The MCF5441x devices are a family of highly-integrated 32-bit microprocessors based on the Version 4m ColdFire microarchitecture, comprising of the V4 integer core, memory management unit (MMU) and enhanced multiply-accumulate unit (EMAC). This product line is well-suited for processing data from and moving data between a variety of common serial interfaces (CAN, I2C, SSI, SPI, UART, and USB) and Ethernet networks, especially for factory automation, process control, and motor control applications. Support for low-cost memory and connectivity options also make this family ideal for a range of consumer digital lifestyle products.

All MCF5441x devices operate at up to 250 MHz and include 64 Kbytes of single-cycle SRAM, memory controllers for DDR2 SDRAM and NAND flash, the highly configurable FlexBus for interfacing components like NOR flash, SRAM, and programmable logic devices (FPGAs and CPLDs), a 64-channel DMA controller, and serial memory boot and configuration support.

Communications peripheral interfaces include: USB host and On-the-Go controllers with integrated full-speed/low-speed transceivers and a switchable port for an external ULPI high-speed PHY, dual smart card ports, an enhanced controller for MMC, MMCplus, SD, and SDHC memory cards, dual CAN modules, a pair of synchronous serial interfaces, a 1-wire interface for low speed communication to devices (thermostats, batteries, etc.) and a maximum of ten UARTs, six I2C controllers, and four DMA serial peripheral interfaces.



Unique to the MCF5441x family is a flexible 10/100 Mbps Ethernet subsystem configurable as: a single media access controller (MAC) with a media independent interface (MII) or reduced MII (RMII), a pair of MACs with dual RMIs, or, on specific devices, as a 3-port switch with two external ports and the third port internally connected to the processor. The Ethernet MACs incorporate hardware CRC checking/generation and Magic Packet power management. The entire Ethernet subsystem supports the IEEE 1588-2002 standard, a precision clock synchronization protocol for networked measurement and control systems. Certain MCF5441x family members also include the cryptographic acceleration unit (CAU), a CPU coprocessor for the DES, 3DES, AES, MD5, SHA-1, and SHA-256 algorithms implemented in network security protocols like SSL and IPsec.



Additional features include four 32-bit timers that can optionally be linked to the Ethernet subsystem's IEEE 1588 timestamp logic for network-triggered event recognition and generation, a flexible multi-channel pulse width modulation timer suitable for motor control which can also be linked to the IEEE 1588 timestamp logic for synchronizing motors through Ethernet, a fast, 12-bit analog-to-digital converter (ADC) with 8-shared input channels capable of simultaneous parallel conversions, and two 12-bit digital-to-analog converters (DACs). The standard 17 mm × 17 mm 256-ball MAPBGA package has a 1 mm ball pitch and can be escape-routed on 4-layer printed circuit boards.

## 1.2 MCF5441x Family Comparison

The following table compares the various device ~~derivatives~~ available within the MCF5441x family.

**Table 1-1. MCF5441x family configurations**

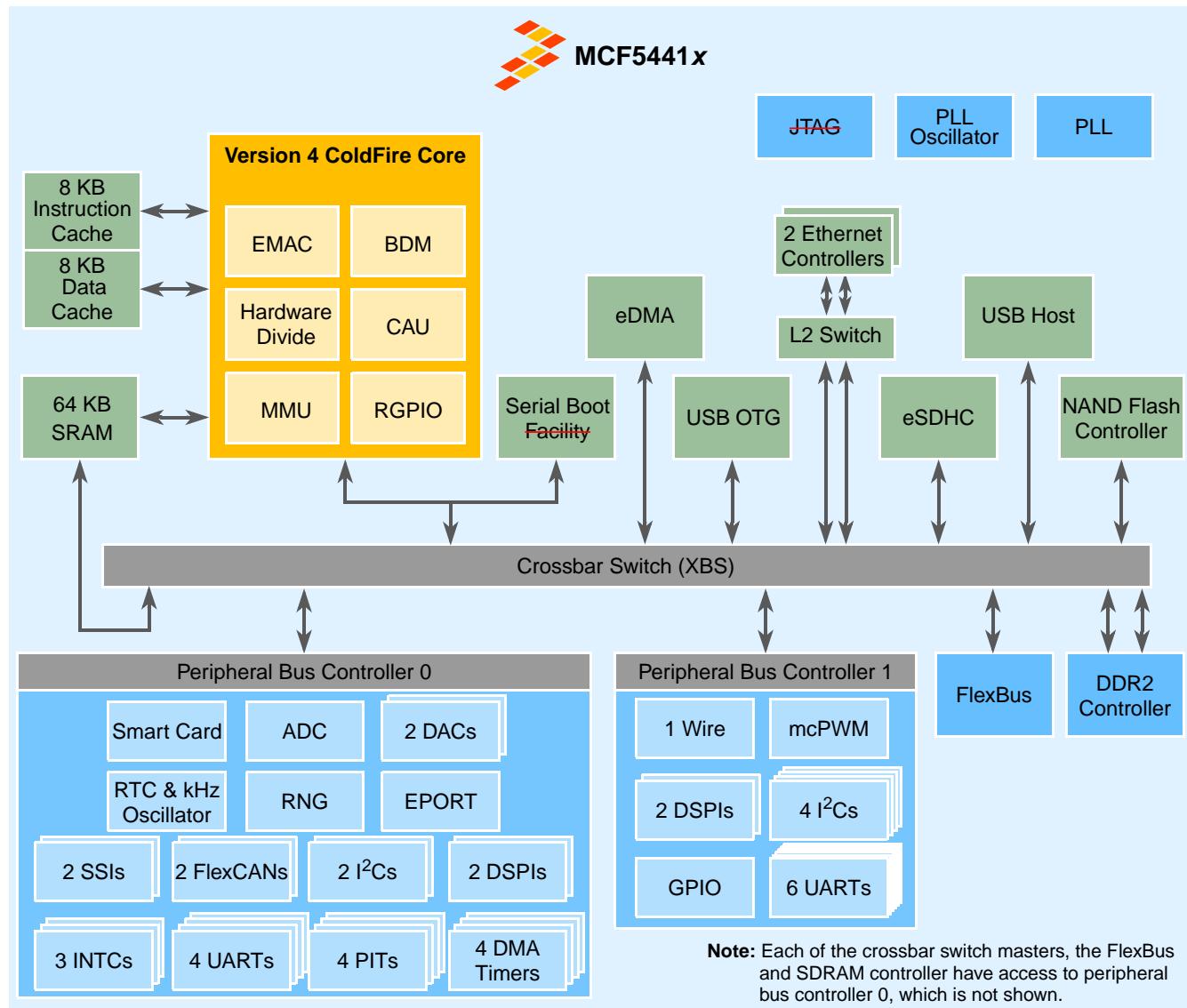
Module	MCF54410	MCF54415	MCF54416	MCF54417	MCF54418
Version 4 ColdFire core with <b>EMAC</b> (enhanced multiply-accumulate unit) and <b>MMU</b> (memory management unit)	•	•	•	•	•
Cryptography acceleration unit ( <b>CAU</b> )	—	—	•	—	•
Core (system) and SDRAM clock			up to 250 MHz		
Peripheral clock (Core clock ÷ 2)			up to 125 MHz		
External bus (FlexBus) clock			up to 100 MHz		
Performance (Dhrystone 2.1 MIPS)			up to 385		
Static RAM (SRAM)			64 Kbytes		
Independent data/instruction cache			8 Kbytes each		
USB 2.0 Host controller	—	•	•	•	•
USB 2.0 Host/Device/On-the-Go controller	•	•	•	•	•
UTMI+ Low Pin Interface (ULPI) for external high-speed USB PHY	—	•	•	•	•
10/100 Mbps Ethernet controller with IEEE 1588 support	1	2	2	2	2
Level 2 IEEE 1588-compliant 3-port Ethernet switch	—	—	—	•	•
Enhanced Secure Digital host controller (eSDHC)	•	•	•	•	•
Smart card/Subscriber Identity Module (SIM)	—	2 ports	2 ports	2 ports	2 ports
<b>UARTs</b>	6	10	10	10	10
<b>DSPI</b>	3	4	4	4	4
<b>CAN 2.0B controllers</b>	1	2	2	2	2
<b>I<sup>2</sup>C</b>	4	6	6	6	6
<b>Synchronous serial interface (SSI)</b>	1	2	2	2	2
<b>12-bit ADC</b>	—	•	•	•	•
<b>12-bit DAC</b>	—	2	2	2	2
<b>32-bit DMA timers</b>	4	4	4	4	4
Periodic interrupt timers (PIT)	4	4	4	4	4
Motor control PWM timer (mcPWM)	—	8 channel	8 channel	8 channel	8 channel
64-channel DMA controller	•	•	•	•	•

**Table 1-1. MCF5441x family configurations (continued)**

<b>Module</b>	<b>MCF54410</b>	<b>MCF54415</b>	<b>MCF54416</b>	<b>MCF54417</b>	<b>MCF54418</b>
Real-time clock with 2 KB standby RAM and battery back-up input	•	•	•	•	•
DDR2 SDRAM controller	•	•	•	•	•
FlexBus external memory controller	•	•	•	•	•
<b>NAND flash controller</b>	•	•	•	•	•
1-Wire® interface	•	•	•	•	•
Serial boot facility	•	•	•	•	•
<b>Watchdog timer</b>	•	•	•	•	•
Interrupt controllers (INTC)	3	3	3	3	3
Edge port module (EPORT)	3 IRQs	5 IRQs	5 IRQs	5 IRQs	5 IRQs
<b>Rapid GPIO pins</b>	9	16	16	16	<b>16</b>
<b>General-purpose I/O (GPIO) pins</b>	48	87	87	87	<b>87</b>
JTAG - IEEE® 1149.1 Test Access Port	•	•	•	•	•
Package	196 MAPBGA	256 MAPBGA			

## 1.3 Block Diagram

Figure 1 shows a top-level block diagram of the MCF5441x superset device.



<b>ADC</b>	– Analog-to-digital converter
<b>BDM</b>	– Background debug module
<b>CAU</b>	– Cryptography acceleration unit
<b>DAC</b>	– Digital-to-analog
<b>DSPI</b>	– DMA serial peripheral interface
<b>eDMA</b>	– Enhanced direct memory access module
<b>eSDHC</b>	– Enhanced Secure Digital host controller
<b>EMAC</b>	– Enhance multiply-accumulate unit
<b>EPORT</b>	– Edge port module
<b>GPIO</b>	– General purpose input/output module
<b>I<sup>2</sup>C</b>	– Inter-Integrated Circuit

<b>INTC</b>	– Interrupt controller
<b>JTAG</b>	– Joint Test Action Group interface
<b>mcPWM</b>	– Motor control pulse width modulator
<b>PIT</b>	– Programmable interrupt timers
<b>PLL</b>	– Phase locked loop module
<b>RGPIO</b>	– Rapid GPIO
<b>RNG</b>	– Random number generator
<b>RTC</b>	– Real time clock
<b>SSI</b>	– Synchronous serial interface
<b>USB OTG</b>	– Universal Serial Bus On-the-Go controller

Figure 1. MCF5441x Block Diagram

## 1.4 Operating Parameters

- ~~-40°C to 85°C ambient temperature~~
- ~~1.2V Core, 3.3V I/O, 1.8–3.3V external memory bus~~

## 1.5 ~~Packages~~

Depending on device, the MCF5441x family is available in the following packages:

- ~~196-pin~~ molded array process ball grid array (MAPBGA)
- ~~256-pin~~ molded array process ball grid array (MAPBGA)

## 1.6 Chip Level Features

- Version 4 ColdFire® Core with EMAC and MMU
- Up to 385 Dhystone 2.1 MIPS @ 250 MHz
- ~~8 Kbytes instruction cache and 8 Kbytes data cache~~
- ~~64 Kbytes internal SRAM~~ dual-ported to processor local bus and other crossbar switch masters
- System boot from NOR, NAND, SPI flash, EEPROM, or FRAM
- Crossbar switch technology (XBS) for concurrent access to peripherals or RAM from multiple bus masters
- 64-channel DMA controller
- ~~SDRAM~~ controller supporting full-speed operation from a single x8 DDR2 component up to ~~250 MHz~~
- 32-bit FlexBus external memory interface for RAM, ROM, MRAM, and programmable logic
- USB 2.0 host controller
- USB 2.0 host/device/On-the-Go controller
- 8-bit single data rate ULPI port usable by the dedicated USB host module or the USB host/device/OTG module
- Dual 10/100 Ethernet MACs with hardware CRC checking/genration, IEEE 1588-2002 support support, and optional Ethernet switch
- CPU direct-attached hardware accelerator for DES, 3DES, AES, MD5, SHA-1, and SHA-256 algorithms
- ~~Random number generator~~
- Enhanced Secure Digital host controller for SD, SDHC, SDIO, MMC, and MMCplus cards
- Two ISO7816 smart card interfaces
- Two FlexCAN modules
- ~~Six I<sup>2</sup>C bus interfaces~~ with DMA support in master mode
- ~~Two synchronous serial interfaces~~
- ~~Four 32-bit timers with DMA support~~
- Four programmable interrupt timers
- 8-channel, 16-bit motor control PWM timer

- Dual 12-bit ADCs with shared input channels and multiple conversion trigger sources
- Dual 12-bit DACs with DMA support
- 1-wire module with DMA support
- NAND flash controller
- Real-time clock with 32-kHz oscillator, 2 KB standby SRAM, and battery backup supply input
- Up to four DMA-supported serial peripheral interfaces (DSPI)
- Up to ten UARTs with single-wire mode support
- Up to five external IRQ interrupts and 2 external DMA request/acknowledge pairs
- Up to 16 processor local bus Rapid GPIO pins
- Up to 87 standard GPIO pins

## 1.7 Module-by-Module Feature List

The following is a brief summary of the functional blocks in the MCF54418 superset device. For more details refer to the *MCF54418 ColdFire Microprocessor Reference Manual* (MCF54418RM).

### 1.7.1 Version 4 ColdFire Variable-Length RISC Processor

- Static CMOS operation
- 32-bit address and data path on-chip
- Maximum 250 MHz processor core and 125 MHz bus frequency
- Sixteen total general-purpose 32-bit address and data registers
- Enhanced multiply-accumulate unit (EMAC) for DSP and fast multiply operations
- Hardware divide execution unit supporting various 32-bit operations
- Virtual memory management unit (MMU) providing virtual-to-physical address translation and memory access control
- Implements the ColdFire Instruction Set Architecture, ISA\_C

### 1.7.2 On-chip Memories

- 64 Kbyte dual-ported SRAM on CPU internal bus
  - Accessible to non-core bus masters (e.g. DMA, USB modules, and Ethernet subsystem) via the crossbar switch
- Non-blocking, independent 8-KB data and instruction caches organized as 4-way set associative with 16 bytes per cache line and 512 cache lines, supporting copy-back and write-through modes

### 1.7.3 Phase Locked Loop (PLL) and Crystal Oscillator

- 14–50 MHz input clock
- Programmable frequency multiplication factor settings generating voltage-controlled oscillator (VCO) frequencies from 240–500 MHz, resulting in a core frequency of 7.5 MHz ( $f_{vco} \div 32$ ) to 250 MHz (maximum rated frequency).

- Loss-of-lock and loss-of-clock detection and reset
- Direct clocking of system by input clock, bypassing the PLL

#### 1.7.4 Power Management

- Fully static operation with processor sleep and whole chip stop modes
- ~~Limp mode~~ for very low frequency operation while major peripherals are disabled
- Rapid response to interrupts from the low-power sleep mode (wake-up feature)
- Peripheral power management register to enable/disable clocks to most modules
- Software controlled disable of external clock input for low power consumption
- Battery backup supply for RTC and 2 KB standby SRAM for when main processor supply is removed

#### 1.7.5 Chip Configuration Module (CCM)

- System configuration during reset
- Bus monitor, abort monitor
- Part identification number and part revision number
- Serial boot and configuration capability
  - Supports SPI-~~compatible~~ EEPROM, flash, and FRAM
  - Pre-boot control of multiple chip configuration options

#### 1.7.6 Reset Controller

- Separate reset in and reset out signals
- ~~Seven sources~~ of reset: power-on reset (POR), external, ~~software~~, watchdog timer, loss of lock, loss of clock, JTAG instruction
- ~~Status flag~~ indication of source of last reset

#### 1.7.7 System Control Module

- Access control registers
- Watchdog timer with  $2^n$  (where  $n = 8\text{--}31$ ) clock cycle selectable timeout period
- ~~On-chip watchdog timer sourced from RTC or system clock~~
- Core fault reporting

#### 1.7.8 Crossbar Switch

- ~~Concurrent~~ access from different masters to different slaves
- Slave ~~arbitration~~ attributes configured on a per basis
- Fixed or ~~round-robin~~ arbitration

### 1.7.9 Universal Serial Bus (USB) Host Controller

- Fully compliant with the *Universal Serial Bus Specification, Revision 2.0*
- Support for full speed (FS = 12 Mbps) and low speed (LS = 1.5 Mbps) with on-chip FS/LS transceiver
- Compatible with the *Enhanced Host Controller Interface (EHCI) Specification for Universal Serial Bus, Revision 1.0*
- Connects to external 5V power control chip for downstream power
- Optional UTMI+ Low Pin Interface (ULPI) to support high speed (HS = 480 Mbps) transfers with external PHY, shared with USB OTG module
- Uses 60 MHz reference clock based off of the system clock or from an external pin

### 1.7.10 Universal Serial Bus (USB) 2.0 On-The-Go (OTG) Controller

- Fully compliant with the *Universal Serial Bus Specification, Revision 2.0* and *On-The-Go Supplement to the USB 2.0 Specification, Revision 1.0a*
- Support for full speed (FS) and low speed (LS) via on-chip FS/LS transceiver in host mode
- Compatible with the *Enhanced Host Controller Interface (EHCI) Specification for Universal Serial Bus, Revision 1.0* (host mode)
- Connects to external 5V power control chip for downstream power (host mode)
- Support for full speed with on-chip transceiver in device mode
- Optional UTMI+ Low Pin Interface (ULPI) to support high speed (HS = 480 Mbps) transfers with external PHY, shared with USB host module
- Connects to external OTG charge pump and resistor chip via I<sup>2</sup>C bus
- Uses 60 MHz reference clock based off of the system clock or from an external pin

### 1.7.11 DDR SDRAM Controller

- Supports glueless interface to DDR2 SDRAM devices
- 8-bit wide memory port runs at the CPU clock frequency to deliver up to 500 MBps of memory bandwidth at 250 MHz
- Unique interface design reduces memory subsystem cost to the cost of a single x8 commodity DDR2 component while delivering performance equivalent to a 32-bit wide single data rate SDRAM subsystem running at up to 125 MHz and requiring two x16 or four x8 components
- Innovative controller uses two crossbar switch slave ports, multiple read/write queues, and intelligent transfer ordering to maximize memory channel utilization and reduce multi-master access latency
- 16 bytes critical word first burst transfer
- Up to 15 lines of row address, up to 12 column address lines, 8 banks (3-bits), and one chip select
- Supports 16, 32, 64, 128, or 256 MB of memory using a single 16M × 8 (128 Mb), 32M × 8 (256 Mb), 64M × 8 (512 Mb), 128M × 8 (1 Gb), or 256M × 8 (2 Gb) component
- Supports page mode to maximize the data rate

- Supports sleep mode and self-refresh mode

### 1.7.12 FlexBus (External Interface)

- 32-bit external ~~bidirectional~~ multiplexed address/data bus
- ~~Glueless~~ connections to 8-, 16-, and 32-bit external memory devices (SRAM, flash, ROM, etc.)
- Support for independent primary and secondary wait states per chip select
- Programmable address setup and hold time with respect to chip-select ~~assertion~~, per transfer direction
- Glueless interface to SRAM devices with or without byte ~~strobe~~ inputs
- Supports asynchronous and synchronous memories with a dedicated address latch signal available
- Programmable wait state generator
- Up to six chip selects available
- Byte/write enables (byte strobes)
- Ability to boot from external memories that are 8, 16, or 32 bits wide

### 1.7.13 Ethernet Assembly

- On-chip layer 2 (L2) Ethernet switch
  - 3-port switch (one port internal to the switch core)
  - IEEE 1588 support
  - Fast cut-through mode
  - QoS with 8 queues per port
  - Port mirroring
  - Level 3 IP snooping
  - Link aggregation
- Two Ethernet controllers with 10/100 BaseT/TX capability; half duplex or full duplex
  - Hardware support for *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, IEEE 1588
  - Media independent interface (MII) and reduced media independent interface (RMII) support
- Built-in unified DMA
  - On-chip transmit and receive FIFOs
  - Supports legacy buffer descriptor programming models and functionality
  - Enhanced buffer descriptor programming model for new Ethernet functionality
- Supports wake-up from low power mode through magic packets
- During chip reset, ability to route traffic from one port to another
- Multiple clock source options for time-stamping clock

### 1.7.14 Cryptography Acceleration Unit (CAU)

- Instruction-level coprocessor
- Supports DES, 3DES, AES, MD5, SHA-1, and SHA-256

### 1.7.15 Random Number Generator

- National Institute of Standards and Technology (NIST)-approved pseudo-random number generator
- Supports the key generation algorithm defined in the Digital Signature Standard
- Integrated entropy sources capable of providing the PRNG with entropy for its seed.

### 1.7.16 Secure Digital Host Controller (SDHC)

- Compatible with the following specifications:
  - SD Host Controller Standard Specification, Version 2.0 ([www.sdcards.org](http://www.sdcards.org))
  - MultiMediaCard System Specification, Version 4.2 ([www.mmca.org](http://www.mmca.org))
  - SD Memory Card Specification, Version 2.0 ([www.sdcards.org](http://www.sdcards.org))
  - SDIO Card Specification, Version 2.0 ([www.sdcards.org](http://www.sdcards.org))
  - CE-ATA Card Specification, Version 1.0 ([www.sdcards.org](http://www.sdcards.org))
- Designed to work with CE-ATA, SD Memory, miniSD Memory, SDIO, miniSDIO, SD Combo, MMC, MMC plus, MMC 4x, and MMC RS cards
- SD bus clock up to 52 MHz
- Supports 1- and 4-bit SD and SDIO modes, 1-, 4-, and 8-bit MMC modes, and 1-, 4-, and 8-bit CE-ATA devices
- Contains a fully configurable 128 x 32-bit FIFO for read/write data

### 1.7.17 Subscriber Identity Module (SIM)

- Two ISO7816 smart card interfaces
- Internal one-wire interface
- Programmable clock divisor
- Fourteen total interrupt sources (six transmit, six receive, two control functions)

### 1.7.18 Synchronous Serial Interfaces (SSI)

- Two SSIs each support shared (synchronous) transmit and receive sections
- Normal mode operation using frame sync
- Network mode operation allowing multiple devices to share the port with as many as 32 time slots
- Gated clock mode operation requiring no frame sync
- Programmable data interface modes such as I<sup>2</sup>S, LSB aligned, and MSB aligned
- Programmable word length up to 24 bits

- AC97 support

### 1.7.19 FlexCAN Modules

- Two FlexCAN modules support the full implementation of the *CAN Specification Version 2.0, Part B*
  - Standard data and remote frames (up to 109 bits long)
  - Extended data and remote frames (up to 127 bits long)
  - 0–8 bytes data length
  - Programmable bit rate up to 1 Mbit/sec
- Flexible message buffers (MBs), totalling up to 16 message buffers of 0–8 bytes data length each, configurable as Rx or Tx, all supporting standard and extended messages
- Unused MB space can be used as general purpose RAM space
- Listen-only mode capability
- Content-related addressing
- Three programmable mask registers: global (MBs 0–13), special for MB14 and special for MB15
- Programmable transmit-first scheme: lowest ID or lowest buffer number
- Time stamp based on 16-bit free-running timer
- Global network time, synchronized by a specific message

### 1.7.20 Analog-Digital Converters (ADC)

- Two 4-channel, 12-bit ADCs with ~~sequential~~ and parallel sampling
- Software, external, and on-chip (such as mcPWM and DMA timers) triggered conversions
- DMA interface to the on-chip 64 channel DMA controller
- Supports multiple low power modes, including auto-power down

### 1.7.21 Digital-Analog Converters (DAC)

- Dual 12-bit DACs
- DMA interface to the on-chip 64 channel DMA controller
- High-/low-speed programmable operation
- Software, external, and on-chip (such as mcPWM and DMA timers) triggered conversions
- Automatic waveform generation, including square, triangle, and sawtooth
- DAC output connected to the on-chip ADC for digital to digital comparision for debug

### 1.7.22 NAND Flash Controller

- 8- or 16-bit NAND flash interface
- Internal 9-KByte RAM buffer configurable as boot RAM
- Supports all NAND flash products regardless of density/organization

- Automatic write protection during power-up
- Handshaking to indicate ready/busy status
- Supports time division multiplexing external memory interface (FlexBus) cycles with long access times of NAND flash devices
- Integrated DMA engine to support self-initiated data transfer from NAND flash to system memory or any slave devices on crossbar switch

### 1.7.23 1-Wire Interface

- Communicates with an external 1-Wire device as defined by Maxim
- Clock divider to generate the bus reference clock from the peripheral bus clock
- DMA interface to the on-chip 64 channel DMA controller

### 1.7.24 Robust Real Time Clock

- Full clock: hours, minutes, and seconds with storing option
- Calendar: day, month, year, and day of the week with storing option
- Automatic adjustment for daylight savings with user-defined parameters
- Automatic month and leap year adjustment
- Programmable alarm with interrupt
- Eight periodic interrupts
- Minute stopwatch
- Once-per-day, once-per-hour, once-per-minute, and once-per-second interrupts
- Operation determined by reference input oscillator clock frequency and value programmed into user-accessible registers
- Ability to wake the processor from low-power modes (wait, doze, and stop) via the RTC interrupts
  - The RTC is enabled during stop mode
- Battery operation (standby mode) ensures seamless operation when processor power is removed
- Protection against tampering and spurious memory/register updates
- Supports hardware compensation to improve accuracy against crystal frequency variations
- Dedicated 2 KB RAM for storing the system contents during standby operation when processor power is removed
- Built-in write protection prevents tampering

### 1.7.25 Programmable Interrupt Timers (PIT)

- Four programmable interrupt timers each with a 16-bit counter
- Configurable as a down counter or free-running counter

## 1.7.26 DMA Timers

- Four 32-bit timers with DMA and interrupt request trigger capability
- Input capture and reference compare modes
- Support for the Ethernet assembly's IEEE 1588 timebase and count values
- Programmable delay support

## 1.7.27 DMA Serial Peripheral Interfaces (DSPI)

- Four DSPIs with up to four chip selects available per DSPI module
- Full-duplex, three-wire synchronous transfer
- Master and slave modes with programmable master bit-rates
- Up to 16 pre-programmed transfers

## 1.7.28 Motor Control Pulse Width Modulation (mcPWM) Module

- 16-bit resolution for center-aligned, edge-aligned, and asymmetrical PWMs
- Four PWM output pairs (8 total outputs) that can operate as complementary pairs or independent channels
- Synchronization to external hardware, other PWM channels, or the IEEE 1588 timestamp via the on-chip DMA timer
- Can accept signed numbers for PWM generation
- Independent control of both edges of each PWM output
- Multiple output trigger events can be generated per PWM cycle via hardware
- Support for double-switching PWM outputs
- Fault inputs can be assigned to control multiple PWM outputs
- Programmable filters for fault inputs
- Independently programmable PWM output polarity
- Independent top and bottom deadtime insertion
- Each complementary pair can operate with its own PWM frequency and deadtime values
- Individual software-control for each PWM output
- All outputs can be programmed to change simultaneously via a force-out event
- Channels not used for PWM generation can be used for buffered output compare functions
- Channels not used for PWM generation can be used for input capture functions
- Enhanced dual-edge capture functionality
- Enhanced triggering capability
  - Multiple ADC trigger events can be generated per PWM cycle

## 1.7.29 Universal Asynchronous Receiver Transmitters (UARTs)

- Ten UARTs each with a 16-bit divider for clock generation

- DMA support with separate transmit and receive requests
- Support for single wire and wired-OR mode
- Programmable clock-rate generator
- Data formats can be 5, 6, 7 or 8 bits with even, odd or no parity
- Up to 2 stop bits in 1/16 increments
- Error-detection capabilities

### **1.7.30 I<sup>2</sup>C Modules**

- Six interchip bus interfaces for EEPROMs, LCD controllers, A/D converters, and keypads
- DMA support in master mode
- Fully compatible with industry-standard I<sup>2</sup>C bus
- Master or slave modes support multiple masters
- Automatic interrupt generation with programmable level

### **1.7.31 Interrupt Controllers**

- Three interrupt controllers, supporting up to 64 interrupt sources each, organized as seven programmable levels
- Unique vector number for each interrupt source
- Ability to mask any individual interrupt source plus a global mask-all capability
- Support for service routine software interrupt acknowledge (IACK) cycles
- Combinational path to provide wake-up from low power modes

### **1.7.32 Edge Port Module**

- Each pin can be individually configured as low level sensitistive interrupt pin or edge-detecting interrupt pin (rising, falling, or both)
- Exit stop mode via level-detect function

### **1.7.33 DMA Controller**

- 64 fully programmable channels with 32-byte transfer control descriptors
- Data movement via dual-address transfers for 8-, 16-, 32- and 128-bit data values
- Programmable source, destination addresses, transfer size, support for enhanced address modes
- Support for major and minor nested counters with one request and one interrupt per channel
- Support for channel-to-channel linking and scatter/gather for continuous transfers with fixed priority and round-robin channel arbitration
- External request/acknowledge pins

### 1.7.34 Rapid GPIO Interface

- 16 bits of high-speed GPIO functionality connected to the processor's local bus
- Pin toggle rates typically 1.5–3.5x faster than traditional GPIO pin connected to the peripheral bus
- Unused Rapid GPIO pins can be used as traditional GPIO

### 1.7.35 General Purpose I/O Interface

- Up to 48 bits of GPIO for the 196 MAPBGA device
- Up to 71 bits of GPIO for the 256 MAPBGA device
- Bit manipulation supported via set/clear functions
- Various unused peripheral pins may be used as GPIO
- Drive strength and slew rate control

### 1.7.36 System Debug Support

- Background debug mode (BDM) Revision D+
- Real time debug support, with four PC breakpoint registers and a pair of address breakpoint registers with optional data
- Compressed processor status and debug data captured into on-chip trace buffer

### 1.7.37 JTAG Support

- JTAG part identification and part revision numbers

## 1.8 Memory Map Overview

Table 1-2 illustrates the overall memory map of the device, which depends on the selected boot mode.

**Table 1-2. System Memory Map per Boot Mode**

Address Range <sup>1</sup>	Boot Source		
	FlexBus	Flash Controller	Serial Boot
0x0000_0000 0x0000_FFFF	FlexBus	NAND flash controller <sup>2</sup>	Internal SRAM <sup>2</sup>
0x0001_0000 0x00FF_FFFF			
0x1000_0000		FlexBus	FlexBus
0x3FFF_FFFF			
0x4000_0000  0x7FFF_FFFF	SDRAM controller		
0x8000_0000 0x8BFF_FFFF <sup>3</sup>	Internal SRAM backdoor <sup>3</sup>		
0x8C00_0000 0x8FFF_FFFF	Rapid GPIO		
0x9000_0000  0xBFFF_FFFF	Reserved		
0xC000_0000 0xDFFF_FFFF	FlexBus		
0xE000_0000 0xFFFF_FFFF	Peripheral bus controller 1 <sup>4</sup>		
0xF000_0000 0xFFFF_FFFF	Peripheral bus controller 0 <sup>4</sup>		

<sup>1</sup> See the various peripheral chapters for their memory maps. Any unused space by these peripherals within this memory range is reserved and must not be accessed.

<sup>2</sup> After NFC or serial boot completes, this space is available to the FlexBus.

<sup>3</sup> The SRAM backdoor is used by the non-core crossbar switch masters (DMA, USB, etc.) to access the internal SRAM. The 64 KByte SRAM wraps around in this 192 MByte address space.

<sup>4</sup> See [Section 1.8.1, “Internal Peripheral Space”](#) for a list of which modules are connected to each peripheral bus controller.

## NOTE

This memory map provides two disjoint regions mapped to the FlexBus controller to support glueless connections to external memories (e.g., flash and SRAM), and a second space with one (or more) unique chip-selects that can be used for non-cacheable, non-memory devices (addresses 0xC000\_0000–0xDFFF\_FFFF). Additionally, this mapping is selected since it easily maps into the ColdFire access control registers, which provide a coarse association between memory addresses and their attributes (e.g., cacheable, non-cacheable). For this device, one possible configuration defines the default memory attribute as non-cacheable, and one ACR is then used to identify cacheable addresses, e.g., ADDR[31] equals 0 identifies the cacheable space.

### 1.8.1 Internal Peripheral Space

The internal peripheral space contains locations for all internal registers used to program and control the device's functional blocks and external interfaces. [Table 1-3](#) and [Table 1-4](#) summarize the various register spaces and their base addresses for each of the peripheral bus controllers. Each slot is 16 kB in size, which is not necessarily taken up entirely by the functional blocks. Any slot not illustrated is reserved. See the corresponding chapter for details on their individual memory maps.

**Table 1-3. Peripheral Bus Controller 0 Memory Map**

Base Address	Slot Number	Peripheral
0xFC00_4000	1	Crossbar switch
0xFC00_8000	2	FlexBus
0xFC02_0000	8	FlexCAN 0
0xFC02_4000	9	FlexCAN 1
0xFC03_8000	14	I <sup>2</sup> C 1
0xFC03_C000	15	DSPI 1
0xFC04_0000	16	SCM
0xFC04_4000	17	eDMA controller
0xFC04_8000	18	Interrupt controller 0
0xFC04_C000	19	Interrupt controller 1
0xFC05_0000	20	Interrupt controller 2
0xFC05_4000	21	Interrupt controller IACK
0xFC05_8000	22	I <sup>2</sup> C 0
0xFC05_C000	23	DSPI 0
0xFC06_0000	24	UART0
0xFC06_4000	25	UART1
0xFC06_8000	26	UART2

**Table 1-3. Peripheral Bus Controller 0 Memory Map (continued)**

<b>Base Address</b>	<b>Slot Number</b>	<b>Peripheral</b>
0xFC06_C000	27	UART3
0xFC07_0000	28	DMA timer 0
0xFC07_4000	29	DMA timer 1
0xFC07_8000	30	DMA timer 2
0xFC07_C000	31	DMA timer 3
0xFC08_0000	32	PIT 0
0xFC08_4000	33	PIT 1
0xFC08_8000	34	PIT 2
0xFC08_C000	35	PIT 3
0xFC09_0000	36	Edge port 0
0xFC09_4000	37	ADC
0xFC09_8000	38	DAC 0
0xFC09_C000	39	DAC 1
0xFC0A_8000	42	Robust real-time clock
0xFC0A_C000	43	SIM
0xFC0B_0000	44	USB On-the-Go
0xFC0B_4000	45	USB host
0xFC0B_8000	46	DDR controller
0xFC0B_C000	47	SSI 0
0xFC0C_0000	48	PLL
0xFC0C_4000	49	Random number generator (RNG)
0xFC0C_8000	50	SSI 1
0xFC0C_C000	51	eSDHC
0xFC0D_4000	53	MAC-NET0
0xFC0D_8000	54	MAC-NET1
0xFC0D_C000	55	L2 Ethernet switch 0
0xFC0E_0000	56	L2 Ethernet switch 1
0xFC0F_C000	63	NAND flash controller

**Table 1-4. Peripheral Bus Controller 1 Memory Map**

<b>Base Address</b>	<b>Slot Number</b>	<b>Peripheral</b>
0xEC00_8000	2	1-Wire
0xEC01_0000	4	I <sup>2</sup> C 2

**Table 1-4. Peripheral Bus Controller 1 Memory Map (continued)**

Base Address	Slot Number	Peripheral
0xEC01_4000	5	I <sup>2</sup> C 3
0xEC01_8000	6	I <sup>2</sup> C 4
0xEC01_C000	7	I <sup>2</sup> C 5
0xEC03_8000	14	DSPI 2
0xEC03_C000	15	DSPI 3
0xEC06_0000	24	UART4
0xEC06_4000	25	UART5
0xEC06_8000	26	UART6
0xEC06_C000	27	UART7
0xEC07_0000	28	UART8
0xEC07_4000	29	UART9
0xEC08_8000	34	mcPWM
0xEC09_0000	36	CCM, reset controller, power management
0xEC09_4000	37	Pin multiplexing and control (GPIO)

## 1.9 Documentation

Documentation is available from a local Freescale distributor, a Freescale sales office, the Freescale Literature Distribution Center, or through the Freescale world-wide web address at  
<http://www.freescale.com/coldfire>.



# Chapter 2

## Signal Descriptions

### 2.1 Introduction

This chapter describes the external signals on the device. It includes an alphabetical signal listing of signals that characterizes each signal as an input or output, defines its state at reset, and identifies whether a pull-up resistor should be used.

#### NOTE

The terms assertion and negation are used to avoid confusion when dealing with a mixture of active-low and active-high signals. The term asserted indicates that a signal is active, independent of the voltage level. The term negated indicates that a signal is inactive.

Active-low signals, such as  $\overline{\text{SD\_RAS}}$  and  $\overline{\text{FB\_TA}}$ , are indicated with an overbar.

### 2.2 Signal Properties Summary

The below table lists the signals grouped by functionality.

#### NOTE

In this table and throughout this document a single signal within a group is designated without square brackets (i.e., FB\_AD23), while designations for multiple signals within a group use brackets (i.e., FB\_AD[23:21]) and is meant to include all signals within the two bracketed numbers when these numbers are separated by a colon.

#### NOTE

The primary functionality of a pin is not necessarily its default functionality. Most pins that are muxed with GPIO default to their GPIO functionality. See [Table 2-1](#) for a list of the exceptions.

**Table 2-1. Special-case default signal functionality**

Pin	Default signal
FB_CLK, $\overline{\text{FB\_OE}}$ , FB_R/ $\overline{W}$ , $\overline{\text{FB\_BE/BWE}}[1:0]$ , $\overline{\text{FB\_CS}}[5:4]$	FB_CLK, $\overline{\text{FB\_OE}}$ , FB_R/ $\overline{W}$ , $\overline{\text{FB\_BE/BWE}}[1:0]$ , $\overline{\text{FB\_CS}}[5:4]$
FB_ALE	FB_ALE or $\overline{\text{FB\_TS}}$ (depending on RCON[3])

**Table 2-1. Special-case default signal functionality (continued)**

Pin	Default signal
$\overline{\text{FB\_BE/BWE3}}$	Boot from NFC, NF_ALE. Otherwise, $\overline{\text{FB\_BE/BWE3}}$ .
$\overline{\text{FB\_BE/BWE2}}$	Boot from NFC, NF_CLE. Otherwise, $\overline{\text{FB\_BE/BWE2}}$ .
FB_CS1	Boot from NFC, NFC_CE. Otherwise, GPIO.
FB_CS0	Boot from FlexBus, FB_CS0. Otherwise, GPIO.
$\overline{\text{FB\_TA}}$	Boot from NFC, NFC_R/ $\overline{\text{B}}$ . Otherwise, $\overline{\text{FB\_TA}}$ .
ALLPST, PST[3:0], DDATA[3:0]	ALLPST, PST[3:0], DDATA[3:0]

**Table 2-2. MCF5441x Signal information and muxing**

Signal name	GPIO	Alternate 1	Alternate 2	Pullup (U) <sup>1</sup> Pulldown (D)	Direction <sup>2</sup>	Voltage domain	Pad type <sup>3</sup>	196 MABGA <sup>4</sup>	256 MABGA
<b>Reset</b>									
RESET	—	—	—	U	I	EVDD	ssr	K14	K15
$\overline{\text{RSTOUT}}$	—	—	—	—	O	EVDD	msr	L13	L16
<b>Clock</b>									
EXTAL/ RMII_REF_CLK	—	—	—	—	$\overline{\text{I}}^5$	EVDD	ae	G14	G16
XTAL	—	—	—	—	O	EVDD	ae	H14	H16
<b>Mode selection</b>									
BOOTMOD[1:0]	—	—	—	—	I	EVDD	msr	G5,H5	K5, L5
<b>FlexBus</b>									
FB_AD[31:24]/ NFC_IO[15:8] <sup>6</sup>	—	—	—	—	I/O	FBVDD	fsr	A10, A9, B9, C9, A8, B8, C8, A7	B9, C8, A9, B8, D8, A8, D7, B7
FB_AD[23:16]/ NFC_IO[7:0] <sup>6</sup>	—	—	—	—	I/O	FBVDD	fsr	B7, C7, C6, B6, A6, A5, B5, A4	C7, A7, D6, A6, B6, D5, C6, A5
FB_AD[15:10]	—	—	—	$\overline{\text{U}}^7$	I/O	FBVDD	fsr	C5, A3, B4, C4, B3, A2	B5, A4, A3, D4, B4, C5
FB_AD[9:8]	—	—	—	U <sup>8</sup>	I/O	FBVDD	fsr	B2, C3	C4, B3

**Table 2-2. MCF5441x Signal information and muxing (continued)**

Signal name	GPIO	Alternate 1	Alternate 2	Pullup (U) <sup>1</sup> Pulldown (D)	Direction <sup>2</sup>	Voltage domain	Pad type <sup>3</sup>	196 MAPBGA <sup>4</sup>	256 MAPBGA
FB_AD[7:0]	—	—	—	—	I/O	FBVDD	fsr	D4, B1, C2, D3, C1, D2, E3, D1	C3, E4, D3, E3, A2, B2, C2, F3
FB_ALE	PA7	FB_TS	—	—	O	FBVDD	fsr	E2	D2
FB_OE/ NFC_RE	PA6	FB_TBST/ NFC_RE	—	—	O	FBVDD	fsr	H1	F1
FB_R/W/ NFC_WE	PA5	—	—	—	O	FBVDD	fsr	H2	G2
FB_TA	PA4	—	NFC_R/B	U <sup>9</sup>	O	FBVDD	fsr	H3	H3
FB_BE/BWE3	PA3	FB_CS3	FB_A1/ NFC_ALE <sup>10</sup>	—	O	FBVDD	fsr	F3	C1
FB_BE/BWE2	PA2	FB_CS2	FB_A0/ NFC_CLE <sup>11</sup>	—	O	FBVDD	fsr	E1	E2
FB_BE/BWE[1:0]	PA[1:0]	FB_TSIZ[1:0]	—	—	O	FBVDD	fsr	F2, F1	D1, F4
FB_CLK	PB7	—	—	—	O	FBVDD	fsr	G1	G1
FB_CS5	PB6	DACK1	—	—	O	FBVDD	fsr	—	F2
FB_CS4	PB5	DREQ1	—	—	O	FBVDD	fsr	—	B1
FB_CS1	PB4	—	NFC_CE	—	O	FBVDD	fsr	G3	E1
FB_CS0	PB3	—	—	—	O	FBVDD	fsr	G2	G3
<b>I<sup>2</sup>C 0</b>									
I2C0_SCL	PB2	UART8_TXD	CAN0_TX	—	I/O	EVDD	ssr	H12	G15
I2C0_SDA	PB1	UART8_RXD	CAN0_RX	—	I/O	EVDD	ssr	G12	G14
<b>FlexCAN 1</b>									
CAN1_TX	PB0	UART9_TXD	I2C1_SCL	—	I/O	EVDD	ssr	—	D14
CAN1_RX	PC7	UART9_RXD	I2C1_SDA	—	I/O	EVDD	ssr	—	D15
<b>SDRAM controller</b>									
SD_A14	—	—	—	—	O	SDVDD	st_dec_ap	—	P6

**Table 2-2. MCF5441x Signal information and muxing (continued)**

Signal name	GPIO	Alternate 1	Alternate 2	Pullup (U) <sup>1</sup> Pulldown (D)	Direction <sup>2</sup>	Voltage domain	Pad type <sup>3</sup>	196 MAPBGA <sup>4</sup>	256 MAPBGA
SD_A[13:0]	—	—	—	—	O	SDVDD	st_dec_ap	P3, M1, M3, L2, L1, N4, M2, P2, L3, L4, N1, N2, K1, N3, A2, A1, A0	R4, R1, R3, N4, P3, T4, R2, T2, N3, P5, P4, N5, P2, T3
SD_BA[2:0]	—	—	—	—	O	SDVDD	st_dec_ap	M6, L5, P4	P7, N6, R5
<u>SD_CAS</u>	—	—	—	—	O	SDVDD	st_dec_ap	L6	N8
SD_CKE	—	—	—	—	O	SDVDD	st_dec_ap	N6	R7
SD_CLK	—	—	—	—	O	SDVDD	st_ck	P6	T5
<u>SD_CLK</u>	—	—	—	—	O	SDVDD	st_ck	P7	T6
<u>SD_CS</u>	—	—	—	—	O	SDVDD	st_dec_ap	M5	N7
SD_D[7:0]	—	—	—	—	I/O	SDVDD	st_odi	P11, M10, N10, M9, P10, M8, N8, M7	T12, R11, T11, R10, N9, T10, P9, R9
SD_DM	—	—	—	—	O	SDVDD	st_odi	P7	T7
SD_DQS	—	—	—	—	I/O	SDVDD	st_dqs	P8	T8
<u>SD_DQS</u>	—	—	—	—	I/O	SDVDD	st_dqs	P9	T9
SD_ODT	—	—	—	—	O	SDVDD	st_dec_ap	P5	P8
<u>SD_RAS</u>	—	—	—	—	O	SDVDD	st_dec_ap	M4	R6
<u>SD_WE</u>	—	—	—	—	O	SDVDD	st_dec_ap	N5	R8
SD_VREF	—	—	—	—	—	SDVDD	st_vref	N9	P10
SD_VTT	—	—	—	—	—	SDVDD	st_vtt	L7	N10
<b>External interrupts port</b>									
<u>IRQ7</u>	PC6	—	—	—	I	EVDD	ssr	F12	F12
<u>IRQ6</u>	PC5	—	USB_CLKIN <sup>12</sup>	—	I	EVDD	ssr	—	N1
<u>IRQ4</u>	PC4	DREQ0	—	—	I	EVDD	ssr	E11	F14

Table 2-2. MCF5441x Signal information and muxing (continued)

Signal name	GPIO	Alternate 1	Alternate 2	Pullup (U) <sup>1</sup> Pulldown (D)	Direction <sup>2</sup>	Voltage domain	Pad type <sup>3</sup>	196 MAPBGA <sup>4</sup>	256 MAPBGA
<u>IRQ3</u>	PC3	DSPI0_PCS3	USBH_VBUS_EN	—	I	EVDD	ssr	—	M1
<u>IRQ2</u>	PC2	DSPI0_PCS2	USBH_VBUS_OC	— <sup>13</sup>	I	EVDD	ssr	—	M2
<u>IRQ1</u>	PC1	—	—	—	I	EVDD	ssr	E13	F13
<b>USB On-the-Go</b>									
USBO_DM	—	—	—	—	I/O	VDD_USB0	ae	B13	A14
USBO_DP	—	—	—	—	I/O	VDD_USB0	ae	A13	B14
<b>USB host</b>									
USBH_DM	—	—	—	—	I/O	VDD_USBH	ae	—	A15
USBH_DP	—	—	—	—	I/O	VDD_USBH	ae	—	B15
<b>ADC</b>									
ADC_IN7/ DAC1_OUT	—	—	—	—	I	VDDA_DAC_ADC	ae	—	K3
ADC_IN[6:4]	—	—	—	—	I	VDDA_ADC	ae	—	H2, J3, G4
ADC_IN3/ DAC0_OUT	—	—	—	—	I	VDDA_DAC_ADC	ae	—	K4
ADC_IN[2:0]	—	—	—	—	I	VDDA_ADC	ae	—	J2, J1, H1
<b>Real time clock</b>									
RTC_EXTAL	—	—	—	—	I <sup>5</sup>	VSTBY	ae	B14	B16
RTC_XTAL	—	—	—	—	O	VSTBY	ae	C14	C16
<b>DSPI/SBF<sup>14</sup></b>									
DSPI0_PCS1/ SBF_CS	PC0	—	—	—	I/O	EVDD	msr	K3	L1
DSPI0_PCS0/SS	PD7	I2C3_SDA	SDHC_DAT3	—	I/O	EVDD	msr	J1	K2
DSPI0_SCK/ SBF_CK	PD6	I2C3_SCL	SDHC_CLK	—	I/O	EVDD	msr	J3	L2

**Table 2-2. MCF5441x Signal information and muxing (continued)**

Signal name	GPIO	Alternate 1	Alternate 2	Pullup (U) <sup>1</sup> Pulldown (D)	Direction <sup>2</sup>	Voltage domain	Pad type <sup>3</sup>	196 MAPBGA <sup>4</sup>	256 MAPBGA
DSPI0_SIN/ SBF_DI	PD5	UART3_RXD	SDHC_CMD	U <sup>15</sup>	I	EVDD	msr	K2	L3
DSPI0_SOUT/ SBF_DO	PD4	UART3_TXD	SDHC_DAT0	—	O	EVDD	msr	J2	K1
<b>One wire</b>									
OW_DAT	GPIO0/PD3	DACK0	—	—	I/O	EVDD	ssr	M11	N11
<b>DMA timers</b>									
T3IN/PWM_EXTA3	GPIO1/PD2	T3OUT	USBO_VBUS_EN/ ULPI_DIR <sup>16</sup>	—	I	EVDD	msr	G13	G13
T2IN/PWM_EXTA2	GPIO2/PD1	T2OUT	SDHC_DAT2	—	I	EVDD	msr	J12	H14
T1IN/PWM_EXTA1	GPIO3/PD0	T1OUT	SDHC_DAT1	—	I	EVDD	msr	H13	H13
T0IN/PWM_EXTA0	GPIO4/PE7	T0OUT	USBO_VBUS_OC/ ULPI_NXT <sup>17</sup>	— <sup>18</sup>	I	EVDD	msr	J13	H15
<b>UART 2</b>									
UART2_CTS	GPIO14/PE6	UART6_TXD	SSI1_BCLK	—	I	EVDD	msr	—	M4
UART2_RTS	GPIO15/PE5	UART6_RXD	SSI1_FS	—	O	EVDD	msr	—	M3
UART2_RXD	PE4	PWM_A3	SSI1_RXD	—	I	EVDD	msr	—	P1
UART2_TXD	PE3	PWM_B3	SSI1_TXD	—	I/O 19	EVDD	msr	—	N2
<b>UART 1</b>									
UART1_CTS	GPIO7/PE2	UART5_TXD	DSPI3_SCK	—	I	EVDD	msr	D11	C10
UART1_RTS	GPIO8/PE1	UART5_RXD	DSPI3_PCS0	—	O	EVDD	msr	C12	D10
UART1_RXD	PE0	I2C5_SDA	DSPI3_SIN	—	I	EVDD	msr	B10	C9
UART1_TXD	PF7	I2C5_SCL	DSPI3_SOUT	—	I/O 19	EVDD	msr	C10	D9
<b>UART 0</b>									
UART0_CTS	GPIO5/PF6	UART4_TXD	DSPI2_SCK	—	I	EVDD	msr	E12	E13
UART0_RTS	GPIO6/PF5	UART4_RXD	DSPI2_PCS0	—	O	EVDD	msr	D12	B11
UART0_RXD	PF4	I2C4_SDA	DSPI2_SIN	—	I	EVDD	msr	C11	B10
UART0_TXD	PF3	I2C4_SCL	DSPI2_SOUT	—	I/O 19	EVDD	msr	B11	D11
<b>Enhanced secure digital host controller</b>									
SDHC_DAT3	PF2	PWM_A1	DSPI1_PCS0	—	I/O	EVDD	msr	—	B13

**Table 2-2. MCF5441x Signal information and muxing (continued)**

Signal name	GPIO	Alternate 1	Alternate 2	Pullup (U) <sup>1</sup> Pulldown (D)	Direction <sup>2</sup>	Voltage domain	Pad type <sup>3</sup>	196 MAPBGA <sup>4</sup>	256 MAPBGA
SDHC_DAT2	PF1	PWM_B1	DSPI1_PCS2	—	I/O	EVDD	msr	—	E14
SDHC_DAT1	PF0	PWM_A2	DSPI1_PCS1	—	I/O	EVDD	msr	—	D12
SDHC_DAT0	PG7	PWM_B2	DSPI1_SOUT	—	I/O	EVDD	msr	—	B12
SDHC_CMD	PG6	PWM_B0	DSPI1_SIN	—	I/O	EVDD	msr	—	C11
SDHC_CLK	PG5	PWM_A0	DSPI1_SCK	—	O	EVDD	msr	—	A10
<b>Smart card interface 0</b>									
SIM0_DATA	GPIO13/PG4	PWM_FAULT2	SDHC_DAT7	—	I/O	EVDD	msr	—	E12
SIM0_VEN	GPIO12/PG3	PWM_FAULT0	—	—	O	EVDD	msr	—	D13
SIM0_RST	GPIO11/PG2	PWM_FORCE	SDHC_DAT6	—	O	EVDD	msr	—	C15
SIM0_PD	GPIO10/PG1	PWM_SYNC	SDHC_DAT5	—	I	EVDD	msr	—	C14
SIM0_CLK	GPIO9/PG0	PWM_FAULT1	SDHC_DAT4	—	O	EVDD	msr	—	A11
<b>Synchronous serial interface 0</b>									
SSI0_RXD	PH7	I2C2_SDA	SIM1_VEN	—	I	EVDD	msr	B12	C12
SSI0_TXD	PH6	I2C2_SCL	SIM1_DATA	—	O	EVDD	msr	A11	C13
SSI0_FS	PH5	UART7_TXD	SIM1_RST	—	I/O	EVDD	msr	C13	E15
SSI0_MCLK	PH4	SSI_CLKIN	SIM1_CLK	—	O	EVDD	msr	A12	A12
SSI0_BCLK	PH3	UART7_RXD	SIM1_PD	—	I/O	EVDD	msr	D13	A13
<b>Ethernet subsystem</b>									
MII0_MDC	PI1	RMII0_MDC <sup>20</sup>	—	—	O	EVDD	fsr	N14	P16
MII0_MDIO	PI0	RMII0_MDIO <sup>20</sup>	—	—	I/O	EVDD	fsr	M14	N16
MII0_RXDV	PJ7	RMII0_CRS_DV <sup>20</sup>	—	—	I	EVDD	fsr	M13	P14
MII0_RXD[1:0]	PJ[6:5]	RMII0_RXD[1:0] <sup>20</sup>	—	—	I	EVDD	fsr	P13, N13	R15, T15
MII0_RXER	PJ4	RMII0_RXER <sup>20</sup>	—	—	I	EVDD	fsr	M12	N14
MII0_TXD[1:0]	PJ[3:2]	RMII0_TXD[1:0] <sup>20</sup>	—	—	O	EVDD	fsr	P12, N12	R13, P13
MII0_TXEN	PJ1	RMII0_TXEN <sup>20</sup>	—	D <sup>21</sup>	O	EVDD	fsr	N11	P12
MII0_COL	PJ0	RMII1_MDC	ULPI_STP	—	I	EVDD	fsr	—	R12
MII0_TXER	PK7	RMII1_MDIO	ULPI_DATA4	—	O	EVDD	fsr	—	R14
MII0_CRS	PK6	RMII1_CRS_DV	ULPI_DATA5	—	I	EVDD	fsr	—	P11
MII0_RXD[3:2]	PK[5:4]	RMII1_RXD[1:0]	ULPI_DATA[1:0]	—	I	EVDD	fsr	—	P15, N13

## Signal Descriptions

**Table 2-2. MCF5441x Signal information and muxing (continued)**

Signal name	GPIO	Alternate 1	Alternate 2	Pullup (U) <sup>1</sup> Pulldown (D)	Direction <sup>2</sup>	Voltage domain	Pad type <sup>3</sup>	196 MAPBGA <sup>4</sup>	256 MAPBGA
MII0_RXCLK	PK3	RMII1_RXER	ULPI_DATA6	—	I	EVDD	fsr	—	M14
MII0_TXD[3:2]	PK[2:1]	RMII1_TXD[1:0]	ULPI_DATA[3:2]	—	O	EVDD	fsr	—	T13, N12
MII0_TXCLK	PK0	RMII1_TXEN	ULPI_DATA7	D <sup>21</sup>	I	EVDD	fsr	—	T14
<b>BDM/JTAG</b>									
ALLPST <sup>22</sup>	PH2	—	—	—	O	EVDD	fsr	K12	—
DDATA[3:2]	PH[1:0]	—	—	—	O	EVDD	fsr	—	L15, M13
DDATA[1:0]	PI[7:6]	—	—	—	O	EVDD	fsr	—	M15, L14
PST[3:0]	PI[5:2]	—	—	—	O	EVDD	fsr	—	J13, J16, J15, J14
JTAG_EN	—	—	—	D	I	EVDD	msr	L9	N15
PSTCLK	—	TCLK <sup>23</sup>	—	—	I	EVDD	fsr	L14	M16
DSI	—	TDI <sup>23</sup>	—	U	I	EVDD	msr	L11	L13
DSO	—	TDO <sup>23</sup>	—	—	O	EVDD	msr	L12	K14
<u>BKPT</u>	—	TMS <sup>23</sup>	—	U	I	EVDD	msr	K13	K16
DSCLK	—	TRST <sup>23</sup>	—	U	I	EVDD	msr	L10	K13
<b>Test</b> (this signal must be grounded)									
TEST	—	—	—	D	I	EVDD	ssr	L8	R16
<b>Power supplies</b>									
IVDD	—	—	—	—	—	—	—	D9, D10, E9, E10, F9, F10, G10	E9–E11, F9–F11
EVDD	—	—	—	—	—	—	—	F4–F7, G6, G7, H6, H7, J5, J6	H8, J7–J10, K6–K11, L6
FB_VDD	—	—	—	—	—	—	—	D5–D7, E4–E7	E5–E7, F5, F6, G5
SD_VDD	—	—	—	—	—	—	—	J8–J10, K7–K10	M7–M12
VDD_OSC_A_PLL	—	—	—	—	—	—	vddint	F13	F15
VSS_OSC_A_PLL	—	—	—	—	—	—	vddint	F14	F16

**Table 2-2. MCF5441x Signal information and muxing (continued)**

Signal name	GPIO	Alternate 1	Alternate 2	Pullup (U) <sup>1</sup> Pulldown (D)	Direction <sup>2</sup>	Voltage domain	Pad type <sup>3</sup>	196 MAPBGA <sup>4</sup>	256 MAPBGA
VDD_USBO	—	—	—	—	—	—	vdde	F11	G12
VDD_USBH	—	—	—	—	—	—	vdde	—	H12
VDDA_ADC	—	—	—	—	—	—	—	—	H4
VSSA_ADC	—	—	—	—	—	—	vssint	—	H5
VDDA_DAC_ADC	—	—	—	—	—	—	vddint	—	J4
VSSA_DAC_ADC	—	—	—	—	—	—	vssint	—	J5
VSTBY <sup>24</sup>	—	—	—	—	—	—	vddint	E14	E16
VSS	—	—	—	—	—	—	—	A1, A14, D8, D14, E8, F8, G4, G8, G9, G11, H4, H8–11, J4, J7, J11, J14, K4–K6, K11, P1, P14	A1, A16, D16, E8, F7, F8, G6–G11, H6, H7, H9–H11, J6, J11, J12, K12, L4, L7–L12, M5, M6, T1, T16

<sup>1</sup> All pins available with GPIO contain a configurable pull-up/down. This column indicates the pull devices that are enabled automatically at reset. Pull-ups are generally only enabled on pins with their primary function, except as noted.

<sup>2</sup> Refers to pin's primary function.

<sup>3</sup> For details on the available slew rates of the various pad types see section "Output Pad Loading and Slew Rate" of the *MCF5441x Data Sheet* or section "Slew Rate Control Registers (SRCR\_x)" in chapter "Pin-Multiplexing and Control" of the *MCF5441x Reference Manual*.

<sup>4</sup> This is tentative information — as of September 21, 2010 the 196 MAPBGA ball map has not yet been finalized.

<sup>5</sup> Enabled as input only in oscillator bypass mode (internal crystal oscillator is disabled).

<sup>6</sup> These pins are time-division multiplexed between the FlexBus and NFC. An arbitration mechanism determines which module drives these pins at any point in time.

<sup>7</sup> An internal pulldown circuit is enabled during system reset for FB\_AD[10].

<sup>8</sup> An internal pullup circuit is enabled when the system is in reset state.

<sup>9</sup> Configurable pull that is enabled and pulled up after reset.

<sup>10</sup> When configured for FB\_A1, this pin is time-division multiplexed between the FlexBus and NFC. An arbitration mechanism determines which module drives the pin at any point in time. When not configured as FB\_A1, NFC\_ALE cannot be used.

<sup>11</sup> When configured for FB\_A0, this pin is time-division multiplexed between the FlexBus and NFC. An arbitration mechanism determines which module drives the pin at any point in time. When not configured as FB\_A0, NFC\_CLE cannot be used.

<sup>12</sup> Since USB\_CLKIN is a clock signal, it must be dedicated to the USB system. Do not implement this pin as dual-use.

<sup>13</sup> When Alternate 2 is selected, then internal pullup/pulldown control will come from the MISCCR[3] register of CIM.

<sup>14</sup> When booting from serial boot flash, the SBF function is enabled automatically. After the SBF function completes its reset sequence, the signals are returned to GPIO functionality.

<sup>15</sup> Automatic pull-up when SBF controls the pin during reset only. Configurable pull when UART, DSPI, or SDHC control the pin.

## Signal Descriptions

- <sup>16</sup> If ULPI is enabled, ULPI\_DIR is available as the Alternate 2 function. If ULPI is disabled, USBO\_VBUS\_EN is available.
- <sup>17</sup> If ULPI is enabled, ULPI\_NXT is available as the Alternate 2 function. If ULPI is disabled, USBO\_VBUS\_OC is available.
- <sup>18</sup> When Alternate 2 is selected, then internal pullup/pulldown control will come from the MISCCR[2] register of CIM.
- <sup>19</sup> UARTx\_TXD pad can act as RXD(input) pad when UART One Wire mode is enabled.
- <sup>20</sup> These RMII functions are selected by the mode chosen by the MAC-NET, not by the pin-multiplexing and control (GPIO) module.
- <sup>21</sup> Configurable pull that is enabled and pulled down after reset.
- <sup>22</sup> The ALLPST signal is available only on the 196 MAPBGA package and allows limited debug trace functionality compared to the 256 MAPBGA package.
- <sup>23</sup> If JTAG\_EN is asserted, these pins default to Alternate 1 (JTAG) functionality. The GPIO module is not responsible for assigning these pins.
- <sup>24</sup> VSTBY is for optional standby lithium battery. If not used, connect to EVDD.

## 2.3 Signal Primary Functions

### 2.3.1 Reset Signals

Table 2-3 describes signals used to reset the chip or to indicate a reset.

Table 2-3. Reset Signals

Signal Name	Abbreviation	Function	I/O
Reset in	RESET	Primary reset input to the device. Asserting $\overline{\text{RESET}}$ resets the core and peripherals after four FB_CLK cycles. Asserting $\overline{\text{RESET}}$ also causes $\overline{\text{RSTOUT}}$ to be asserted.	I
Reset out	$\overline{\text{RSTOUT}}$	Reset output ( $\overline{\text{RSTOUT}}$ ) is an indicator that the chip is in reset. $\overline{\text{RSTOUT}}$ is asserted at least 512 internal system bus clock (FB_CLK) cycles in response to any internal or external reset. (The exact time depends on how long it takes for the PLL to lock and/or the serial boot sequence to complete.)	O

### 2.3.2 PLL and Clock Signals

Table 2-4 describes signals that are used to support the on-chip clock generation circuitry.

Table 2-4. PLL and Clock Signals

Signal Name	Abbreviation	Function	I/O
External clock in	EXTAL	Always driven by an external clock input except when used as a connection to the external crystal if the internal oscillator circuit is used. Clock source may be configured during reset. See <a href="#">Chapter 10, "Chip Configuration Module (CCM)</a> , for more details.	I
Crystal	XTAL	Used as a connection to the external crystal when the internal oscillator circuit is used to drive the crystal. To enable oscillator bypass mode, an external pull-up resistor on XTAL is required.	O
RTC external clock in	RTC_EXTAL	Crystal input clock for the real-time clock module.	I

**Table 2-4. PLL and Clock Signals (continued)**

Signal Name	Abbreviation	Function	I/O
RTC crystal	RTC_XTAL	Oscillator output to EXTAL RTC crystal.	O
FlexBus clock out	FB_CLK	Reflects the internal bus clock (or one-half the core/system clock). ( $f_{sys}/2$ )	O
USB clock in	USB_CLKIN	This pin allows the user to drive the reference clock to the USB module as an alternate method of generating the USB reference clock during ULPI operation. This pin should be driven only with a 60 MHz clock. The 60 MHz input can also be used without ULPI (in case the PLL system setup prevents 60 MHz generation for the USB systems).	I
SSI clock in	SSI_CLKIN	This pin allows the user to drive a specific clock frequency to the SSI module.	I

### 2.3.3 Mode Selection

**Table 2-5. Mode Selection Signals**

Signal Name	Abbreviation	Function	I/O
Boot mode	BOOTMOD[1:0]	Selects the device's boot mode and chip configuration at reset. See Chapter 10, "Chip Configuration Module (CCM)," for the signal encodings.	I

### 2.3.4 Enhanced Secure Digital Host Controller

**Table 2-6. eSDHC Signals**

Signal Name	Abbreviation	Function	I/O
SDHC data bus	SDHC_DAT[7:0]	Data lines. SDHC_DAT3 can be used as card-detection input.	I/O
SDHC command	SDHC_CMD	Command line	I/O
SDHC clock	SDHC_CLK	Clock for MMC/SD/SDIO card	O

### 2.3.5 SmartCard Interface Ports

**Table 2-7. SIM Port Signals**

Signal Name	Abbreviation	Function	I/O
SIM data	SIM $n$ _DATA	Bidirectional transmit/receive data signal	I/O
SIM supply enable	SIM $n$ _VEN	Power supply enable signal	O
SIM reset	SIM $n$ _RST	Reset signal	O
SIM card detection	SIM $n$ _PD	Card insertion detect signal	O
SIM clock	SIM $n$ _CLK	Clock for the smart card. Typical frequencies are 1–5 MHz. This clock is 372 times the data rate that is on SIM_DATA. There is no required timing relationship between this clock signal and any of the other data signals. This is because of the asynchronous nature of the protocol.	O

## 2.3.6 FlexBus Signals

Table 2-8 describes signals that are used for performing transactions on the external bus.

**Table 2-8. FlexBus Signals**

Signal Name	Abbreviation	Function	I/O
Address/Data Bus	FB_AD[31:0]	Defines address and data of external byte, word, and longword accesses. This three-state, bi-directional bus is the general-purpose address/data path to external SRAM and flash devices.	I/O
Byte enables	FB_BE/BWE[3:0]	Defines flow of data on data bus. During peripheral accesses, these output signals indicate that data is to be latched or driven onto a byte of the data bus when driven low. The BE/BWE[3:0] signals are asserted only to the memory bytes used during a read or write access. BE/BWE0 controls access to the most significant byte lane of data, and BE/BWE3 controls access to the least significant byte lane of data.  For SRAM or Flash devices, the $\overline{BE/BWE}_n$ outputs should be connected to individual byte strobe signals.  The $\overline{BE/BWE}_n$ signals are asserted during accesses to on-chip peripherals, but not to on-chip SRAM or cache.	O
Output enable	FB_OE	Indicates when an external device can drive data during external read cycles.	O
Transfer acknowledge	FB_TA	Indicates external data transfer is complete. During a read cycle, when the processor recognizes $\overline{TA}$ , it latches the data and then terminates the bus cycle. During a write cycle, when the processor recognizes $\overline{TA}$ , the bus cycle is terminated.	I
Read/Write	FB_R/W	Indicates direction of the data transfer on the bus for SRAM (R/W) accesses. A logic 1 indicates a read from a slave device and a logic 0 indicates a write to a slave device.	O
Transfer start	FB_TS	Bus control output signal indicating the start of a transfer.	O
Address Latch Enable	FB_ALE	Indicates device has begun a bus transaction and the address and attributes are valid. FB_ALE is asserted for one bus clock cycle. In multiplexed mode, ALE is used externally as an address latch enable to capture the address phase of the bus transfer.	O
Chip selects	FB_CS[5:0]	Select external devices for external bus transactions.	O

## 2.3.7 SDRAM Controller Signals

Table 2-9 describes signals used for SDRAM accesses.

**Table 2-9. SDRAM Controller Signals**

Signal Name	Abbreviation	Function	I/O
SDRAM address bus	SD_A[14:0]	Address bus used for multiplexed row and column addresses during SDRAM bus cycles	O
SDRAM data bus	SD_D[7:0]	Bidirectional, non-multiplexed data bus for SDRAM accesses	I/O

**Table 2-9. SDRAM Controller Signals (continued)**

Signal Name	Abbreviation	Function	I/O
SDRAM bank address	SD_BA[2:0]	Selects one of the SDRAM row banks	O
SDRAM clock enable	SD_CKE	SDRAM clock enable	O
DDR SDRAM clock	SD_CLK	Output clock for DDR SDRAM	O
DDR SDRAM clock	SD_CLK̄	Inverted output clock for DDR SDRAM	O
SDRAM chip selects	SD_CS	SDRAM chip select	O
SDRAM data strobes	SD_DQS SD_DQS̄	Read: input, edge-aligned with read data Write: output, center-aligned with write data	I/O
SDRAM write data byte mask	SD_DM	Determines which data beat of the burst should be latched during a write cycle	O
SDRAM column address strobe	SD_CAS	SDRAM column address strobe.	O
SDRAM row address strobe	SD_RAS	SDRAM row address strobe.	O
SDRAM write enable	SD_WE	Indicates direction of data transfer on bus for SDRAM accesses. A logic 1 indicates a read from a slave device and a logic 0 indicates a write to a slave device.	O
SDRAM on-die termination	SD_ODT	On-die-termination	O
Voltage reference	SD_VREF	Voltage supply reference	—
Termination supply	SD_VTT	Voltage supply for termination resistors	—

### 2.3.8 Serial Boot Facility Signals

**Table 2-10. SBF Signals**

Signal Name	Abbreviation	Function	I/O
SBF Chip Select	SBF_CS	Chip select used to access external SPI memory.	O
SBF Clock	SBF_CK	Clock source for external SPI memory.	O
SBF Data In	SBF_DI	Data being driven by SPI memory.	I
SBF Data Out	SBF_DO	Data out to SPI memory. SBF uses this output solely for the purpose of issuing the SPI memory READ command. SBF does not write data to SPI memory.	O

### 2.3.9 External Interrupt Signals

**Table 2-11. External Interrupt Signals**

Signal Name	Abbreviation	Function	I/O
Edge port	IRQ[7,6,4:1]	External interrupt sources.	I

## 2.3.10 DMA Signals

Table 2-12. DMA Signals

Signal Name	Abbreviation	Function	I/O
DMA request	DREQ[1:0]	Asserted by an external device to request a DMA transfer.	I
DMA acknowledge	DACK[1:0]	Asserted by processor to indicate DMA request has been recognized.	O

## 2.3.11 Ethernet Controllers (MACNET0–1) Signals

The processor contains two Ethernet controllers. However, due to external pin limitations, if MII mode is used, only FEC0 can be used. In RMII mode, both Ethernet controllers are available.

Table 2-13. ENET Signal Descriptions

Signal		Description
MII	RMII	
MII_COL	—	Asserted upon detection of a collision and remains asserted while the collision persists. This signal is not defined for full-duplex mode.
MII_CRS	—	Carrier sense. When asserted, indicates transmit or receive medium is not idle. In RMII mode, this signal is present on the RMII_CRS_DV pin.
MII_MDC	RMII_MDC	Output clock provides a timing reference to the PHY for data transfers on the MDIO signal.
MII_MDIO	RMII_MDIO	Transfers control information between the external PHY and the media-access controller. Data is synchronous to MDC. This signal is an input after reset.
MII_RXCLK	—	Provides a timing reference for RXDV, RXD[3:0], and RXER.
MII_RXDV	RMII_CRS_DV	Asserting this input indicates the PHY has valid nibbles present on the MII. RXDV must remain asserted from the first recovered nibble of the frame through to the last nibble. Asserting RXDV must start no later than the SFD and exclude any EOF. In RMII mode, this pin also generates the CRS signal.
MII_RXD0	RMII_RXD0	Contains the Ethernet input data transferred from PHY to the media-access controller when RXDV is asserted.
MII_RXD1	RMII_RXD1	
MII_RXD[3:2]	—	
MII_RXER	RMII_RXER	When asserted with RXDV, indicates the PHY detects an error in the current frame. When RXDV is negated, RXER has no effect.
MII_TXCLK	—	Input clock which provides a timing reference for TXEN, TXD[3:0], and TXER.
MII_TXD0	RMII_TXD0	The serial output Ethernet data and only valid during the assertion of TXEN.
MII_TXD1	RMII_TXD1	
MII_TXD[3:2]	—	
MII_TXEN	RMII_TXEN	Indicates when valid nibbles are present on the MII. This signal is asserted with the first nibble of a preamble and is negated before the first TXCLK following the final nibble of the frame.
MII_TXER	—	When asserted for one or more clock cycles while TXEN is also asserted, PHY sends one or more illegal symbols. TXER has no effect at 10 Mbps or when TXEN is negated.
—	RMII_REF_CLK	In RMII mode, this signal is the reference clock for receive, transmit, and the control interface.

## 2.3.12 NAND Flash Controller Signals

**Table 2-14. NFC Signal Properties**

Name	Function	I/O
NFC_ALE	Flash address latch enable	O
<u>NFC_CE</u>	Flash chip enable	O
NFC_CLE	Flash command latch enable	O
NFC_R/B	Flash ready/busy	I
NFC_RE	Flash read enable	O
<u>NFC_WE</u>	Flash write enable	O
NFC_IO[15:0]	Flash data bus	I/O

## 2.3.13 Analog-to-Digital Converter Signals

Table 2-15 shows the ADC signal interface.

**Table 2-15. ADC Signal Description**

Signal	I/O	Function
ADC_IN[7:0]	I	Analog input to be converted.

## 2.3.14 Digital-to-Analog Converter Signals

This device contains two DACs.

**Table 2-16. External Signal Properties**

Name	I/O	Function
DACn_OUT	O	Analog output. Each DAC has a single current mode analog output pin. The digital words to be converted are 12 bits long with an lsb representing 0.806 mV. Analog output ranges from $\sim V_{SSA} + 40$ mV to $\sim V_{DDA} - 40$ mV (actual output range can be found in the device's data sheet) and can drive a 3-k $\Omega$ load.

## 2.3.15 Rapid GPIO Signals

**Table 2-17. GPIO Module External I/O Signals**

Signal Name	Description	Type
GPIO[15:0]	GPIO data input/output	I/O

## 2.3.16 1-Wire Signals

**Table 2-18. 1-Wire Module Signal Descriptions**

Signal	I/O	Function
OW_DAT	I/O	One-Wire bus Requires an external pull-up resistor. The recommended resistor value is specified by the generic 1-Wire device used in a given system.

## 2.3.17 PWM I/O Signals

**Table 2-19. PWM Signal Descriptions**

Signal	Description	I/O
PWM_A[3:0] PWM_B[3:0]	External PWM pair. They can be independent output PWM signals or a complementary pair. When not needed as an output, they can be used for input capture.	I/O
PWMFAULT[2:0]	Fault inputs for disabling selected PWM outputs	I
PWM_SYNC	External synchronization signal. Allows a source external to the PWM to initialize the PWM counter.	I
PWM_FORCE	External output force signal. Allows a source external to the PWM to force an update of the PWM outputs. For example, simultaneously switching all PWM outputs on a commutation boundary for trapezoidal control of a BLDC motor. The boundary can be established by external logic or an on-chip timer.	I
PWM_EXTA[3:0]	Alternate PWM control signals. These pins allow an alternate source to control the PWMA outputs. Although typically, the PWM_EXTA $n$ input is used for the generation of a complementary pair. Typical connections include ADC results registers, timer outputs, GPIO inputs, and comparator outputs.	I

## 2.3.18 FlexCAN Signals

This device contains two FlexCAN modules.

**Table 2-20. FlexCAN Signals**

Signal Name	Abbreviation	Function	I/O
FlexCAN Transmit	CAN $n$ _TX	Controller area network transmit data output.	O
FlexCAN Receive	CAN $n$ _RX	Controller area network receive data input.	I

### 2.3.19 I<sup>2</sup>C I/O Signals

This device contains six identical I<sup>2</sup>C modules.

**Table 2-21. I<sup>2</sup>C I/O Signals**

Signal Name	Abbreviation	Function	I/O
Serial clock	I2Cn_SCL	Open-drain clock signal for I <sup>2</sup> C interface. It is driven by the I <sup>2</sup> C module when the bus is in master mode, or it becomes the clock input when the I <sup>2</sup> C is in slave mode.	I/O
Serial data	I2Cn_SDA	Open-drain signal serving as the data input/output for the I <sup>2</sup> C interface.	I/O

### 2.3.20 DMA Serial Peripheral Interface (DSPI) Signals

This device contains four DSPI modules.

**Table 2-22. DMA Serial Peripheral Interface (DSPI) Signals**

Signal Name	Abbreviation	Function	I/O
DSPI synchronous serial output	DSPI <sub>n</sub> _SOUT	Provides the serial data from the DSPI and can be programmed to be driven on the rising or falling edge of DSPI_SCK.	O
DSPI synchronous serial data input	DSPI <sub>n</sub> _SIN	Provides the serial data to the DSPI and can be programmed to be sampled on the rising or falling edge of DSPI_SCK.	I
DSPI serial clock	DSPI <sub>n</sub> _SCK	Provides the serial clock from the DSPI. In master mode, the processor generates DSPI_SCK, while in slave mode, DSPI_SCK is an input from an external bus master.	I/O
DSPI peripheral chip selects	DSPI <sub>n</sub> _PCS[3:1]	Provide DSPI peripheral chip selects that can be programmed to be active high or low. <b>Note:</b> There are no DSPI1_PCS3, DSPI2_PCS[3:1], DSPI3_PCS[3:1] signals on this device.	O
DSPI peripheral chip select 0/slave select	DSPI <sub>n</sub> _PCS0/ DSPI <sub>n</sub> _SS	In master mode, DSPI_PCS0 is a peripheral chip select output that selects which slave device the current transmission is intended. In slave mode, the SS signal is a slave select input that allows an SPI master to select the processor as the target for transmission.	I/O

### 2.3.21 Synchronous Serial Interface (SSI) Signals

This device contains two SSI modules.

**Table 2-23. SSI Module Signals**

Signal Name	Abbreviation	Function	I/O
Serial bit clock	SSIn_BCLK	Used by the receive and transmit blocks. In gated clock mode, SSIn_BCLK is only valid during transmission of data, otherwise it is pulled to an inactive state.	I/O
Serial master clock	SSIn_MCLK	This clock signal is output from the device when it is the master. When in I <sup>2</sup> S master mode, this signal is referred to as the oversampling clock. The frequency of SSIn_MCLK is a multiple of the frame clock. <b>Note:</b> There is no SSIn1_MCLK signal on this device.	O
Serial frame sync	SSIn_FS	Used by transmitter/receiver to synchronize the transfer of data. In gated clock mode, this signal is not used. When configured as an input, the external device should drive SSIn_FS during the rising edge of SSIn_BCLK.	I/O
Serial receive data	SSIn_RXD	Receives data into the receive data shift register	I
Serial transmit data	SSIn_TXD	Transmits data from the serial transmit shift register.	O

### 2.3.22 Universal Serial Bus (USB) Signals

**Table 2-24. USB Module Signals**

Signal Name	Abbreviation	Function	I/O
USB On-the-Go D-	USBO_DM	D- output of the dual-speed transceiver for the On-the-Go module.	O
USB On-the-Go D+	USBO_DP	D+ output of the dual-speed transceiver for the On-the-Go module.	O
USB OTG VBUS enable	USBO_VBUS_EN	Enables the off-chip VBUS charge pump when USB OTG module is configured as a host.	O
USB OTG VBUS over-current	USBO_VBUS_OC	Indicates to the processor that a short has occurred on USB data bus.	I
USB host D-	USBH_DM	D- output of the dual-speed transceiver for the host module.	O
USB host D+	USBH_DP	D+ output of the dual-speed transceiver for the host module.	O
USB host VBUS enable	USBH_VBUS_EN	Enables the off-chip VBUS charge pump when USB host module is configured as a host.	O
USB host VBUS over-current	USBH_VBUS_OC	Indicates to the processor that a short has occurred on USB data bus.	I
ULPI Data Bus	ULPI_DATA[7:0]	These bi-directional signals are ULPI data bus. Synchronous to USB_CLKIN.	I/O
ULPI Next Data	ULPI_NXT	This input is the ULPI next data. Synchronous to USB_CLKIN.	I
ULPI Stop Data	ULPI_STP	This output is the ULPI stop data. Synchronous to USB_CLKIN.	O
ULPI Data Bus Direction	ULPI_DIR	This input is the ULPI data bus direction. Synchronous to USB_CLKIN.	I

### 2.3.23 UART Module Signals

**Table 2-25** describes the signals of the ten UART modules, where  $n$  equals 0–9 and  $m$  equals 0–2. Baud-rate clock inputs are not supported.

**Table 2-25. UART Module Signals**

Signal Name	Abbreviation	Function	I/O
Transmit serial data output	UnTXD	Data is shifted out lsb first at the falling edge of the serial clock source. Output is held high when transmitter is disabled, idle, or in local loopback mode.	O
Receive serial data input	UnRXD	Data is sampled lsb first at the serial clock source's rising edge. When the UART clock is stopped for power-down mode, any transition on this pin restarts it.	I
Clear-to-send	UmCTS	Indicates UART modules can begin data transmission <b>Note:</b> There are no U3CTS–U9CTS signals on this device.	I
Request-to-send	UmRTS	Automatic request-to-send outputs from UART modules. They may also be asserted and negated as a function of the received FIFO level. <b>Note:</b> There are no U3RTS–U9RTS signals on this device.	O

### 2.3.24 DMA Timer Signals

**Table 2-26** describes the signals of the four DMA timer modules, where  $n$  equals 0–3.

**Table 2-26. DMA Timer Signals**

Signal Name	Abbreviation	Function	I/O
DMA timer $n$ input	DT $n$ IN	Can be programmed to cause events in the respective timer. It can clock the event counter or provide a trigger to the timer value capture logic.	I
DMA timer $n$ output	DT $n$ OUT	Output from respective timer.	O

### 2.3.25 Debug Support Signals

These signals are used as the interface to the on-chip JTAG controller and the BDM logic. Pin functionality between JTAG and BDM is dependent upon the JTAG\_EN pin.

**Table 2-27. Debug Support Signals**

Signal Name	Abbreviation	Function	I/O
JTAG enable	JTAG_EN	Enables JTAG (asserted) or BDM (negated) operation.	I
<b>JTAG Signals</b>			
Test reset	TRST	Active-low signal used to initialize the JTAG logic asynchronously.	I
Test clock	TCLK	Used to synchronize the JTAG logic.	I
Test mode select	TMS	Used to sequence the JTAG state machine. TMS is sampled on the rising edge of TCLK.	I

**Table 2-27. Debug Support Signals (continued)**

<b>Signal Name</b>	<b>Abbreviation</b>	<b>Function</b>	<b>I/O</b>
Test data input	TDI	Serial input for test instructions and data. TDI is sampled on the rising edge of TCLK.	I
Test data output	TDO	Serial output for test instructions and data. TDO is three-stateable and actively driven in the shift-IR and shift-DR controller states. TDO changes on the falling edge of TCLK.	O

**BDM Signals**

Development serial clock	DSCLK	Clocks the serial communication port to the BDM module during packet transfers.	I
Breakpoint	BKPT	Used to request a manual breakpoint.	I
Development serial input	DSI	Internally-synchronized signal provides data input for the serial communication port to the BDM module.	I
Development serial output	DSO	Internally-registered signal provides serial output communication for BDM module responses.	O
Processor status clock	PSTCLK	Used by the development system to know when to sample DDATA and PST signals.	O
Debug data	DDATA[3:0]	Display captured processor data and breakpoint status. The PSTCLK signal can be used by the development system to know when to sample DDATA[3:0]. <b>Note:</b> Only present on the BGA devices.	O
Processor status outputs	PST[3:0]	Indicate core status, as shown in <a href="#">Table 2-28</a> . Debug mode timing is synchronous with the processor clock; status is unrelated to the current bus transfer. The PSTCLK signal can be used by the development system to know when to sample PST[3:0]. <b>Note:</b> Only present on the BGA devices.	O
All processor status outputs	ALLPST	ALLPST is a logical AND of the four PST signals and is present in place of PST[3:0] and DDATA[3:0] on the QFP devices. When asserted, reflects that the core is halted.	O

**Table 2-28. Processor Status**

<b>PST[3:0] (BGA Devices)</b>	<b>ALLPST (QFP Devices)</b>	<b>Processor Status</b>
0000	0	Continue execution
0001	0	Begin execution of one instruction
0010	0	Reserved
0011	0	Entry into user mode
0100	0	Begin execution of PULSE and WDDATA instructions
0101	0	Begin execution of taken branch
0110	0	Reserved
0111	0	Begin execution of RTE instruction

**Table 2-28. Processor Status (continued)**

PST[3:0] (BGA Devices)	ALLPST (QFP Devices)	Processor Status
1000	0	Begin one-byte transfer on PSTDDATA
1001	0	Begin two-byte transfer on PSTDDATA
1010	0	Begin three-byte transfer on PSTDDATA
1011	0	Begin four-byte transfer on PSTDDATA
1100	0	Exception processing
1101	0	Reserved
1110	0	Processor is stopped
1111	1	Processor is halted

### 2.3.26 Test Signals

Table 2-29 describes test signals reserved for factory testing.

**Table 2-29. Test Signals**

Signal Name	Abbreviation	Function	I/O
Test	TEST	Reserved for factory testing only and in normal modes of operation should be connected to VSS to prevent unintentional activation of test functions.	I

### 2.3.27 Power and Ground Pins

The pins described in Table 2-30 provide system power and ground to the device. Multiple pins are provided for adequate current capability. All power supply pins must have adequate bypass capacitance for high-frequency noise suppression.

**Table 2-30. Power and Ground Pins**

Signal Name	Abbreviation	Function	I/O
PLL and oscillator analog supply	VDD_OSC_A_PLL	Dedicated power supply signal to isolate the sensitive oscillator and PLL analog (VCO) circuitry from the normal levels of noise present on the digital power supply.	—
Positive I/O supply	EVDD	These pins supply positive power to the I/O pads.	—
Positive core supply	IVDD	These pins supply positive power to the core logic.	—
SDRAM supply	SD_VDD	These pins supply positive power to the DDR controller.	—
FlexBus supply	FB_VDD	These pins supply positive power to the FlexBus controller.	—
USB On-the-Go supply	VDD_USBO	This pin supplies positive power to the USB OTG controller.	—
USB host supply	VDD_USBH	This pin supplies positive power to the USB host controller.	—
Ground	VSS	These pins are the negative supply (ground) for the device.	—

**Table 2-30. Power and Ground Pins (continued)**

Signal Name	Abbreviation	Function	I/O
RTC standby supply	VSTBY_RTC	Standby voltage for the real-time clock module	—
DAC/ADC supply	VDDA_DAC_ADC VSSA_DAC_ADC	These pins supply power to the DAC and also the shared ADC signals, ADC_IN3 and ADC_IN7	—
ADC supply	VDDA_ADC	Dedicated power supply pins to reduce noise coupling and to improve accuracy. The power provided to these pins is suggested to come from a low noise filtered source. Connect uncoupling capacitors between VDDA_ADC and VSSA_ADC. VSSA_ADC is shared among the analog and digital circuitry.	—
	VSSA_ADC		—

## 2.4 External Boot Mode

After reset the address bus, data bus, FlexBus control signals, and SDRAM control signals default to their bus functionalities. All other signals default to GPIO inputs (if applicable).

# **Chapter 3**

## **ColdFire Core**

### **3.1 Introduction**

This section describes the organization of the Version 4 (V4) ColdFire® processor core and an overview of the program-visible registers. For detailed information on instructions, see the ISA\_C definition in the *ColdFire Family Programmer's Reference Manual*. The V4 ColdFire core includes the enhanced multiply-accumulate unit (EMAC), and memory management unit (MMU), which are explained in detail in their own chapters. This chapter also includes a full description of exception handling, data formats, an instruction set summary, and a table of instruction timings.

#### **3.1.1 Overview**

As with all ColdFire cores, the V4 ColdFire core is comprised of two separate pipelines decoupled by an instruction buffer.

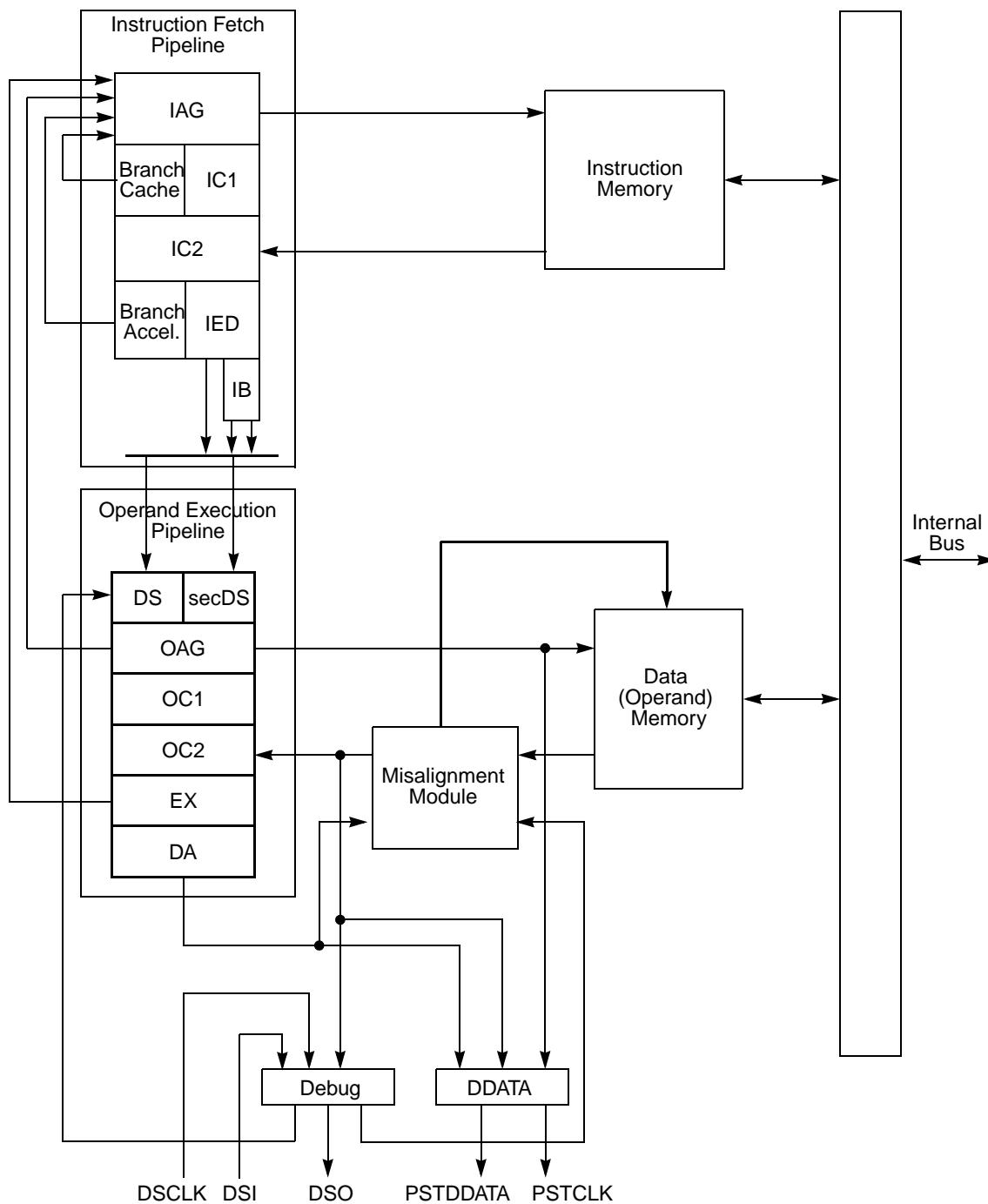


Figure 3-1. V4 ColdFire Core Pipelines

The instruction fetch pipeline (IFP) is a four-stage pipeline for prefetching instructions. The prefetched instruction stream is then gated into the five-stage operand execution pipeline (OEP), that decodes the instruction, fetches the required operands, and then executes the required function. Because the IFP and OEP pipelines are decoupled by an instruction buffer serving as a FIFO queue, the IFP is able to prefetch instructions in advance of their actual use by the OEP thereby minimizing time stalled waiting for instructions.

The V4 ColdFire core pipeline stages include the following:

- Four-stage instruction fetch pipeline (IFP) (plus optional instruction buffer stage)
  - Instruction address generation (IAG) — Calculates the next prefetch address
  - Instruction fetch cycle 1 (IC1) — Prefetch on the processor's local bus
  - Instruction fetch cycle 2 (IC2) — Completes prefetch on the processor's local bus
  - Instruction early decode (IED) — Generates time-critical decode signals needed for the OEP
  - Instruction buffer (IB) — Optional buffer stage minimizes fetch latency effects using FIFO queue
- Five-stage operand execution pipeline (OEP) with two optional processor bus write cycles
  - Decode and select (DS/secDS) — Decodes and selects two sequential instructions and selects operands for effective address calculation
  - Operand address generation (OAG) — Generates the effective (logical) address
  - Operand fetch cycle 1 (OC1) — Initiates memory operand fetch on the processor's local bus
  - Operand fetch cycle 2 (OC2) — Completes memory operand fetch on the processor's local bus, as well as immediate and/or register operand fetches
  - Execute (EX) — Performs prescribed operations on previously fetched data operands
  - Write data available (DA) — Makes data available for operand write operations only
  - Store data (ST) — Updates memory element for operand write operations only

When the instruction buffer is empty, opcodes are loaded directly from the IED cycle into the operand execution pipeline. If the buffer is not empty, the IFP stores the contents of the fetched instruction and its early decode information in the IB until it is required by the OEP.

The five stage operand execution pipeline structure is a key factor in the performance of the Version 4 ColdFire design. The pipeline structure is termed a limited superscalar design because there are certain, heavily-used instruction constructs that support multiple-instruction dispatch. In particular, folding two consecutive instructions into a single pipeline issue effectively creates zero-cycle execution times for certain instructions.

With the increased performance, the bandwidth needed to support operand references requires a split bus (or Harvard architecture) where there are separate instruction and operand memory connections. These connections may be accessed concurrently to double the amount of available bandwidth to the processor's pipelines.

The resulting pipeline and local bus structure allow the V4 ColdFire core to deliver sustained high performance across a variety of demanding embedded applications.

### **3.1.1.1 Change-of-Flow Acceleration**

To maximize the performance of conditional branch instructions, the IFP implements a sophisticated two-level acceleration mechanism. The first level is an 8-entry, direct-mapped branch cache with 2 bits for indicating four prediction states (strongly or weakly; taken or not-taken) for each entry. The branch cache also provides the association between instruction addresses and the corresponding target address. In the event of a branch cache hit, if the branch is predicted as taken, the branch cache sources the target address

from the IC1 stage back into the IAG to redirect the prefetch stream to the new location as shown in [Figure 3-1](#).

The branch cache implements instruction folding, so conditional branch instructions correctly predicted as taken can execute in zero cycles. For conditional branches with no information in the branch cache, a second-level, direct-mapped prediction table is accessed. Each of its 128 entries uses the same 2-bit prediction mechanism as the branch cache.

If a branch is predicted as taken, branch acceleration logic in the IED stage generates the target address. Other change-of-flow instructions, including unconditional branches, jumps, and subroutine calls, use a similar mechanism where the IFP calculates the target address. The performance of the subroutine return instruction (RTS) is improved through the use of a four-entry, LIFO hardware return stack. In all cases, these mechanisms allow the IFP to redirect the fetch stream down the predicted path ahead of instruction execution.

### 3.1.1.2 Operand Execution Pipeline (OEP)

The two instruction registers in the decode stage (DS) of the OEP are loaded from the FIFO instruction buffer or are bypassed directly from the instruction early decode (IED). The OEP consists of two traditional, two-stage RISC compute engines with a dual-ported register file access feeding an arithmetic logic unit (ALU).

The compute engine at the top of the OEP (the address ALU) is used typically for operand address calculations; the execution ALU at the bottom is used for instruction execution. The resulting structure provides almost 4 GB/s read operand bandwidth (at 250 MHz) to the two compute engines and supports single-cycle execution speeds for most instructions, including all load and store operations and most embedded-load operations. The V4 OEP supports the ColdFire instruction set architecture (ISA) revision C.

Advanced performance features implemented by the OEP:

- Stalls are minimized by dynamically basing the choice between the address ALU or execution ALU for instruction execution on the pipeline state.
- The address ALU and register renaming resources together can execute heavily used opcodes and forward results to subsequent instructions with no pipeline stalls.
- Instruction folding involving MOVE instructions allows two instructions to be issued in one cycle. The resulting microarchitecture approaches full superscalar performance at a much lower silicon cost.

## 3.2 Memory Map/Register Description

The following sections describe the processor registers in the user and supervisor programming models. The programming model is selected based on the processor privilege level (user mode or supervisor mode) as defined by the S bit of the status register (SR). [Table 3-1](#) lists the processor registers.

The user-programming model consists of the following registers:

- 16 general-purpose 32-bit registers (D0–D7, A0–A7)
- 32-bit program counter (PC)

- 8-bit condition code register (CCR)
- EMAC registers (described fully in [Chapter 5, “Enhanced Multiply-Accumulate Unit \(EMAC”](#))
  - Four 48-bit accumulator registers partitioned as follows:
    - Four 32-bit accumulators (ACC0–ACC3)
    - Eight 8-bit accumulator extension bytes (two per accumulator). These are grouped into two 32-bit values for load and store operations (ACCEXT01 and ACCEXT23).
  - Accumulators and extension bytes can be loaded, copied, and stored; results from EMAC arithmetic operations generally affect the entire 48-bit destination.
  - One 16-bit mask register (MASK)
  - One 32-bit Status register (MACSR) including four indicator bits signaling product or accumulation overflow (one for each accumulator: PAV0–PAV3)

The supervisor programming model is to be used only by system control software to implement restricted operating system functions, I/O control, and memory management. All accesses that affect the control features of ColdFire processors are in the supervisor programming model, that consists of registers available in user mode as well as the following control registers:

- 16-bit status register (SR)
- 32-bit supervisor stack pointer (SSP)
- 32-bit vector base register (VBR)
- 32-bit cache control register (CACR)
- 32-bit access control registers (ACR0, ACR1, ... ACR7)
- One 32-bit memory base address register (RAMBAR)
- 32-bit address space ID register (ASID)
- 32-bit MMU base address register (MMUBAR)

**Table 3-1. ColdFire Core Programming Model**

BDM <sup>1</sup>	Register	Width (bits)	Access	Reset Value	Written with MOVEC	Section/Page
<b>Supervisor/User Access Registers</b>						
Load: 0x080 Store: 0x180	Data Register 0 (D0)	32	R/W	0xCF40_6C2F	No	<a href="#">3.2.1/3-6</a>
Load: 0x081 Store: 0x181	Data Register 1 (D1)	32	R/W	0x0500_2580	No	<a href="#">3.2.1/3-6</a>
Load: 0x082–7 Store: 0x182–7	Data Register 2–7 (D2–D7)	32	R/W	Undefined	No	<a href="#">3.2.1/3-6</a>
Load: 0x088–8E Store: 0x188–8E	Address Register 0–6 (A0–A6)	32	R/W	Undefined	No	<a href="#">3.2.2/3-7</a>
Load: 0x08F Store: 0x18F	Supervisor/User A7 Stack Pointer (A7)	32	R/W	Undefined	No	<a href="#">3.2.3/3-7</a>
0x804	MAC Status Register (MACSR)	32	R/W	0x0000_0000	No	<a href="#">5.2.1/5-3</a>

**Table 3-1. ColdFire Core Programming Model (continued)**

BDM <sup>1</sup>	Register	Width (bits)	Access	Reset Value	Written with MOVEC	Section/Page
0x805	MAC Address Mask Register (MASK)	32	R/W	0xFFFF_FFFF	No	5.2.2/5-5
0x806, 0x809, 0x80A, 0x80B	MAC Accumulators 0–3 (ACC0–3)	32	R/W	Undefined	No	5.2.3/5-7
0x807	MAC Accumulator 0,1 Extension Bytes (ACCext01)	32	R/W	Undefined	No	5.2.4/5-7
0x808	MAC Accumulator 2,3 Extension Bytes (ACCext23)	32	R/W	Undefined	No	5.2.4/5-7
0x80E	Condition Code Register (CCR)	8	R/W	Undefined	No	3.2.4/3-8
0x80F	Program Counter (PC)	32	R/W	Contents of location 0x0000_0004	No	3.2.5/3-9
<b>Supervisor Access Only Registers</b>						
0x002	Cache Control Register (CACR)	32	R/W	0x0000_0000	Yes	3.2.6/3-9
0x003	Address Space Identifier (ASID)	8	R/W	0x00	Yes	4.2.1/4-4
0x004–7	Access Control Register 0–3 (ACR0–3)	32	R/W	See Section	Yes	6.3.2/6-8
0x008	MMU Base Address Register (MMUBAR)	32	R/W	0x0000_0000	Yes	4.2.2/4-4
0x009	GPIO Base Address Register (RGPIOBAR)	32	R/W	0x8C00_0034	Yes	16.3.1/16-5
0x00C–F	Access Control Register 4–7 (ACR4–7)	32	R/W	See Section	Yes	6.3.2/6-8
0x800	User/Supervisor A7 Stack Pointer (OTHER_A7)	32	R/W	Contents of location 0x0000_0000	No	3.2.3/3-7
0x801	Vector Base Register (VBR)	32	R/W	0x0000_0000	Yes	3.2.8/3-9
0x80E	Status Register (SR)	16	R/W	0x27--	No	3.2.9/3-10
0xC05	RAM Base Address Register (RAMBAR)	32	R/W	See Section	Yes	3.2.10/3-11

<sup>1</sup> The values listed in this column represent the Rc field used when accessing the core registers via the BDM port. For more information see [Chapter 43, “Debug Module”](#).

### 3.2.1 Data Registers (D0–D7)

D0–D7 data registers are for bit (1-bit), byte (8-bit), word (16-bit) and longword (32-bit) operations; they can also be used as index registers.

#### NOTE

Registers D0 and D1 contain hardware configuration details after reset. See [Section 3.3.4.15, “Reset Exception”](#) for more details.

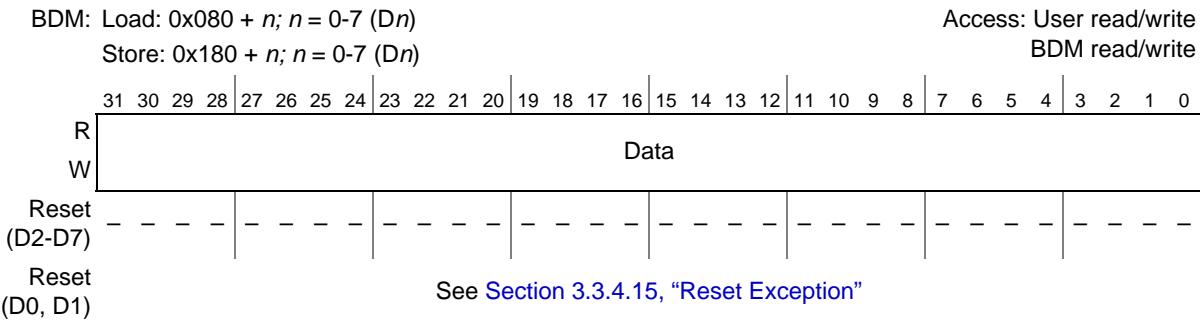


Figure 3-2. Data Registers (D0–D7)

### 3.2.2 Address Registers (A0–A6)

These registers can be used as software stack pointers, index registers, or base address registers. They can also be used for word and longword operations.

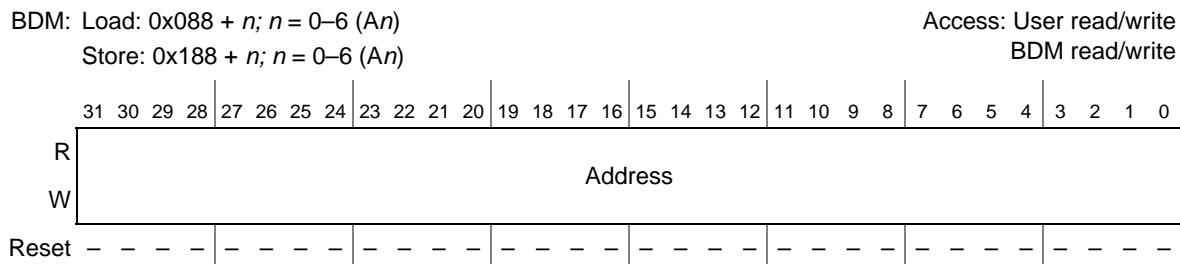


Figure 3-3. Address Registers (A0–A6)

### 3.2.3 Supervisor/User Stack Pointers (A7 and OTHER\_A7)

The ColdFire architecture supports two independent stack pointer (A7) registers—the supervisor stack pointer (SSP) and the user stack pointer (USP). The hardware implementation of these two program-visible 32-bit registers does not identify one as the SSP and the other as the USP. Instead, the hardware uses one 32-bit register as the active A7 and the other as OTHER\_A7. Thus, the register contents are a function of the processor operation mode, as shown in the following:

```
if SR[S] = 1
    then      A7 = Supervisor Stack Pointer
              OTHER_A7 = User Stack Pointer
    else      A7 = User Stack Pointer
              OTHER_A7 = Supervisor Stack Pointer
```

The BDM programming model supports direct reads and writes to the (active) A7 and OTHER\_A7. It is the responsibility of the external development system to determine, based on the setting of SR[S], the mapping of A7 and OTHER\_A7 to the two program-visible definitions (SSP and USP). This functionality is enabled by setting the enable user stack pointer bit, CACR[EUSP]. If this bit is cleared, only a single stack pointer (A7), originally defined for ColdFire ISA\_A, is available. EUSP is cleared at reset.

To support dual stack pointers, the following two supervisor instructions are included in the ColdFire instruction set architecture to load/store the USP:

```
move.l Ay,USP ; move to USP
```

```
move.l USP,Ax;move from USP
```

These instructions are described in the *ColdFire Family Programmer's Reference Manual*. All other instruction references to the stack pointer, explicit or implicit, access the active A7 register.

### NOTE

The SSP is loaded during reset exception processing with the contents of location 0x0000\_0000.

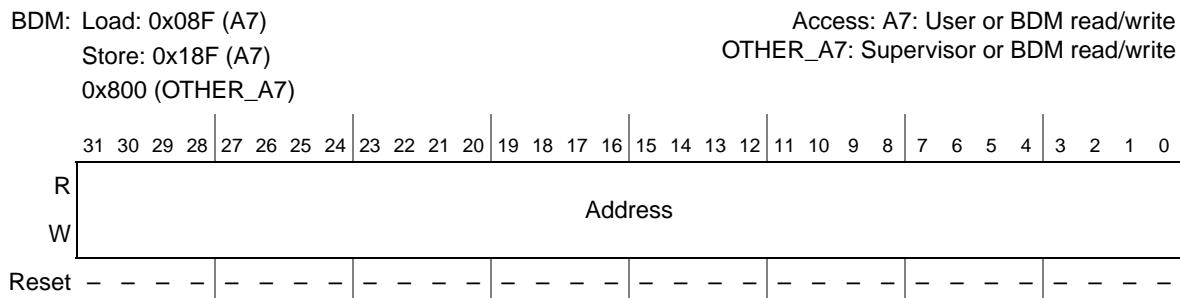


Figure 3-4. Stack Pointer Registers (A7 and OTHER\_A7)

### 3.2.4 Condition Code Register (CCR)

The CCR is the LSB of the processor status register (SR). Bits 4–0 act as indicator flags for results generated by processor operations. The extend bit (X) is also an input operand during multiprecision arithmetic computations.

### NOTE

The CCR register must be explicitly loaded after reset and before any compare (CMP), Bcc, or Scc instructions are executed.

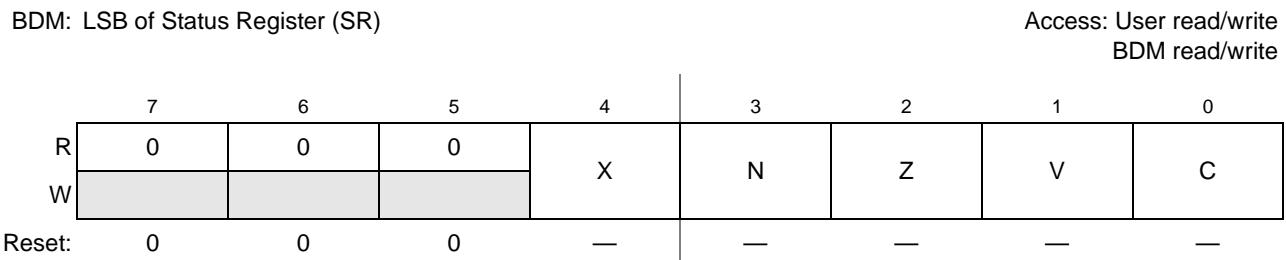


Figure 3-5. Condition Code Register (CCR)

Table 3-2. CCR Field Descriptions

Field	Description
7–5	Reserved, must be cleared.
4 X	Extend condition code bit. Set to the C-bit value for arithmetic operations; otherwise not affected or set to a specified result.
3 N	Negative condition code bit. Set if most significant bit of the result is set; otherwise cleared.

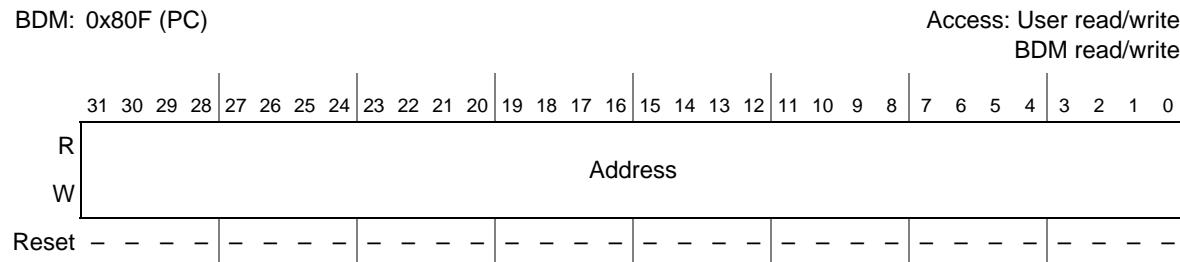
**Table 3-2. CCR Field Descriptions (continued)**

Field	Description
Z	Zero condition code bit. Set if result equals zero; otherwise cleared.
V	Overflow condition code bit. Set if an arithmetic overflow occurs implying the result cannot be represented in operand size; otherwise cleared.
C	Carry condition code bit. Set if a carry out of the operand msb occurs for an addition or if a borrow occurs in a subtraction; otherwise cleared.

### 3.2.5 Program Counter (PC)

The PC contains the currently executing instruction address. During instruction execution and exception processing, the processor automatically increments PC contents or places a new value in the PC. The PC is a base address for PC-relative operand addressing.

The PC is initially loaded during reset exception processing with the contents at location 0x0000\_0004.

**Figure 3-6. Program Counter Register (PC)**

### 3.2.6 Cache Programming Model

The registers in the cache portion of the programming model are described in [Chapter 6, “Cache.”](#)

### 3.2.7 MMU Programming Model

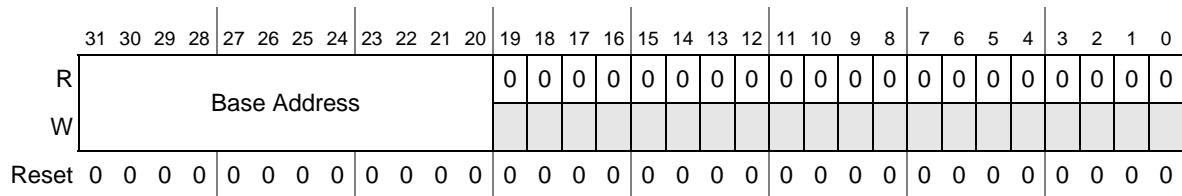
The registers in the MMU portion of the programming model are described in [Chapter 4, “Memory Management Unit \(MMU\).”](#)

### 3.2.8 Vector Base Register (VBR)

The VBR contains the base address of the exception vector table in the memory. To access the vector table, the displacement of an exception vector is added to the value in VBR. The lower 20 bits of the VBR are not implemented by ColdFire processors. They are assumed to be zero, forcing the table to be aligned on a 1 MB boundary.

BDM: 0x801 (VBR)

Access: Supervisor read/write  
BDM read/write



**Figure 3-7. Vector Base Register (VBR)**

### **3.2.9 Status Register (SR)**

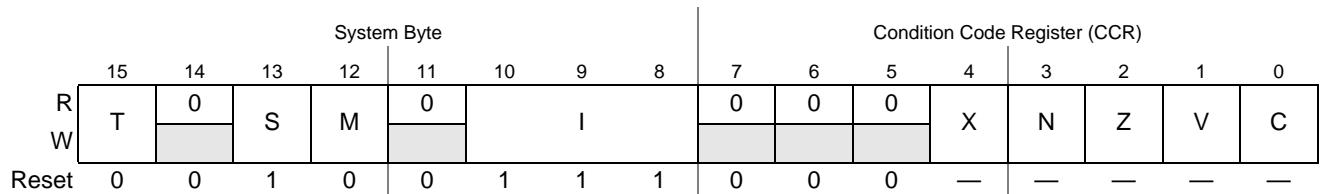
The SR stores the processor status and includes the CCR, the interrupt priority mask, and other control bits. In supervisor mode, software can access the entire SR. In user mode, only the lower 8 bits (CCR) are accessible. The control bits indicate the following states for the processor: trace mode (T bit), supervisor or user mode (S bit), and master or interrupt state (M bit). All defined bits in the SR have read/write access when in supervisor mode.

## **NOTE**

The lower byte of the SR (the CCR) must be loaded explicitly after reset and before any compare (CMP), Bcc, or Scc instructions execute.

BDM: 0x80E (SR)

Access: Supervisor read/write  
BDM read/write



**Figure 3-8. Status Register (SR)**

**Table 3-3. SR Field Descriptions**

Field	Description
15 T	Trace enable. When set, the processor performs a trace exception after every instruction.
14	Reserved, must be cleared.
13 S	Supervisor/user state. 0 User mode 1 Supervisor mode
12 M	Master/interrupt state. Bit is cleared by an interrupt exception and software can set it during execution of the RTE or move to SR instructions.
11	Reserved, must be cleared.

**Table 3-3. SR Field Descriptions (continued)**

Field	Description
10–8 I	Interrupt level mask. Defines current interrupt level. Interrupt requests are inhibited for all priority levels less than or equal to current level, except edge-sensitive level 7 requests, which cannot be masked.
7–0 CCR	Refer to <a href="#">Section 3.2.4, “Condition Code Register (CCR)”</a> .

### 3.2.10 Memory Base Address Register (RAMBAR)

The memory base address register is used to specify the base address of the internal SRAM module and indicates the types of references mapped to it. The base address register includes a base address, write-protect bit, address space mask bits, and an enable bit. RAMBAR determines the base address of the on-chip RAM. For more information, refer to [Section 7.2.1, “SRAM Base Address Register \(RAMBAR\)”](#).

## 3.3 Functional Description

### 3.3.1 Version 4 ColdFire Microarchitecture

As previously discussed, the unrolling of the operand execution pipeline into a five-stage structure is a key factor in the improved performance of the Version 4 ColdFire design. The resulting pipeline structure is termed a limited superscalar design because there are certain, heavily-used instruction constructs that support multiple-instruction dispatch. The following figure presents the top-level spatial block diagram of the Version 4 ColdFire operand execution pipeline, where the major hardware structures associated with each pipeline stage are clearly visible.

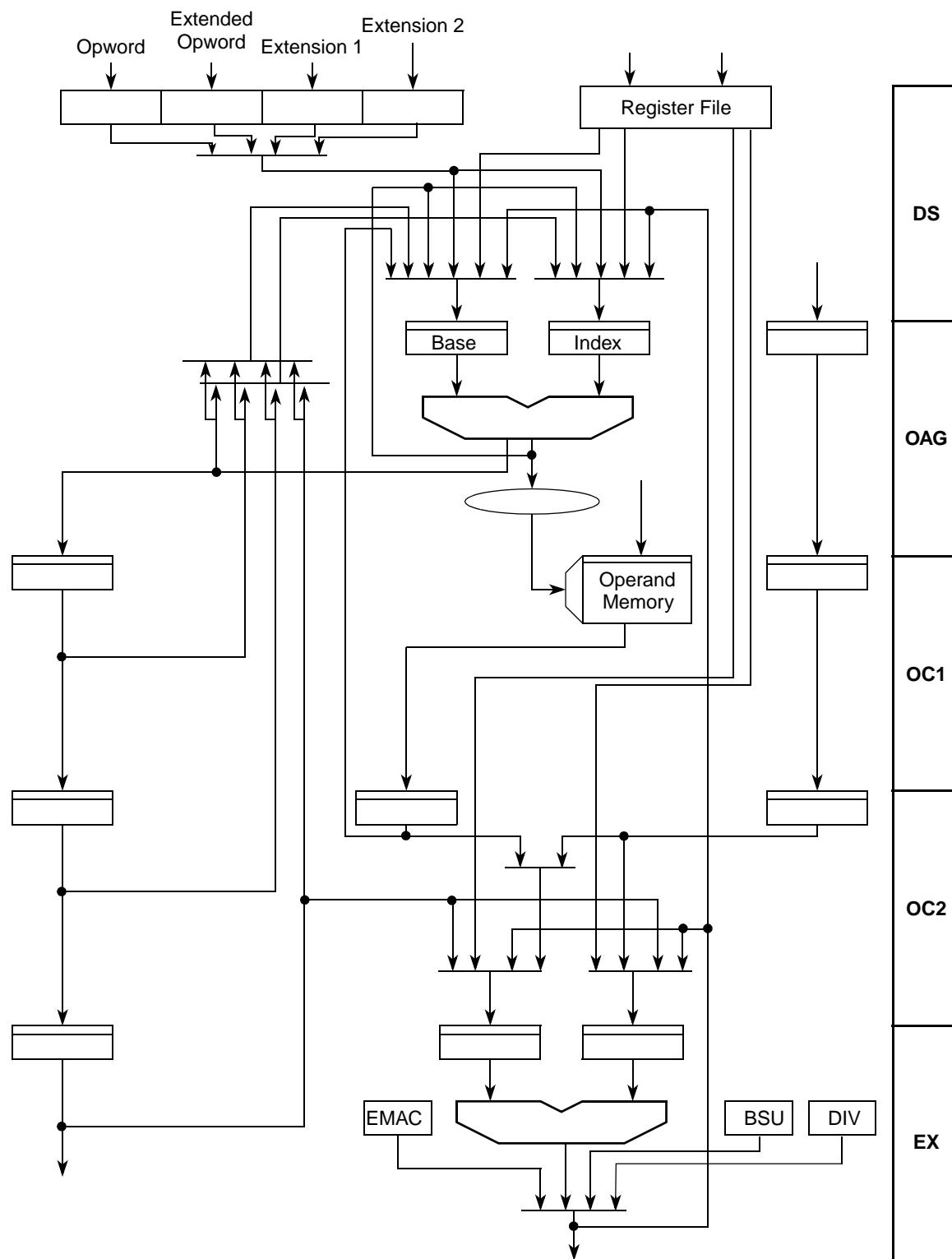


Figure 3-9. Version 4 ColdFire Processor Operand Execution Pipeline Diagram

### 3.3.2 Instruction Set Architecture (ISA\_C)

The original ColdFire instruction set architecture (ISA\_A) was derived from the M68000 family opcodes based on extensive analysis of embedded application code. The ISA was optimized for code compiled from high-level languages where the dominant operand size was the 32-bit integer declaration. This approach minimized processor complexity and cost, while providing excellent performance for compiled applications.

After the initial ColdFire compilers were created, developers noted there were certain ISA additions that would enhance code density and overall performance. Additionally, as users implemented ColdFire-based designs into a wide range of embedded systems, they found certain frequently-used instruction sequences that could be improved by the creation of additional instructions.

The original ISA definition minimized support for instructions referencing byte- and word-sized operands. Full support for the move byte and move word instructions was provided, but the only other opcodes supporting these data types are CLR (clear) and TST (test). A set of instruction enhancements has been implemented in subsequent ISA revisions, ISA\_B and ISA\_C. The added opcodes primarily addressed three areas:

1. Enhanced support for byte and word-sized operands
2. Enhanced support for position-independent code
3. Miscellaneous instruction additions to address new functionality

**Table 3-4** summarizes the instructions added to revision ISA\_A to form revision ISA\_C. For more details see the *ColdFire Family Programmer's Reference Manual*.

**Table 3-4. Instruction Enhancements over Revision ISA\_A**

Instruction	Description
BITREV	The contents of the destination data register are bit-reversed; that is, new Dn[31] equals old Dn[0], new Dn[30] equals old Dn[1], ..., new Dn[0] equals old Dn[31].
BYTEREV	The contents of the destination data register are byte-reversed; that is, new Dn[31:24] equals old Dn[7:0], ..., new Dn[7:0] equals old Dn[31:24].
FF1	The data register, Dn, is scanned, beginning from the most-significant bit (Dn[31]) and ending with the least-significant bit (Dn[0]), searching for the first set bit. The data register is then loaded with the offset count from bit 31 where the first set bit appears.
INTOUCH	Loads blocks of instructions to be locked in the instruction cache.
MOV3Q.L	Moves 3-bit immediate data to the destination location.
Move from USP	User Stack Pointer → Destination register
Move to USP	Source register → User Stack Pointer
MVS.{B,W}	Sign-extends source operand and moves it to destination register.
MVZ.{B,W}	Zero-fills source operand and moves it to destination register.
SATS.L	Performs saturation operation for signed arithmetic and updates destination register, depending on CCR[V] and bit 31 of the register.
TAS.B	Performs indivisible read-modify-write cycle to test and set addressed memory byte.
Bcc.L	Branch conditionally, longword

**Table 3-4. Instruction Enhancements over Revision ISA\_A (continued)**

Instruction	Description
BSR.L	Branch to sub-routine, longword
CMP.{B,W}	Compare, byte and word
CMPA.W	Compare address, word
CMPI.{B,W}	Compare immediate, byte and word
MOVEI	Move immediate, byte and word to memory using Ax with displacement

### 3.3.3 Exception Processing Overview

Exception processing for ColdFire processors is streamlined for performance. The ColdFire processors differ from the M68000 family because they include:

- A simplified exception vector table
- Reduced relocation capabilities using the vector-base register
- A single exception stack frame format
- A precise instruction restart model for translation (TLB miss) and access faults. This functionality extends the existing ColdFire access error fault vector in the exception stack frames.

All ColdFire processors use an instruction restart exception model. Exception processing includes all actions from fault condition detection to the initiation of fetch for first handler instruction. Exception processing is comprised of four major steps:

1. The processor makes an internal copy of the SR and then enters supervisor mode by setting the S bit and disabling trace mode by clearing the T bit. The interrupt exception also forces the M bit to be cleared and the interrupt priority mask to set to current interrupt request level.
2. The processor determines the exception vector number. For all faults except interrupts, the processor performs this calculation based on exception type. For interrupts, the processor performs an interrupt-acknowledge (IACK) bus cycle to obtain the vector number from the interrupt controller. The IACK cycle is mapped to special locations within the interrupt controller's address space with the interrupt level encoded in the address.
3. The processor saves the current context by creating an exception stack frame on the system stack. The exception stack frame is created at a 0-modulo-4 address on top of the system stack pointed to by the supervisor stack pointer (SSP). As shown in [Figure 3-10](#), the processor uses a simplified fixed-length stack frame for all exceptions with additional fault status (FS) encodings to support the MMU. The exception type determines whether the program counter placed in the exception stack frame defines the location of the faulting instruction (fault) or the address of the next instruction to be executed (next).
4. The processor calculates the address of the first instruction of the exception handler. By definition, the exception vector table is aligned on a 1 MB boundary. This instruction address is generated by fetching an exception vector from the table located at the address defined in the vector base register.

The index into the exception table is calculated as  $(4 \times \text{vector number})$ . After the exception vector has been fetched, the vector contents determine the address of the first instruction of the desired handler. After the instruction fetch for the first opcode of the handler has initiated, exception processing terminates and normal instruction processing continues in the handler.

All ColdFire processors support a 1024-byte vector table aligned on any 1 Mbyte address boundary (see [Table 3-5](#)).

The table contains 256 exception vectors; the first 64 are defined for the core and the remaining 192 are device-specific peripheral interrupt vectors. See [Chapter 17, “Interrupt Controller Modules”](#) for details on the device-specific interrupt sources.

**Table 3-5. Exception Vector Assignments**

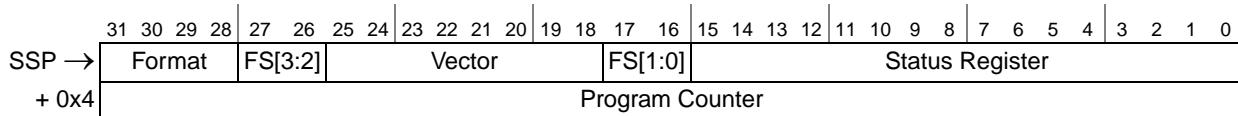
Vector Number(s)	Vector Offset (Hex)	Stacked Program Counter	Assignment
0	0x000	—	Initial supervisor stack pointer
1	0x004	—	Initial program counter
2	0x008	Fault	Access error
3	0x00C	Fault	Address error
4	0x010	Fault	Illegal instruction
5	0x014	Fault	Divide by zero
6–7	0x018–0x01C	—	Reserved
8	0x020	Fault	Privilege violation
9	0x024	Next	Trace
10	0x028	Fault	Unimplemented line-A opcode
11	0x02C	Fault	Unimplemented line-F opcode
12	0x030	Next	Non-PC breakpoint debug interrupt
13	0x034	Next	PC breakpoint debug interrupt
14	0x038	Fault	Format error
15	0x03C	Next	Uninitialized interrupt
16–23	0x040–0x05C	—	Reserved
24	0x060	Next	Spurious interrupt
25–31	0x064–0x07C	Next	Level 1–7 autovectored interrupts
32–47	0x080–0x0BC	Next	Trap # 0–15 instructions
48–60	0x0C0–0x0F0	—	Reserved
61	0x0F4	Fault	Unsupported instruction
62–63	0x0F8–0x0FC	—	Reserved
64–255	0x100–0x3FC	Next	Device-specific interrupts

<sup>1</sup> Fault refers to the PC of the instruction that caused the exception. Next refers to the PC of the instruction that follows the instruction that caused the fault.

All ColdFire processors inhibit interrupt sampling during the first instruction of all exception handlers. This allows any handler to disable interrupts effectively, if necessary, by raising the interrupt mask level contained in the status register. For more details, see *ColdFire Family Programmer's Reference Manual*.

### 3.3.3.1 Exception Stack Frame Definition

Figure 3-10 shows exception stack frame. The first longword contains the 16-bit format/vector word (F/V) and the 16-bit status register, and the second longword contains the 32-bit program counter address.



**Figure 3-10. Exception Stack Frame Form**

The 16-bit format/vector word contains three unique fields:

- A 4-bit format field at the top of the system stack is always written with a value of 4, 5, 6, or 7 by the processor, indicating a two-longword frame format. See Table 3-6.

**Table 3-6. Format Field Encodings**

Original SSP @ Time of Exception, Bits 1:0	SSP @ 1st Instruction of Handler	Format Field
00	Original SSP - 8	0100
01	Original SSP - 9	0101
10	Original SSP - 10	0110
11	Original SSP - 11	0111

- There is a 4-bit fault status field, FS[3:0], at the top of the system stack. This field is defined for access and address errors only and written as zeros for all other exceptions. See Table 3-7.

**Table 3-7. Fault Status Encodings**

FS[3:0]	Definition
0000	Not an access or address error nor an interrupted debug service routine
0001	Reserved
0010	Interrupt during a debug service routine for faults other than access errors <sup>1</sup>
0011	Reserved
0100	Error on instruction fetch
0101	TLB miss on opword of instruction fetch
0110	TLB miss on extension word of instruction fetch
0111	IFP access error while executing in emulator mode
1000	Error on operand write
1001	Attempted write to write-protected space

**Table 3-7. Fault Status Encodings (continued)**

FS[3:0]	Definition
1010	TLB miss on data write
1011	Reserved
1100	Error on operand read
1101	Attempted read, read-modify-write of protected space
1110	TLB miss on data read, or read-modify-write
1111	OEP access error while executing in emulator mode

<sup>1</sup> This refers to taking an I/O interrupt during a debug service routine. If an access error occurs during a debug service routine, FS is set to 0111 if it is due to an instruction fetch or to 1111 for a data access.

- The 8-bit vector number, vector[7:0], defines the exception type and is calculated by the processor for all internal faults and represents the value supplied by the interrupt controller in case of an interrupt. See [Table 3-5](#).

### 3.3.4 Processor Exceptions

#### 3.3.4.1 Access Error Exception

The exact processor response to an access error depends on the memory reference being performed. For an instruction fetch, the processor postpones the error reporting until the faulted reference is needed by an instruction for execution. Therefore, faults during instruction prefetches followed by a change of instruction flow do not generate an exception. When the processor attempts to execute an instruction with a faulted opword and/or extension words, the access error is signaled and the instruction is aborted. For this type of exception, the programming model has not been altered by the instruction generating the access error.

If the access error occurs on an operand read, the processor immediately aborts the current instruction's execution and initiates exception processing. The operand execution pipeline includes logic to fully recover program-visible register updates in the event of a bus transfer error acknowledge on an operand memory reference. This allows for a precise instruction restart from this class of exceptions. See [Section 3.3.4.16, “Precise Faults”](#), for additional information.

If the MMU is disabled, access errors are reported only with an attempted store to write-protected memory. Therefore, access errors associated with instruction fetch or operand read accesses are not possible. The Version 4 ColdFire processor, unlike the Version 2 and 3 ColdFire processors, updates the condition code register if a write-protect error occurs during a CLR or MOV3Q operation to memory.

Internal memory accesses that fault (terminate with an internal memory transfer error acknowledge) generate an access error exception. MMU TLB misses and access violations use the same fault. If the MMU is enabled, all TLB misses and protection violations generate an access error exception. To determine if a fault is due to a TLB miss or another type of access error, new FS encodings (described in [Table 3-7](#)) signal TLB misses on instruction fetch, instruction extension fetch, and data read and writes.

### 3.3.4.2 Address Error Exception

Any attempted execution transferring control to an odd instruction address (if bit 0 of the target address is set) results in an address error exception.

Any attempted use of a word-sized index register (Xn.w) or a scale factor of eight on an indexed effective addressing mode generates an address error, as does an attempted execution of a full-format indexed addressing mode, which is defined by bit 8 of extension word 1 being set.

If an address error occurs on a JSR instruction, the Version 4 ColdFire processor first pushes the return address onto the stack and then calculates the target address. If an address error occurs on an RTS instruction, the Version 4 ColdFire processor preserves the original return PC and writes the exception stack frame above this value.

### 3.3.4.3 Illegal Instruction Exception

The ColdFire variable-length instruction set architecture supports three instruction sizes: 16, 32, or 48 bits. The first instruction word is known as the operation word (or opword), while the optional words are known as extension word 1 and extension word 2. The opword is further subdivided into three sections: the upper four bits segment the entire ISA into 16 instruction lines, the next 6 bits define the operation mode (opmode), and the low-order 6 bits define the effective address. See [Figure 3-11](#). The opword line definition is shown in [Table 3-8](#).

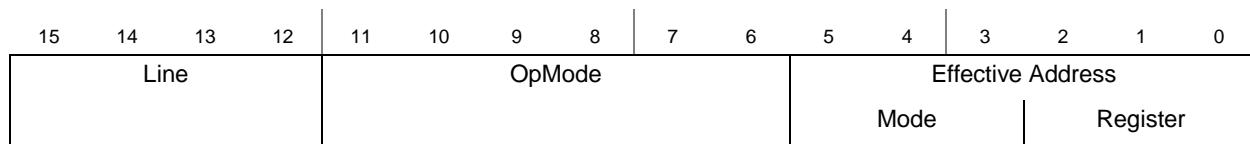


Figure 3-11. ColdFire Instruction Operation Word (Opword) Format

Table 3-8. ColdFire Opword Line Definition

Opword[Line]	Instruction Class
0x0	Bit manipulation, Arithmetic and Logical Immediate
0x1	Move Byte
0x2	Move Long
0x3	Move Word
0x4	Miscellaneous
0x5	Add (ADDQ) and Subtract Quick (SUBQ), Set according to Condition Codes (Scc)
0x6	PC-relative change-of-flow instructions Conditional (Bcc) and unconditional (BRA) branches, subroutine calls (BSR)
0x7	Move Quick (MOVEQ), Move with sign extension (MVS) and zero fill (MVZ)
0x8	Logical OR (OR)
0x9	Subtract (SUB), Subtract Extended (SUBX)
0xA	EMAC, Move 3-bit Quick (MOV3Q)
0xB	Compare (CMP), Exclusive-OR (EOR)

**Table 3-8. ColdFire Opword Line Definition (continued)**

Opword[Line]	Instruction Class
0xC	Logical AND (AND), Multiply Word (MUL)
0xD	Add (ADD), Add Extended (ADDX)
0xE	Arithmetic and logical shifts (ASL, ASR, LSL, LSR)
0xF	Cache Push (CPUSHL), Write DDATA (WDDATA), Write Debug (WDEBUG)

In the original M68000 ISA definition, lines A and F were effectively reserved for user-defined operations (line A) and co-processor instructions (line F). Accordingly, there are two unique exception vectors associated with illegal opwords in these two lines.

Any attempted execution of an illegal 16-bit opcode (except for line-A and line-F opcodes) generates an illegal instruction exception (vector 4). Additionally, any attempted execution of any non-MAC line-A and most line-F opcodes generate their unique exception types, vector numbers 10 and 11, respectively. ColdFire cores do not provide illegal instruction detection on the extension words on any instruction, including MOVEC.

### 3.3.4.4 Divide-By-Zero

Attempting to divide by zero causes an exception (vector 5, offset equal 0x014).

### 3.3.4.5 Privilege Violation

The attempted execution of a supervisor mode instruction while in user mode generates a privilege violation exception. See *ColdFire Programmer's Reference Manual* for a list of supervisor-mode instructions.

There is one special case involving the HALT instruction. Normally, this opcode is a supervisor mode instruction, but if the debug module's CSR[UHE] is set, then this instruction can be also be executed in user mode for debugging purposes.

### 3.3.4.6 Trace Exception

To aid in program development, all ColdFire processors provide an instruction-by-instruction tracing capability. While in trace mode, indicated by setting of the SR[T] bit, the completion of an instruction execution (for all but the stop instruction) signals a trace exception. This functionality allows a debugger to monitor program execution.

The stop instruction has the following effects:

1. The instruction before the stop executes and then generates a trace exception. In the exception stack frame, the PC points to the stop opcode.
2. When the trace handler is exited, the stop instruction executes, loading the SR with the immediate operand from the instruction.
3. The processor then generates a trace exception. The PC in the exception stack frame points to the instruction after the stop, and the SR reflects the value loaded in the previous step.

If the processor is not in trace mode and executes a stop instruction where the immediate operand sets SR[T], hardware loads the SR and generates a trace exception. The PC in the exception stack frame points to the instruction after the stop, and the SR reflects the value loaded in step 2.

Because ColdFire processors do not support any hardware stacking of multiple exceptions, it is the responsibility of the operating system to check for trace mode after processing other exception types. As an example, consider a TRAP instruction execution while in trace mode. The processor initiates the trap exception and then passes control to the corresponding handler. If the system requires that a trace exception be processed, it is the responsibility of the trap exception handler to check for this condition (SR[T] in the exception stack frame set) and pass control to the trace handler before returning from the original exception.

### **3.3.4.7 Unimplemented Line-A Opcode**

A line-A opcode is defined when bits 15-12 of the opword are 0b1010. This exception is generated by the attempted execution of an undefined line-A opcode.

### **3.3.4.8 Unimplemented Line-F Opcode**

A line-F opcode is defined when bits 15-12 of the opword are 0b1111. This exception is generated when attempting to execute an undefined line-F opcode.

### **3.3.4.9 Debug Interrupts**

See [Chapter 43, “Debug Module,”](#) for a detailed explanation of these exceptions, which are generated in response to hardware breakpoint register triggers. The processor does not generate an IACK cycle, but rather calculates the vector number internally (vector number 12 or 13, depending on the type of breakpoint trigger). Additionally, SR[M,I] are unaffected by the interrupt.

Separate exception vectors are provided for PC breakpoints and for address/data breakpoints. In the case of a two-level trigger, the last breakpoint determines the vector. There are two unique vectors for these exceptions: vector 0x030 corresponds to non-PC breakpoints and vector 0x034 corresponds to PC breakpoints.

### **3.3.4.10 RTE and Format Error Exception**

When an RTE instruction is executed, the processor first examines the 4-bit format field to validate the frame type. For a ColdFire core, any attempted RTE execution (where the format is not equal to {4,5,6,7}) generates a format error. The exception stack frame for the format error is created without disturbing the original RTE frame and the stacked PC pointing to the RTE instruction.

The selection of the format value provides some limited debug support for porting code from M68000 applications. On M68000 family processors, the SR was located at the top of the stack. On those processors, bit 30 of the longword addressed by the system stack pointer is typically zero. Thus, if an RTE is attempted using this old format, it generates a format error on a ColdFire processor.

If the format field defines a valid type, the processor: (1) reloads the SR operand, (2) fetches the second longword operand, (3) adjusts the stack pointer by adding the format value to the auto-incremented address

after the fetch of the first longword, and then (4) transfers control to the instruction address defined by the second longword operand within the stack frame.

### 3.3.4.11 TRAP Instruction Exception

The TRAP #n instruction always forces an exception as part of its execution and is useful for implementing system calls. The TRAP instruction may be used to change from user to supervisor mode.

### 3.3.4.12 Unsupported Instruction Exception

If execution of a valid instruction is attempted but the required hardware is not present in the processor, an unsupported instruction exception is generated. The instruction functionality can then be emulated in the exception handler, if desired.

All ColdFire cores record the processor hardware configuration in the D0 register immediately after the negation of  $\overline{\text{RESET}}$ . See [Section 3.3.4.15, “Reset Exception,”](#) for details.

### 3.3.4.13 Interrupt Exception

Interrupt exception processing includes interrupt recognition and the fetch of the appropriate vector from the interrupt controller using an IACK cycle. See [Chapter 17, “Interrupt Controller Modules,”](#) for details on the interrupt controller.

### 3.3.4.14 Fault-on-Fault Halt

If a ColdFire processor encounters any type of fault during the exception processing of another fault, the processor immediately halts execution with the catastrophic fault-on-fault condition. A reset is required to exit this state.

### 3.3.4.15 Reset Exception

Asserting the reset input signal ( $\overline{\text{RESET}}$ ) to the processor causes a reset exception. The reset exception has the highest priority of any exception; it provides for system initialization and recovery from catastrophic failure. Reset also aborts any processing in progress when the reset input is recognized. Processing cannot be recovered.

The reset exception places the processor in the supervisor mode by setting the SR[S] bit and disables tracing by clearing the SR[T] bit. This exception also clears the SR[M] bit and sets the processor’s SR[I] field to the highest level (level 7, 0b111). Next, the VBR is initialized to zero (0x0000\_0000). The control registers specifying the operation of any memories (e.g., cache and/or RAM modules) connected directly to the processor are disabled.

#### NOTE

Other implementation-specific registers are also affected. Refer to each module in this reference manual for details on these registers.

After the processor is granted the bus, it performs two longword read-bus cycles. The first longword at address 0x0000\_0000 is loaded into the supervisor stack pointer and the second longword at address

## ColdFire Core

0x0000\_0004 is loaded into the program counter. After the initial instruction is fetched from memory, program execution begins at the address in the PC. If an access error or address error occurs before the first instruction is executed, the processor enters the fault-on-fault state.

ColdFire processors load hardware configuration information into the D0 and D1 general-purpose registers after system reset. The hardware configuration information is loaded immediately after the reset-in signal is negated. This allows an emulator to read out the contents of these registers via the BDM to determine the hardware configuration.

Information loaded into D0 defines the processor hardware configuration as shown in [Figure 3-12](#).

BDM: Load: 0x080 (D0) Store: 0x180 (D0)																Access: User read-only BDM read-only							
<b>PF</b>																<b>VER</b>							
<b>REV</b>																							
R																W							
Reset																1 1 0 0 1 1 1 1 0 1 0 0 0 0 0 0							
R																W							
MAC DIV EMAC FPU MMU CAU 0 0 7 6 5 4 3 2 1 0																ISA DEBUG							
Reset																0 1 1 0 1 1 1 1 0 0 0 1 1 1 1							

**Figure 3-12. D0 Hardware Configuration Info**

**Table 3-9. D0 Hardware Configuration Info Field Description**

Field	Description
31–24 PF	Processor family. This field is fixed to a hex value of 0xCF indicating a ColdFire core is present.
23–20 VER	ColdFire core version number. Defines the hardware microarchitecture version of ColdFire core. 0001 V1 ColdFire core 0010 V2 ColdFire core 0011 V3 ColdFire core 0100 V4 ColdFire core (This is the value used for this device.) 0101 V5 ColdFire core Else Reserved for future use
19–16 REV	Processor revision number. The default is 0b0000.
15 MAC	MAC present. This bit signals if the optional multiply-accumulate (MAC) execution engine is present in processor core. 0 MAC execute engine not present in core. (This is the value used for this device.) 1 MAC execute engine is present in core.
14 DIV	Divide present. This bit signals if the hardware divider (DIV) is present in the processor core. 0 Divide execute engine not present in core. 1 Divide execute engine is present in core. (This is the value used for this device.)
13 EMAC	EMAC present. This bit signals if the optional enhanced multiply-accumulate (EMAC) execution engine is present in processor core. 0 EMAC execute engine not present in core. 1 EMAC execute engine is present in core. (This is the value used for this device.)

**Table 3-9. D0 Hardware Configuration Info Field Description (continued)**

Field	Description
12 FPU	FPU present. This bit signals if the optional floating-point (FPU) execution engine is present in processor core. 0 FPU execute engine not present in core. (This is the value used for this device.) 1 FPU execute engine is present in core.
11 MMU	MMU present. This bit signals if the optional virtual memory management unit (MMU) is present in processor core. 0 MMU execute engine not present in core. 1 MMU execute engine is present in core. (This is the value used for this device.)
10 CAU	Cryptographic acceleration unit present. This bit signals if the optional cryptographic acceleration unit (CAU) is present in the processor core. 0 CAU coprocessor engine not present in core. 1 CAU coprocessor engine is present in core. (This is the value used for this device.)
9–8	Reserved.
7–4 ISA	ISA revision. Defines the instruction-set architecture (ISA) revision level implemented in ColdFire processor core. 0000 ISA_A 0001 ISA_B 0010 ISA_C (This is the value used for this device.) 1000 ISA_A+ Else Reserved
3–0 DEBUG	Debug module revision number. Defines revision level of the debug module used in the ColdFire processor core. 0000 DEBUG_A 0001 DEBUG_B 0010 DEBUG_C 0011 DEBUG_D 0100 DEBUG_E 1001 DEBUG_B+ 1011 DEBUG_D+ 1111 DEBUG_D+PST Buffer (This is the value used for this device.) Else Reserved

Information loaded into D1 defines the local memory hardware configuration as shown in the figure below.

BDM: Load: 0x081 (D1)					Access: User read-only							
Store: 0x181 (D1)					BDM read-only							
31	30	29	28		27	26	25	24				
R	CLSZ		ICAS		ICSZ				0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			
W												
Reset	0	0	0	0	0	1	0	1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			
15	14	13	12		11	10	9	8				
R	MBSZ		CPES	DCAS	DCSZ				SRAMSZ			
W												
Reset	0	0	1	0	0	1	0	1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0			

**Figure 3-13. D1 Hardware Configuration Info**

**Table 3-10. D1 Hardware Configuration Information Field Description**

Field	Description
31–30 CLSZ	Cache line size. This field is fixed to a hex value of 0x0 indicating a 16-byte cache line size.
29–28 ICAS	Instruction cache associativity. 00 Four-way (This is the value used for this device) 01 Direct mapped Else Reserved for future use
27–24 ICSZ	Instruction cache size. Indicates the amount of instruction cache. 0000 No instruction cache 0001 512 B instruction cache 0010 1 KB instruction cache 0011 2 KB instruction cache 0100 4 KB instruction cache 0101 8 KB instruction cache (This is the value used for this device) 0110 16 KB instruction cache 0111 32 KB instruction cache 1000 64 KB instruction cache Else Reserved
23–16	Reserved.
15–14 MBSZ	Bus size. Defines the width of the ColdFire master bus datapath. 00 32-bit system bus datapath (This is the value used for this device) 01 64-bit system bus datapath Else Reserved
13 CPES	CPUSHL enhancements supported. Specifies whether the enhancements to the CPUSHL instructions are supported by the processor core. See <a href="#">Section 6.4.8, “CPUSHL Enhancements,”</a> for details. 0 CPUSHL instruction enhancements are not supported 1 CPUSHL instruction enhancements are supported (This is the value used for this device)
12 DCAS	Data cache associativity. Defines the data cache set-associativity. 0 Four-way (This is the value used for this device) 1 Direct mapped
11–8 DCSZ	Data cache size. Indicates the size of the unified cache. 0000 No data cache 0001 512 bytes 0010 1 KB 0011 2 KB 0100 4 KB 0101 8 KB (This is the value used for this device) 0110 16 KB 0111 32 KB Else Reserved for future use

**Table 3-10. D1 Hardware Configuration Information Field Description (continued)**

Field	Description
7-3 SRAMSZ	SRAM bank size. 00000 No SRAM 00010 512 bytes 00100 1 KB 00110 2 KB 01000 4 KB 01010 8 KB 01100 16 KB 01111 24 KB 01110 32 KB 10000 64 KB (This is the value used for this device) 10010 128 KB Else Reserved for future use
2-0	Reserved.

### 3.3.4.16 Precise Faults

To support a demand-paged virtual-memory environment, all memory references require precise, recoverable faults. The ColdFire instruction restart mechanism ensures that a faulted instruction restarts from the execution beginning. No internal state information is saved when an exception occurs nor is any restored when the handler ends. Given the PC address defined in the exception stack frame, the processor re-establishes program execution by transferring control to the given location as part of the RTE (return from exception) instruction.

The instruction restart recovery model requires program-visible register changes made during execution to be undone if that instruction subsequently faults.

The Version 4 (and later) ColdFire OEP structure naturally supports this concept for most instructions; program-visible registers are updated only in the final OEP stage when fault collection is complete. If any exception occurs, pending register updates are discarded.

For V4 ColdFire cores and later, most single-cycle instructions naturally support precise faults and instruction restart, while complex instruction do not. Consider the following memory-to-memory move:

```
move.1    (Ay)+, (Ax)+      # copy 4 bytes from source to destination
```

This instruction takes one cycle to read the source operand (Ay) and one to write the data into Ax. Source and destination address pointers are updated as part of execution. [Table 3-11](#) lists the operations performed in execute stage (EX).

**Table 3-11. OEP EX Cycle Operations**

EX Cycle	Operations
1	Read source operand from memory @ (Ay), update Ay, new Ay = old Ay + 4
2	Write operand into destination memory @ (Ax), update Ax, new Ax = old Ax + 4, update CCR

A fault detected with the destination memory write is reported during the second cycle. At this point, operations performed in the first cycle are complete, so if the destination write takes any type of access

error, Ay is updated. After the access error handler executes and the faulting instruction restarts, the processor's operation would be incorrect (without the special register recovery hardware) because the source-address register has an incorrect (post-incremented) value.

To recover the original state of the programming model for all instructions, the Version 4 ColdFire core adds the needed hardware to support full-register recovery. This hardware allows program-visible registers to be restored to their original state for multi-cycle instructions so that the instruction restart mechanism is supported. Memory-to-memory moves and move-multiple loads are representative of the complex instructions needing the special recovery support.

Recall the IFP and OEP are decoupled by a FIFO instruction buffer. In the V4 ColdFire IFP, each buffer entry includes 48 bits of instruction data fetched from memory and 64 bits of early decode and branch prediction information. This datapath also includes IFP fault-status information. Therefore, every IFP access can be tagged if an instruction fetch terminates with an error acknowledge. IFP access errors are recognized after the buffered instruction enters the OEP.

#### **NOTE**

For access errors signaled on instruction prefetches, an access error exception is generated only if instruction execution is attempted. If an instruction fetch access error exception is generated and the FS field indicates the fault occurred on an extension word, it may be necessary for the exception PC to be rounded-up to the next page address to determine the faulting instruction fetch address.

### **3.3.5 Instruction Execution Timing**

This section presents processor instruction execution times in terms of processor-core clock cycles. The number of operand references for each instruction is enclosed in parentheses following the number of processor clock cycles. Each timing entry is presented as C(R/W) where:

- C is the number of processor clock cycles, including all applicable operand fetches and writes, and all internal core cycles required to complete the instruction execution.
- R/W is the number of operand reads (R) and writes (W) required by the instruction. An operation performing a read-modify-write function is denoted as (1/1).

This section includes the assumptions concerning the timing values and the execution time details.

#### **3.3.5.1 Timing Assumptions**

For the timing data presented in this section, these assumptions apply:

1. The OEP is loaded with the opword and all required extension words at the beginning of each instruction execution. This implies that the OEP does not wait for the IFP to supply opwords and/or extension words.
2. Execution times for individual instructions make no assumptions concerning the OEP's ability to dispatch multiple instructions in one machine cycle. For sequences where instruction pairs are issued, the execution time of the first instruction defines the execution time of pair; the second instruction effectively executes in zero cycles.

3. The OEP does not experience any sequence-related pipeline stalls. The most common example of stall occurs when a register is modified in the EX engine and a subsequent instruction generates an address that uses the previously modified register. The second instruction stalls in the OEP until the previous instruction updates the register. For example, in the following code:

```
muls.l #<data>,d0
move.l (a0,d0.1*4),d1
```

the move.l instruction waits three cycles for the muls.l to update D0. If consecutive instructions update a register and use that register as a base of index value with a scale factor of 1 ( $X_i.1^*1$ ) in an address calculation, a 2-cycle pipeline stall occurs. If the destination register is used as an index register with any other scale factor ( $X_i.1^*2$ ,  $X_i.1^*4$ ), a 3-cycle stall occurs.

### NOTE

Address register results from post-increment and pre-decrement modes are available to subsequent instructions without stalls.

4. The OEP completes all memory accesses without any stall conditions caused by the memory itself. Thus, the timing details provided in this section assume that an infinite zero-wait state memory is attached to the processor core.
5. All operand data accesses are aligned on the same byte boundary as the operand size; for example, 16-bit operands aligned on 0-modulo-2 addresses, 32-bit operands aligned on 0-modulo-4 addresses.

The processor core decomposes misaligned operand references into a series of aligned accesses as shown in [Table 3-12](#).

**Table 3-12. Misaligned Operand References**

address[1:0]	Size	Bus Operations	Additional C(R/W)
01 or 11	Word	Byte, Byte	2(1/0) if read 1(0/1) if write
01 or 11	Long	Byte, Word, Byte	3(2/0) if read 2(0/2) if write
10	Long	Word, Word	2(1/0) if read 1(0/1) if write

### 3.3.5.2 MOVE Instruction Execution Times

[Table 3-13](#) lists execution times for MOVE.{B,W} instructions; [Table 3-14](#) lists timings for MOVE.L.

### NOTE

For all tables in this section, the execution time of any instruction using the PC-relative effective addressing modes is the same for the comparable An-relative mode.

ET with { $<ea> = (d16,PC)$ }  
 ET with { $<ea> = (d8,PC,Xi^*SF)$ }

equals ET with { $<ea> = (d16,An)$ }  
 equals ET with { $<ea> = (d8,An,Xi^*SF)$ }

The nomenclature xxx.wl refers to both forms of absolute addressing, xxx.w and xxx.l.

**Table 3-13. MOVE Byte and Word Execution Times**

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d16,Ax)	(d8,Ax,Xi*SF)	xxx.wl
Dy	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
Ay	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
(Ay)	1(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1))	2(1/1)
(Ay)+	1(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1))	2(1/1)
-(Ay)	1(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1))	2(1/1)
(d16,Ay)	1(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—
(d8,Ay,Xi*SF)	2(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
xxx.w	1(1/0)	2(1/1)	2(1/1)	2(1/1)	—	—	—
xxx.l	1(1/0)	2(1/1)	2(1/1)	2(1/1)	—	—	—
(d16,PC)	1(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—
(d8,PC,Xi*SF)	2(1/0)	3(1/1)	3(1/1))	3(1/1))	—	—	—
#xxx	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	—	—

**Table 3-14. MOVE Long Execution Times**

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d16,Ax)	(d8,Ax,Xi*SF)	xxx.wl
Dy	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
Ay	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
(Ay)	1(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
(Ay)+	1(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
-(Ay)	1(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
(d16,Ay)	1(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—
(d8,Ay,Xi*SF)	2(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
xxx.w	1(1/0)	2(1/1)	2(1/1)	2(1/1)	—	—	—
xxx.l	1(1/0)	2(1/1)	2(1/1)	2(1/1)	—	—	—
(d16,PC)	1(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—
(d8,PC,Xi*SF)	2(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
#xxx	1(0/0)	1(0/1)	1(0/1)	1(0/1)	—	—	—

### 3.3.5.3 Standard One Operand Instruction Execution Times

Table 3-15. One Operand Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
BITREV	Dx	1(0/0)	—	—	—	—	—	—	—
BYTEREV	Dx	1(0/0)	—	—	—	—	—	—	—
CLR.B	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
CLR.W	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
CLR.L	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
EXT.W	Dx	1(0/0)	—	—	—	—	—	—	—
EXT.L	Dx	1(0/0)	—	—	—	—	—	—	—
EXTB.L	Dx	1(0/0)	—	—	—	—	—	—	—
FF1	Dx	1(0/0)	—	—	—	—	—	—	—
NEG.L	Dx	1(0/0)	—	—	—	—	—	—	—
NEGX.L	Dx	1(0/0)	—	—	—	—	—	—	—
NOT.L	Dx	1(0/0)	—	—	—	—	—	—	—
SATS.L	Dx	1(0/0)	—	—	—	—	—	—	—
SCC	Dx	1(0/0)	—	—	—	—	—	—	—
SWAP	Dx	1(0/0)	—	—	—	—	—	—	—
TAS.B	<ea>	—	1(1/1)	1(1/1)	1(1/1)	1(1/1)	2(1/1)	1(1/1)	—
TST.B	<ea>	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)
TST.W	<ea>	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)
TST.L	<ea>	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)

### 3.3.5.4 Standard Two Operand Instruction Execution Times

Table 3-16. Two Operand Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xn*SF) (d8,PC,Xn*SF)	xxx.wl	#xxx
ADD.L	<ea>,Rx	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)
ADD.L	Dy,<ea>	—	1(1/1)	1(1/1)	1(1/1)	1(1/1)	2(1/1)	1(1/1)	—
ADDI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
ADDQ.L	#imm,<ea>	1(0/0)	1(1/1)	1(1/1)	1(1/1)	1(1/1)	2(1/1)	1(1/1)	—
ADDX.L	Dy,Dx	1(0/0)	—	—	—	—	—	—	—

**Table 3-16. Two Operand Instruction Execution Times (continued)**

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xn*SF) (d8,PC,Xn*SF)	xxx.wl	#xxx
AND.L	<ea>,Rx	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)
AND.L	Dy,<ea>	—	1(1/1)	1(1/1)	1(1/1)	1(1/1)	2(1/1)	1(1/1)	—
ANDI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
ASL.L	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
ASR.L	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
BCHG	Dy,<ea>	2(0/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)	—
BCHG	#imm,<ea>	2(0/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—	—
BCLR	Dy,<ea>	2(0/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)	—
BCLR	#imm,<ea>	2(0/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—	—
BSET	Dy,<ea>	2(0/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)	—
BSET	#imm,<ea>	2(0/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—	—
BTST	Dy,<ea>	2(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	—
BTST	#imm,<ea>	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	—	—	—
CMP.B	<ea>,Rx	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)
CMP.W	<ea>,Rx	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)
CMPL	<ea>,Rx	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)
CMPI.B	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
CMPI.W	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
CMPI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
DIVS.W	<ea>,Dx	20(0/0)	20(1/0)	20(1/0)	20(1/0)	20(1/0)	21(1/0)	20(1/0)	20(0/0)
DIVU.W	<ea>,Dx	20(0/0)	20(1/0)	20(1/0)	20(1/0)	20(1/0)	21(1/0)	20(1/0)	20(0/0)
DIVS.L	<ea>,Dx	≤35(0/0)	≤35(1/0)	≤35(1/0)	≤35(1/0)	≤35(1/0)	—	—	—
DIVU.L	<ea>,Dx	≤35(0/0)	≤35(1/0)	≤35(1/0)	≤35(1/0)	≤35(1/0)	—	—	—
EOR.L	Dy,<ea>	1(0/0)	1(1/1)	1(1/1)	1(1/1)	1(1/1)	2(1/1)	1(1/1)	—
EORI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
LEA	<ea>,Ax	—	1(0/0)	—	—	1(0/0)	2(0/0)	1(0/0)	—
LSL.L	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
LSR.L	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
MOVEQ.L	#imm,Dx	—	—	—	—	—	—	—	1(0/0)
OR.L	<ea>,Rx	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)
OR.L	Dy,<ea>	—	1(1/1)	1(1/1)	1(1/1)	1(1/1)	2(1/1)	1(1/1)	—
ORI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
REMS.L	<ea>,Dx	≤35(0/0)	≤35(1/0)	≤35(1/0)	≤35(1/0)	≤35(1/0)	—	—	—
REMUL	<ea>,Dx	≤35(0/0)	≤35(1/0)	≤35(1/0)	≤35(1/0)	≤35(1/0)	—	—	—
SUB.L	<ea>,Rx	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)

**Table 3-16. Two Operand Instruction Execution Times (continued)**

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xn*SF) (d8,PC,Xn*SF)	xxx.wl	#xxx
SUB.L	Dy,<ea>	—	1(1/1)	1(1/1)	1(1/1)	1(1/1)	2(1/1)	1(1/1)	—
SUBI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
SUBQ.L	#imm,<ea>	1(0/0)	1(1/1)	1(1/1)	1(1/1)	1(1/1)	2(1/1)	1(1/1)	—
SUBX.L	Dy,Dx	1(0/0)	—	—	—	—	—	—	—

### 3.3.5.5 Miscellaneous Instruction Execution Times

**Table 3-17. Miscellaneous Instruction Execution Times**

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
CPUSHL	(Ax)	—	9(0/1)	—	—	—	—	—	—
CPUSHL	bc,Ax	—	18(0/1)	—	—	—	—	—	—
CPUSHL	dc,Ax	—	12(0/1)	—	—	—	—	—	—
CPUSHL	ic,Ax	—	18(0/1)	—	—	—	—	—	—
INTOUCH	(Ay)	—	19(1/0)	—	—	—	—	—	—
LINK.W	Ay,#imm	2(0/1)	—	—	—	—	—	—	—
MOV3QL	#imm,<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
MOVE.L	Ay,USP	3(0/0)	—	—	—	—	—	—	—
MOVE.L	USP,Ax	3(0/0)	—	—	—	—	—	—	—
MOVE.W	CCR,Dx	1(0/0)	—	—	—	—	—	—	—
MOVE.W	<ea>,CCR	1(0/0)	—	—	—	—	—	—	1(0/0)
MOVE.W	SR,Dx	1(0/0)	—	—	—	—	—	—	—
MOVE.W	<ea>,SR	4(0/0)	—	—	—	—	—	—	4(0/0) <sup>2</sup>
MOVEC	Ry,Rc	20(0/1)	—	—	—	—	—	—	—
MOVEM.L	<ea>, and list	—	n(n/0)	—	—	n(n/0)	—	—	—
MOVEM.L	and list,<ea>	—	n(0/n)	—	—	n(0/n)	—	—	—
MVS	<ea>,Dx	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)
MVZ	<ea>,Dx	1(0/0)	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	1(0/0)
NOP		6(0/0)	—	—	—	—	—	—	—
PEA	<ea>	—	1(0/1)	—	—	1(0/1) <sup>4</sup>	2(0/1) <sup>5</sup>	1(0/1)	—
PULSE		1(0/0)	—	—	—	—	—	—	—
STOP	#imm	—	—	—	—	—	—	—	6(0/0) <sup>3</sup>
TRAP	#imm	—	—	—	—	—	—	—	18(1/2)
TPF		1(0/0)	—	—	—	—	—	—	—

**Table 3-17. Miscellaneous Instruction Execution Times (continued)**

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
TPF.W		1(0/0)	—	—	—	—	—	—	—
TPFL		1(0/0)	—	—	—	—	—	—	—
UNLK	Ax	1(1/0)	—	—	—	—	—	—	—
WDDATA	<ea>	—	1(1/0)	1(1/0)	1(1/0)	1(1/0)	2(1/0)	1(1/0)	—
WDEBUG	<ea>	—	3(2/0)	—	—	3(2/0)	—	—	—

<sup>1</sup>The n is the number of registers moved by the MOVEM opcode.<sup>2</sup>If a MOVE.W #imm,SR instruction is executed and imm[13] equals 1, the execution time is 1(0/0).<sup>3</sup>The execution time for STOP is the time required until the processor begins sampling continuously for interrupts.<sup>4</sup>PEA execution times are the same for (d16,PC).<sup>5</sup>PEA execution times are the same for (d8,PC,Xn\*SF).

### 3.3.5.6 EMAC Instruction Execution Times

**Table 3-18. EMAC Instruction Execution Times**

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An, Xn*SF)	xxx.wl	#xxx
MAC.L	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—
MAC.L	Ry, Rx, <ea>, Rw, Raccx	—	1(1/0)	1(1/0)	1(1/0)	1(1/0) <sup>1</sup>	—	—	—
MAC.W	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—
MAC.W	Ry, Rx, <ea>, Rw, Raccx	—	1(1/0)	1(1/0)	1(1/0)	1(1/0) <sup>1</sup>	—	—	—
MOVE.L	<ea>y, Raccx	1(0/0)	—	—	—	—	—	—	1(0/0)
MOVE.L	Raccy, Raccx	1(0/0)	—	—	—	—	—	—	—
MOVE.L	<ea>y, MACSR	8(0/0)	—	—	—	—	—	—	8(0/0)
MOVE.L	<ea>y, Rmask	7(0/0)	—	—	—	—	—	—	7(0/0)
MOVE.L	<ea>y, Raccext01	1(0/0)	—	—	—	—	—	—	1(0/0)
MOVE.L	<ea>y, Raccext23	1(0/0)	—	—	—	—	—	—	1(0/0)
MOVE.L	Raccx, <ea>x	1(0/0) <sup>2</sup>	—	—	—	—	—	—	—
MOVE.L	MACSR, <ea>x	1(0/0)	—	—	—	—	—	—	—
MOVE.L	Rmask, <ea>x	1(0/0)	—	—	—	—	—	—	—
MOVE.L	Raccext01,<ea>x	1(0/0)	—	—	—	—	—	—	—
MOVE.L	Raccext23,<ea>x	1(0/0)	—	—	—	—	—	—	—
MSAC.L	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—
MSAC.W	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—

**Table 3-18. EMAC Instruction Execution Times (continued)**

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An, Xn*SF)	xxx.wl	#xxx
MSAC.L	Ry, Rx, <ea>, Rw, Raccx	—	1(1/0)	1(1/0)	1(1/0)	1(1/0) <sup>1</sup>	—	—	—
MSAC.W	Ry, Rx, <ea>, Rw, Raccx	—	1(1/0)	1(1/0)	1(1/0)	1(1/0) <sup>1</sup>	—	—	—
MULS.L	<ea>y, Dx	4(0/0)	4(1/0)	4(1/0)	4(1/0)	4(1/0)	—	—	—
MULS.W	<ea>y, Dx	4(0/0)	4(1/0)	4(1/0)	4(1/0)	4(1/0)	5(1/0)	4(1/0)	4(0/0)
MULU.L	<ea>y, Dx	4(0/0)	4(1/0)	4(1/0)	4(1/0)	4(1/0)	—	—	—
MULU.W	<ea>y, Dx	4(0/0)	4(1/0)	4(1/0)	4(1/0)	4(1/0)	5(1/0)	4(1/0)	4(0/0)

<sup>1</sup> Effective address of (d16,PC) not supported<sup>2</sup> Storing an accumulator requires one additional processor clock cycle when saturation is enabled, or fractional rounding is performed (MACSR[7:4] equals 1---, -11-, --11)

## NOTE

The execution times for moving the contents of the Racc, Raccext[01,23], MACSR, or Rmask into a destination location <ea>x shown in this table represent the best-case scenario when the store instruction is executed and there are no load or M{S}AC instructions in the EMAC execution pipeline. In general, these store operations require only a single cycle for execution, but if preceded immediately by a load, MAC, or MSAC instruction, the depth of the EMAC pipeline is exposed and the execution time is four cycles.

### 3.3.5.7 Branch Instruction Execution Times

**Table 3-19. General Branch Instruction Execution Times**

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xi*SF) (d8,PC,Xi*SF)	xxx.wl	#xxx
BRA		—	—	—	—	1(0/1) <sup>1</sup>	—	—	—
BSR		—	—	—	—	1(0/1) <sup>2</sup>	—	—	—
JMP	<ea>	—	5(0/0)	—	—	5(0/0) <sup>1</sup>	6(0/0)	1(0/0) <sup>1</sup>	—
JSR	<ea>	—	5(0/1)	—	—	5(0/1)	6(0/1)	1(0/1) <sup>2</sup>	—
RTE		—	—	15(2/0)	—	—	—	—	—
RTS		—	—	2(1/0) <sup>3</sup> 9(1/0) <sup>3</sup> 8(1/0) <sup>3</sup>	—	—	—	—	—

**Table 3-20. Bcc Instruction Execution Times**

Opcode	Branch Cache Correctly Predicts Taken	Prediction Table Correctly Predicts Taken	Predicted Correctly as Not Taken	Predicted Incorrectly
Bcc	0(0/0)	1(0/0)	1(0/0)	8(0/0)

The following notes apply to the branch execution times:

1. For BRA and JMP <ea> instructions, where <ea> is (d16,PC) or xxx.wl, the branch acceleration logic of the IFP calculates the target address and begins prefetching the new path. Because the IFP and OEP are decoupled by the FIFO instruction buffer, the execution time can vary from one to three cycles, depending on the decoupling amount.

For all other <ea> values of the JMP instruction, the branch acceleration logic is not used, and the execution times are fixed.

2. For BSR and JSR xxx.wl opcodes, the same branch acceleration mechanism is used to initiate the fetch of the target instruction. Depending on the amount of decoupling between the IFP and OEP, the resulting execution times can vary from 1 to 3 cycles.

For the remaining <ea> values for the JSR instruction, the branch acceleration logic is not used, and the execution times are fixed.

3. For the RTS opcode, the timing depends on the prediction results of the hardware return stack:
  - a) If predicted correctly, 2(1/0).
  - b) If mispredicted, 9(1/0).
  - c) If not predicted, 8(1/0).

# **Chapter 4**

## **Memory Management Unit (MMU)**

### **4.1 Introduction**

This chapter describes the ColdFire virtual memory management unit (MMU), which provides virtual-to-physical address translation and memory access control. The MMU consists of memory-mapped control, status, and fault registers that provide access to translation-lookaside buffers (TLBs). Software can control address translation and access attributes of a virtual address by configuring MMU control registers and loading TLBs. With software support, the MMU provides demand-paged, virtual addressing.

#### **4.1.1 Block Diagram**

[Figure 4-1](#) shows the placement of the MMU/TLB hardware. It follows a traditional model closely coupled to the processor local-memory controllers.

## Memory Management Unit (MMU)

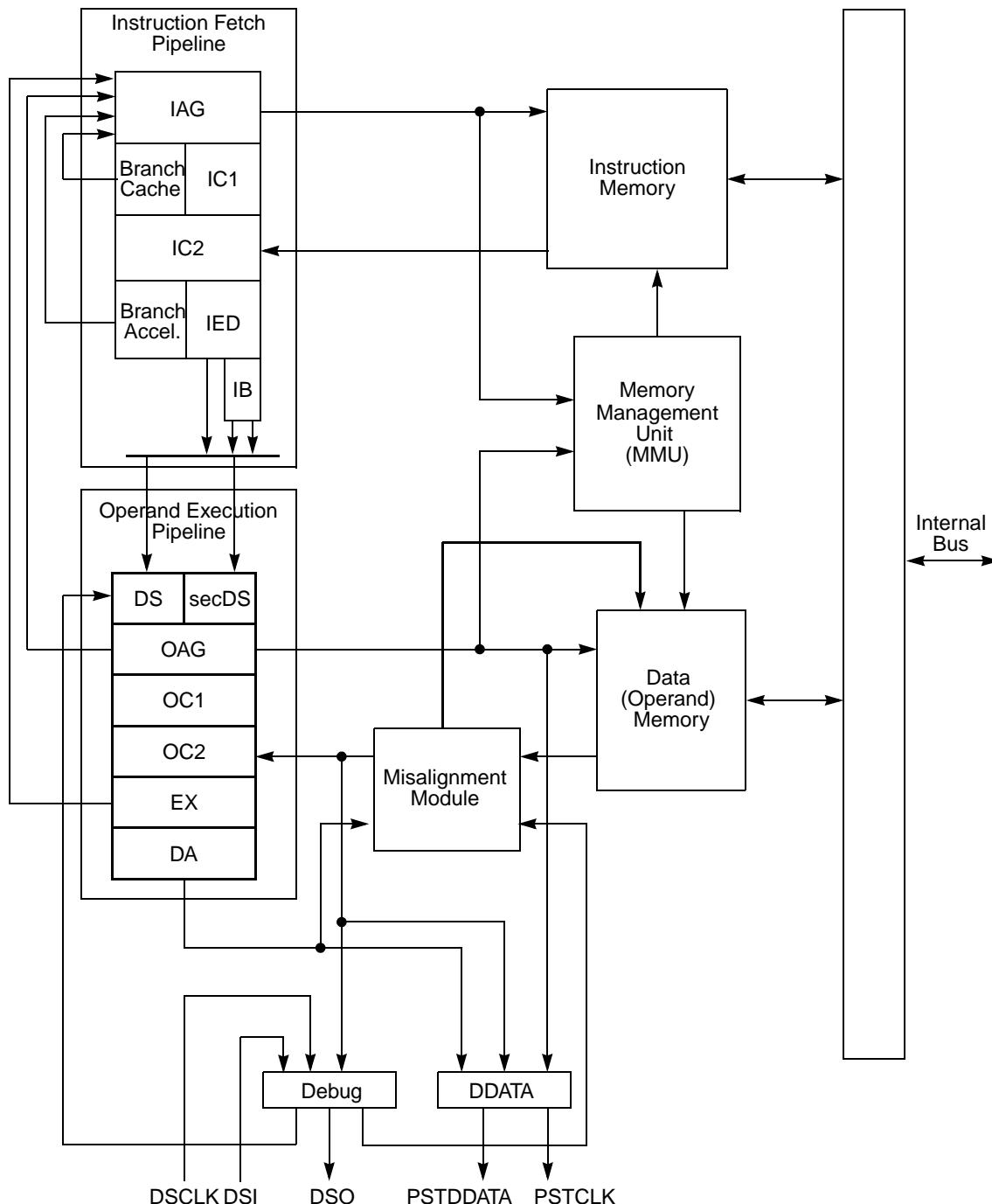


Figure 4-1. CF4 Processor Core Block with MMU

### 4.1.2 Features

The MMU has the following features:

- MMU memory-mapped control, status, and fault registers
  - Supports a flexible, software-defined virtual environment

- Provides control and maintenance of TLBs
- Provides fault status and recovery information functions
- Separate, 32-entry, fully associative instruction and data TLBs (Harvard TLBs)
  - Resides in the processor local bus-controller
  - Operates in parallel with internal memory
  - Suffers no performance penalty on TLB hits
  - Supports 4- and 8-Kbyte, and 1- and 16-Mbyte page sizes concurrently
  - Contains register-based TLB entries
- Core extensions:
  - User stack pointer
  - All access error exceptions are precise and recoverable
- Harvard TLB provides 97% of baseline performance on large embedded applications without MMU support

## 4.2 Memory Map/Register Definition

Access to the MMU memory-mapped region is controlled by MMUBAR, a 32-bit supervisor control register at 0x008 accessed using MOVEC or the serial BDM debug port. The *ColdFire Programmers Reference Manual* describes the MOVEC instruction.

MMUBAR holds the base address for the 64-Kbyte MMU memory map (Table 4-1). The MMU memory map area is not visible unless the MMUBAR is valid and must be referenced aligned. A large map portion is reserved for future use.

**Table 4-1. MMU Memory Map**

Address	Register	Width	Access	Reset Value	Section/Page
Rc[11:0] = 0x003 <sup>1</sup>	ASID—Address Space ID	8	R/W	0x00	<a href="#">4.2.1/4-4</a>
Rc[11:0] = 0x008 <sup>1</sup>	MMUBAR—MMU Base Address Register	32	R/W	0x0000_0000	<a href="#">4.2.2/4-4</a>
MMUBAR + 0x0000	MMUCR—MMU control register	32	R/W	0x0000_0000	<a href="#">4.2.3/4-5</a>
MMUBAR + 0x0004	MMUOR—MMU operation register	32	R/W	0x0000_0000	<a href="#">4.2.4/4-6</a>
MMUBAR + 0x0008	MMUSR—MMU status register	32	R/W	0x0000_0000	<a href="#">4.2.5/4-7</a>
MMUBAR + 0x0010	MMUAR—MMU fault, test, or TLB address register	32	R/W	0x0000_0000	<a href="#">4.2.6/4-8</a>
MMUBAR + 0x0014	MMUTR—MMU read/write TLB tag register	32	R/W	0x0000_0000	<a href="#">4.2.7/4-9</a>

**Table 4-1. MMU Memory Map (continued)**

Address	Register	Width	Access	Reset Value	Section/Page
MMUBAR + 0x0018	MMUDR—MMU read/write TLB data register	32	R/W	0x0000_0000	<a href="#">4.2.8/4-10</a>

<sup>1</sup> The address listed here represents the value of the Rc field used when accessing the core registers via the BDM port. For more information see Chapter 43, "Debug Module."

#### **4.2.1 Address Space ID (ASID)**

The address space ID (ASID) is located in a CPU space-control register. The 8-bit ASID value is mapped into CPU space at address 0x003 and is accessed using a MOVEC instruction. The *ColdFire Family Programmer's Reference Manual* describes MOVEC.

**Figure 4-2. Address Space ID (ASID)**

**Table 4-2. ASID Field Descriptions**

Field	Description
7–0 ID	This 8-bit field is the current user ASID. The ASID is an extension to the virtual address. Address space 0x00 may be reserved for supervisor mode. See address space mode functionality in <a href="#">Section 4.2.3, “MMU Control Register (MMUCR)</a> .“ The other 255 address spaces are used to tag user processes. The TLB entry ASID values are compared to this value for user mode unless the TLB entry is marked shared (MMUTR[SG] is set). The TLB entry ASID value may be compared to 0x00 for supervisor accesses.

#### 4.2.2 MMU Base Address Register (MMUBAR)

The default reset state is an invalid MMUBAR; The MMU is disabled and the memory-mapped space is not visible.

**Figure 4-3. MMU Base Address Register (MMUBAR)**

**Table 4-3. MMUBAR Field Descriptions**

Field	Description
31–16 BA	Base address. Defines the base address for the 64-Kbyte address space mapped to the MMU.
15–1	Reserved, must be cleared.
0 V	Valid. Indicates when MMUMBAR contents are valid. BA is not used unless V is set. 0 MMUBAR contents are not valid. 1 MMUBAR contents are valid.

### 4.2.3 MMU Control Register (MMUCR)

MMUCR contains the address space mode and virtual mode enable bits. The user must force pipeline synchronization after writing to this register. Therefore, all writes to this register must be immediately followed by a NOP instruction.

MMUBAR 0x000 (MMUCR)

Access: User read/write

Offset:

	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	UVE	SAL	ASM	EN		
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	

**Figure 4-4. MMU Control Register (MMUCR)****Table 4-4. MMUCR Field Descriptions**

Bits	Description
31–4	Reserved, must be cleared.
3 UVE	User virtual mode enable. Controls whether virtual mode is automatically enabled and disabled when entering or exiting user mode as defined by SR[S]. The virtual mode qualification also requires that MMUCR[EN] be set. 0 Virtual mode is based solely on the state of MMUCR[EN] 1 Virtual mode is based on MMUCR[EN] and SR[S] being cleared Virtual mode is defined by the boolean expression: MMUCR[EN] && (SR[S]    MMUCR[UVE])
2 SAL	Select ASID Location. Defines the source of the ASID value used for certain operations associated with the MMUOR, specifically those defined by MMUOR[STLB, CAS]. 0 Address space identifier is defined by ASID register contents 1 Address space identifier is defined by MMUAR[9:2]

**Table 4-4. MMUCR Field Descriptions (continued)**

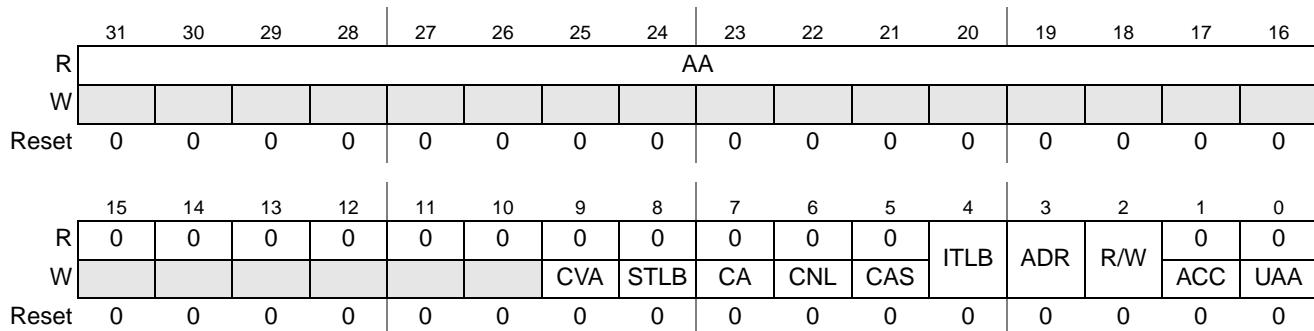
Bits	Description
1 ASM	Address space mode. Controls how the address space ID is used for TLB hits. 0 TLB entry ASID values are compared to the ASID register value for user or supervisor mode unless the TLB entry is marked shared (MMUTR[SG] = 1). The address space ID register value is the effective address space for all requests, supervisor and user. 1 Address space 0x00 is reserved for supervisor mode, and the effective address space is forced to 0x00 for all supervisor accesses. The other 255 address spaces are used to tag user processes. The TLB entry ASID values are compared to the ASID register for user mode unless the TLB entry is marked shared (SG = 1). The TLB entry ASID value is always compared to 0x00 for supervisor accesses. This allows two levels of sharing. All users, but not the supervisor, share an entry if SG is set and ASID does not equal 0. All users and the supervisor share an entry if SG is set and ASID equals 0
0 EN	Virtual mode enable. 0 Virtual mode is disabled 1 Virtual mode is enabled

#### 4.2.4 MMU Operation Register (MMUOR)

MMUBAR 0x004 (MMUOR)

Access: User read/write

Offset:

**Figure 4-5. MMU Operation Register (MMUOR)****Table 4-5. MMUOR Field Descriptions**

Field	Description
31–16 AA	TLB allocation address. This read-only field is maintained by MMU hardware. Its range and format depend on the TLB implementation (specific TLB size in entries, associativity, and organization). The access TLB function can use AA to read or write the addressed TLB entry. The MMU loads AA on the following three events: <ul style="list-style-type: none"><li>• On DTLB access errors, it loads the TLB entry address that caused the error.</li><li>• If MMUOR[UAA] is set, it loads the address of the TLB entry chosen by the MMU for replacement.</li><li>• If MMUOR[STLB] is set, it uses the data in MMUAR to search the TLB. If the TLB hits, it loads the address of the TLB entry that hits; if the TLB misses, it loads the TLB entry chosen by the MMU for replacement.</li></ul> The MMU never picks a locked entry for replacement, and TLB hits of locked entries do not update hardware replacement algorithm information. This is so access error handlers mapped with locked TLB entries do not influence the replacement algorithm. Further, TLB search operations do not update the hardware replacement algorithm information; TLB writes (loads) do update the hardware replacement algorithm information. The algorithm that chooses the allocation address depends on the TLB implementation (such as LRU, round-robin, pseudo-random).
15–10	Reserved, must be cleared.

**Table 4-5. MMUOR Field Descriptions (continued)**

Field	Description
9 CVA	Clears a common virtual address entry from the TLB. CVA always reads as a zero. 0 No operation 1 Clear all non-locked and non-shared TLB entries matching the virtual address contained in the MMUAR. No address space identifier is included in the comparisons. <b>Note:</b> The MMUOR[ADR, R/W] bits must also be set for correct operation.
8 STLB	Search TLB. STLB always reads as zero. 0 No operation 1 The MMU searches the TLB using the virtual address in MMUAR[31:10], address space identifier in MMUAR[9:2] if MMUCR[SAL] is set, else ASID[7:0], supervisor mode in MMUAR[0]. This operation updates the probe TLB hit bit in the status register plus loads the AA field as described above.
7 CA	Clear all TLB entries. CA always reads as zero. 0 No operation 1 Clear all TLB entries and all hardware TLB replacement algorithm information.
6 CNL	Clear all non-locked TLB entries. Setting CNL clears all TLB entries that do not have locked bits. CNL always reads as zero. 0 No operation 1 Clear all non-locked TLB entries
5 CAS	Clear all non-locked TLB entries that match ASID. If MMUCR[SAL] is set, then the address space identifier is defined by MMUAR[9:2]; Else, ASID[7:0] is used in the match. CAS always reads as a zero. 0 No operation 1 Clear all non-locked TLB entries that match ASID register
4 ITLB	ITLB operation. Used by TLB search and access operations that use the TLB allocation address. 0 MMU uses DTLB to search or update allocation address 1 MMU uses ITLB for of the allocation address searches and updates
3 ADR	TLB address select. Indicates which address to use when accessing the TLB. 0 Use the TLB allocation address for the TLB address 1 Use MMUAR for the TLB address
2 R/W	TLB access read/write select. Indicates whether to perform a read or a write when accessing the TLB. 0 Write 1 Read
1 ACC	MMU TLB access. This bit always reads as a zero. STLB is used for search operations. 0 No operation. ACC must be a zero to search the TLB. 1 The MMU reads or writes the TLB depending on R/W. For TLB reads, TLB tag and data results are loaded into MMUTR and MMUDR. For TLB writes, the contents of these registers are written to the TLB. The TLB is accessed using the TLB allocation address if ADR is zero or using MMUAR if ADR is set.
0 UAA	Update allocation address. UAA always reads as a zero. 0 No operation 1 MMU updates the allocation address field with the MMU's choice for the allocation address in the ITLB or DTLB depending on the ITLB instruction operation bit.

## 4.2.5 MMU Status Register (MMUSR)

MMUSR is updated on all data access faults and search TLB operations.

## Memory Management Unit (MMU)

MMUBAR 0x008 (MMUSR)

Access: User read/write

Offset:

	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SPF	RF	WF	0	HIT	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	

Figure 4-6. MMU Status Register (MMUSR)

Table 4-6. MMUSR Field Descriptions

Field	Description
31–6	Reserved, must be cleared.
5 SPF	Supervisor-protect fault. Indicates if last data fault was a user-mode access that hit in a TLB entry with its supervisor protect bit set. 0 Last data access fault did not have a supervisor protect fault 1 Last data access fault had a supervisor protect fault
4 RF	Read-access fault. Indicates if last data fault was a data-read access that hit in a TLB entry without its read bit set. 0 Last data access fault did not have a read protect fault 1 Last data access fault had a read protect fault
3 WF	Write-access fault. Indicates if the last data fault was a data-write access that hit in a TLB entry without its write bit set. 0 Last data access fault did not have a write protect fault 1 Last data access fault had a write protect fault
2	Reserved, must be cleared.
1 HIT	Search TLB hit. Indicates if last data fault or last search TLB operation hit in the TLB. 0 Last data access fault or search TLB operation did not hit in the TLB 1 Last data access fault or search TLB operation hit in the TLB
0	Reserved, must be cleared.

## 4.2.6 MMU Fault, Test, or TLB Address Register (MMUAR)

The MMUAR format depends on how register is used.

MMUBAR 0x010 (MMUAR)

Access: User read/write

Offset:

	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																																
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	

Figure 4-7. MMU Fault, Test, or TLB Address Register (MMUAR)

#### **Table 4-7. MMUAR Field Descriptions**

Field	Description
31–0 FA	<p>Form address.</p> <ul style="list-style-type: none"> <li>Written by the MMU with the virtual-address on DTLB misses and access faults. For this case, all 32 bits are address bits.</li> <li>This register may be written with a virtual address and (optionally) address space identifier information for searching the TLB (MMUOR[STLB]), globally clearing the TLB of a particular virtual address (MMUOR[CVA]), globally clearing a particular address space identifier (MMUOR[CAS]). For these cases, FA[31:10] is the virtual page number, FA[9:2] is the address space identifier if MMUCR[SAL] is set, and FA[0] is the supervisor bit.</li> <li>For MMUOR[CVA], virtual address = MMUAR[31:10], address space identifier = MMUAR[9:2] if MMUCR[SAL] is set, else ASID[7:0], supervisor mode = MMUAR[0]</li> <li>For MMUOR[STLB], virtual address = MMUAR[31:10], address space identifier = MMUAR[9:2] if MMUCR[SAL] is set, else ASID[7:0], supervisor mode = MMUAR[0]</li> <li>For MMUOR[CAS], address space identifier = MMUAR[9:2] if MMUCR[SAL] is set, else ASID[7:0]</li> <li>MMUAR can also be written with a TLB address for use with the access TLB function (using MMUCR[ACC]).</li> </ul>

#### **4.2.7 MMU Read/Write Tag Entry Registers (MMUTR)**

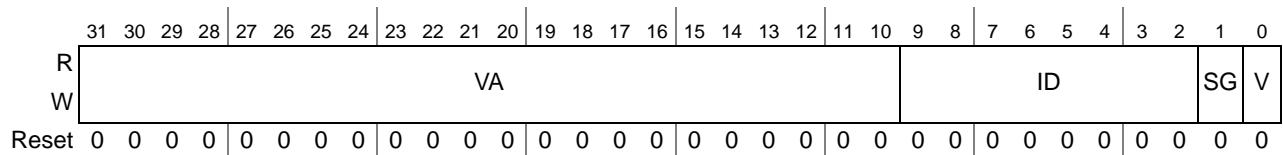
Each TLB entry consists of a 32-bit TLB tag entry and a 32-bit TLB data entry. TLB entries are referenced through MMUTR and MMUDR registers. For read TLB accesses, the contents of the TLB tag and data entries referenced by the allocation address or MMUAR are loaded in MMUTR and MMUDR. TLB write accesses place MMUTR and MMUDR contents into the TLB tag and data entries defined by the allocation address or MMUAR.

The MMUTR register contains the virtual address tag, the address space ID (ASID), a shared page indicator, and the valid bit.

## MMUBAR 0x014 (MMUTR)

## Access: User read/write

Offset:



**Figure 4-8. MMU Read/Write TLB Tag Register (MMUTR)**

**Table 4-8. MMUTR Field Descriptions**

Field	Description
31–10 VA	Virtual address. Defines the virtual address mapped by this entry. The number of bits used in TLB hit determination depends on the page-size field in the corresponding TLB data entry.
9–2 ID	Address space ID (ASID). This extension to the virtual address marks this entry as part of 1 of 256 possible address spaces. Address space 0x00 can be reserved for supervisor mode. The other 255 address spaces are used to tag user processes. TLB entry ASID values are compared to the ASID register value for user mode unless the TLB entry is marked shared (SG = 1). The TLB entry ASID value may be compared to 0x00 for supervisor accesses or to the ASID. The description of MMUCR[ASM] in <a href="#">Table 4-4</a> gives details on supervisor mode and ASID compares.

**Table 4-8. MMUTR Field Descriptions (continued)**

Field	Description
1 SG	Shared global. Indicates when the entry is shared among user address spaces. If an entry is shared, its ASID is not part of the TLB hit determination for user accesses. 0 This entry is not shared globally. 1 This entry is shared globally. <b>Note:</b> The ASID can determine supervisor mode hits to allow two sharing levels. If SG and MMUCR[ASM] are set and the ASID is not zero, all users (but not the supervisor) share an entry. If SG and MMUCR[ASM] are set and the ASID is zero, all users and the supervisor share an entry. The ASM description in <a href="#">Table 4-4</a> details supervisor mode and ASID compares.
0 V	Valid. Indicates when the entry is valid. Only valid entries generate a TLB hit. 0 Entry is not valid. 1 Entry is valid.

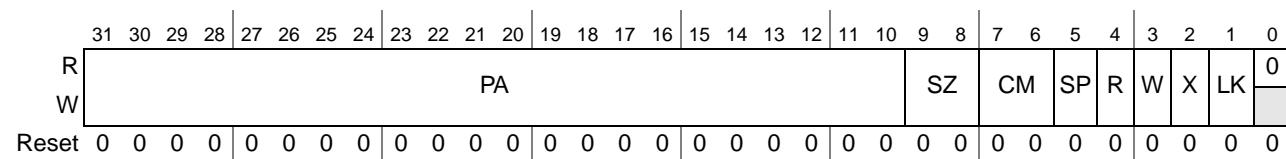
#### 4.2.8 MMU Read/Write Data Entry Register (MMUDR)

The MMUDR register contains the physical address, page size, cache-mode field, supervisor-protect bit, read, write, execute permission bits, and lock-entry bit.

MMUBAR 0x018 (MMUTR)

Access: User read/write

Offset:

**Figure 4-9. MMU Read/Write TLB Data Register (MMUDR)****Table 4-9. MMUDR Field Descriptions**

Field	Descriptions
31–10 PA	Physical address. Defines the physical address mapped by this entry. The number of bits used to build the effective physical address if this TLB entry hits depends on the page size field.
9–8 SZ	Page size. Page size for this entry: 00 1 Mbyte: VA[31–20] used for TLB hit 01 4 Kbytes: VA[31–12] used for TLB hit 10 8 Kbytes: VA[31–13] used for TLB hit 11 16 Mbytes: VA[31–24] used for TLB hit
7–6 CM	Cache mode. <b>Instruction cache modes:</b> 1x Page is non-cacheable. 0x Page is cacheable. <b>Data cache modes:</b> 00 Page is cacheable write-through. 01 Page is cacheable copy-back. 10 Page is non-cacheable precise. 11 Page is non-cacheable imprecise.

**Table 4-9. MMUDR Field Descriptions (continued)**

Field	Descriptions
5 SP	Supervisor protect. Controls user mode access to the page mapped by this entry. 0 Entry is not supervisor protected. 1 Entry is supervisor protected. An attempted user mode access that matches this entry generates an access error exception.
4 R	Read access enable. Indicates if data read accesses to this entry are allowed. If a Harvard TLB implementation is used, this bit is a don't care for the ITLB. This bit is ignored on writes and always reads as zero for the ITLB. 0 Do not allow data read accesses. Attempted data read accesses that match this entry generate an access error exception. 1 Allow data-read accesses.
3 W	Write access enable. Indicates if data write accesses are allowed to this entry. If separate ITLB and DTLBs are used, this bit is a don't care for the ITLB. This bit is ignored on writes and always reads as zero for the ITLB. 0 Do not allow data write accesses. Attempted data write accesses that match this entry generate an access error exception. 1 Allow data-write accesses.
2 X	Execute access enable. Indicates if instruction fetches to this entry are allowed. If separate ITLB and DTLBs are used, this bit is a don't care for the DTLB. This bit is ignored on writes and reads as zero for the DTLB. 0 Do not allow instruction fetches. Attempted instruction fetches that match this entry cause an access error exception. 1 Allow instruction-fetch accesses.
1 LK	Lock entry bit. Indicates if this entry is included in the replacement algorithm. TLB hits of locked entries do not update replacement algorithm information. 0 Include this entry when determining the best entry for a TLB allocation. 1 Do not allow this entry to be selected by the replacement algorithm.
0	Reserved, must be cleared.

## 4.3 Functional Description

The ColdFire MMU provides a virtual address, demand-paged memory architecture. The MMU supports hardware address translation acceleration using software-managed TLBs. It enforces permission checking on a per-memory request basis, and has control, status, and fault registers for MMU operation.

This device adds the following new features that were not present in previous devices containing an MMU:

- Support to clear all non-locked and non-shared TLB entries that match a virtual address.
- Ability to clear TLB entries based on an arbitrary address space identifier (ASID). Previously, operations to clear TLB entries based on the ASID only use the ASID register contents. This improvement adds the ability to clear any address space identifier, regardless of the contents of the ASID register.
- Optional capabilities to automatically enable and disable virtual mode based on entry (enable) and exit (disable) into user mode.
- Four additional access control registers (ACRs): two for instruction fetches and two for operand references. These registers are accessible via the privileged MOVEC instruction.

The default configuration is fully backward compatible with the previous MMU definition. The above enhancements are explicitly enabled by new control and configuration register fields.

### 4.3.1 Virtual Memory Management Architecture

The ColdFire memory management architecture provides a demand-paged, virtual-address environment with hardware address translation acceleration. It supports supervisor/user, read, write, and execute permission checking on a per-memory request basis.

The architecture defines the MMU TLB, associated control logic, TLB hit/miss logic, address translation based on the TLB contents, and access faults due to TLB misses and access violations. It intentionally leaves some virtual environment details undefined to maximize the software-defined flexibility. These include the exact structure of the memory-resident pointer descriptor/page descriptor tables, the base registers for these tables, the exact information stored in the tables, the methodology (if any) for access maintenance, and written information on a per-page basis.

#### 4.3.1.1 MMU Architecture Features

To add optional virtual-addressing support, demand-page support, permission checking, and hardware address translation acceleration to the ColdFire architecture, the MMU architecture features:

- Addresses from the core to the MMU are treated as physical or virtual addresses.
- The address access control logic, address attribute logic, internal memories, and internal to external memory bus controller function as in previous ColdFire versions with the addition of MMU.
- MMU, its TLB, and associated control reside in the processor local bus logic.
- MMU appears as a memory-mapped device in the processor local bus space. Information for access error fault processing is stored in MMU.
- A precise processor local-bus fault (transfer-error acknowledge) signals the core on translation (TLB miss) and access faults. The core supports an instruction restart model for this fault class. This structure uses the existing ColdFire access error fault vector and needs no new ColdFire exception stack frames.
- New ACR bits improve address granularity, supervisor mode protection, and memory functionality for physical and virtual address environments.

#### 4.3.1.2 MMU Architecture Implementation

This section describes ColdFire design additions and changes for the MMU architecture. It includes precise faults, MMU access, virtual mode, virtual memory references, instruction and data cache addresses, supervisor/user stack pointers, access error stack frame additions, expanded control register space, ACR address improvements, supervisor protection, and debugging in a virtual environment.

##### 4.3.1.2.1 Precise Faults

The MMU architecture performs virtual-to-physical address translation and permission checking in the core. To support demand-paging, the core design provides a precise, recoverable fault for all processor local bus references.

#### 4.3.1.2.2 MMU Access

The MMU TLB control registers are memory-mapped. The TLB entries are read and written indirectly through MMU control registers. Memory space for these resources is defined by a new supervisor program model register, the MMU base-address register (MMUBAR). This defines a supervisor-mode, data-only space. It has the highest priority for the data-processor local-bus address-mode determination.

#### 4.3.1.2.3 Virtual Mode

Every processor local-bus instruction and data reference is a virtual or physical address mode access. The following are always physical accesses:

- All addresses for special mode (interrupt acknowledges, emulator mode operations, etc.) accesses
- All addresses if the MMU is not enabled

If the MMU is enabled, the address mode for normal accesses is determined by the MMUBAR, RAMBARs, and ACRs in the priority order listed:

- Addresses that hit in these registers are treated as physical references. These addresses are not translated and their address attributes are sourced from the highest priority mapping register they hit.
- If an address hits none of these mapping registers, it is a virtual address and is sent to the MMU. If the MMU is enabled, the default CACR information is not used.

Virtual mode can be automatically enabled/disabled based on entry (enable) and exit (disable) into user mode. See the MMUCR[UVE] bit description, in [Section 4.2.3, “MMU Control Register \(MMUCR\)”](#) for more details.

#### 4.3.1.2.4 Virtual Memory References

The ColdFire MMU architecture references the MMU for all virtual mode accesses to the processor local bus. MMU, SRAM and ACR memory spaces are treated as physical address spaces and all permissions applying to these spaces are contained in the respective mapping register. The virtual mode access either hits or misses in the TLB of the MMU. A TLB miss generates an access fault in the processor, allowing software to either load the appropriate translation into the TLB and restart the faulting instruction or abort the process. Each TLB hit checks permissions based on the access control information in the referenced TLB entry.

#### 4.3.1.2.5 Instruction and Data Cache Addresses

For a given page size, virtual address bits that reference within a page are called the in-page address. All bits above this are the virtual page number. Likewise, the physical address has a physical page number and in-page address bits. Virtual and physical in-page address bits are the same; the MMU translates the virtual page number to the physical page number.

Instruction and data caches are accessed with the untranslated processor local-bus address. The translated address is used for cache allocation. That is, caches are virtual-address accessed and physical-address tagged. If instruction and data cache addresses are not larger than the in-page address for the smallest active MMU page, the cache is physically accessed; if they are larger, the cache can have aliasing problems between virtual and cache addresses. Software handles these problems by forcing the virtual address to be

equal to the physical address for those bits addressing the cache, but above the in-page address of the smallest active page size. The number of these bits depends on cache and page sizes.

Caches are addressed with the virtual address (the cache uses synchronous memory elements), and an access starts at the rising-clock edge of the first processor local bus pipeline stage. The MMU provides a physical address midway through this cycle.

If the cache-set address has fewer bits than the in-page address, the cache is considered physically addressed because these bits are the same in the virtual and physical addresses. If the cache set address has more bits than the in-page address, one or more of the low-order virtual page number bits are used to address the cache. The MMU translates these bits; the resulting low-order physical page number bits are used to determine cache hits.

Address aliasing problems occur when two virtual addresses access one physical page. This is generally allowed and, if the page is cacheable, one coherent copy of the page image is mapped in the cache at any time.

If multiple virtual addresses pointing to the same physical address differ only in the low-order virtual page number bits, conflicting copies can be allocated. For an 8-Kbyte, 4-way, set-associative cache with a 16-byte line size, the cache set address uses address bits 10–4. If virtual addresses 0x0\_1000 and 0x0\_1400 are mapped to physical address 0x0\_1000, using virtual address 0x0\_1000 loads cache set 0x00; using virtual address 0x0\_1400 loads cache set 0x40. This puts two copies of the same physical address in the cache, making this memory space not coherent. To avoid this problem, software must force low-order virtual page number bits to be equal to low-order physical address bits for all bits used to address the cache set.

#### **4.3.1.2.6 Supervisor/User Stack Pointers**

To isolate supervisor and user modes, the Version 4 ColdFire core implements two A7 register stack pointers: one for supervisor mode (SSP) and one for user mode (USP). Two former M68000 family-privileged instructions to load and store the user stack pointer are restored in the instruction set architecture.

#### **4.3.1.2.7 Access Error Stack Frame**

Processor local bus accesses that fault (that is, terminate with a transfer error acknowledge) to generate an access error exception. MMU TLB misses and access violations use the same fault. New fault status field (FS) encodings in the exception stack frame signal TLB misses on the following to quickly determine if a fault was due to a TLB miss or another type of access error:

- Instruction fetch
- Instruction extension fetch
- Data read
- Data write

See [Section 4.3.3.3, “Access Error Stack Frame Additions,”](#) for more information.

#### 4.3.1.2.8 Expanded Control Register Space

The MMU base-address register (MMUBAR) is added for ColdFire virtual mode. Like other control registers, it can be accessed from the debug module or written using the privileged MOVEC instruction. See [Section 4.2.2, “MMU Base Address Register \(MMUBAR\).”](#)

#### 4.3.1.2.9 Changes to ACRs and CACR

Four access control registers (ACRs) are added; two for instruction fetches and two for operand fetches. These registers are accessible via the privileged MOVEC instruction.

New ACR and CACR bits improve address granularity and supervisor mode protection and memory functionality for physical- and virtual- address environments.

**Table 4-10. New ACR and CACR Bits**

Field	Description
ACR $n[10]$ AMM	Address mask mode. Determines access to the associated address space. 0 The ACR hit function is the same as previous versions, allowing control of a 16-Mbyte or greater memory region. 1 The upper 8 bits of the address and ACR are compared without a mask function; bits 23–20 of the address and ACR are compared masked by ACR[19–16], allowing control of a 1- to 16-Mbyte region. Reset value is 0.
ACR $n[3]$ SP	Supervisor protect. Determines access to the associated address space. 0 Supervisor and user access allowed. 1 Only supervisor access allowed. Attempted user access causes an access error exception. Reset value is 0.
CACR[23] DDSP	Default data supervisor protect. Determines access to the associated data space. 0 Supervisor and user access allowed. 1 Only supervisor access allowed. Attempted user access causes an access error exception. Reset value is 0.
CACR[7] DISP	Default instruction supervisor protect. Determines access to the associated instruction space. 0 Supervisor and user access allowed. 1 Only supervisor access allowed. Attempted user access causes access error exception Reset value is 0.

#### 4.3.1.2.10 ACR Address Improvements

The ACR registers provide a 16-Mbyte address window. For a given request address, if the ACR is valid and the request mode matches the mode specified in the supervisor mode field (ACR $n[S]$ ), hit determination is specified as:

```
ACRx_Hit = 0;
if ((address[31:24] and ~ACRn[23:16]) == (ACRn[31:24] and ~ACRn[23:16]))
    ACRx_Hit = 1;
```

With this hit function, ACRs can assign address attributes for user or supervisor requests to memory spaces of at least 16 Mbytes (through the address mask). With the MMU definition, the ACR hit function is improved by the address mask mode bit (ACR $n[AMM]$ ), which supports finer address granularity. See [Table 4-10](#).

The revised hit determination becomes:

## Memory Management Unit (MMU)

```
ACRx_Hit = 0;
if (ACRn[10] == 1)
    if ((address[31-24] == ACRn[31-24])) &&
        ((address[23-20] and ~ACRn[19-16]) == (ACRn[23-20] and ~ACRn[19-16]))
            ACRx_Hit = 1;
else if (address[31-24] and ~ACRn[23-16]) == (ACRn[31-24] and ~ACRn[23-16])
    ACRx_Hit = 1;
```

### 4.3.1.2.11 Supervisor Protection

Each instruction or data reference is a supervisor or user access. The CPU's status register supervisor bit (SR[S]) determines the operating mode. New ACR and CACR bits protect supervisor space. See [Table 4-10](#).

## 4.3.2 Debugging in a Virtual Environment

To support debugging in a virtual environment, numerous enhancements are implemented in the ColdFire debug architecture. These enhancements are collectively called debug revision D and primarily relate to the addition of an 8-bit address space identifier (ASID) to yield a 40-bit virtual address. This expansion affects two major debug functions:

- The ASID is optionally included in the hardware breakpoint registers specification. For example, the four PC breakpoint registers are expanded by 8 bits each, so that a specific ASID value can be part of the breakpoint instruction address. Likewise, data address/data breakpoint registers are expanded to include an ASID value. The new control registers define if and how the ASID is included in the breakpoint comparison trigger logic.
- The debug module implements the concept of ownership trace in which an ASID value can be optionally displayed as part of real-time trace. When enabled, real-time trace displays instruction addresses on any change-of-flow instruction that is not absolute or PC-relative. For debug revision D architecture, the address display is expanded to include ASID contents optionally, thus providing the complete instruction virtual address on these instructions. Additionally, when a SYNC\_PC serial BDM command is loaded from the external development system, the processor displays the complete virtual-instruction address, including the 8-bit ASID value.

The MMU control registers are accessible through serial BDM commands. See [Chapter 43, “Debug Module.”](#)

## 4.3.3 Virtual Memory Architecture Processor Support

To support the MMU, enhancements have been made to the exception model, the stack pointers, and the access error stack frame.

### 4.3.3.1 Precise Faults

To support demand-paging, all memory references require precise, recoverable faults. The ColdFire instruction-restart mechanism ensures a faulted instruction restarts from the beginning of execution; in other words, no internal state information is saved when an exception occurs and none is restored when the handler ends. Given the PC address defined in the exception stack frame, the processor reestablishes

program execution by transferring control to the given location as part of the RTE (return from exception) instruction.

For a detailed description, see [Section 3.3.4.16, “Precise Faults.”](#)

### 4.3.3.2 Supervisor/User Stack Pointers

To provide the required isolation among these operating modes as dictated by a virtual memory management scheme, a user stack pointer (A7-USP) is added. The appropriate stack pointer register (SSP, USP) is accessed as a function of the processor’s operating mode.

In addition, the following two privileged M68000 family instructions to load/store the USP are added to the ColdFire instruction set architecture:

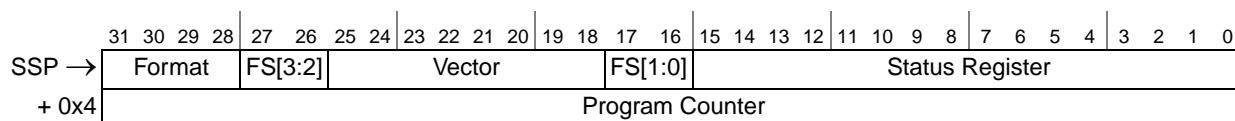
```
move.l Ay,USP    # move to USP: opcode = 0x4E6{0-7}
move.l USP,Ax    # move from USP: opcode = 0x4E6{8-F}
```

The address register number is encoded in the three low-order bits of the opcode.

These instructions are described in detail in [Section 4.3.9, “MMU Instructions.”](#)

### 4.3.3.3 Access Error Stack Frame Additions

ColdFire exceptions generate a standard 2-longword stack frame, signaling the contents of the SR and PC at the time of the exception, the exception type, and a 4-bit fault status field (FS). The first longword contains the 16-bit format/vector word (F/V) and the 16-bit status register. The second contains the 32-bit program counter address of the faulted instruction. For more information, see [Section 3.3.3.1, “Exception Stack Frame Definition.”](#)



**Figure 4-10. Exception Stack Frame Form**

The FS field is used for access and address errors. To optimize TLB miss-exception handling, new FS encodings (as shown in [Table 4-11](#)) allow quick error classification.

**Table 4-11. Fault Status Encodings**

FS[3:0]	Definition
0000	Not an access or address error
0001 – 0011	Reserved
0100	Error (for example, protection fault) on instruction fetch
0101	TLB miss on opword of instruction fetch (New for MMU)
0110	TLB miss on extension word of instruction fetch (New for MMU)
0111	IFP access error while executing in emulator mode (New for MMU)
1000	Error on data write
1001	Attempted write of protected space

**Table 4-11. Fault Status Encodings (continued)**

<b>FS[3:0]</b>	<b>Definition</b>
1010	TLB miss on data write (New for MMU)
1011	Reserved
1100	Error on data read
1101	Attempted read, read-modify-write of protected space (New for MMU)
1110	TLB miss on data read, or read-modify-write (New for MMU)
1111	OEP access error while executing in emulator mode (New for MMU)

#### 4.3.4 Effective Address Attribute Determination

The ColdFire core generates an effective memory address for all instruction fetches and data read and write memory accesses. The previous ColdFire memory access control model was based strictly on physical addresses. Every memory request address is a physical address analyzed by this memory access control logic and assigned address attributes, including:

- Cache mode
- SRAM enable information
- Write protect information
- Write mode information

These attributes control processing of the memory request. The address itself is not affected by memory access control logic.

Instruction and data references base effective address attributes and access mode on the instruction type and the effective address. There are two types of accesses:

- Special mode accesses, including interrupt acknowledges, reads/writes to program-visible control registers (CACR, RAMBARs, and ACRs), cache-control commands (CPUSHL and INTOUCH), and emulator-mode operations. These accesses have the following attributes:
  - Non-cacheable
  - Precise
  - No write protection
 Unless the CPU space/IACK mask bit is set, interrupt acknowledge cycles and emulator mode operations are allowed to hit in RAMBAR. All other operations are normal mode accesses.
- Normal mode accesses. For these accesses, an effective cache mode, precision, and write-protection are calculated for each request.

For data, a normal mode access address is compared with the following priority, from highest to lowest: RAMBAR, ACR0, ACR1, ACR4, and ACR5. If no match is found, default attributes in the CACR are used. The priority for instruction accesses is RAMBAR, ACR2, ACR3, ACR6, and ACR7. Again, if no match is found, default CACR attributes are used.

Only the test-and-set (TAS) instruction generates a normal mode access with implied cache mode and precision. TAS is a special, byte-sized, read-modify-write instruction used in synchronization routines. A TAS data access that does not hit in the RAMBAR is non-cacheable and precise. TAS uses the normal effective write protection.

If the MMU is enabled, it adds two factors for calculating effective address attributes:

- MMUBAR defines a memory-mapped, privileged data-only space with the highest priority in effective address attribute calculation for the data bus (that is, the MMUBAR has priority over RAMBAR).
- If virtual mode is enabled, any normal mode access that does not hit in the MMUBAR, RAMBAR or ACRs is considered a normal mode virtual address request and generates its access attributes from the MMU. For this case, the default CACR address attributes are not used.

The MMU also uses TLB contents to perform virtual-to-physical address translation.

### 4.3.5 MMU Functionality

The MMU provides virtual-to-physical address translation and memory access control. The MMU consists of memory-mapped, control, status, and fault registers, and a TLB that can be accessed through MMU registers. Supervisor software can access these resources through MMUBAR. Software can control address translation and virtual address access attributes by configuring MMU control registers and loading the MMU's TLB, which functions as a cache, associating virtual addresses to corresponding physical addresses and providing access attributes. Each TLB entry maps a virtual page. Several page sizes are supported. Features such as clear-all and probe-for-hit help maintain TLBs.

Fault-free, virtual address accesses that hit in the TLB incur no pipeline delay. Accesses that miss the TLB or hit the TLB but violate an access attribute generate an access-error exception. On an access error, software can reference address and information registers in the MMU to retrieve data. Depending on the fault source, software can obtain and load a new TLB entry, modify the attributes of an existing entry, or abort the faulting process.

### 4.3.6 MMU TLB

Each TLB entry consists of two 32-bit fields. The first is the TLB tag entry, and the second is the TLB data entry. TLB entries can be read and written through MMU registers. TLB contents are unaffected by reset.

### 4.3.7 MMU Operation

The processor sends instruction-fetch requests and data read/write requests to the MMU internal bus in the instruction- and operand-address generation cycles (IAG and OAG). The processor local bus controller and memories occupy the next two pipeline stages, instruction fetch cycles 1 and 2 (IC1 and IC2) and operand fetch cycles 1 and 2 (OC1 and OC2). For late writes, optional data pipeline stages are added to the processor local bus controller as well as any writable memories.

[Table 4-12](#) shows the association between internal memory pipeline stages and the processor's pipeline structures ([Figure 4-1](#)).

**Table 4-12. Version 4 Processor Local Bus Memory Pipelines**

Processor Local Bus Memory Pipeline Stage	Instruction Fetch Pipeline	Operand Execution Pipeline
J stage	IAG	OAG
KC1 stage	IC1	OC1
KC2 stage	IC2	OC2
Operand-execute stage	n/a	EX
Late-write stage	n/a	DA

Version 4 ColdFire processor local buses use the same 2-cycle read pipeline developed for Version 3 ColdFire cores. Each processor local bus has 32-bit address and 32-bit read data paths. Version 4 ColdFire cores use synchronous memory elements for all memory-control units. To support this, certain control information and all address bits are sent on the processor local buses at the end of the cycle before the initial bus access cycle (J cycle). The data processor local bus has an additional 32-bit write data path. For processor-store operations, Version 4 ColdFire uses a late-write strategy, which can require two additional data processor local bus cycles. This strategy yields the processor local bus pipeline behavior described in [Table 4-13](#).

**Table 4-13. Processor Local Bus Pipeline Cycles**

Cycle	Description
J	Control and partial address broadcast (to start synchronous memories)
KC1	Complete address and control broadcast plus MMU information. During this cycle, all memory element read operations are performed; memory arrays are accessed.
KC2	Select appropriate memory as source, return data to processor, handle cache misses or hold processor local bus pipeline as needed.
EX	Optional write stage, pipeline address and control for store operations.
DA	Data available for stores from processor; memory element update occurs in the next cycle.

The processor core contains two independent memory unit access controllers and two independent processor local bus controllers. Each instruction and data processor local bus request is analyzed to see which, if any, memory controller is referenced. This information, along with cache mode, store precision, and fault information, is sourced during KC1.

The MMU is referenced concurrently with the memory unit access controllers. It has two independent control sections to process simultaneously an instruction and data processor local bus request. [Figure 4-1](#) shows how the MMU and memory unit access controllers fit in the processor local bus pipeline. As the diagram shows, core address and attributes access the mapping registers and the MMU. By the middle of the KC1 cycle, the physical-memory address is available along with its corresponding access control.

[Figure 4-11](#) shows more details of the MMU structure. At the beginning of the KC1 pipeline stage, the TLB is accessed so the resulting physical address can be sourced to the cache controllers to factor into the cache hit/miss determination. This is required because caches are virtually indexed but physically mapped.

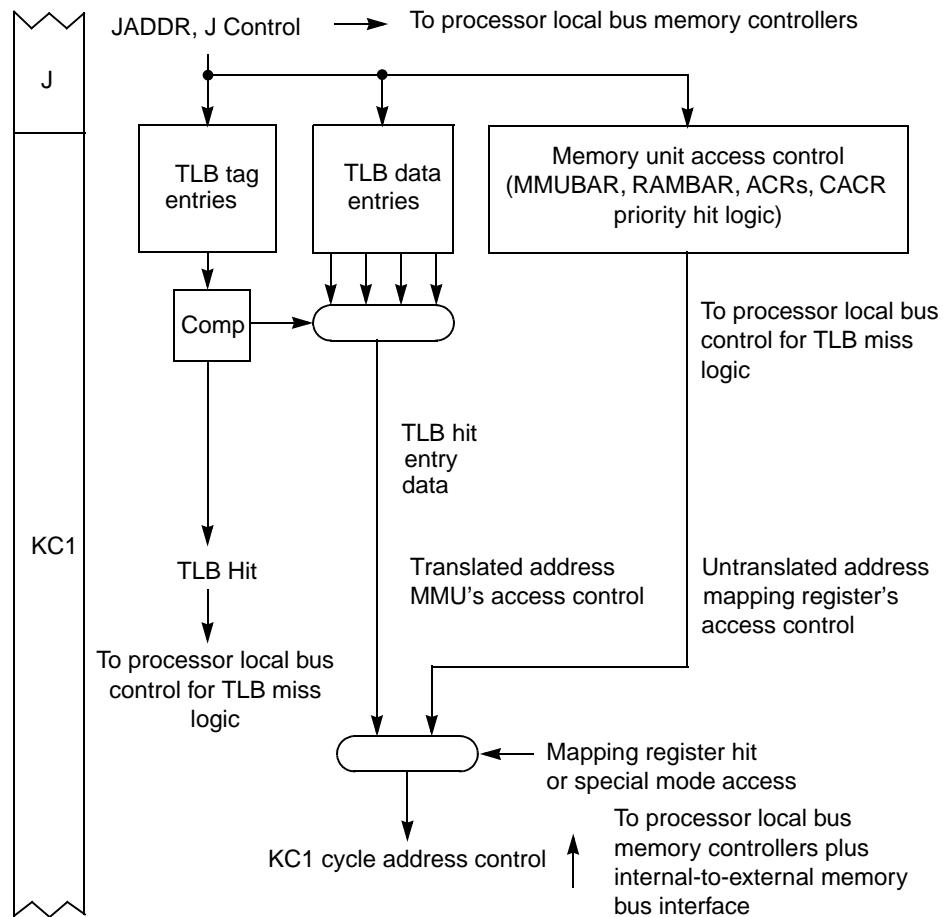


Figure 4-11. Processor Local Bus Address and Attributes Generation

## 4.3.8 MMU Implementation

The MMU implements a 64-entry full-associative Harvard TLB architecture with 32-entry ITLB and DTLB. This section details the operation and looks at the size, frequency, miss rate, and miss recovery time of this TLB implementation.

### 4.3.8.1 TLB Address Fields

Because the TLB has a total of 64 entries (32 each for the instruction and data TLBs), a 6-bit address field is necessary. TLB addresses 0–31 reference the ITLB; TLB addresses 32–63 reference the DTLB.

In the MMUOR register, bits 0–5 of the TLB allocation address (AA[5–0]) have this address format. The remaining TLB allocation address bits (AA[15–6]) are ignored on updates and always read as zero.

When the MMUAR register is used for a TLB address, bits FA[5–0] also have this address format. The remaining form address bits (FA[31–6]) are ignored when this register is used for a TLB address.

### 4.3.8.2 TLB Replacement Algorithm

The instruction and data TLBs provide low-latency access to recently used instruction and operand translation information. The ITLBs and DTLBs are 32-entry fully associative caches. The 32 ITLB entries are searched on each instruction reference; the 32 DTLB entries are searched on each operand reference.

The TLBs are software controlled. The TLB clear-all function clears valid bits on every TLB entry and resets the replacement logic. A new valid entry loaded in the TLBs may be designated as locked and unavailable for allocation. TLB hits to locked entries do not update replacement algorithm information.

When a new TLB entry needs to be allocated, the user can specify the exact TLB entry to be updated (through MMUOR[ADR] and MMUAR) or let TLB hardware pick the entry to update based on the replacement algorithm. A pseudo least recently used (PLRU) algorithm picks the entry to replace on a TLB miss. The algorithm works as follows:

- If any element is empty (non-valid), use the lowest empty element as the allocate entry (entry 0 before 1, 2, 3, and so on).
- If all entries are valid, use the entry indicated by the PLRU as the allocate entry.

The PLRU algorithm uses 31 most recently used state bits per TLB to track the TLB hit history. [Table 4-14](#) lists these state bits.

**Table 4-14. PLRU State Bits**

State Bits	Meaning
rdRecent31To16	A 1 indicates 31To16 is more recent than 15To00
rdRecent31To24	A 1 indicates 31To24 is more recent than 23To16
rdRecent15To08	A 1 indicates 15To08 is more recent than 07To00
rdRecent31To28	A 1 indicates 31To28 is more recent than 27To24
rdRecent23To20	A 1 indicates 23To20 is more recent than 19To16
rdRecent15To12	A 1 indicates 15To12 is more recent than 11To08
rdRecent07To04	A 1 indicates 07To04 is more recent than 03To00
rdRecent31To30	A 1 indicates 31To30 is more recent than 29To28
rdRecent27To26	A 1 indicates 27To26 is more recent than 25To24
rdRecent23To22	A 1 indicates 23To22 is more recent than 21To20
rdRecent19To18	A 1 indicates 19To18 is more recent than 17To16
rdRecent15To14	A 1 indicates 15To14 is more recent than 13To12
rdRecent11To10	A 1 indicates 11To10 is more recent than 09To08
rdRecent07To06	A 1 indicates 07To06 is more recent than 05To04
rdRecent03To02	A 1 indicates 03To02 is more recent than 01To00
rdRecent31	A 1 indicates 31 is more recent than 30
rdRecent29	A 1 indicates 29 is more recent than 28
rdRecent27	A 1 indicates 27 is more recent than 26

**Table 4-14. PLRU State Bits (continued)**

<b>State Bits</b>	<b>Meaning</b>
rdRecent25	A 1 indicates 25 is more recent than 24
rdRecent23	A 1 indicates 23 is more recent than 22
rdRecent21	A 1 indicates 21 is more recent than 20
rdRecent19	A 1 indicates 19 is more recent than 18
rdRecent17	A 1 indicates 17 is more recent than 16
rdRecent15	A 1 indicates 15 is more recent than 14
rdRecent13	A 1 indicates 13 is more recent than 12
rdRecent11	A 1 indicates 11 is more recent than 10
rdRecent09	A 1 indicates 09 is more recent than 08
rdRecent07	A 1 indicates 07 is more recent than 06
rdRecent05	A 1 indicates 05 is more recent than 04
rdRecent03	A 1 indicates 03 is more recent than 02
rdRecent01	A 1 indicates 01 is more recent than 00

Binary state bits are updated on all TLB write (load) operations, as well as normal non-locked entries ITLB and DTLB hits. Also, if all entries in a binary state are locked, then that state is always set. That is, if entries 15, 14, 13, and 12 were locked, LRU state bit rdRecent15To14 is forced to 1.

For a completely valid TLB, binary state information determines the LRU entry. The replacement algorithm is deterministic and, for the case of a full TLB (with no locked entries and always touching new pages), the replacement entry repeats every 32 TLB loads.

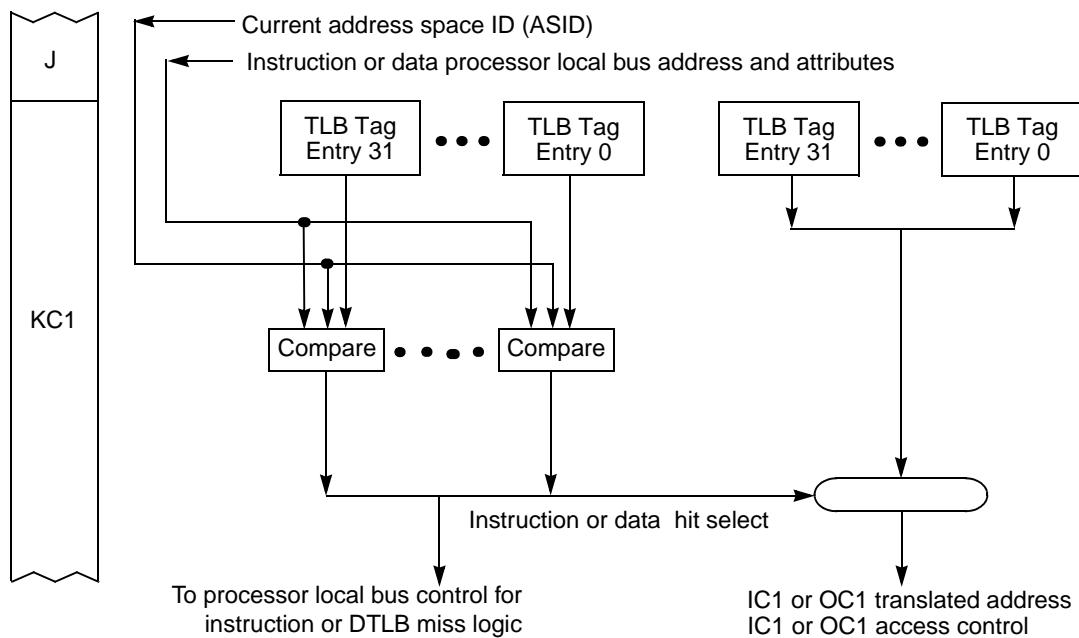
#### 4.3.8.3 TLB Locked Entries

[Figure 4-12](#) is a ColdFire MMU Harvard TLB block diagram.

For TLB miss faults, the instruction restart model re-executes an instruction on returning from the exception handler. An instruction can touch two instruction pages (a 32- or 48-bit instruction can straddle two pages) or four data pages (a memory-to-memory word or longword move where misaligned source and destination operands straddle two pages). Therefore, one instruction may take two ITLB misses and allocate two ITLB pages before completion. Likewise, one instruction may require four DTLB misses and allocate four DTLB pages. Because of this, a pool of unlocked TLB entries must be available if virtual memory is used.

The above examples show the fewest entries needed to guarantee an instruction can complete execution. For good MMU performance, more unlocked TLB entries should be available.

## Memory Management Unit (MMU)



**Figure 4-12. Version 4 ColdFire MMU Harvard TLB**

### 4.3.9 MMU Instructions

The MOVE to USP and MOVE from USP instructions are added for accessing the USP. Refer to the *ColdFire Programmer's Reference Manual* for more information.

# Chapter 5

## Enhanced Multiply-Accumulate Unit (EMAC)

### 5.1 Introduction

This chapter describes the functionality, microarchitecture, and performance of the enhanced multiply-accumulate (EMAC) unit in the ColdFire family of processors.

#### 5.1.1 Overview

The EMAC design provides a set of DSP operations that can improve the performance of embedded code while supporting the integer multiply instructions of the baseline ColdFire architecture.

The MAC provides functionality in three related areas:

1. Signed and unsigned integer multiplication
2. Multiply-accumulate operations supporting signed and unsigned integer operands as well as signed, fixed-point, and fractional operands
3. Miscellaneous register operations

The ColdFire family supports two MAC implementations with different performance levels and capabilities. The original MAC features a three-stage execution pipeline optimized for 16-bit operands, with a  $16 \times 16$  multiply array and a single 32-bit accumulator. The EMAC features a four-stage pipeline optimized for 32-bit operands, with a fully pipelined  $32 \times 32$  multiply array and four 48-bit accumulators.

The first ColdFire MAC supported signed and unsigned integer operands and was optimized for  $16 \times 16$  operations, such as those found in applications including servo control and image compression. As ColdFire-based systems proliferated, the desire for more precision on input operands increased. The result was an improved ColdFire MAC with user-programmable control to optionally enable use of fractional input operands.

EMAC improvements target three primary areas:

- Improved performance of  $32 \times 32$  multiply operation.
- Addition of three more accumulators to minimize MAC pipeline stalls caused by exchanges between the accumulator and the pipeline's general-purpose registers
- A 48-bit accumulation data path to allow a 40-bit product, plus 8 extension bits increase the dynamic number range when implementing signal processing algorithms

The three areas of functionality are addressed in detail in following sections. The logic required to support this functionality is contained in a MAC module ([Figure 5-1](#)).

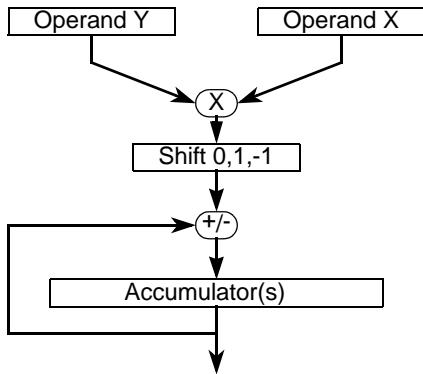


Figure 5-1. Multiply-Accumulate Functionality Diagram

### 5.1.1.1 Introduction to the MAC

The MAC is an extension of the basic multiplier in most microprocessors. It is typically implemented in hardware within an architecture and supports rapid execution of signal processing algorithms in fewer cycles than comparable non-MAC architectures. For example, small digital filters can tolerate some variance in an algorithm's execution time, but larger, more complicated algorithms such as orthogonal transforms may have more demanding speed requirements beyond scope of any processor architecture and may require full DSP implementation.

To balance speed, size, and functionality, the ColdFire MAC is optimized for a small set of operations that involve multiplication and cumulative additions. Specifically, the multiplier array is optimized for single-cycle pipelined operations with a possible accumulation after product generation. This functionality is common in many signal processing applications. The ColdFire core architecture is also modified to allow an operand to be fetched in parallel with a multiply, increasing overall performance for certain DSP operations.

Consider a typical filtering operation where the filter is defined as in [Equation 5-1](#).

$$y(i) = \sum_{k=1}^{N-1} a(k)y(i-k) + \sum_{k=0}^{N-1} b(k)x(i-k) \quad \text{Eqn. 5-1}$$

Here, the output  $y(i)$  is determined by past output values and past input values. This is the general form of an infinite impulse response (IIR) filter. A finite impulse response (FIR) filter can be obtained by setting coefficients  $a(k)$  to zero. In either case, the operations involved in computing such a filter are multiplies and product summing. To show this point, reduce [Equation 5-1](#) to a simple, four-tap FIR filter, shown in [Equation 5-2](#), in which the accumulated sum is a past data values and coefficients sum.

$$y(i) = \sum_{k=0}^3 b(k)x(i-k) = b(0)x(i) + b(1)x(i-1) + b(2)x(i-2) + b(3)x(i-3) \quad \text{Eqn. 5-2}$$

3

## 5.2 Memory Map/Register Definition

The following table and sections explain the MAC registers:

**Table 5-1. EMAC Memory Map**

BDM <sup>1</sup>	Register	Width (bits)	Access	Reset Value	Section/Page
0x804	MAC Status Register (MACSR)	32	R/W	0x0000_0000	<a href="#">5.2.1/5-3</a>
0x805	MAC Address Mask Register (MASK)	32	R/W	0xFFFF_FFFF	<a href="#">5.2.2/5-5</a>
0x806	MAC Accumulator 0 (ACC0)	32	R/W	Undefined	<a href="#">5.2.3/5-7</a>
0x807	MAC Accumulator 0,1 Extension Bytes (ACCext01)	32	R/W	Undefined	<a href="#">5.2.4/5-7</a>
0x808	MAC Accumulator 2,3 Extension Bytes (ACCext23)	32	R/W	Undefined	<a href="#">5.2.4/5-7</a>
0x809	MAC Accumulator 1 (ACC1)	32	R/W	Undefined	<a href="#">5.2.3/5-7</a>
0x80A	MAC Accumulator 2 (ACC2)	32	R/W	Undefined	<a href="#">5.2.3/5-7</a>
0x80B	MAC Accumulator 3 (ACC3)	32	R/W	Undefined	<a href="#">5.2.3/5-7</a>

<sup>1</sup> The values listed in this column represent the Rc field used when accessing the core registers via the BDM port. For more information see [Chapter 43, “Debug Module.”](#)

### 5.2.1 MAC Status Register (MACSR)

The MAC status register (MACSR) contains a 4-bit operational mode field and condition flags.

Operational mode bits control whether operands are signed or unsigned and whether they are treated as integers or fractions. These bits also control the overflow/saturation mode and the way in which rounding is performed. Negative, zero, and multiple overflow condition flags are also provided.

BDM: 0x804 (MACSR)																Access: Supervisor read/write BDM 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	
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PAV <sub>n</sub>	OMC	S/U	F/I	R/T
W	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	N	Z	V	EV		
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

**Figure 5-2. MAC Status Register (MACSR)**

**Table 5-2. MACSR Field Descriptions**

Field	Description
31–12	Reserved, must be cleared.
11–8 PAV $n$	Product/accumulation overflow flags. Contains four flags, one per accumulator, that indicate if past MAC or MSAC instructions generated an overflow during product calculation or the 48-bit accumulation. When a MAC or MSAC instruction is executed, the PAV $n$ flag associated with the destination accumulator forms the general overflow flag, MACSR[V]. Once set, each flag remains set until V is cleared by a <code>move .1 , MACSR</code> instruction or the accumulator is loaded directly. Bit 11: Accumulator 3 ... Bit 8: Accumulator 0
7 OMC	Overflow saturation mode. Enables or disables saturation mode on overflow. If set, the accumulator is set to the appropriate constant (see S/U field description) on any operation that overflows the accumulator. After saturation, the accumulator remains unaffected by any other MAC or MSAC instructions until the overflow bit is cleared or the accumulator is directly loaded.
6 S/U	Signed/unsigned operations. <b>In integer mode:</b> S/U determines whether operations performed are signed or unsigned. It also determines the accumulator value during saturation, if enabled. 0 Signed numbers. On overflow, if OMC is enabled, an accumulator saturates to the most positive (0x7FFF_FFFF) or the most negative (0x8000_0000) number, depending on the instruction and the product value that overflowed. 1 Unsigned numbers. On overflow, if OMC is enabled, an accumulator saturates to the smallest value (0x0000_0000) or the largest value (0xFFFF_FFFF), depending on the instruction. <b>In fractional mode:</b> S/U controls rounding while storing an accumulator to a general-purpose register. 0 Move accumulator without rounding to a 16-bit value. Accumulator is moved to a general-purpose register as a 32-bit value. 1 The accumulator is rounded to a 16-bit value using the round-to-nearest (even) method when moved to a general-purpose register. See <a href="#">Section 5.3.1.1, "Rounding"</a> . The resulting 16-bit value is stored in the lower word of the destination register. The upper word is zero-filled. This rounding procedure does not affect the accumulator value.
5 F/I	Fractional/integer mode. Determines whether input operands are treated as fractions or integers. 0 Integers can be represented in signed or unsigned notation, depending on the value of S/U. 1 Fractions are represented in signed, fixed-point, two's complement notation. Values range from -1 to $1 - 2^{-15}$ for 16-bit fractions and -1 to $1 - 2^{-31}$ for 32-bit fractions. See <a href="#">Section 5.3.4, "Data Representation."</a>
4 R/T	Round/truncate mode. Controls rounding procedure for <code>move .1 ACCx , Rx</code> , or <code>MSAC.L</code> instructions when in fractional mode. 0 Truncate. The product's lsbs are dropped before it is combined with the accumulator. Additionally, when a store accumulator instruction is executed ( <code>move .1 ACCx , Rx</code> ), the 8 lsbs of the 48-bit accumulator logic are truncated. 1 Round-to-nearest (even). The 64-bit product of two 32-bit, fractional operands is rounded to the nearest 40-bit value. If the low-order 24 bits equal 0x80_0000, the upper 40 bits are rounded to the nearest even (lsb = 0) value. See <a href="#">Section 5.3.1.1, "Rounding"</a> . Additionally, when a store accumulator instruction is executed ( <code>move .1 ACCx , Rx</code> ), the lsbs of the 48-bit accumulator logic round the resulting 16- or 32-bit value. If MACSR[S/U] is cleared and MACSR[R/T] is set, the low-order 8 bits are used to round the resulting 32-bit fraction. If MACSR[S/U] is set, the low-order 24 bits are used to round the resulting 16-bit fraction.

**Table 5-2. MACSR Field Descriptions (continued)**

Field	Description
3 N	Negative. Set if the msb of the result is set, otherwise cleared. N is affected only by MAC, MSAC, and load operations; it is not affected by MULS and MULU instructions.
2 Z	Zero. Set if the result equals zero, otherwise cleared. This bit is affected only by MAC, MSAC, and load operations; it is not affected by MULS and MULU instructions.
1 V	Overflow. Set if an arithmetic overflow occurs on a MAC or MSAC instruction, indicating that the result cannot be represented in the limited width of the EMAC. V is set only if a product overflow occurs or the accumulation overflows the 48-bit structure. V is evaluated on each MAC or MSAC operation and uses the appropriate PAV/n flag in the next-state V evaluation.
0 EV	Extension overflow. Signals that the last MAC or MSAC instruction overflowed the 32 lsbs in integer mode or the 40 lsbs in fractional mode of the destination accumulator. However, the result remains accurately represented in the combined 48-bit accumulator structure. Although an overflow has occurred, the correct result, sign, and magnitude are contained in the 48-bit accumulator. Subsequent MAC or MSAC operations may return the accumulator to a valid 32/40-bit result.

Table 5-3 summarizes the interaction of the MACSR[S/U,F/I,R/T] control bits.

**Table 5-3. Summary of S/U, F/I, and R/T Control Bits**

S/U	F/I	R/T	Operational Modes
0	0	x	Signed, integer
0	1	0	Signed, fractional Truncate on MAC.L and MSAC.L No round on accumulator stores
0	1	1	Signed, fractional Round on MAC.L and MSAC.L Round-to-32-bits on accumulator stores
1	0	x	Unsigned, integer
1	1	0	Signed, fractional Truncate on MAC.L and MSAC.L Round-to-16-bits on accumulator stores
1	1	1	Signed, fractional Round on MAC.L and MSAC.L Round-to-16-bits on accumulator stores

## 5.2.2 Mask Register (MASK)

The 32-bit MASK implements the low-order 16 bits to minimize the alignment complications involved with loading and storing only 16 bits. When the MASK is loaded, the low-order 16 bits of the source operand are actually loaded into the register. When it is stored, the upper 16 bits are all forced to ones.

This register performs a simple AND with the operand address for MAC instructions. The processor calculates the normal operand address and, if enabled, that address is then ANDed with {0xFFFF, MASK[15:0]} to form the final address. Therefore, with certain MASK bits cleared, the operand address

## Enhanced Multiply-Accumulate Unit (EMAC)

can be constrained to a certain memory region. This is used primarily to implement circular queues with the (An)+ addressing mode.

This minimizes the addressing support required for filtering, convolution, or any routine that implements a data array as a circular queue. For MAC + MOVE operations, the MASK contents can optionally be included in all memory effective address calculations. The syntax is as follows:

```
mac.sz Ry,RxSF,<ea>y&,Rw
```

The & operator enables the MASK use and causes bit 5 of the extension word to be set. The exact algorithm for the use of MASK is:

```
if extension word, bit [5] = 1, the MASK bit, then
    if <ea> = (An)
        oa = An & {0xFFFF, MASK}

    if <ea> = (An) +
        oa = An
        An = (An + 4) & {0xFFFF, MASK}

    if <ea> =-(An)
        oa = (An - 4) & {0xFFFF, MASK}
        An = (An - 4) & {0xFFFF, MASK}

    if <ea> = (d16,An)
        oa = (An + se_d16) & {0xFFFF0x, MASK}
```

Here, *oa* is the calculated operand address and *se\_d16* is a sign-extended 16-bit displacement. For auto-addressing modes of post-increment and pre-decrement, the updated An value calculation is also shown.

Use of the post-increment addressing mode, {(An)+} with the MASK is suggested for circular queue implementations.

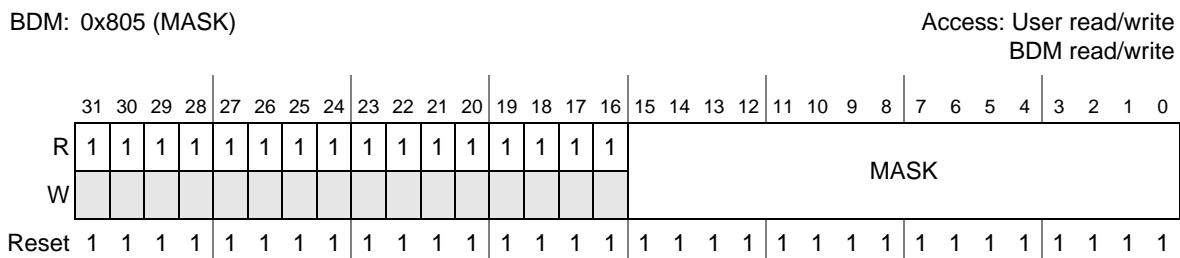


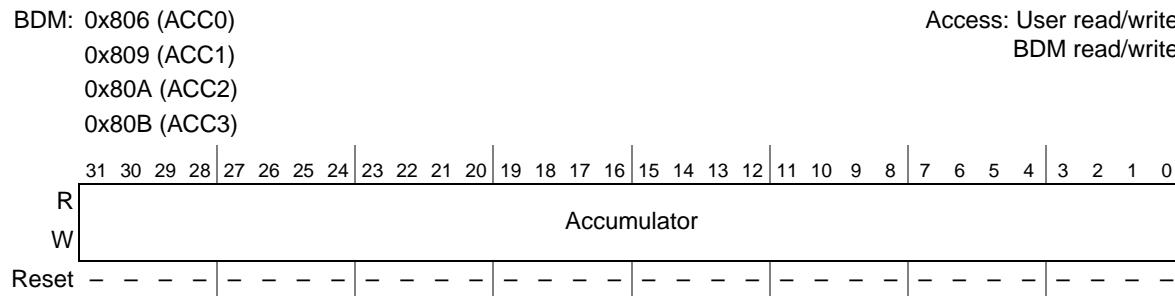
Figure 5-3. Mask Register (MASK)

Table 5-4. MASK Field Descriptions

Field	Description
31–16	Reserved, must be set.
15–0 MASK	Performs a simple AND with the operand address for MAC instructions.

### 5.2.3 Accumulator Registers (ACC0–3)

The accumulator registers store 32-bits of the MAC operation result. The accumulator extension registers form the entire 48-bit result.



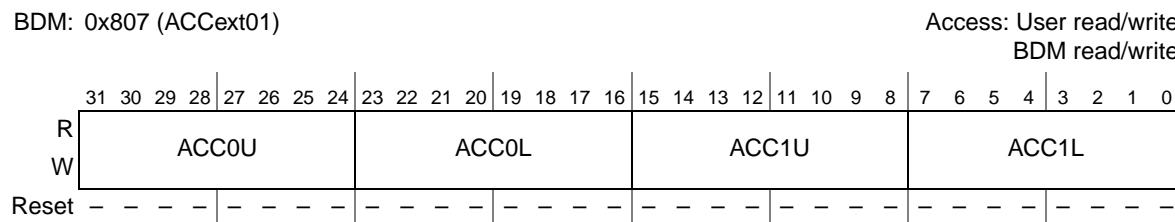
**Figure 5-4. Accumulator Registers (ACC0–3)**

**Table 5-5. ACC0–3 Field Descriptions**

Field	Description
31–0 Accumulator	Store 32-bits of the result of the MAC operation.

### 5.2.4 Accumulator Extension Registers (ACCext01, ACCext23)

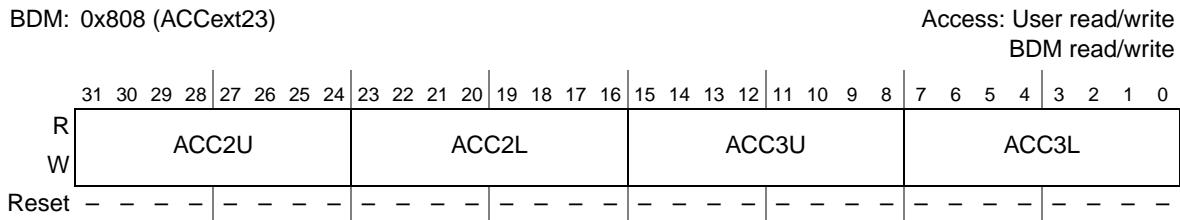
Each pair of 8-bit accumulator extension fields are concatenated with the corresponding 32-bit accumulator register to form the 48-bit accumulator. For more information, see [Section 5.3, “Functional Description.”](#)



**Figure 5-5. Accumulator Extension Register (ACCext01)**

**Table 5-6. ACCext01 Field Descriptions**

Field	Description
31–24 ACC0U	Accumulator 0 upper extension byte
23–16 ACC0L	Accumulator 0 lower extension byte
15–8 ACC1U	Accumulator 1 upper extension byte
7–0 ACC1L	Accumulator 1 lower extension byte

**Figure 5-6. Accumulator Extension Register (ACCext23)****Table 5-7. ACCext23 Field Descriptions**

Field	Description
31–24 ACC2U	Accumulator 2 upper extension byte
23–16 ACC2L	Accumulator 2 lower extension byte
15–8 ACC3U	Accumulator 3 upper extension byte
7–0 ACC3L	Accumulator 3 lower extension byte

### 5.3 Functional Description

The MAC speeds execution of ColdFire integer-multiply instructions (MULS and MULU) and provides additional functionality for multiply-accumulate operations. By executing MULS and MULU in the MAC, execution times are minimized and deterministic compared to the 2-bit/cycle algorithm with early termination that the OEP normally uses if no MAC hardware is present.

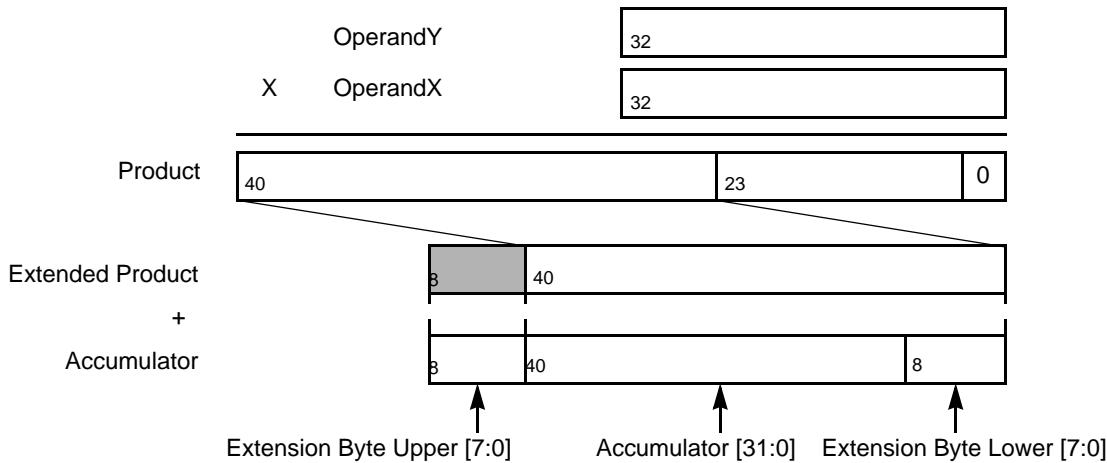
The added MAC instructions to the ColdFire ISA provide for the multiplication of two numbers, followed by the addition or subtraction of the product to or from the value in an accumulator. Optionally, the product may be shifted left or right by 1 bit before addition or subtraction. Hardware support for saturation arithmetic can be enabled to minimize software overhead when dealing with potential overflow conditions. Multiply-accumulate operations support 16- or 32-bit input operands in these formats:

- Signed integers
- Unsigned integers
- Signed, fixed-point, fractional numbers

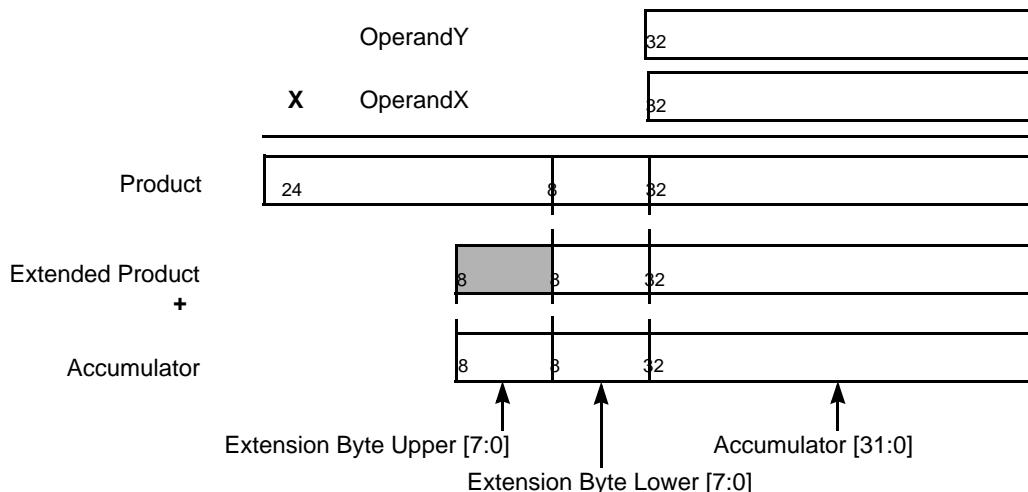
The EMAC is optimized for single-cycle, pipelined  $32 \times 32$  multiplications. For word- and longword-sized integer input operands, the low-order 40 bits of the product are formed and used with the destination accumulator. For fractional operands, the entire 64-bit product is calculated and truncated or rounded to the most-significant 40-bit result using the round-to-nearest (even) method before it is combined with the destination accumulator.

For all operations, the resulting 40-bit product is extended to a 48-bit value (using sign-extension for signed integer and fractional operands, zero-fill for unsigned integer operands) before being combined with the 48-bit destination accumulator.

Figure 5-7 and Figure 5-8 show relative alignment of input operands, the full 64-bit product, the resulting 40-bit product used for accumulation, and 48-bit accumulator formats.



**Figure 5-7. Fractional Alignment**



**Figure 5-8. Signed and Unsigned Integer Alignment**

Therefore, the 48-bit accumulator definition is a function of the EMAC operating mode. Given that each 48-bit accumulator is the concatenation of 16-bit accumulator extension register (ACCextn) contents and 32-bit ACCn contents, the specific definitions are:

```

if MACSR[6:5] == 00      /* signed integer mode */
    Complete Accumulator[47:0] = {ACCextn[15:0], ACCn[31:0]}
if MACSR[6:5] == 01 or 11 /* signed fractional mode */
    Complete Accumulator [47:0] = {ACCextn[15:8], ACCn[31:0], ACCextn[7:0]}
if MACSR[6:5] == 10      /* unsigned integer mode */
    Complete Accumulator[47:0] = {ACCextn[15:0], ACCn[31:0]}

```

The four accumulators are represented as an array, ACCn, where n selects the register.

Although the multiplier array is implemented in a four-stage pipeline, all arithmetic MAC instructions have an effective issue rate of 1 cycle, regardless of input operand size or type.

All arithmetic operations use register-based input operands, and summed values are stored in an accumulator. Therefore, an additional MOVE instruction is needed to store data in a general-purpose register. One new feature in EMAC instructions is the ability to choose the upper or lower word of a register as a 16-bit input operand. This is useful in filtering operations if one data register is loaded with the input data and another is loaded with the coefficient. Two 16-bit multiply accumulates can be performed without fetching additional operands between instructions by alternating word choice during calculations.

The EMAC has four accumulator registers versus the MAC's single accumulator. The additional registers improve the performance of some algorithms by minimizing pipeline stalls needed to store an accumulator value back to general-purpose registers. Many algorithms require multiple calculations on a given data set. By applying different accumulators to these calculations, it is often possible to store one accumulator without any stalls while performing operations involving a different destination accumulator.

The need to move large amounts of data presents an obstacle to obtaining high throughput rates in DSP engines. Existing ColdFire instructions can accommodate these requirements. A MOVEM instruction can efficiently move large data blocks by generating line-sized burst references. The ability to load an operand simultaneously from memory into a register and execute a MAC instruction makes some DSP operations such as filtering and convolution more manageable.

The programming model includes a mask register (MASK), which can optionally be used to generate an operand address during MAC + MOVE instructions. The register application with auto-increment addressing mode supports efficient implementation of circular data queues for memory operands.

### 5.3.1 Fractional Operation Mode

This section describes behavior when the fractional mode is used (MACSR[F/I] is set).

#### 5.3.1.1 Rounding

When the processor is in fractional mode, there are two operations during which rounding can occur:

- Execution of a store accumulator instruction (`move.1 ACCx, Rx`). The lsbs of the 48-bit accumulator logic are used to round the resulting 16- or 32-bit value. If MACSR[S/U] is cleared, the low-order 8 bits round the resulting 32-bit fraction. If MACSR[S/U] is set, the low-order 24 bits are used to round the resulting 16-bit fraction.
- Execution of a MAC (or MSAC) instruction with 32-bit operands. If MACSR[R/T] is zero, multiplying two 32-bit numbers creates a 64-bit product truncated to the upper 40 bits; otherwise, it is rounded using round-to-nearest (even) method.

To understand the round-to-nearest-even method, consider the following example involving the rounding of a 32-bit number, R0, to a 16-bit number. Using this method, the 32-bit number is rounded to the closest 16-bit number possible. Let the high-order 16 bits of R0 be named R0.U and the low-order 16 bits be R0.L.

- If R0.L is less than 0x8000, the result is truncated to the value of R0.U.
- If R0.L is greater than 0x8000, the upper word is incremented (rounded up).

- If R0.L is 0x8000, R0 is half-way between two 16-bit numbers. In this case, rounding is based on the lsb of R0.U, so the result is always even (lsb = 0).
  - If the lsb of R0.U equals 1 and R0.L equals 0x8000, the number is rounded up.
  - If the lsb of R0.U equals 0 and R0.L equals 0x8000, the number is rounded down.

This method minimizes rounding bias and creates as statistically correct an answer as possible.

The rounding algorithm is summarized in the following pseudocode:

```
if R0.L < 0x8000
    then Result = R0.U
else if R0.L > 0x8000
    then Result = R0.U + 1
else if lsb of R0.U = 0           /* R0.L = 0x8000 */
    then Result = R0.U
else Result = R0.U + 1
```

The round-to-nearest-even technique is also known as convergent rounding.

### 5.3.1.2 Saving and Restoring the EMAC Programming Model

The presence of rounding logic in the EMAC output datapath requires special care during the EMAC's save/restore process. In particular, any result rounding modes must be disabled during the save/restore process so the exact bit-wise contents of the EMAC registers are accessed. Consider the memory structure containing the EMAC programming model:

```
struct macState {
    int acc0;
    int acc1;
    int acc2;
    int acc3;
    int accext01;
    int accext02;
    int mask;
    int macsr;
} macState;
```

The following assembly language routine shows the proper sequence for a correct EMAC state save. This code assumes all Dn and An registers are available for use, and the memory location of the state save is defined by A7.

```
EMAC_state_save:
    move.l  macsr,d7          ; save the macsr
    clr.l   d0                ; zero the register to ...
    move.l  d0,macsr          ; disable rounding in the macsr
    move.l  acc0,d0            ; save the accumulators
    move.l  acc1,d1
    move.l  acc2,d2
    move.l  acc3,d3
    move.l  accext01,d4        ; save the accumulator extensions
    move.l  accext23,d5
    move.l  mask,d6            ; save the address mask
    movem.l #0x00ff,(a7)       ; move the state to memory
```

This code performs the EMAC state restore:

```
EMAC_state_restore:
```

## Enhanced Multiply-Accumulate Unit (EMAC)

```
movem.1 (a7),#0x00ff      ; restore the state from memory
move.1 #0,macsr           ; disable rounding in the macsr
move.1 d0,acc0             ; restore the accumulators
move.1 d1,acc1
move.1 d2,acc2
move.1 d3,acc3
move.1 d4,accext01        ; restore the accumulator extensions
move.1 d5,accext23
move.1 d6,mask              ; restore the address mask
move.1 d7,macsr            ; restore the macsr
```

Executing this sequence type can correctly save and restore the exact state of the EMAC programming model.

### 5.3.1.3 MULS/MULU

MULS and MULU are unaffected by fractional-mode operation; operands remain assumed to be integers.

### 5.3.1.4 Scale Factor in MAC or MSAC Instructions

The scale factor is ignored while the MAC is in fractional mode.

## 5.3.2 EMAC Instruction Set Summary

Table 5-8 summarizes EMAC unit instructions.

Table 5-8. EMAC Instruction Summary

Command	Mnemonic	Description
Multiply Signed	muls <ea>y,Dx	Multiplies two signed operands yielding a signed result
Multiply Unsigned	mulu <ea>y,Dx	Multiplies two unsigned operands yielding an unsigned result
Multiply Accumulate	mac Ry,RxSF,ACCx msac Ry,RxSF,ACCx	Multiplies two operands and adds/subtracts the product to/from an accumulator
Multiply Accumulate with Load	mac Ry,Rx,<ea>y,Rw,ACCx msac Ry,Rx,<ea>y,Rw,ACCx	Multiplies two operands and combines the product to an accumulator while loading a register with the memory operand
Load Accumulator	move.1 {Ry,#imm},ACCx	Loads an accumulator with a 32-bit operand
Store Accumulator	move.1 ACCx,Rx	Writes the contents of an accumulator to a CPU register
Copy Accumulator	move.1 ACCy,ACCx	Copies a 48-bit accumulator
Load MACSR	move.1 {Ry,#imm},MACSR	Writes a value to MACSR
Store MACSR	move.1 MACSR,Rx	Write the contents of MACSR to a CPU register
Store MACSR to CCR	move.1 MACSR,CCR	Write the contents of MACSR to the CCR
Load MAC Mask Reg	move.1 {Ry,#imm},MASK	Writes a value to the MASK register
Store MAC Mask Reg	move.1 MASK,Rx	Writes the contents of the MASK to a CPU register
Load Accumulator Extensions 01	move.1 {Ry,#imm},ACCext01	Loads the accumulator 0,1 extension bytes with a 32-bit operand

**Table 5-8. EMAC Instruction Summary (continued)**

Command	Mnemonic	Description
Load Accumulator Extensions 23	move.l {Ry,#imm},ACCext23	Loads the accumulator 2,3 extension bytes with a 32-bit operand
Store Accumulator Extensions 01	move.l ACCext01,Rx	Writes the contents of accumulator 0,1 extension bytes into a CPU register
Store Accumulator Extensions 23	move.l ACCext23,Rx	Writes the contents of accumulator 2,3 extension bytes into a CPU register

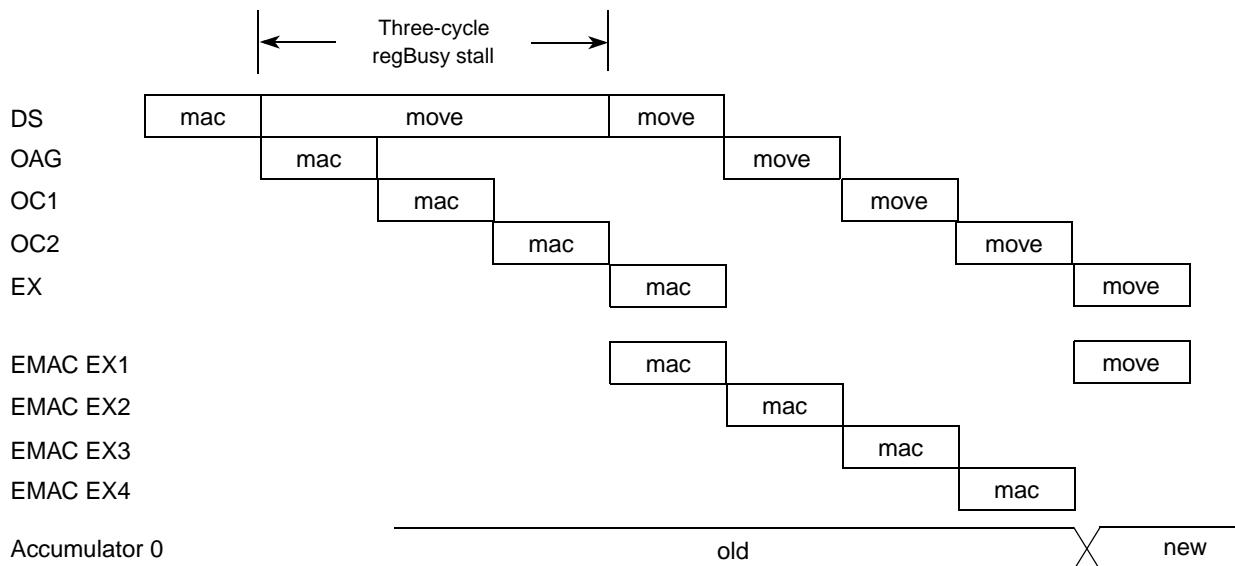
### 5.3.3 EMAC Instruction Execution Times

The instruction execution times for the EMAC can be found in [Section 3.3.5.6, “EMAC Instruction Execution Times”](#).

The EMAC execution pipeline overlaps the EX stage of the OEP (the first stage of the EMAC pipeline is the last stage of the basic OEP). EMAC units are designed for sustained, fully-pipelined operation on accumulator load, copy, and multiply-accumulate instructions. However, instructions that store contents of the multiply-accumulate programming model can generate OEP stalls that expose the EMAC execution pipeline depth:

```
mac.w Ry, Rx, Acc0
move.l Acc0, Rz
```

The MOVE.L instruction that stores the accumulator to an integer register (Rz) stalls until the program-visible copy of the accumulator is available. [Figure 5-9](#) shows EMAC timing.

**Figure 5-9. EMAC-Specific OEP Sequence Stall**

In [Figure 5-9](#), the OEP stalls the store-accumulator instruction for three cycles: the EMAC pipeline depth minus 1. The minus 1 factor is needed because the OEP and EMAC pipelines overlap by a cycle, the EX stage. As the store-accumulator instruction reaches the EX stage where the operation is performed, the recently-updated accumulator 0 value is available.

As with change or use stalls between accumulators and general-purpose registers, introducing intervening instructions that do not reference the busy register can reduce or eliminate sequence-related store-MAC instruction stalls. A major benefit of the EMAC is the addition of three accumulators to minimize stalls caused by exchanges between accumulator(s) and general-purpose registers.

### 5.3.4 Data Representation

MACSR[S/U,F/I] selects one of the following three modes, where each mode defines a unique operand type:

1. Two's complement signed integer: In this format, an N-bit operand value lies in the range  $-2^{(N-1)} \leq \text{operand} \leq 2^{(N-1)} - 1$ . The binary point is right of the lsb.
2. Unsigned integer: In this format, an N-bit operand value lies in the range  $0 \leq \text{operand} \leq 2^N - 1$ . The binary point is right of the lsb.
3. Two's complement, signed fractional: In an N-bit number, the first bit is the sign bit. The remaining bits signify the first N-1 bits after the binary point. Given an N-bit number,  $a_{N-1}a_{N-2}a_{N-3}\dots a_2a_1a_0$ , its value is given by the equation in [Equation 5-3](#).

$$\text{value} = -(1 \cdot a_{N-1}) + \sum_{i=0}^{N-2} 2^{-(i+1-N)} \cdot a_i \quad \text{Eqn. 5-3}$$

This format can represent numbers in the range  $-1 \leq \text{operand} \leq 1 - 2^{-(N-1)}$ .

For words and longwords, the largest negative number that can be represented is -1, whose internal representation is 0x8000 and 0x8000\_0000, respectively. The largest positive word is 0x7FFF or  $(1 - 2^{-15})$ ; the most positive longword is 0x7FFF\_FFFF or  $(1 - 2^{-31})$ . Thus, the number range for these signed fractional numbers is [-1.0, ..., 1.0].

### 5.3.5 MAC Opcodes

MAC opcodes are described in the *ColdFire Programmer's Reference Manual*.

Remember the following:

- Unless otherwise noted, the value of MACSR[N,Z] is based on the result of the final operation that involves the product and the accumulator.
- The overflow (V) flag is managed differently. It is set if the complete product cannot be represented as a 40-bit value (this applies to  $32 \times 32$  integer operations only) or if the combination of the product with an accumulator cannot be represented in the given number of bits. The EMAC design includes an additional product/accumulation overflow bit for each accumulator that are treated as sticky indicators and are used to calculate the V bit on each MAC or MSAC instruction. See [Section 5.2.1, “MAC Status Register \(MACSR\)”](#).

- For the MAC design, the assembler syntax of the MAC (multiply and add to accumulator) and MSAC (multiply and subtract from accumulator) instructions does not include a reference to the single accumulator. For the EMAC, assemblers support this syntax and no explicit reference to an accumulator is interpreted as a reference to ACC0. Assemblers also support syntaxes where the destination accumulator is explicitly defined.
- The optional 1-bit shift of the product is specified using the notation {<< | >>} SF, where <<1 indicates a left shift and >>1 indicates a right shift. The shift is performed before the product is added to or subtracted from the accumulator. Without this operator, the product is not shifted. If the EMAC is in fractional mode (MACSR[F/I] is set), SF is ignored and no shift is performed. Because a product can overflow, the following guidelines are implemented:
  - For unsigned word and longword operations, a zero is shifted into the product on right shifts.
  - For signed, word operations, the sign bit is shifted into the product on right shifts unless the product is zero. For signed, longword operations, the sign bit is shifted into the product unless an overflow occurs or the product is zero, in which case a zero is shifted in.
  - For all left shifts, a zero is inserted into the lsb position.

The following pseudocode explains basic MAC or MSAC instruction functionality. This example is presented as a case statement covering the three basic operating modes with signed integers, unsigned integers, and signed fractionals. Throughout this example, a comma-separated list in curly brackets, {}, indicates a concatenation operation.

```

switch (MACSR[6:5])      /* MACSR[S/U, F/I] */
{
  case 0:                /* signed integers */
    if (MACSR.OMC == 0 || MACSR.PAVn == 0)
      then {
        MACSR.PAVn = 0
        /* select the input operands */
        if (sz == word)
          then {if (U/Ly == 1)
                  then operandY[31:0] = {sign-extended Ry[31], Ry[31:16]}
                  else operandY[31:0] = {sign-extended Ry[15], Ry[15:0]}
                  if (U/Lx == 1)
                    then operandX[31:0] = {sign-extended Rx[31], Rx[31:16]}
                    else operandX[31:0] = {sign-extended Rx[15], Rx[15:0]}
                  }
          else {operandY[31:0] = Ry[31:0]
                 operandX[31:0] = Rx[31:0]
               }
        }

        /* perform the multiply */
        product[63:0] = operandY[31:0] * operandX[31:0]

        /* check for product overflow */
        if ((product[63:39] != 0x0000_00_0) && (product[63:39] != 0xffff_ff_1))
          then {/* product overflow */
                  MACSR.PAVn = 1
                  MACSR.V = 1
                  if (inst == MSAC && MACSR.OMC == 1)
                    then if (product[63] == 1)
                            then result[47:0] = 0x0000_7fff_ffff
                            else result[47:0] = 0xffff_8000_0000
                  }
            }
      }
}

```

## Enhanced Multiply-Accumulate Unit (EMAC)

```
        else if (MACSR.OMC == 1)
            then /* overflowed MAC,
                   saturationMode enabled */
                if (product[63] == 1)
                    then result[47:0] = 0xffff_8000_0000
                    else result[47:0] = 0x0000_7fff_ffff
            }

        /* sign-extend to 48 bits before performing any scaling */
        product[47:40] = {8{product[39]}} /* sign-extend */

        /* scale product before combining with accumulator */
        switch (SF)      /* 2-bit scale factor */
        {
            case 0:    /* no scaling specified */
            break;
            case 1:    /* SF = "<< 1" */
            product[40:0] = {product[39:0], 0}
            break;
            case 2:    /* reserved encoding */
            break;
            case 3:    /* SF = ">> 1" */
            product[39:0] = {product[39], product[39:1]}
            break;
        }

        if (MACSR.PAVn == 0)
            then {if (inst == MSAC)
                    then result[47:0] = ACCx[47:0] - product[47:0]
                    else result[47:0] = ACCx[47:0] + product[47:0]
            }

        /* check for accumulation overflow */
        if (accumulationOverflow == 1)
            then {MACSR.PAVn = 1
                  MACSR.V = 1
                  if (MACSR.OMC == 1)
                      then /* accumulation overflow,
                             saturationMode enabled */
                          if (result[47] == 1)
                              then result[47:0] = 0x0000_7fff_ffff
                              else result[47:0] = 0xffff_8000_0000
                      }
            /* transfer the result to the accumulator */
            ACCx[47:0] = result[47:0]
        }

        MACSR.V = MACSR.PAVn
        MACSR.N = ACCx[47]
        if (ACCx[47:0] == 0x0000_0000_0000)
            then MACSR.Z = 1
            else MACSR.Z = 0
        if ((ACCx[47:31] == 0x0000_0) || (ACCx[47:31] == 0xffff_1))
            then MACSR.EV = 0
            else MACSR.EV = 1
        break;
        case 1,3:          /* signed fractionals */
        if (MACSR.OMC == 0 || MACSR.PAVn == 0)
```

```

then {
    MACSR.PAVn = 0
    if (sz == word)
        then {if (U/Ly == 1)
            then operandY[31:0] = {Ry[31:16], 0x0000}
            else operandY[31:0] = {Ry[15:0], 0x0000}
            if (U/Lx == 1)
                then operandX[31:0] = {Rx[31:16], 0x0000}
                else operandX[31:0] = {Rx[15:0], 0x0000}
        }
        else {operandY[31:0] = Ry[31:0]
            operandX[31:0] = Rx[31:0]
        }
    /* perform the multiply */
    product[63:0] = (operandY[31:0] * operandX[31:0]) << 1
    /* check for product rounding */
    if (MACSR.R/T == 1)
        then { /* perform convergent rounding */
            if (product[23:0] > 0x80_0000)
                then product[63:24] = product[63:24] + 1
            else if ((product[23:0] == 0x80_0000) && (product[24] == 1))
                then product[63:24] = product[63:24] + 1
        }
    /* sign-extend to 48 bits and combine with accumulator */
    /* check for the -1 * -1 overflow case */
    if ((operandY[31:0] == 0x8000_0000) && (operandX[31:0] == 0x8000_0000))
        then product[71:64] = 0x00          /* zero-fill */
        else product[71:64] = {8{product[63]}}      /* sign-extend */
    if (inst == MSAC)
        then result[47:0] = ACCx[47:0] - product[71:24]
        else result[47:0] = ACCx[47:0] + product[71:24]
    /* check for accumulation overflow */
    if (accumulationOverflow == 1)
        then {MACSR.PAVn = 1
            MACSR.V = 1
            if (MACSR.OMC == 1)
                then /* accumulation overflow,
                    saturationMode enabled */
                    if (result[47] == 1)
                        then result[47:0] = 0x007f_ffff_ff00
                        else result[47:0] = 0xff80_0000_0000
            }
        /* transfer the result to the accumulator */
        ACCx[47:0] = result[47:0]
    }
    MACSR.V = MACSR.PAVn
    MACSR.N = ACCx[47]
    if (ACCx[47:0] == 0x0000_0000_0000)
        then MACSR.Z = 1
        else MACSR.Z = 0
    if ((ACCx[47:39] == 0x00_0) || (ACCx[47:39] == 0xff_1))
        then MACSR.EV = 0
        else MACSR.EV = 1
break;
case 2:           /* unsigned integers */
    if (MACSR.OMC == 0 || MACSR.PAVn == 0)
        then {

```

## Enhanced Multiply-Accumulate Unit (EMAC)

```
MACSR.PAVn = 0
/* select the input operands */
if (sz == word)
    then {if (U/Ly == 1)
        then operandY[31:0] = {0x0000, Ry[31:16]}
        else operandY[31:0] = {0x0000, Ry[15:0]}
        if (U/Lx == 1)
            then operandX[31:0] = {0x0000, Rx[31:16]}
            else operandX[31:0] = {0x0000, Rx[15:0]}
    }
    else {operandY[31:0] = Ry[31:0]
        operandX[31:0] = Rx[31:0]
    }

/* perform the multiply */
product[63:0] = operandY[31:0] * operandX[31:0]

/* check for product overflow */
if (product[63:40] != 0x0000_00)
    then {/* product overflow */
        MACSR.PAVn = 1
        MACSR.V = 1
        if (inst == MSAC && MACSR.OMC == 1)
            then result[47:0] = 0x0000_0000_0000
        else if (MACSR.OMC == 1)
            then /* overflowed MAC,
                saturationMode enabled */
                result[47:0] = 0xffff_ffff_ffff
    }

/* zero-fill to 48 bits before performing any scaling */
product[47:40] = 0 /* zero-fill upper byte */

/* scale product before combining with accumulator */
switch (SF) /* 2-bit scale factor */
{
    case 0: /* no scaling specified */
        break;
    case 1: /* SF = "<< 1" */
        product[40:0] = {product[39:0], 0}
        break;
    case 2: /* reserved encoding */
        break;
    case 3: /* SF = ">> 1" */
        product[39:0] = {0, product[39:1]}
        break;
}

/* combine with accumulator */
if (MACSR.PAVn == 0)
    then {if (inst == MSAC)
        then result[47:0] = ACCx[47:0] - product[47:0]
        else result[47:0] = ACCx[47:0] + product[47:0]
    }

/* check for accumulation overflow */
if (accumulationOverflow == 1)
```

```
then {MACSR.PAVn = 1
      MACSR.V = 1
      if (inst == MSAC && MACSR.OMC == 1)
          then result[47:0] = 0x0000_0000_0000
          else if (MACSR.OMC == 1)
              then /* overflowed MAC,
                     saturationMode enabled */
                  result[47:0] = 0xffff_ffff_ffff
      }
      /* transfer the result to the accumulator */
      ACCx[47:0] = result[47:0]
}
MACSR.V = MACSR.PAVn
MACSR.N = ACCx[47]
if (ACCx[47:0] == 0x0000_0000_0000)
    then MACSR.Z = 1
    else MACSR.Z = 0
if (ACCx[47:32] == 0x0000)
    then MACSR.EV = 0
    else MACSR.EV = 1
break;
}
```



# Chapter 6

## Cache

### 6.1 Introduction

This section describes the cache module, including organization, configuration, and coherency. It describes cache operations and how the cache interacts with other memory structures.

#### 6.1.1 Block Diagram

Figure 6-1 shows the organization and integration of the data cache.

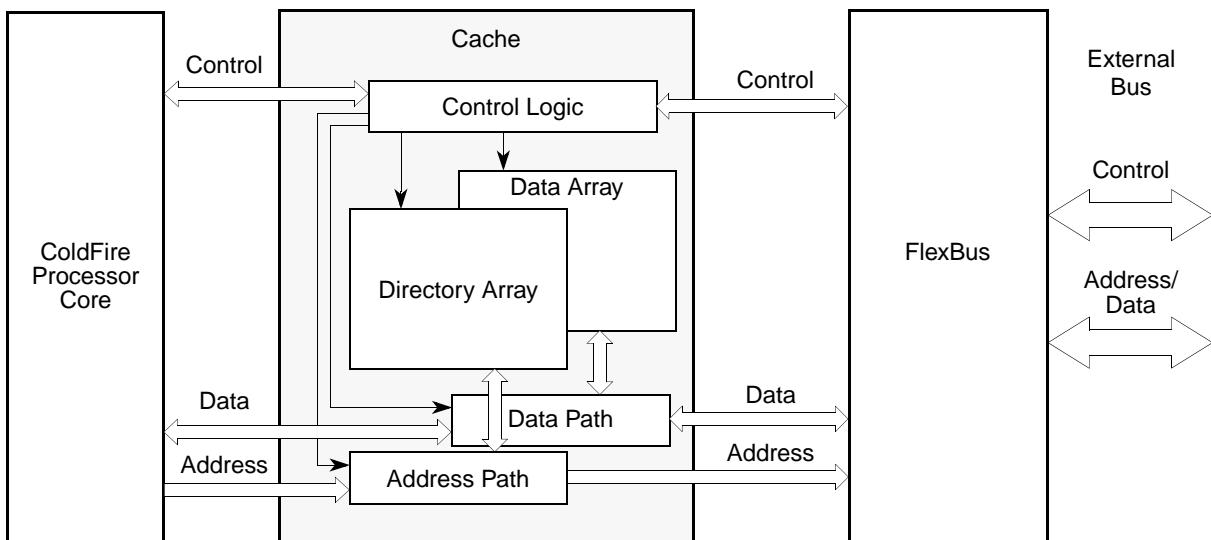


Figure 6-1. Data Cache Organization

#### 6.1.2 Overview

The processor's memory structure includes a 8-Kbyte data cache and a 8-Kbyte instruction cache. Both are non-blocking and four-way set-associative with a 16-byte line size. The cache improves system performance by providing single-cycle access to the instruction and data pipelines. This decouples processor performance from system-memory performance, increasing bus availability for on-chip DMA or external devices.

This device implements a special branch instruction cache for accelerating branches, enabled by a bit in the cache access control register (CACR[BEC]). The branch cache is described in [Section 3.1.1.1, “Change-of-Flow Acceleration.”](#)

Instruction and data caches implement line-fill buffers to optimize line-sized burst accesses. The data cache supports operation of copyback, write-through, or cache-inhibited modes. A four-entry, 32-bit buffer supports cache line-push operations, and can be configured to defer write buffering in write-through or cache-inhibited modes. The cache lock feature can be used to guarantee deterministic response for critical code or data areas.

A non-blocking cache services read or write hits from the processor while a fill (caused by a cache allocation) is in progress. As [Figure 6-1](#) shows, accesses use a single bus connected to the cache.

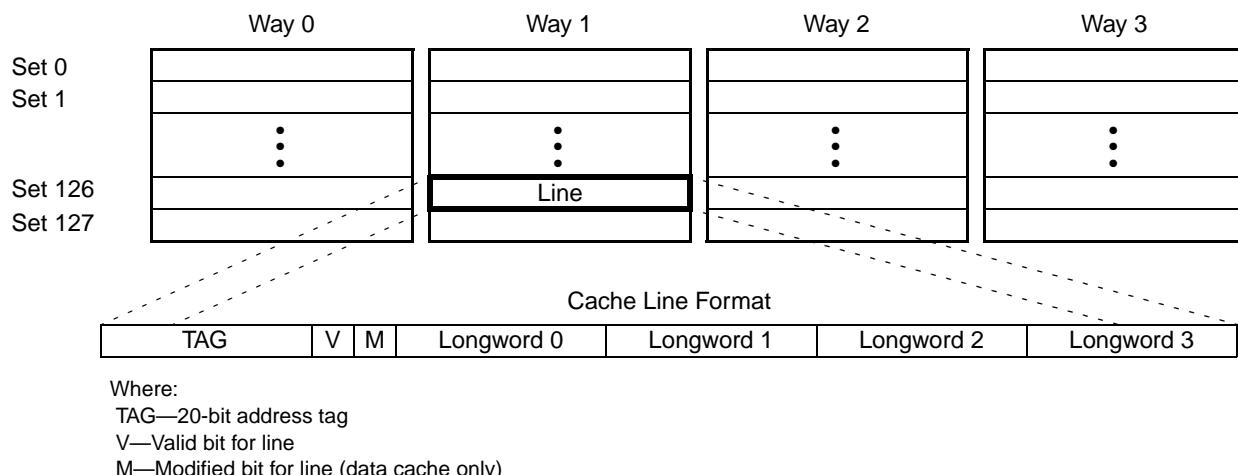
All addresses from the processor to the cache are physical addresses. A cache hit occurs when an address matches a cache entry. For a read, the cache supplies data to the processor. For a write, which is permitted to the data cache only, the processor updates the cache. If an access does not match a cache entry (misses the cache) or if a write access must be written through to memory, the cache performs a bus cycle on the internal bus and correspondingly on the external bus.

The cache module does not implement bus snooping; cache coherency with other possible bus masters must be maintained in software.

## 6.2 Cache Organization

A four-way set-associative cache is organized as four ways (levels). There are 128 sets in the 8-Kbyte data cache with each set defined as the grouping of four lines (one from each level, or way), corresponding to the same index into the cache array. Each line contains 16 bytes (4 longwords). The 8-Kbyte instruction cache has 128 sets as well. Entire cache lines are loaded from memory by burst-mode accesses that cache four longwords of data or instructions. All four longwords must be loaded for the cache line to be valid.

[Figure 6-2](#) shows data cache organization, as well as terminology used.



**Figure 6-2. Data Cache Organization and Line Format**

## 6.2.1 Cache Line States: Invalid, Valid-Unmodified, and Valid-Modified

As shown in [Table 6-1](#), a data cache line is always in one of three states: invalid, valid-unmodified (often referred to as exclusive), or valid-modified. An instruction cache line can be valid or invalid. A valid line can be explicitly invalidated by executing a CPUSHL instruction.

**Table 6-1. Valid and Modified Bit Settings**

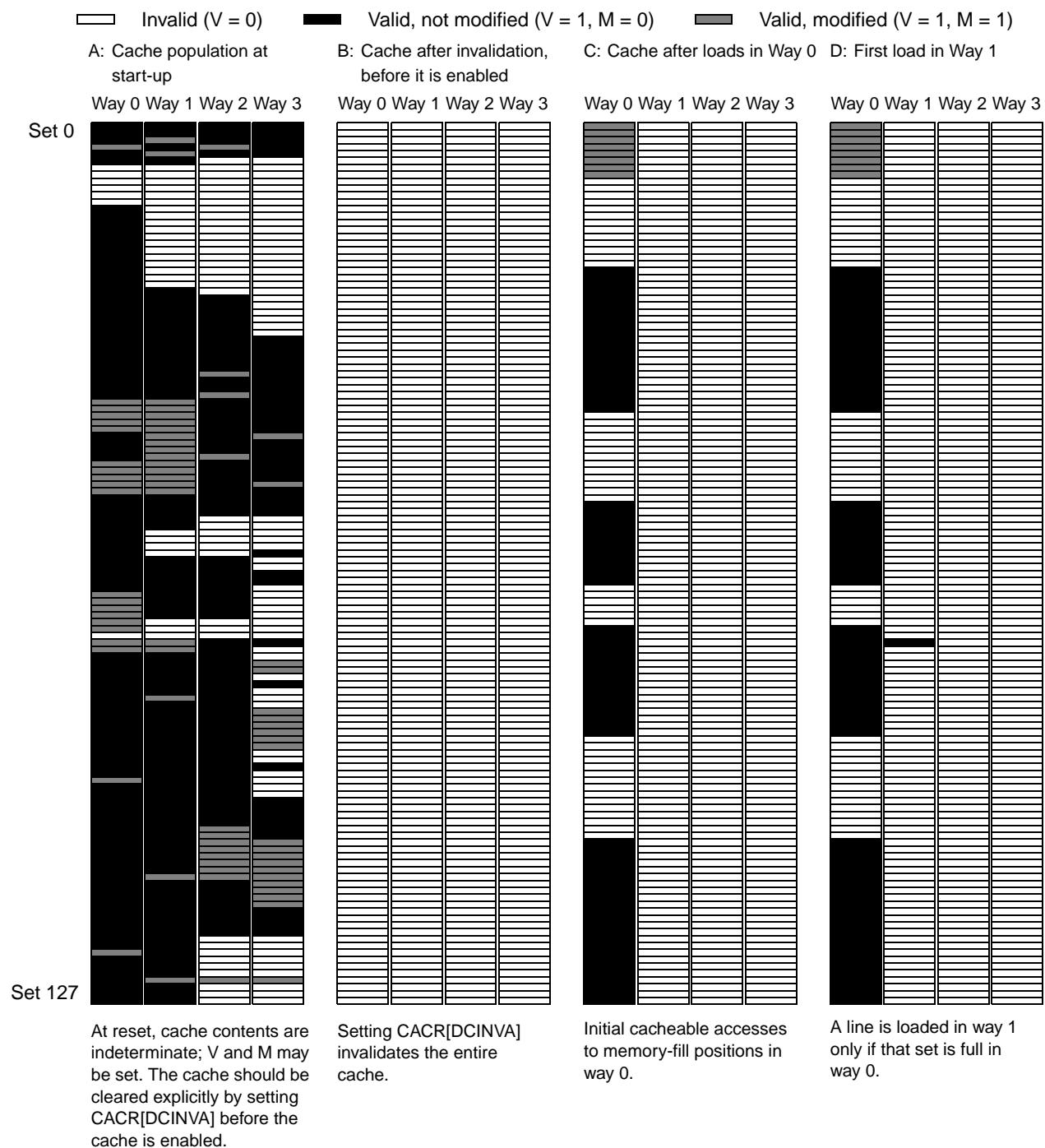
V	M	Description
0	x	Invalid. Ignored during lookups.
1	0	Valid, unmodified. Cache line has valid data that matches system memory.
1	1	Valid, modified. Cache line contains most recent data, data at system memory location is stale.

## 6.2.2 The Cache at Start-Up

As [Figure 6-3](#) (A) shows, after power-up, cache contents are undefined; V and M may be set on some lines even though the cache may not contain the appropriate data for start up. Because reset and power-up do not invalidate cache lines automatically, the cache should be cleared explicitly by setting CACR[DCINVA,ICINVA] before the cache is enabled (B).

After the entire cache is flushed, cacheable entries are loaded first in way 0. If way 0 is occupied, the cacheable entry is loaded into the same set in way 1, as shown in [Figure 6-3](#) (D). This process is described in detail in [Section 6.4, “Functional Description.”](#)

## Cache



**Figure 6-3. Data Cache: A) at Reset; B) after Invalidation; C and D) Loading Pattern**

## 6.3 Memory Map/Register Definition

This section describes the implementation of the cache registers.

**Table 6-2. Cache Memory Map**

BDM <sup>1</sup>	Register	Width (bits)	Access	Reset Value	Written with MOVEC	Section/Page
0x002	Cache Control Register (CACR)	32	R/W	0x0000_0000	Yes	6.3.1/6-5
0x004	Access Control Register 0 (ACR0)	32	R/W	Undefined	Yes	6.3.2/6-8
0x005	Access Control Register 1 (ACR1)	32	R/W	Undefined	Yes	6.3.2/6-8
0x006	Access Control Register 2 (ACR2)	32	R/W	Undefined	Yes	6.3.2/6-8
0x007	Access Control Register 3 (ACR3)	32	R/W	Undefined	Yes	6.3.2/6-8
0x00C	Access Control Register 4 (ACR4)	32	R/W	Undefined	Yes	6.3.2/6-8
0x00D	Access Control Register 5 (ACR5)	32	R/W	Undefined	Yes	6.3.2/6-8
0x00E	Access Control Register 6 (ACR6)	32	R/W	Undefined	Yes	6.3.2/6-8
0x00F	Access Control Register 7 (ACR7)	32	R/W	Undefined	Yes	6.3.2/6-8

<sup>1</sup> The values listed in this column represent the Rc field used when accessing the core registers via the BDM port. For more information see [Chapter 43, "Debug Module"](#).

### 6.3.1 Cache Control Register (CACR)

The CACR register contains bits for configuring the cache. It can be written by the MOVEC register instruction and can be read or written from the debug facility. A hardware reset clears CACR, which disables the cache; however, reset does not affect tags, state information, or data in the cache.

BDM: 0x002

Access: MOVEC write-only  
Debug read/write

31 30 29 28				27 26 25 24				23 22 21 20				19 18 17 16			
R	DEC	DW	DESB	DDPI	DHLCK	DDCM	DC INVA	DDSP	0	0	IVO	BEC	BC INVA	0	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	IEC	SPA	DNFB	IDPI	IHLCK	IDCM	0	IC INVA	0	0	EUSP	0	0	0	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 6-4. Cache-Control Register (CACR)**

**Table 6-3. CACR Field Descriptions**

Field	Description
31 DEC	Enable data cache. 0 Cache disabled. The data cache is not operational, but data and tags are preserved. 1 Cache enabled.
30 DW	Data default write-protect. For normal operations that do not hit in the RAMBARs or ACRs, this field defines write-protection. See <a href="#">Section 6.4.1, “Caching Modes.”</a> 0 Not write protected. 1 Write protected. Write operations cause an access error exception.
29 DESB	Enable data store buffer. Affects the precision of transfers. 0 Imprecise-mode, write-through or cache-inhibited writes bypass the store buffer and generate bus cycles directly. <a href="#">Section 6.4.4.2.1, “Push and Store Buffers,”</a> describes the associated performance penalty. 1 The four-entry FIFO store buffer is enabled; to maximize performance, this buffer defers pending imprecise-mode, write-through or cache-inhibited writes. Precise-mode, cache-inhibited accesses always bypass the store buffer. Precise and imprecise modes are described in <a href="#">Section 6.4.1.2, “Cache-Inhibited Accesses.”</a>
28 DDPI	Data disable CPUSHL invalidate. 0 Normal operation. A CPUSHL instruction causes the selected line to be pushed if modified, then invalidated. 1 No clear operation. A CPUSHL instruction causes the selected line to be pushed if modified, then left valid.
27 DHLCK	Data cache half-data lock. 0 Normal operation. The cache allocates the lowest invalid way. If all ways are valid, the cache allocates the way pointed at by the round-robin counter and then increments this counter. 1 Half-cache operation. The cache allocates to the lower invalid way of levels 2 and 3; if both are valid, the cache allocates to Way 2 if the high-order bit of the round-robin counter is zero; otherwise, it allocates Way 3 and increments the round-robin counter. This locks the contents of ways 0 and 1. Ways 0 and 1 are still updated on write hits and may be pushed or cleared by specific cache push/invalidate instructions.
26–25 DDCM	Default data-cache mode. For normal operations that do not hit in the RAMBARs or ACRs, this field defines the effective cache mode. 00 Cacheable write-through imprecise 01 Cacheable copyback 10 Cache-inhibited precise 11 Cache-inhibited imprecise Precise and imprecise accesses are described in <a href="#">Section 6.4.1.2, “Cache-Inhibited Accesses.”</a>
24 DCINVA	Data cache invalidate all. Setting this bit initiates entire cache invalidation. After invalidation is complete, this bit automatically clears; it is not necessary to clear it explicitly. The caches are not cleared on power-up or normal reset, as shown in <a href="#">Figure 6-3.</a> 0 No invalidation is performed. 1 Initiate invalidation of the entire data cache. The cache controller sequentially clears V and M bits in all sets. Subsequent data accesses stall until the invalidation is finished, at which point, this bit is automatically cleared. In copyback mode, the cache should be flushed using a CPUSHL instruction before setting this bit.
23 DDSP	Data default supervisor-protect. For normal operations that do not hit in the RAMBAR or ACRs, this field defines supervisor-protection 0 Not supervisor protected 1 Supervisor protected. User operations cause a fault
22–21	Reserved, must be cleared.
20 IVO	Invalidate only. Setting this bit forces the invalidation of only the referenced cache line when a CPUSHL instruction executes. See <a href="#">Section 6.4.8, “CPUSHL Enhancements,”</a> for more information.

**Table 6-3. CACR Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>
19 BEC	Enable branch cache. 0 Branch cache disabled. This may be useful if code is unlikely to be reused. 1 Branch cache enabled.
18 BCINVA	Branch cache invalidate all. Invalidation occurs when this bit is set. Branch caches are not cleared on power-up or normal reset. 0 No invalidation is performed. 1 Initiate an invalidation of the entire branch cache.
17–16	Reserved, must be cleared.
15 IEC	Enable instruction cache 0 Instruction cache disabled. All instructions and tags in the cache are preserved. 1 Instruction cache enabled.
14 SPA	Search by physical address. Setting this bit forces the cache to search the accessed set with the supplied physical cache address when a CPUSHL instruction is executed. See <a href="#">Section 6.4.8, “CPUSHL Enhancements,”</a> for more information.
13 DNFB	Default cache-inhibited fill buffer 0 Fill buffer does not store cache-inhibited instruction accesses (16 or 32 bits). 1 Fill buffer can store cache-inhibited accesses. The buffer is used only for normal (TT = 0) instruction reads of a cache-inhibited region. Instructions are loaded into the buffer by a burst access (line fill). They stay in the buffer until they are displaced; subsequent accesses may not appear on the external bus. Setting DNFB can cause a coherency problem for self-modifying code. If a cache-inhibited access uses the buffer while DNFB is set, instructions remain valid in the buffer until a cache-invalidate-all instruction, another cache-inhibited burst, or a miss that initiates a fill. A write to the line in the fill goes to the external bus without updating or invalidating the buffer. Subsequent reads are serviced by the fill buffer and receive stale information. <b>Note:</b> Freescale discourages the use of self-modifying code.
12 IDPI	Instruction CPUSHL invalidate disable. 0 Normal operation. A CPUSHL instruction invalidates the selected line. 1 No clear operation. A CPUSHL instruction causes the selected line to remain valid.
11 IHLCK	Instruction cache half-lock. 0 Normal operation. The cache allocates to the lowest invalid way; if all ways are valid, the cache allocates to the way pointed at by the round-robin counter and then increments this counter. 1 Half cache operation. The cache allocates to the lowest invalid way of ways 2 and 3; if both of these ways are valid, the cache allocates to way 2 if the high-order bit of the round-robin counter is zero; otherwise, it allocates way 3 and then increments the round-robin counter. This locks the contents of ways 0 and 1. Ways 0 and 1 are still updated on write hits and may be pushed or cleared by specific cache push/invalidate instructions.
10 IDCM	Instruction default cache mode. For normal operations that do not hit in the RAMBARs or ACRs, this field defines the effective cache mode. 0 Cacheable 1 Cache-inhibited
9	Reserved, must be cleared.
8 ICINVA	Instruction cache invalidate. Invalidation occurs when this bit is set. Caches are not cleared on power-up or normal reset. 0 No invalidation is performed. 1 Initiate instruction cache invalidation. The cache controller clears all V bits sequentially. Subsequent local memory bus accesses stall until invalidation completes, at which point ICINVA is cleared automatically without software intervention. For copyback mode, use CPUSHL before setting ICINVA.

**Table 6-3. CACR Field Descriptions (continued)**

Field	Description
7 IDSP	Instruction default supervisor-protect. For normal operations that do not hit in the RAMBAR or ACRs, this field defines supervisor-protection. 0 Not supervisor protected 1 Supervisor protected. User operations cause a fault
6	Reserved, must be cleared.
5 EUSP	Enable USP. Enables user stack pointer. 0 USP disabled. Core uses a single stack pointer. 1 USP enabled. Core uses separate supervisor and user stack pointers.
4–0	Reserved, must be cleared.

### 6.3.2 Access Control Registers (ACR*n*)

The ACR*n* registers assign control attributes, such as cache mode and write protection, to specified memory regions. ACR0, ACR1, ACR4, and ACR5 control data attributes; ACR2, ACR3, ACR6, and ACR7 control instruction attributes. Registers are accessed with the MOVEC instruction with the R<sub>c</sub> encodings in [Figure 6-5](#).

For overlapping regions, the lower ACR number takes priority. Data transfers to and from these registers are longword transfers.

#### NOTE

ACR0–7 are read/write by the debug module.

BDM: 0x004 (ACR0)	0x00C (ACR4)	Access: MOVEC write-only
0x005 (ACR1)	0x00D (ACR5)	Debug read/write
0x006 (ACR2)	0x00E (ACR6)	
0x007 (ACR3)	0x00F (ACR7)	
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 BA	ADMSK E S 0 0 AMM 0 0 0 CM 0 SP W <sup>1</sup> 0 0	
W		
Reset – – – –   – – – –   – – – –   – – – –   0 – – 0 0 0 0 0 0 0 – – 0 0 0 – 0 0		

<sup>1</sup> Reserved in ACR2, ACR3, ACR6, and ACR7

**Figure 6-5. Access Control Register Format (ACR*n*)****Table 6-4. ACR*n* Field Descriptions**

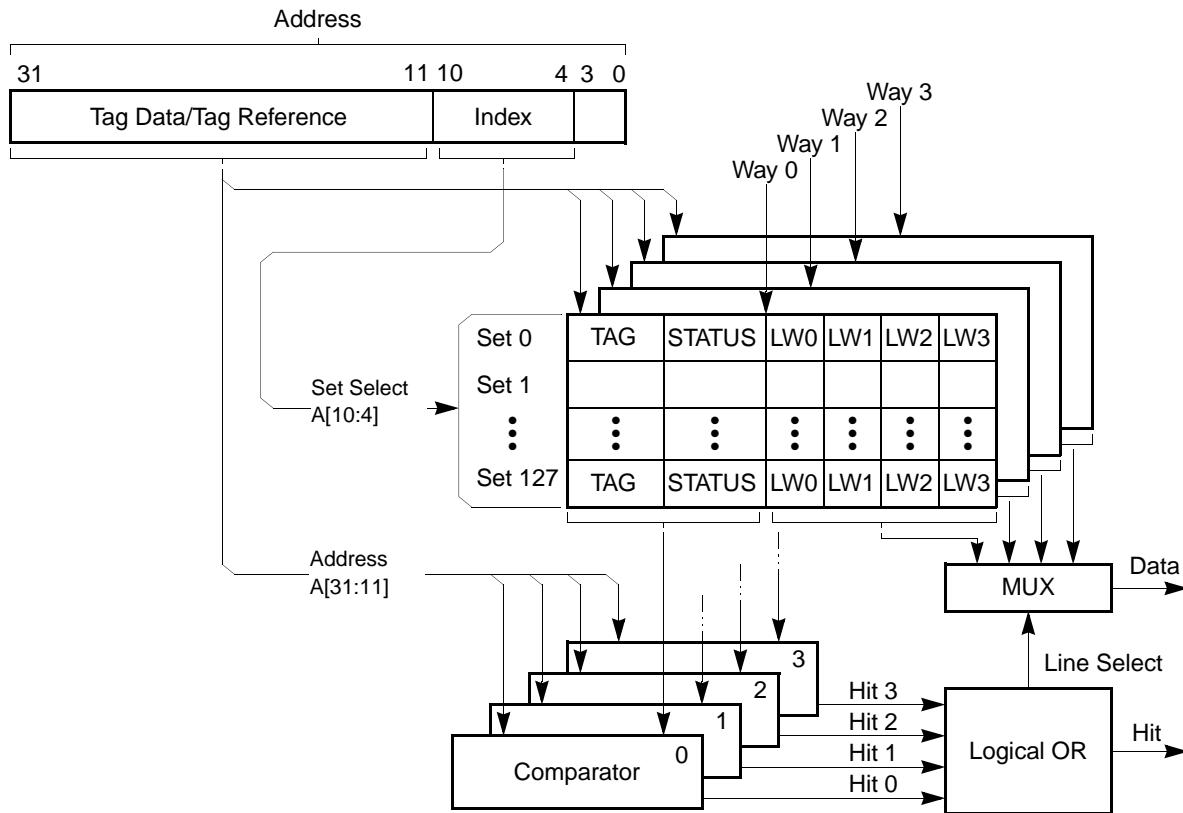
Field	Description
31–24 BA	Base address. Compared with address bits A[31:24]. Eligible addresses that match are assigned the access control attributes.
23–16 ADMSK	Address mask. Setting a mask bit causes the corresponding address base bit to be ignored. The low-order mask bits can be set to define contiguous regions larger than 16 Mbytes. The mask can define multiple non-contiguous regions of memory.

**Table 6-4. ACR $n$  Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>
15 E	Enable. Enables or disables the other ACR $n$ bits. 0 Access control attributes disabled 1 Access control attributes enabled
14–13 S	Supervisor mode. Specifies whether only user or supervisor accesses are allowed in this address range or if the type of access is a don't care. 00 Match addresses only in user mode 01 Match addresses only in supervisor mode 1x Execute cache matching on all accesses
12–11	Reserved, must be cleared.
10 AMM	Address mask mode. 0 The ACR hit function allows control of a 16 Mbytes or greater memory region. 1 The upper 8 bits of the address and ACR are compared without a mask function. Address bits [23:20] of the address and ACR are compared using ACR[19:16] as a mask, allowing control of a 1–16 Mbyte memory region.
9–7	Reserved, must be cleared.
6–5 CM	Cache mode. Selects the cache mode and access precision. Precise and imprecise modes are described in <a href="#">Section 6.4.1.2, “Cache-Inhibited Accesses.”</a> 00 Cacheable, write-through 01 Cacheable, copyback 10 Cache-inhibited, precise 11 Cache-inhibited, imprecise
4	Reserved, must be cleared.
3 SP	Supervisor protect. 0 Indicates supervisor and user mode access allowed 1 Indicates only supervisor access is allowed to this address space and attempted user mode accesses generate an access error exception
2 W	Write protect. Selects the write privilege of the memory region. This field is reserved in the instruction attribute ACRs (ACR2–3, ACR6–7). 0 Read and write accesses permitted 1 Write accesses not permitted
1–0	Reserved, must be cleared.

## 6.4 Functional Description

[Figure 6-6](#) shows the general flow of a caching operation using the 8-Kbyte data cache as an example. This chapter assumes a data cache. Instruction cache operations are similar except for writing to the cache has no support; therefore, such notions of modified cache lines and write allocation do not apply.



**Figure 6-6. Data-Caching Operation**

The following steps determine if a data-cache line for a given address is allocated:

1. The cache set index,  $A[10:4]$ , selects one cache set.
2.  $A[31:11]$  and the cache set index are used as a tag reference or to update the cache line tag field.  $A[31:11]$  can specify 19 possible address lines that can be mapped to one of the four ways.
3. The four tags from the selected cache set are compared with the tag reference. A cache hit occurs if a tag matches the tag reference and the V bit is set, indicating that the cache line contains valid data. If a cacheable write access hits in a valid cache line, the write can occur to the cache line without loading it from memory.

If the memory space is copyback, the updated cache line is marked modified ( $M = 1$ ), because the new data made the data in memory stale. If the memory location is write-through, the write is passed to system memory and the  $M$  bit is not used. The tag does not have TT or TM bits.

To allocate a cache entry, the cache set index selects one of the cache's 128 sets. The cache control logic looks for an invalid cache line to use for the new entry. If none are available, the cache controller uses a pseudo-round-robin replacement algorithm to choose the line to be deallocated and replaced. First, the cache controller looks for an invalid line, with way 0 the highest priority. If all lines have valid data, a 2-bit replacement counter chooses the way. After a line is allocated, the pointer increments to point to the next way.

Cache lines from ways 0 and 1 can be protected from deallocation by enabling half-cache locking. If CACR[DHLCK,IHLCK] are set, the replacement pointer is restricted to way 2 or 3.

As part of deallocation, a valid, unmodified cache line is invalidated. It is consistent with system memory, so memory does not need to be updated. To deallocate a modified cache line, data is placed in a push buffer (for an external cache line push) before being invalidated. After invalidation, the new entry can replace it. The old cache line may be written after the new line is read.

When a cache line is selected to host a new cache entry, three things happen:

1. The new address tag bits A[31:11] are written to the tag.
2. The cache line is updated with the new memory data.
3. The cache line status changes to a valid state (V = 1).

Read cycles that miss in the cache allocate normally as previously described. Write cycles that miss in the cache do not allocate on a cacheable write-through region but do allocate for addresses in a cacheable copyback region.

A copyback byte, word, longword, or line write miss causes the following:

1. The cache initiates a line fill or flush.
2. Space is allocated for a new line.
3. V and M are set to indicate valid and modified.
4. Data is written in the allocated space. No write to memory occurs.

#### **NOTE**

Read hits cannot change the status bits and no deallocation or replacement occurs; the data or instructions are read from the cache. If the cache hits on a write access, data is written to the appropriate portion of the accessed cache line. Write hits in cacheable, write-through regions generate an external write cycle and the cache line is marked valid, but is never marked modified. Write hits in cacheable copyback regions do not perform an external write cycle; the cache line is marked valid and modified (V and M are set). Misaligned accesses are broken into at least two cache accesses. Validity is provided only on a line basis. Unless a whole line is loaded on a cache miss, the cache controller does not validate data in the cache line.

Write accesses designated as cache-inhibited by the CACR or ACR bypass the cache and perform a corresponding external write.

Normally, cache-inhibited reads bypass the cache and are performed on the external bus. The exception occurs when all of the following conditions are true during a cache-inhibited read:

- The cache-inhibited fill buffer bit, CACR[DNFB], is set.
- The access is an instruction read.
- The access is normal (TT = 0).

In this case, an entire line is fetched and stored in the fill buffer. It remains valid there, and the cache can service additional read accesses from this buffer until either another fill or a cache-invalidate-all operation occurs.

Valid cache entries that match during cache-inhibited address accesses are neither pushed nor invalidated. Such a scenario suggests that the associated cache mode for this address space was changed. To avoid this, use the CPUSHL instruction to push or invalidate the cache entry or set CACR[DCINVA] to invalidate the data cache before switching cache modes.

## 6.4.1 Caching Modes

For every memory reference the processor or debug module generates, a set of effective attributes is determined based on the address and ACRs. Caching modes determine how the cache handles an access. A data access can be cacheable in write-through or copyback mode; it can be cache-inhibited in precise or imprecise modes. For normal accesses, the  $ACR_n[CM]$  bit corresponding to the address of the access specifies the caching modes. If an address does not match an ACR, the default caching mode is defined by CACR[DDCM,IDCM]. The specific algorithm is as follows:

```
if (address == ACR0-address including mask)
    effective attributes = ACR0 attributes
else if (address == ACR1-address including mask)
    effective attributes = ACR1 attributes
else if (address == ACR4-address including mask)
    effective attributes = ACR4 attributes
else if (address == ACR5-address including mask)
    effective attributes = ACR5 attributes
else effective attributes = CACR default attributes
```

Addresses matching an ACR can also be write-protected using ACR[W]. Addresses that do not match either ACR can be write-protected using CACR[DW].

Reset disables the cache and clears all CACR bits. As shown in [Figure 6-3](#), reset does not automatically invalidate cache entries; the software invalidates them.

The ACRs allow the defaults selected in the CACR to be overridden. In addition, some instructions (for example, CPUSHL) and processor core operations perform accesses that have an implicit caching mode associated with them. The following sections discuss the different caching accesses and their associated cache modes.

### 6.4.1.1 Cacheable Accesses

If  $ACR_n[CM]$  or the default field of the CACR indicates write-through or copyback, the access is cacheable. If matching data is found, a read access to a write-through or copyback region is read from the cache. Otherwise, the data is read from memory, and the cache is updated. When a line is read from memory for either a write-through or copyback read miss, the longword within the line that contains the core-requested data is loaded first, and the requested data is given immediately to the processor, without waiting for the three remaining longwords to reach the cache.

The following sections describe write-through and copyback modes in detail. Some of this information applies to data caches only.

#### 6.4.1.1.1 Write-Through Mode (Data Cache Only)

Write accesses to regions specified as write-through are always passed on to the external bus; although the cycle can be buffered, depending on the state of CACR[DESB]. Writes in write-through mode are handled

with a no-write-allocate policy—that is, writes that miss in the cache are written to the external bus but do not cause the corresponding line in memory to load into the cache. Write accesses that hit always write through to memory and update matching cache lines. The cache supplies data to data-read accesses that hit in the cache; read misses cause a new cache line to load into the cache.

#### **6.4.1.1.2 Copyback Mode (Data Cache Only)**

Copyback regions are typically used for local data structures or stacks to minimize external bus use and reduce write-access latency. Write accesses to regions specified as copyback that hit in the cache update the cache line and set the corresponding M bit without an external bus access.

The cache should be flushed using the CPUSHL instruction before invalidating the cache in copyback mode using the CINV bits. Modified cache data is written to memory only if the line is replaced because of a miss or a CPUSHL instruction pushes the line. If a byte, word, longword, or line write access misses in the cache, the required cache line is read from memory, thereby updating the cache. When a miss selects a modified cache line for replacement, the modified cache data moves to the push buffer. The replacement line is read into the cache, and the push buffer contents are then written to memory.

#### **6.4.1.2 Cache-Inhibited Accesses**

Memory regions can be designated as cache-inhibited, which is useful for memory containing targets such as I/O devices and shared data structures in multiprocessing systems. It is also important to not cache the processor’s memory-mapped registers. If the corresponding ACR $n$ [CM] or CACR[DDCM] indicates cache-inhibited, precise or imprecise, the access is cache-inhibited. The caching operation is identical for both cache-inhibited modes, which differ only regarding recovery from an external bus error.

In determining whether a memory location is cacheable or cache-inhibited, the CPU checks memory-control registers in the following order:

1. RAMBARs
2. ACR0 and ACR2
3. ACR1 and ACR3
4. ACR4 and ACR6
5. ACR5 and ACR7
6. If an access does not hit in the RAMBARs or the ACRs, the default is provided for all accesses in CACR.

Cache-inhibited write accesses bypass the cache, and a corresponding external write is performed. Cache-inhibited reads bypass the cache and are performed on the external bus, except when all of the following conditions are true:

- The cache-inhibited fill buffer bit, CACR[DNFB], is set.
- The access is an instruction read.
- The access is normal (TT = 0).

In this case, a fetched line is stored in the fill buffer and remains valid there; the cache can service additional read accesses from this buffer until another fill occurs or a cache-invalidate-all operation occurs.

If ACR $n$ [CM] indicates cache-inhibited mode, precise or imprecise, the controller bypasses the cache and performs an external transfer. If a line in the cache matches the address and the mode is cache-inhibited, the cache does not automatically push the line if it is modified, nor does it invalidate the line if it is valid. Before switching cache mode, execute a CPUSHL instruction or set CACR[DCINVA,ICINVA] to invalidate the entire cache.

If ACR $n$ [CM] indicates precise mode, the sequence of read and write accesses to the region is guaranteed to match the instruction sequence. In imprecise mode, the processor core allows read accesses that hit to occur before completion of a pending write from a previous instruction. Writes are not deferred past data-read accesses that miss the cache (they must be read from the bus).

Precise operation forces data-read accesses for an instruction to occur only once by preventing the instruction from being interrupted after data is fetched. Otherwise, if the processor is not in precise mode, an exception aborts the instruction and the data may be accessed again when the instruction is restarted. These guarantees apply only when ACR $n$ [CM] indicates precise mode and aligned accesses.

All CPU space-register accesses, such as MOVEC, are treated as cache-inhibited and precise.

## 6.4.2 Cache Protocol

The following sections describe the cache protocol for processor accesses and assumes that the data is cacheable (that is, write-through or copyback). The discussion of write operations applies to the data cache only.

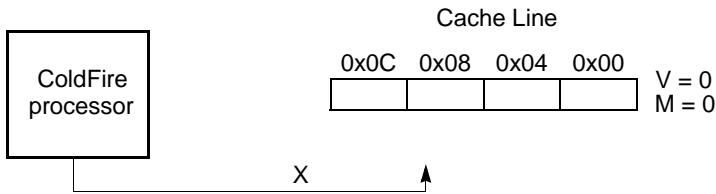
### 6.4.2.1 Read Miss

A processor read that misses in the cache requests the cache controller to generate a bus transaction. This bus transaction reads the needed line from memory and supplies the required data to the processor core. The line is placed in the cache in the valid state.

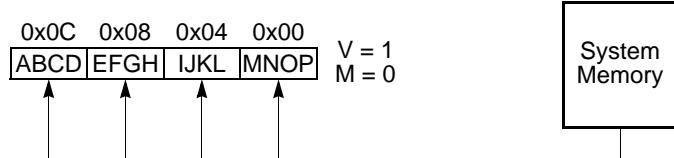
### 6.4.2.2 Write Miss (Data Cache Only)

The cache controller handles processor writes that miss in the data cache differently for write-through and copyback regions. Write misses to copyback regions cause the cache line to be read from system memory, as shown in [Figure 6-7](#).

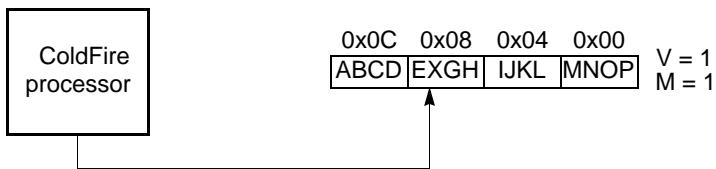
1. Writing character X to 0x0B generates a write miss. Data cannot be written to an invalid line.



2. The cache line (characters A–P) is updated from system memory, and the line is marked valid.



3. After the cache line is filled, the write that initiated the write miss (the character X) completes to 0x0B.



**Figure 6-7. Write-Miss in Copyback Mode**

The new cache line is then updated with write data and the M bit is set for the line, leaving it in modified state. Write misses to write-through regions write directly to memory without loading the corresponding cache line into the cache.

#### 6.4.2.3 Read Hit

On a read hit, the cache provides the data to the processor core and the cache line state remains unchanged. If the cache mode changes for a specific region of address space, lines in the cache corresponding to that region that contain modified data are not pushed out to memory when a read hit occurs within that line. First, execute a CPUSHL instruction or set CACR[DCINVA,ICINVA] before switching the cache mode.

#### 6.4.2.4 Write Hit (Data Cache Only)

The cache controller handles processor writes that hit in the data cache differently for write-through and copyback regions. For write hits to a write-through region, portions of cache lines corresponding to the size of the access are updated with the data. The data is also written to external memory. The cache line state is unchanged. For copyback accesses, the cache controller updates the cache line and sets the M bit for the line. An external write is not performed and the cache line state changes to (or remains in) the modified state.

### 6.4.3 Cache Coherency (Data Cache Only)

The processor provides limited support for maintaining cache coherency in multiple-master environments. Write-through and copyback memory update techniques are supported to maintain coherency between the cache and memory.

The cache does not support snooping (cache coherency is not supported while external or DMA masters use the bus). Therefore, on-chip DMA channels should not access cached local memory locations because coherency is not maintained with the data cache.

### 6.4.4 Memory Accesses for Cache Maintenance

The cache controller performs all maintenance activities that supply data from the cache to the core, including requests for reading new cache lines and writing modified cache lines to memory. The following sections describe memory accesses resulting from cache fill and push operations.

#### 6.4.4.1 Cache Filling

When a new cache line is required, a line read is requested, which generates a burst-read transfer by indicating a line access with the size signals, SIZ[1:0].

The responding device supplies four consecutive longwords of data. Line accesses implicitly request burst-mode operations from memory, but burst operations can be inhibited or enabled through the burst read/write enable bits (CSCR $n$ [BSTR, BSTW]). For more information regarding external bus burst-mode accesses, see [Chapter 20, “FlexBus.”](#)

The first cycle of a cache-line read loads the longword entry corresponding to the requested address. Subsequent transfers load the remaining longword entries.

A burst operation aborts by a write-protection fault, which is the only possible access error. Exception processing proceeds immediately. Unlike Version 2 and Version 3 access errors, the program counter stored on the exception stack frame points to the faulting instruction. See [Section 3.3.4.1, “Access Error Exception.”](#)

#### 6.4.4.2 Cache Pushes

Cache pushes occur for line replacement and as required for the execution of the CPUSHL instruction. To reduce the requested data’s latency in the new line, the modified line being replaced is temporarily placed in the push buffer while the new line is fetched from memory. After the bus transfer for the new line completes, the modified cache line writes back to memory and the push buffer invalidates.

##### 6.4.4.2.1 Push and Store Buffers

The 16-byte push buffer reduces latency for requested new data on a cache miss by holding a displaced modified data cache line while the new data is read from memory.

If a cache miss displaces a modified line, a miss read reference is immediately generated. While waiting for the response, the current contents of the cache location load into the push buffer. When the burst-read

bus transaction completes, the cache controller can generate the appropriate line-write bus transaction to write the push buffer contents into memory.

In imprecise mode, the FIFO store buffer can defer pending writes to maximize performance. The store buffer can support as many as four entries (16 bytes maximum) for this purpose.

Data writes destined for the store buffer cannot stall the core. The store buffer effectively provides a measure of decoupling between the pipeline's ability to generate writes (one per cycle maximum) and the external bus's ability to retire those writes. In imprecise mode, writes stall only if the store buffer is full and a write operation is on the internal bus. The internal write cycle is held, stalling the data execution pipeline.

If the store buffer is not used (store buffer disabled or cache-inhibited precise mode), external bus cycles generate directly for each pipeline-write operation. The instruction is held in the pipeline until external bus transfer termination is received. Therefore, each write is stalled for five cycles, making the minimum write time equal to six cycles when the store buffer is not used. See [Section 3.1.1.2, “Operand Execution Pipeline \(OEP\).”](#)

The data store buffer enable bit, CACR[DESB], controls the enabling of the data-store buffer. The MOVEC instruction can set and clear this bit. At reset, this bit is cleared and all writes perform in order (precise mode). ACR $n$ [CM] or CACR[DDCM] generates the mode used when DESB is set. Cacheable write-through and cache-inhibited imprecise modes use the store buffer.

The store buffer can queue data as much as four bytes wide per entry. Each entry matches the corresponding bus cycle it generates; therefore, a misaligned longword write to a write-through region creates two entries if the address is to an odd-word boundary. It creates three entries if the address is to an odd-byte boundary—one per bus cycle.

#### **6.4.4.2.2 Push and Store Buffer Bus Operation**

As soon as the push or store buffer has valid data, the internal bus controller uses the next available external bus cycle to generate the appropriate write cycles. In the event another cache fill is required (for example, cache miss to process) during the continued instruction execution by the processor pipeline, the pipeline stalls until the push and store buffers empty, before generating the required external bus transaction.

Supervisor instructions, the NOP instruction, and exception processing synchronize the processor core and guarantee the push and store buffers are empty before proceeding. The NOP instruction should be used only to synchronize the pipeline. The preferred no-op function is the TPF instruction. See the *ColdFire Programmer’s Reference Manual* for more information on the TPF instruction.

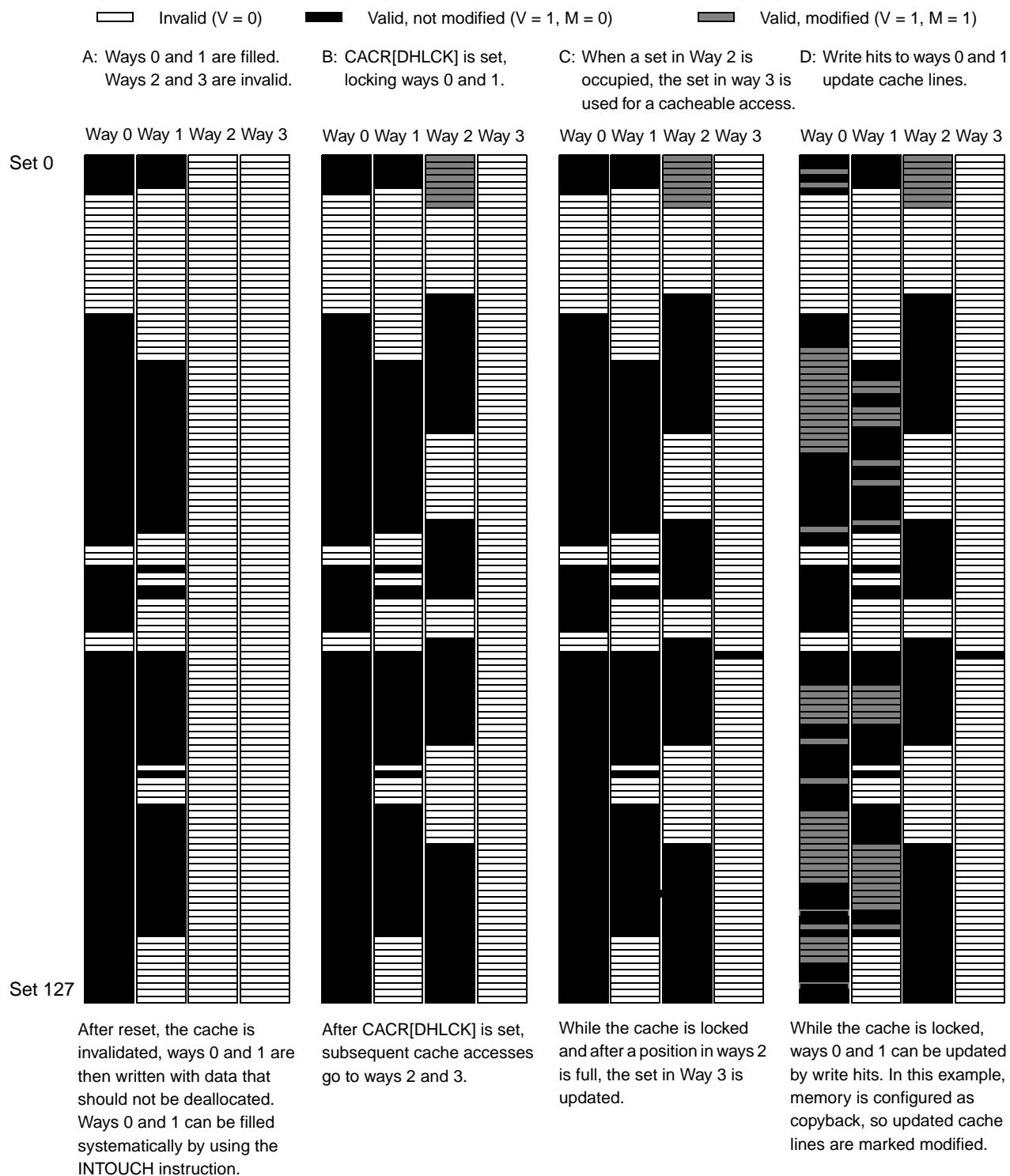
#### **6.4.5 Cache Locking**

Ways 0 and 1 of the data cache can lock by setting CACR[DHLCK]; likewise, ways 0 and 1 of the instruction cache can lock by setting CACR[IHLCK]. If a cache locks, cache lines in ways 0 and 1 are not subject to deallocation by normal cache operations.

As [Figure 6-8 \(B and C\)](#) shows, the algorithm for updating the cache and for identifying cache lines for deallocation does not change. If ways 2 and 3 are entirely invalid, cacheable accesses are first allocated in way 2. Way 3 is not used until the location in way 2 is occupied.

## Cache

Ways 0 and 1 are still updated on write hits (D in [Figure 6-8](#)) and may be pushed or cleared only explicitly by using specific cache push/invalidate instructions. However, new cache lines cannot be allocated in ways 0 and 1.



**Figure 6-8. Data Cache Locking**

## 6.4.6 Cache Management

The cache can be enabled and configured by using a MOVEC instruction to access CACR. A hardware reset clears CACR, disabling the cache and removing all configuration information; however, reset does not affect the tags, state information, and data in the cache.

Set CACR[DCINVA,ICINVA] to invalidate the caches before enabling them.

The privileged CPUSHL instruction supports cache management by selectively pushing and invalidating cache lines. The address register used with CPUSHL directly addresses the cache's directory array. The CPUSHL instruction flushes a cache line.

The value of CACR[DDPI,IDX] determines whether CPUSHL invalidates a cache line after it is pushed. To push the entire cache, implement a software loop to index through all sets and each of the four lines within each set (for a total of 512 lines for the data cache and 1024 lines for the instruction cache). The state of CACR[DEC,IEC] does not affect the operation of CPUSHL or CACR[DCINVA,ICINVA]. Disabling a cache by clearing CACR[IEC] or CACR[DEC] makes the cache non-operational without affecting tags, state information, or contents.

The contents of Ax used with CPUSHL specify cache row and line indexes. This differs from the MC68040 family where a physical address is specified. [Figure 6-9](#) shows the Ax format for the data and instruction cache.

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
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Set Index	Way Index						

**Figure 6-9. Ax Format**

The following code example flushes the entire data cache:

```
_cache_disable:
    nop
    move.w      #0x2700,SR      ;mask off IRQs
    jsr         _cache_flush    ;flush the cache completely
    clr.l      d0
    movec       d0,ACR0        ;ACR0 off
    movec       d0,ACR1        ;ACR1 off
    movec       d0,ACR4        ;ACR4 off
    movec       d0,ACR5        ;ACR5 off
    move.l      #0x01000000,d0  ;Invalidate and disable cache
    movec       d0,CACR
    rts

_cache_flush:
    nop          ; synchronize-flush store buffer
    moveq.l     #0,d0          ; initialize way counter
    moveq.l     #0,d1          ; initialize set counter
    move.l      d0,a0          ; initialize cpushl pointer
setloop:
    cpushl      dc,(a0)        ;push cache line a0
    add.l      #0x0010,a0      ;increment set index by 1
    addq.l      #1,d1          ;increment set counter
    cmpi.l      #128,d1        ;are sets for this way done?
    bne       setloop
    moveq.l     #0,d1          ;set counter to zero again
```

## Cache

```
addq.1      #1,d0          ;increment to next way
move.1       d0,a0          ;set = 0, way = d0
cmpi.1       #4,d0          ;flushed all the ways?
bne         setloop
rts
```

The following CACR loads assume: the instruction cache has been invalidated, the default instruction cache mode is cacheable, the default data cache mode is copyback.

```
dataCacheLoadAndLock:
    move.1   #0xa3080800,d0    ; enable and invalidate data cache ...
    movec    d0,cacr          ; ... in the CACR
```

The following code segments preload half of the data cache (4 Kbytes). It assumes a contiguous block of data is to be mapped into the data cache, starting at a 0-modulo-4K address.

```
move.1   #256,d0          ; 256 16-byte lines in 4K space
lea      data_,a0          ; load pointer defining data area
dataCacheLoop:
    tst.b   (a0)           ; touch location + load into data cache
    lea     16(a0),a0        ; increment address to next line
    subq.1  #1,d0          ; decrement loop counter
    bne.b   dataCacheLoop  ; if done, then exit, else continue

; A 4K region has been loaded into ways 0 and 1 of the 8K data cache. lock it!
    move.1   #0xaa088000,d0  ; set the data cache lock bit ...
    movec    d0,cacr          ; ... in the CACR
    rts

align 16
```

The following CACR loads assume the data cache has been previously invalidated, the default instruction cache mode is cacheable, and the default operand cache mode is copyback.

This function must be mapped into a cache-inhibited or SRAM space or these text lines are to be prefetched into the instruction cache. This may displace some of the 4-Kbyte space being explicitly fetched.

```
instructionCacheLoadAndLock:
    move.1   #0xa2088100,d0  ; enable and invalidate the instruction
    movec    d0,cacr          ; cache in the CACR
```

The following code segments preload half of the instruction cache (4 Kbytes). It assumes a contiguous block of data is to be mapped into the cache, starting at a 0-modulo-4K address

```
move.1   #256,d0          ; 256 16-byte lines in 4K space
lea      code_,a0          ; load pointer defining code area
instCacheLoop:
    intouch (a0)           ; touch location + load into instruction cache

; Note in the assembler we use, there is no INTOUCH opcode. The following
; is used to produce the required binary representation
    cpushl  #nc,(a0)         ;touch location + load into
                           ;instruction cache
    lea     16(a0),a0        ;increment address to next line
    subq.1  #1,d0          ;decrement loop counter
    bne.b   instCacheLoop  ;if done, then exit, else continue
; A 4K region was loaded into levels 0 and 1 of the 8-Kbyte instruction cache. lock it!
```

```

move.l #0xa2088800,d0 ;set the instruction cache lock bit
movec d0,cacr ;in the CACR
rts

```

## 6.4.7 Cache Operation Summary

This section gives operational details for the cache and presents instruction and data cache-line state diagrams.

### 6.4.7.1 Instruction Cache State Transitions

Because the instruction cache does not support writes, it supports fewer operations than the data cache. As Figure 6-10 shows, an instruction cache line can be in one of two states: valid or invalid. Modified state is not supported. Transitions are labeled with a capital letter (indicating the previous state) and a number (indicating the specific case listed in Table 6-5). These numbers correspond to the equivalent operations on data caches as described in Section 6.4.7.2, “Data Cache-State Transitions.”

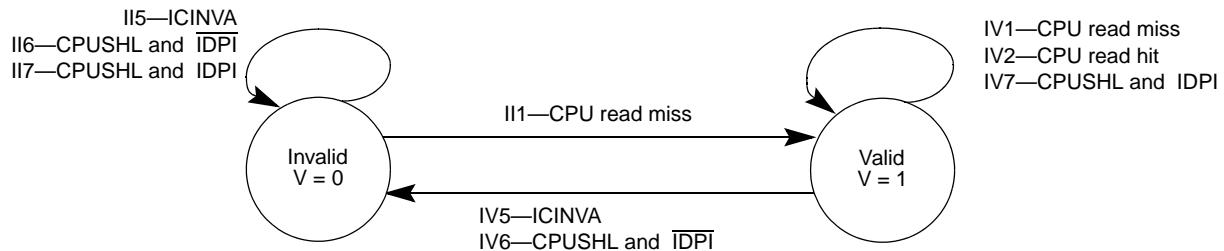


Figure 6-10. Instruction Cache Line State Diagram

Table 6-5 describes the instruction cache-state transitions shown in Figure 6-10.

Table 6-5. Instruction Cache Line State Transitions

Access	Current State			
	Invalid (V = 0)		Valid (V = 1)	
Read miss	II1 Read line from memory and update cache; supply data to processor; go to valid state.		IV1 Read new line from memory and update cache; supply data to processor; stay in valid state.	
Read hit	II2 Not possible		IV2 Supply data to processor; stay in valid state.	
Write miss	II3 Not possible		IV3 Not possible	
Write hit	II4 Not possible		IV4 Not possible	
Cache invalidate	II5 No action; stay in invalid state.		IV5 No action; go to invalid state.	

**Table 6-5. Instruction Cache Line State Transitions (continued)**

Access	Current State	
	Invalid (V = 0)	Valid (V = 1)
Cache push	II6 II7 No action; stay in invalid state.	IV6 No action; go to invalid state.  IV7 No action; stay in valid state.

### 6.4.7.2 Data Cache-State Transitions

Using the V and M bits, the data cache supports a line-based protocol allowing individual cache lines to be invalid, valid, or modified. To maintain coherency with memory, the data cache supports write-through and copyback modes, specified by the corresponding ACR[CM], or CACR[DDCM] if no ACR matches.

Read or write misses to copyback regions cause the cache controller to read a cache line from memory into the cache. If available, tag and data from memory update an invalid line in the selected set. The line state then changes from invalid to valid by setting the V bit for the line. If all lines in the row are already valid or modified, the pseudo-round-robin replacement algorithm selects one of the four lines and replaces the tag and data. Before replacement, modified lines are buffered temporarily and later copied back to memory after the new line has been read from memory.

Figure 6-11 shows the three possible data-cache line states and possible processor-initiated transitions for memory configured as copyback. Transitions are labeled with a capital letter indicating the previous state and a number indicating the specific case listed in Table 6-6.

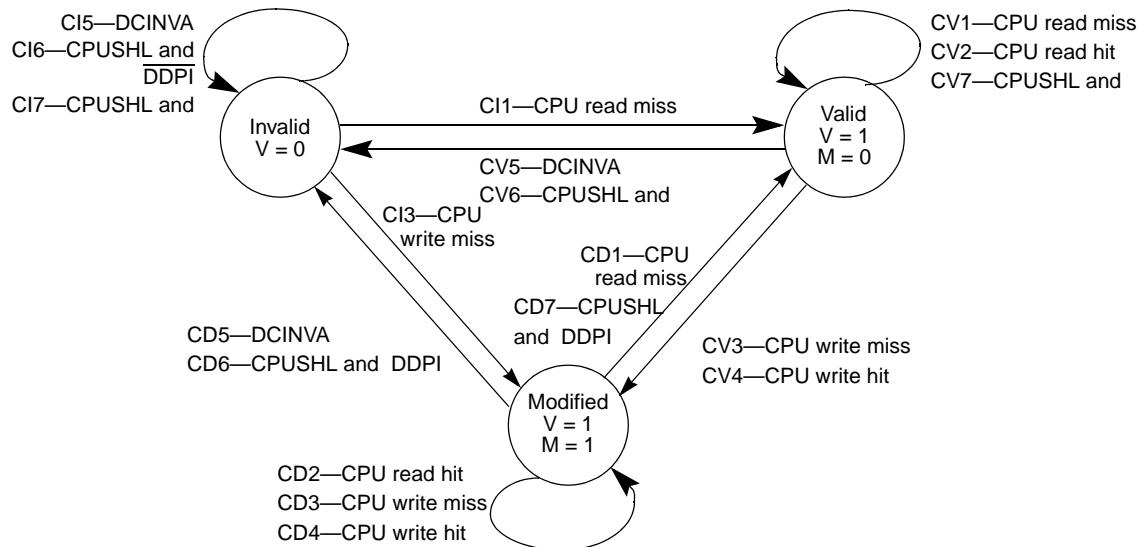
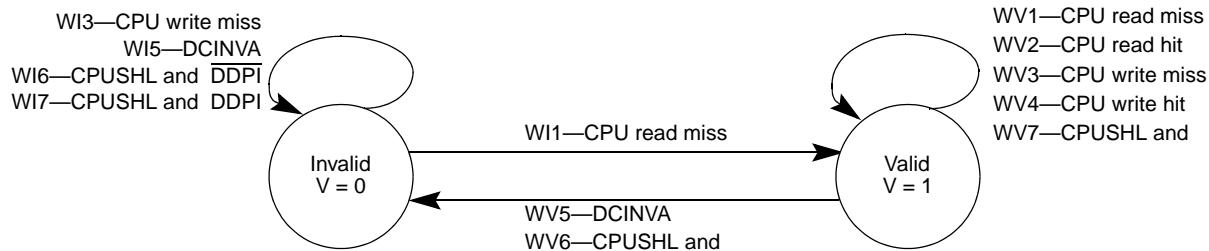
**Figure 6-11. Data Cache Line State Diagram—Copyback Mode**

Figure 6-12 shows the two possible states for a cache line in write-through mode.

**Figure 6-12. Data-Cache Line State Diagram—Write-Through Mode**

**Table 6-6** describes data-cache line transitions and what accesses cause them.

**Table 6-6. Data Cache Line State Transitions**

Access	Current State					
	Invalid (V = 0)		Valid (V = 1, M = 0)		Modified (V = 1, M = 1)	
Read miss	(C,W)I1	Read line from memory and update cache; supply data to processor; go to valid state.	(C,W)V1	Read new line from memory and update cache; supply data to processor; stay in valid state.	CD1	Push modified line to buffer; read new line from memory and update cache; supply data to processor; write push buffer contents to memory; go to valid state.
Read hit	(C,W)I2	Not possible.	(C,W)V2	Supply data to processor; stay in valid state.	CD2	Supply data to processor; stay in modified state.
Write miss (copy-back)	CI3	Read line from memory and update cache; write data to cache; go to modified state.	CV3	Read new line from memory and update cache; write data to cache; go to modified state.	CD3	Push modified line to buffer; read new line from memory and update cache; write push buffer contents to memory; stay in modified state.
Write miss (write-through)	WI3	Write data to memory; stay in invalid state.	WV3	Write data to memory; stay in valid state.	WD3	Write data to memory; stay in modified state. Cache mode changed for the region corresponding to this line. To avoid this state, execute a CPUSHL instruction or set CACR[DCINVA,ICINVA] before switching modes.
Write hit (copy-back)	CI4	Not possible.	CV4	Write data to cache; go to modified state.	CD4	Write data to cache; stay in modified state.

**Table 6-6. Data Cache Line State Transitions (continued)**

Access	Current State					
	Invalid (V = 0)		Valid (V = 1, M = 0)		Modified (V = 1, M = 1)	
Write hit (write-through)	WI4	Not possible.	WV4	Write data to memory and to cache; stay in valid state.	WD4	Write data to memory and to cache; go to valid state. Cache mode changed for the region corresponding to this line. To avoid this state, execute a CPUSHL instruction or set CACR[DCINVA,ICINVA] before switching modes.
Cache invalidate	(C,W)I5	No action; stay in invalid state.	(C,W)V5	No action; go to invalid state.	CD5	No action (modified data lost); go to invalid state.
Cache push	(C,W)I6 (C,W)I7	No action; stay in invalid state.	(C,W)V6	No action; go to invalid state.	CD6	Push modified line to memory; go to invalid state.
			(C,W)V7	No action; stay in valid state.	CD7	Push modified line to memory; go to valid state.

The following tables present the same information as [Table 6-6](#), organized by the current state of the cache line. In [Table 6-7](#) the current state is invalid.

**Table 6-7. Data Cache Line State Transitions (Current State Invalid)**

Access	Response	
Read miss	(C,W)I1	Read line from memory and update cache; supply data to processor; go to valid state.
Read hit	(C,W)I2	Not possible
Write miss (copyback)	CI3	Read line from memory and update cache; write data to cache; go to modified state.
Write miss (write-through)	WI3	Write data to memory; stay in invalid state.
Write hit (copyback)	CI4	Not possible
Write hit (write-through)	WI4	Not possible
Cache invalidate	(C,W)I5	No action; stay in invalid state.
Cache push	(C,W)I6	No action; stay in invalid state.
Cache push	(C,W)I7	No action; stay in invalid state.

In [Table 6-8](#) the current state is valid.

**Table 6-8. Data Cache Line State Transitions (Current State Valid)**

Access	Response	
Read miss	(C,W)V1	Read new line from memory and update cache; supply data to processor; stay in valid state.
Read hit	(C,W)V2	Supply data to processor; stay in valid state.
Write miss (copyback)	CV3	Read new line from memory and update cache; write data to cache; go to modified state.
Write miss (write-through)	WV3	Write data to memory; stay in valid state.
Write hit (copyback)	CV4	Write data to cache; go to modified state.
Write hit (write-through)	WV4	Write data to memory and to cache; stay in valid state.
Cache invalidate	(C,W)V5	No action; go to invalid state.
Cache push	(C,W)V6	No action; go to invalid state.
Cache push	(C,W)V7	No action; stay in valid state.

In [Table 6-9](#) the current state is modified.

**Table 6-9. Data Cache Line State Transitions (Current State Modified)**

Access	Response	
Read miss	CD1	Push modified line to buffer; read new line from memory and update cache; supply data to processor; write push buffer contents to memory; go to valid state.
Read hit	CD2	Supply data to processor; stay in modified state.
Write miss (copyback)	CD3	Push modified line to buffer; read new line from memory and update cache; write push buffer contents to memory; stay in modified state.
Write miss (write-through)	WD3	Write data to memory; stay in modified state. Cache mode changed for the region corresponding to this line. To avoid this state, execute a CPUSHL instruction or set CACR[DCINVA,ICINVA] before switching modes.
Write hit (copyback)	CD4	Write data to cache; stay in modified state.

**Table 6-9. Data Cache Line State Transitions (Current State Modified) (continued)**

Access	Response		
Write hit (write-through)	WD4	Write data to memory and to cache; go to valid state. Cache mode changed for the region corresponding to this line. To avoid this state, execute a CPUSHL instruction or set CACR[DCINVA,ICINVA] before switching modes.	
Cache invalidate	CD5	No action (modified data lost); go to invalid state.	
Cache push	CD6	Push modified line to memory; go to invalid state.	
Cache push	CD7	Push modified line to memory; go to valid state.	

## 6.4.8 CPUSHL Enhancements

The extended CPUSHL functionality adds two new bits in the cache control register (CACR) to support a set search using a physical address. In particular, the added CACR bits are defined as:

```
cacr[14] = cacr[SPA] cpushl Search by physical address
cacr[20] = cacr[IVO] cpushl Invalidate only
```

In many applications where data is shared among bus masters, the performance of the software function to push and/or clear specific lines from the data cache is important. Previously, this function was implemented using a CPUSHL loop that explicitly referenced all four ways in each cache set. By referencing all possible cache entries that might contain the targeted address range, it is guaranteed that the cache data of interest is referenced. It also has the unfortunate side-effect of potentially pushing/clearing other data that happened to be mapped into the targeted cache entries.

For the enhanced CPUSHL functionality, a higher-performance version of this function is possible using a physical address range and a simpler search loop. The enhanced CPUSHL instruction also affects only the specific cache lines being referenced and does not change the state of any other cache entries.

The specific variation of the CPUSHL instruction used to operate only on the data cache:

```
cpushl dc, (ax)
```

where *dc* specifies the data cache, and *ax* is the cache set address and way number for the baseline CPUSHL functionality or *ax* is the physical address for the enhanced CPUSHL.

For the enhanced implementations, the specific operation performed by the CPUSHL instruction is defined by the state of four CACR bits. See [Table 6-10](#).

**Table 6-10. Enhanced CPUSHL Functionality**

Instruction	CACR Bits				Description	
	[14] SPA	[20] IVO	[28] DDPI	[12] IDPI	Search by...	Action
cpushl bc, (ax)	0	0	0	0	Cache address/way	Clear both
cpushl bc, (ax)	0	0	0	1	Cache address/way	Clear data

**Table 6-10. Enhanced CPUSHL Functionality (continued)**

Instruction	CACR Bits				Description	
	[14] SPA	[20] IVO	[28] DDPI	[12] IDPI	Search by...	Action
cpushl bc,(ax)	0	0	1	0	Cache address/way	Push data, clear instruction
cpushl bc,(ax)	0	0	1	1	Cache address/way	Push data
cpushl bc,(ax)	0	1	–	–	Cache address/way	Invalidate both
cpushl bc,(ax)	1	0	0	0	Physical address	Clear both
cpushl bc,(ax)	1	0	0	1	Physical address	Clear data
cpushl bc,(ax)	1	0	1	0	Physical address	Push data, clear instruction
cpushl bc,(ax)	1	0	1	1	Physical address	Push data
cpushl bc,(ax)	1	1	–	–	Physical address	Invalidate both
cpushl dc,(ax)	0	0	0	–	Cache address/way	Clear data
cpushl dc,(ax)	0	0	1	–	Cache address/way	Push data
cpushl dc,(ax)	0	1	–	–	Cache address/way	Invalidate data
cpushl dc,(ax)	1	0	0	–	Physical address	Clear data
cpushl dc,(ax)	1	0	1	–	Physical address	Push data
cpushl dc,(ax)	1	1	–	–	Physical address	Invalidate data
cpushl ic,(ax)	0	0	–	0	Cache address/way	Clear instruction
cpushl ic,(ax)	0	0	–	1	Cache address/way	No operation
cpushl ic,(ax)	0	1	–	–	Cache address/way	Invalidate inst
cpushl ic,(ax)	1	0	–	0	Physical address	Clear instruction
cpushl ic,(ax)	1	0	–	1	Physical address	No operation
cpushl ic,(ax)	1	1	–	–	Physical address	Invalidate instruction
cpushl nc,(ax)	–	–	–	–	Address	intouch instruction

## 6.5 Initialization/Application Information

The following example sets up the cache for flash or ROM space only.

```

move.l #0xA70C8100,D0          //enable cache, invalidate it,
                                //default mode is cache-inhibited imprecise
movec D0, CACR

move.l #0xFF00C000,D0          //cache flash space, enable,
                                //ignore supervisor/user, cacheable, writethrough
movec D0, ACR0

```



# **Chapter 7**

## **Static RAM (SRAM)**

### **7.1 Introduction**

This chapter describes the on-chip static RAM (SRAM) implementation, including general operations, configuration, and initialization. It also provides information and examples showing how to minimize power consumption when using the SRAM.

#### **7.1.1 Overview**

The SRAM module provides a general-purpose memory block that the ColdFire processor can access in a single cycle. The location of the memory block can be specified to any 0-modulo-64K address within the 256-Mbyte address space (0x8000\_0000 – 0x8FFF\_FFFF). The memory is ideal for storing critical code or data structures or for use as the system stack. Because the SRAM module is physically connected to the processor's high-speed local bus, it can service processor-initiated accesses or memory-referencing commands from the debug module.

Depending on configuration information, processor references may be sent to the cache and the SRAM block simultaneously. If the reference maps into the region defined by the SRAM, the SRAM provides the data back to the processor, and the cache data is discarded. Accesses from the SRAM module are not cached.

The SRAM is dual-ported to provide access for any of the bus masters via the SRAM backdoor on the crossbar switch. The SRAM is partitioned into two physical memory arrays to allow simultaneous access to arrays by the processor core and another bus master. For more information on arbitration between multiple masters accessing the SRAM, see [Chapter 13, “System Control Module \(SCM\).”](#)

#### **7.1.2 Features**

The major features includes:

- One 64 Kbyte SRAM
- Single-cycle access
- Physically located on the processor's high-speed local bus
- Memory location programmable on any 0-modulo-64 Kbyte address
- Byte, word, and longword address capabilities
- Backdoor access by crossbar switch masters is clocked separate from the core. If the core is stopped, other masters in the system can still access the SRAM.

## 7.2 Memory Map/Register Description

The SRAM programming model shown in [Table 7-1](#) includes a description of the SRAM base address register (RAMBAR), SRAM initialization, and power management.

**Table 7-1. SRAM Programming Model**

Rc[11:0] <sup>1</sup>	Register	Width (bits)	Access	Reset Value	Written w/ MOVEC	Section/Page
<b>Supervisor Access Only Registers</b>						
0xC05	RAM Base Address Register (RAMBAR)	32	R/W	See Section	Yes	<a href="#">7.2.1/7-2</a>

<sup>1</sup> The values listed in this column represent the Rc field used when accessing the core registers via the BDM port. For more information see [Chapter 43, “Debug Module.”](#)

### 7.2.1 SRAM Base Address Register (RAMBAR)

The configuration information in the SRAM base-address register (RAMBAR) controls the operation of the SRAM module.

- The RAMBAR holds the SRAM base address. The MOVEC instruction provides write-only access to this register.
- The RAMBAR can be read or written from the debug module.
- All undefined bits in the register are reserved. These bits are ignored during writes to the RAMBAR and return zeroes when read from the debug module.
- A reset clears the RAMBAR’s priority, backdoor write-protect, and valid bits, and sets the backdoor enable bit. This enables the backdoor port and invalidates the processor port to the SRAM (The RAMBAR must be initialized before the core can access the SRAM.) All other bits are unaffected.

#### NOTE

The only applicable address ranges for the SRAM module’s base address are 0x8000\_0000 – 0x8FFF\_0000. The address must be 0-modulo-64 K. Set the RAMBAR register appropriately.

By default, the RAMBAR is invalid, but the backdoor is enabled. In this state, any core accesses to the SRAM are routed through the backdoor. Therefore, the SRAM is accessible by the core, but it does not have a single-cycle access time. To ensure that the core has single-cycle access to the SRAM, set the RAMBAR[V] bit.

Any access within the memory range allocated for the on-chip SRAM (0x8000\_0000-0x8FFF\_FFFF) hits in the SRAM even if the address is beyond the defined size for the SRAM. This creates address aliasing for the on-chip SRAM memory. For example, writes to addresses 0x8000\_0000 and 0x8001\_0000 modify the same memory location. System software should ensure SRAM address pointers do not exceed the SRAM size to prevent unwanted overwriting of SRAM.

The RAMBAR contains several control fields. These fields are shown in [Figure 7-1](#).

Rc[11:0]: 0x0C05 (RAMBAR)

Access: User write-only  
Debug 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																	0	0	0	0	PRIU	PRIL	BDE	WP	D/I	BWP	C/I	SC	SD	UC	UD	V
W																					0	0	1	U	0	0	U	U	U	U	0	
Reset	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	0	0	0	0	0	0	1	U	0	0	U	U	U	U	0	

**Figure 7-1. SRAM Base Address Register (RAMBAR)**

**Table 7-2. RAMBAR Field Descriptions**

Field	Description		
31–16 BA	Base Address. Defines the 0-modulo-64K base address of the SRAM module. By programming this field, the SRAM may be located on any 64-Kbyte boundary within the processor's 256-Mbyte address space. For proper operation, the base address must be set to between 0x8000_0000 and 0x8FFF_0000.		
15–12	Reserved, must be cleared.		
11–10 PRIU PRIL	Priority Bit. PRIU determines if the SRAM backdoor or CPU has priority in the upper 32K bank of memory. PRIL determines if the SRAM backdoor or CPU has priority in the lower 32K bank of memory. If a bit is set, the CPU has priority. If a bit is cleared, the SRAM backdoor has priority. Priority is determined according to the following table:		
	PRIU,PRIL	Upper Bank Priority	Lower Bank Priority
	00	SRAM Backdoor	SRAM Backdoor
	01	SRAM Backdoor	CPU
	10	CPU	SRAM Backdoor
	11	CPU	CPU
<b>Note:</b> The recommended setting (maximum performance) for the priority bits is 00.			
9 BDE	Backdoor Enable. Allows access by non-core bus masters via the SRAM backdoor on the crossbar switch		
	0 Non-core crossbar switch master access to memory is disabled.		
	1 Non-core crossbar switch master access to memory is enabled.		
8 WP	Write Protect. Allows only read accesses to the SRAM. When this bit is set, any attempted write access from the core generates an access error exception to the ColdFire processor core.		
	0 Allows core read and write accesses to the SRAM module		
	1 Allows only core read accesses to the SRAM module		
	<b>Note:</b> This bit does not affect non-core write accesses.		
7 D/I	Data/instruction bus. Determines if the SRAM is connected to the internal data or instruction bus.		
	0 Data bus		
	1 Instruction bus		
6 BWP	Backdoor Write Protect. Allows only read accesses from the non-core bus masters. When this bit is set, any attempted write access from the non-core bus masters on the backdoor terminates the bus transfer with an access error.		
	0 Allows read and write accesses to the SRAM module from non-core masters.		
	1 Allows only read accesses to the SRAM module from non-core masters.		

**Table 7-2. RAMBAR Field Descriptions (continued)**

Field	Description
5–1 C/I, SC, SD, UC, UD	<p>Address Space Masks (AS<sub>n</sub>). These five bit fields allow types of accesses to be masked or inhibited from accessing the SRAM module. The address space mask bits are:</p> <ul style="list-style-type: none"> <li>C/I = CPU space/interrupt acknowledge cycle mask</li> <li>SC = Supervisor code address space mask</li> <li>SD = Supervisor data address space mask</li> <li>UC = User code address space mask</li> <li>UD = User data address space mask</li> </ul> <p>For each address space bit:</p> <ul style="list-style-type: none"> <li>0 An access to the SRAM module can occur for this address space</li> <li>1 Disable this address space from the SRAM module. If a reference using this address space is made, it is inhibited from accessing the SRAM module and is processed like any other non-SRAM reference.</li> </ul> <p>These bits do not affect accesses by non-core bus masters using the SRAM backdoor port in any manner. These bits are useful for power management as detailed in <a href="#">Section 7.3.2, “Power Management.”</a> In most applications, the C/I bit is set</p>
0 V	<p>Valid. When set, this bit enables the SRAM module; otherwise, the module is disabled. A hardware reset clears this bit.</p> <ul style="list-style-type: none"> <li>0 Processor accesses of the SRAM are masked</li> <li>1 Processor accesses of the SRAM are enabled</li> </ul>

## 7.3 Initialization/Application Information

After a hardware reset, the SRAM module contents are undefined. The valid bit of the RAMBAR is cleared, disabling the processor port into the memory. RAMBAR[BDE] is set, enabling the system backdoor port into the memory. If the SRAM requires initialization with instructions or data, perform the following steps:

1. Load the RAMBAR, mapping the SRAM module to the desired location within the address space.
2. Read the source data and write it to the SRAM. Various instructions support this function, including memory-to-memory move instructions, or the MOVEM opcode. The MOVEM instruction is optimized to generate line-sized burst fetches on 0-modulo-16 addresses, so this opcode generally provides maximum performance.
3. After the data loads into the SRAM, it may be appropriate to load a revised value into the RAMBAR with a new set of attributes. These attributes consist of the write-protect and address space mask fields.

The ColdFire processor or an external debugger using the debug module can perform these initialization functions.

### 7.3.1 SRAM Initialization Code

The following code segment describes how to initialize the SRAM. The code sets the base address of the SRAM at 0x8000\_0000 and initializes the SRAM to zeros.

```
RAMBASE      EQU 0x80000000          ;set this variable to 0x80000000
RAMVALID     EQU 0x00000001
```

```

move.l #RAMBASE+RAMVALID,D0      ;load RAMBASE + valid bit into D0.
movec.l D0, RAMBAR               ;load RAMBAR and enable SRAM

```

The following loop initializes the entire SRAM to zero:

```

lea.l  RAMBASE,A0                ;load pointer to SRAM
move.l #16384,D0                 ;load loop counter into D0 (SRAM size/4)

SRAM_INIT_LOOP:
clr.l  (A0)+                     ;clear 4 bytes of SRAM
subq.l #4,D0                     ;decrement loop counter
bne.b SRAM_INIT_LOOP             ;if done, then exit; else continue looping

```

### 7.3.2 Power Management

As noted previously, depending on the RAMBAR-defined configuration, instruction fetch and operand read accesses may be sent to the SRAM and cache simultaneously. If the access maps to the SRAM module, it sources the read data and the cache access is discarded. If the SRAM is used only for data operands, setting the AS<sub>n</sub> bits associated with instruction fetches can decrease power dissipation. Additionally, if the SRAM contains only instructions, masking operand accesses can reduce power dissipation. [Table 7-3](#) shows examples of typical RAMBAR settings.

**Table 7-3. Typical RAMBAR Setting Examples**

Data Contained in SRAM	RAMBAR[7:0]
Instruction Only	0x2B
Data Only	0x35
Instructions and Data	0x21



# Chapter 8

## Clock Module

### 8.1 Introduction

The clock module allows the device to be configured for one of several clocking methods. Clocking modes include internal phase-locked loop (PLL) clocking with an external clock reference or an external crystal reference supported by an internal crystal amplifier. The PLL can also be disabled, and an external oscillator can directly clock the device. The clock module contains:

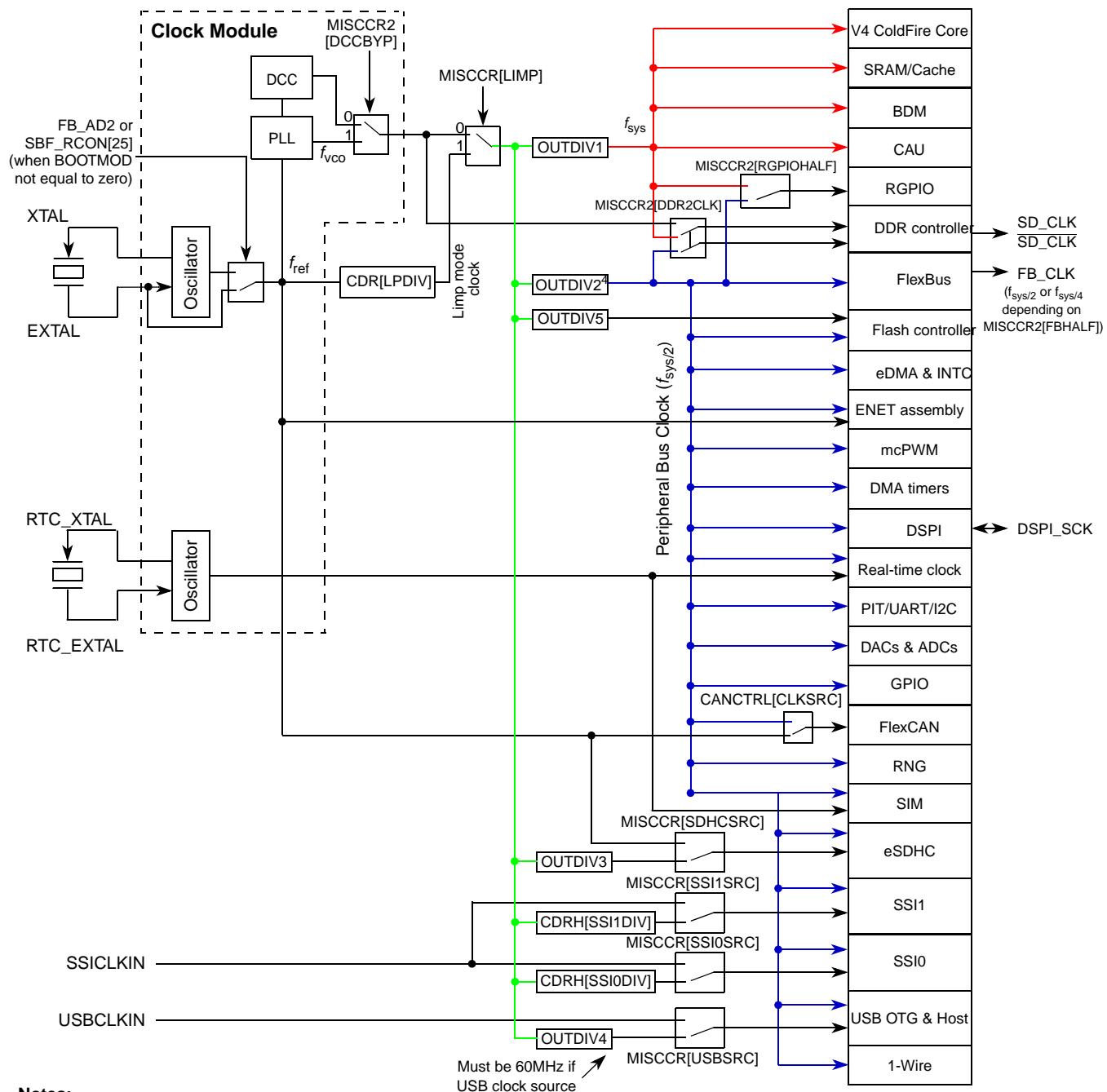
- Crystal amplifier and oscillator (OSC)
- Phase-locked loop (PLL)
- Status and control registers
- Control logic

#### NOTE

Throughout this manual,  $f_{sys}$  refers to the core frequency and  $f_{sys/2}$  refers to the internal bus frequency.

[Figure 8-1](#) is a high-level representation of clock connections. The exact functionality of the blocks is not illustrated (clocks to individual modules may be disabled via the peripheral power management registers as described in [Chapter 9, “Power Management”](#)).

## Clock Module



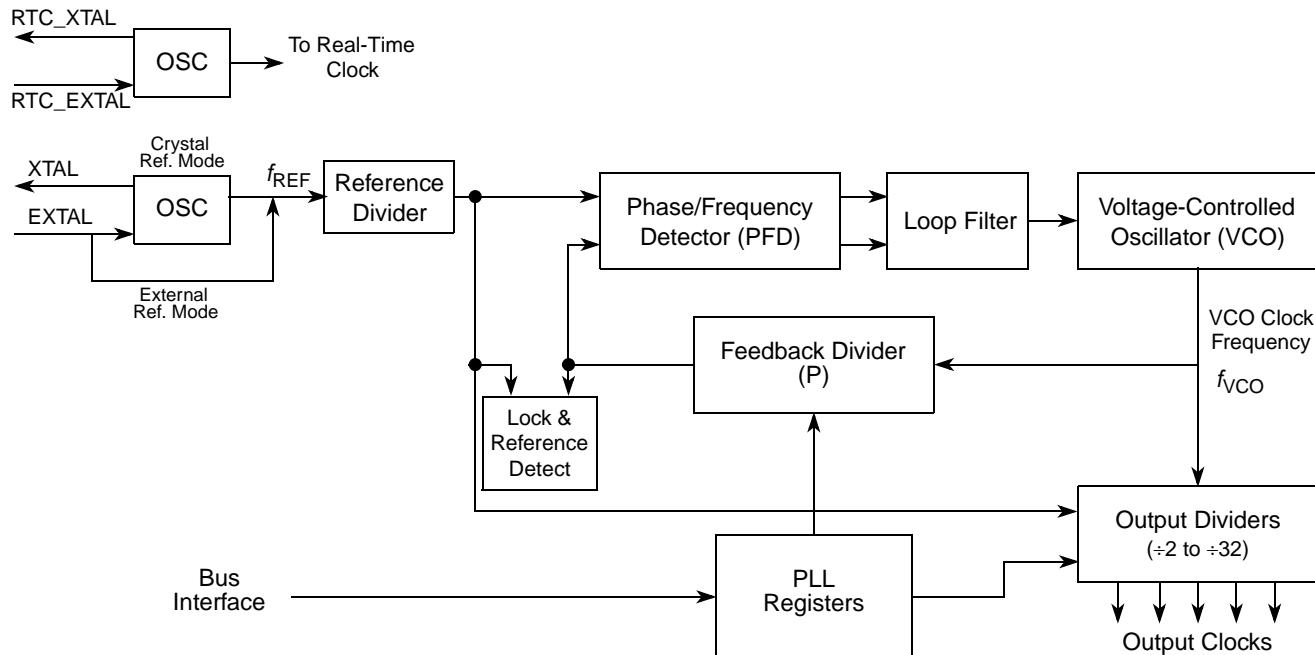
### Notes:

- 1 The DDR controller is disabled in limp mode. The USB controllers are essentially disabled, as they are able to capture a wake-up event to bring the device out of limp mode.
- 2 The DDR controller, flash controller, eSDHC, SSIs, USB, and real time clock contain some logic that uses the  $f_{sys/2}$  clock, in addition to the module-specific clock.
- 3 The OUTDIV $n$  fields are bypassed when in limp mode.
- 4 The OUTDIV2 field controls the divider for the peripheral bus clock ( $f_{sys/2}$ ), which must be half the core clock ( $f_{sys}$ ) frequency.

Figure 8-1. Device Clock Connections

## 8.1.1 Block Diagram

Figure 8-2 shows the clock module block diagram.



**Figure 8-2. Clock Module Diagram**

## 8.1.2 Features

Features of the clock module include:

- 14–50 MHz input clock
- Programmable frequency multiplication factor settings generating voltage-controlled oscillator (VCO) frequencies from 240–500 MHz, resulting in a core frequency of 7.5 MHz ( $f_{vco} \div 32$ ) to 250 MHz (maximum rated frequency).
- Five user-programmable output dividers to produce the following clocks
  - Core clock ( $f_{sys}$ )
  - Bus ( $f_{sys}/2$ )
  - eSDHC clock (250 MHz max)
  - NAND flash controller clock (80 MHz max)
  - USB clock (60 MHz)
- Loss-of-lock detection and reset
- Loss-of-clock detection and reset
- Support for low-power modes
- Direct clocking of system by input clock, bypassing the PLL
- Reference crystal oscillator for the real time clock (RTC) module. Input clock used is programmable within the RTC.

### 8.1.3 Modes of Operation

The PLL operational mode must be configured during reset. The reset configuration pins must be driven to the appropriate state for the desired mode from the time RSTOUT asserts until it negates. Refer to [Chapter 10, “Chip Configuration Module \(CCM\)”,](#) for more details.

The clock module can operate in normal PLL mode with crystal reference, normal PLL mode with external reference, and input-clock limp mode.

#### 8.1.3.1 Normal PLL Mode with Crystal Reference

In normal mode with a crystal reference, the PLL receives an input clock frequency (14–50 MHz) from the crystal oscillator circuit and multiplies the frequency to create the PLL output clock. It can synthesize frequencies ranging from 240–500 MHz. In serial boot mode, this range is programmable and supports a larger range of synthesized frequencies.

You must supply a crystal oscillator within the appropriate input frequency range, the crystal manufacturer’s recommended external support circuitry, and short signal route from the device to the crystal.

#### 8.1.3.2 Normal PLL Mode with External Reference

This second mode is the same as [Section 8.1.3.1, “Normal PLL Mode with Crystal Reference,”](#) except EXTAL is driven by an external clock generator rather than a crystal oscillator. However, the input frequency range is the same as the crystal reference. In this mode, an internal pull-up resistor is enabled on XTAL.

To enter normal mode with external clock generator reference, the PLL configuration must be set at reset by overriding the default reset configuration. See [Chapter 10, “Chip Configuration Module \(CCM\),”](#) for details on setting the device for external reference (oscillator bypass mode).

#### 8.1.3.3 Input Clock (Limp) Mode

Depending on the reset configuration (see [Chapter 10, “Chip Configuration Module \(CCM\)”,](#)) or by setting MISCCR[LIMP] bit, the device is placed into a low-frequency limp mode, in which the PLL is placed in reset and the device runs from a factor of the input clock (EXTAL). In this mode, EXTAL feeds a 5-bit programmable counter that divides the input clock by  $2^n$ , where  $n$  is the value of the programmable counter field, CDR[LPDIV]. For more information on programming the divider, see [Chapter 10, “Chip Configuration Module \(CCM\)”,](#). The programmed value of the divider may be changed without glitches or otherwise negative affects to the system.

While in this mode, the PLL is placed in reset mode to reduce overall system power consumption. A 2:1 ratio is still maintained between the core and the primary bus clock. Because they do not function at speeds as low as the minimum input-clock frequency, the SDRAM controller is not functional in limp mode. Also, the USB controllers must source their system clocks through alternate, external sources.

When switching from limp mode to normal functional mode, you must ensure that any peripheral transactions in progress (Ethernet frame reception/transmission) are allowed to complete to avoid data loss or corruption.

Entering limp mode via the MISCCR[LIMP] bit requires a special procedure for the SDRAM module. As noted above, the SDRAM controller is disabled in limp mode, so follow these two critical steps before setting the MISCCR[LIMP] bit:

1. Code execution must be transferred to another memory resource. Primary options are:
  - Memory device attached to the FlexBus boot chip-select
  - On-chip SRAM (but not the CPU cache, as it may have to be flushed upon limp mode entrance or exit)
2. The SDRAM controller must be placed in self-refresh mode to avoid data loss while the SDRAMC shuts down.

### 8.1.3.4 Low-power Mode Operation

This subsection describes the clock module operation in low-power and halted modes of operation. Low-power modes are described in [Chapter 9, “Power Management”](#). Table 8-1 shows the clock module operation in low-power modes.

**Table 8-1. Clock Module Operation in Low-power Modes**

Low-power Mode	Clock Operation	Mode Exit
Wait	Clocks sent to peripheral modules only. CPU is stopped.	Clock module does not cause exit, but normal clocking resumes upon mode exit
Doze	Clocks sent to peripheral modules only. CPU is stopped.	Clock module does not cause exit, but normal clocking resumes upon mode exit
Stop	All system clocks disabled	Clock module does not cause exit, but clock sources are re-enabled and normal clocking resumes upon mode exit
Halted	Normal	Clock module does not cause exit

In wait and doze modes, the system clocks to the peripherals are enabled, and the clocks to the core are stopped. Each module can disable its clock locally at the module level. The SRAM clock is not gated in wait and doze modes. This allows other masters (like the Ethernet assembly) to access the SRAM if they are operating in wait and doze modes.

In stop mode, all system clocks are disabled (except the real-time clock which continues to run via its external clock). There are several options for enabling or disabling the PLL or crystal oscillator in stop mode, compromising between stop mode current and wake-up recovery time. The PLL can be disabled in stop mode, but requires a wake-up period before it relocks. The oscillator can also be disabled during stop mode, but it requires a wake-up period to restart.

When the PLL is enabled in stop mode (LPCR[STPMD] = 00), the external FB\_CLK signal can support systems using FB\_CLK as the clock source. For more information about operating the PLL in stop mode, see [Section 9.2.5, “Low-Power Control Register \(LPCR\)”](#).

There is also a fast wake-up option for quickly enabling the system clocks during stop recovery (LPCR[FWKUP]). This eliminates the wake-up recovery time but at the risk of sending a potentially unstable clock to the system.

## 8.2 Memory Map/Register Definition

The PLL programming model consists of the following:

**Table 8-2. PLL Memory Map**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0C_0000	PLL Control Register (PLL_CR)	32	R/W	See section	<a href="#">8.2.1/8-6</a>
0xFC0C_0004	PLL Divider register (PLL_DR)	32	R/W	See section	<a href="#">8.2.2/8-7</a>
0xFC0C_0008	PLL Status Register (PLL_SR)	32	R/W	0x0000_0000	<a href="#">8.2.3/8-9</a>

### 8.2.1 PLL Control Register (PLL\_CR)

The PLL\_CR register controls the feedback and reference dividers for generating the core and bus clocks. For details on altering these values after reset, see [Section 8.3.1, “PLL Frequency Multiplication Factor Select.”](#) PLL\_CR also contains control bits for loss-of-clock and loss-of-lock detection.

#### NOTE

Only alter the PLL\_CR[LOLEN] bit when the device is in limp mode.

Address: 0xFC0C_0000 (PLL_CR)																Access: User read/write								
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	LOC	IRQ	LOC	RE	LOC	EN		
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	
<hr/>																								
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	FBKDIV							
W		LOL	LOL	LOL	0	REFDIV				0	0	FBKDIV												
Reset	0	0	0	1	0	See note				0	0	See note												

**Note:** The reset values of REFDIV and FBKDIV depend on the reset configuration:

If default configuration, REFDIV = 000 and FBKDIV = 0x09.

If serial boot, REFDIV = {0,SBF\_RCON[23:22]} and FBKDIV = SBF\_RCON[21:16].

If parallel configuration, REFDIV = 000 and FBKDIV depends on the value of FB\_AD[7:6] (See [Chapter 10, “Chip Configuration Module \(CCM\).](#)

**Figure 8-3. PLL Control Register (PLL\_CR)**

**Table 8-3. PLL\_CR Field Descriptions**

Field	Description
31–19	Reserved, must be cleared.
18 LOCIRQ	Loss of clock interrupt request. Determines how the processor handles a loss-of-clock condition when LOCEN is set. If LOCEN is cleared, this bit has no effect. If PLL_SR[LOCF] is set, setting LOCIRQ causes an immediate interrupt request. 0 Ignore loss of clock, interrupt not requested 1 Request interrupt on loss of clock

**Table 8-3. PLL\_CR Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>
17 LOCRE	Loss of clock reset enable. If LOSEN is set, determines how the processor handles a loss-of-clock condition. If LOSEN is cleared, this bit has no effect. Setting LOCRE causes an immediate reset request upon detection of loss of clock. 0 Ignore loss of clock, reset not requested 1 Request reset on loss of clock
16 LOCEN	Loss of clock enable. Enables the loss-of-clock feature. 0 Loss of clock disabled 1 Loss of clock enabled
15	Reserved, must be cleared.
14 LOLIRQ	Loss of lock interrupt request. If LOLEN is set, determines how the processor handles a loss-of-lock condition. If LOLEN is cleared, this bit has no effect. The PLL must be locked when LOLIR is enabled to avoid an immediate interrupt request. 0 Ignore loss of lock, interrupt not requested 1 Request interrupt on loss of lock
13 LOLRE	Loss of lock reset enable. If LOLEN is set, determines how the processor handles a loss-of-lock condition. If LOLEN is cleared, this bit has no effect. The PLL must be locked when LOLRE is enabled to avoid an immediate reset request. 0 Ignore loss of lock, reset not requested 1 Request reset on loss of lock
12 LOLEN	Loss of lock enable. Enables the loss-of-lock feature. 0 Loss of lock disabled 1 Loss of lock enabled <b>Note:</b> Only change this bit when the device is in limp mode.
11	Reserved, must be cleared.
10–8 REFDIV	Reference divider setting. Along with FBKDIV, determines the VCO clock frequency. See the FBKDIV field for details. 000 1 001 2 010 Reserved 011 Reserved 100 Reserved Else Reserved. A divider value of one is used.
7–6	Reserved, must be cleared.
5–0 FBKDIV	Feedback divider setting. Along with REFDIV, determines the VCO clock frequency. The feedback divider is the value of this bit field plus 1. The resulting VCO frequency is shown below: $f_{VCO} = \frac{f_{REF} \times (FBKDIV + 1)}{2^{REFDIV}}$ <span style="float: right;"><b>Eqn. 8-1</b></span>

## 8.2.2 PLL Divider Register (PLL\_DR)

The PLL\_DR register controls the output dividers for generating the core, bus, eSDHC, NAND flash controller, and USB clocks. For details on altering these values after reset, see [Section 8.3.1, “PLL Frequency Multiplication Factor Select”](#).

## **NOTE**

A single longword (32-bit) write to the PLL\_DR register is required. If back-to-back word or longword writes are attempted, some of the clocks in the system change frequency before others, which can cause the device to hang.

Address: 0xFC0C\_0004 (PLL\_DR)

Access: User read/write

**Note:** The reset values of OUTDIV5, OUTDIV2, and OUTDIV1 depend on the reset configuration.

If default or parallel configuration, OUTDIV5 = 0x1F, OUTDIV2 = 3, and OUTDIV1 = 1.

If serial boot, OUTDIV5 = SBF\_RCON[14:10], OUTDIV2 = SBF\_RCON[9:5], and OUTDIV1 = SBF\_RCON[4:0].

**Figure 8-4. PLL Divider Register (PLL\_DR)**

**Table 8-4. PLL\_DR Field Descriptions**

Field	Description
31–26	Reserved, must be cleared.
25–21 OUTDIV5	Output divider for generating the NAND flash controller clock frequency. The divider is the value of this bit field plus 1. A value of zero disables this clock. $f_{NFC} = \frac{f_{VCO}}{\text{OUTDIV5} + 1}$ <b>Note:</b> The OUTDIV5 resulting frequency must not be greater than 80 MHz.
20–16 OUTDIV4	Output divider for generating the USB controllers clock frequency. The divider is the value of this bit field plus 1. A value of zero disables this clock. $f_{USB} = \frac{f_{VCO}}{\text{OUTDIV4} + 1}$ <b>Note:</b> If used as the USB clock source, the OUTDIV4 resulting frequency must be 60 MHz. This may require that $f_{VCO}$ be less than the maximum.
15	Reserved, must be cleared.
14–10 OUTDIV3	Output divider for generating the eSDHC clock frequency. The divider is the value of this bit field plus 1. A value of zero disables this clock. $f_{eSDHC} = \frac{f_{VCO}}{\text{OUTDIV3} + 1}$ <b>Note:</b> The OUTDIV3 resulting frequency must not be greater than 250 MHz.

**Table 8-4. PLL\_DR Field Descriptions (continued)**

Field	Description
9–5 OUTDIV2	<p>Output divider for generating the internal bus clock frequency, including the FlexBus clock (FB_CLK). The divider is the value of this bit field plus 1. A value of zero disables this clock.</p> $f_{\text{SYS}/2} = \frac{f_{\text{VCO}}}{2} = \frac{f_{\text{VCO}}}{\text{OUTDIV2} + 1}$ <p><b>Note:</b> If the core clock is enabled (<math>\text{OUTDIV1} \neq 0</math>), the internal bus clock frequency must be half the core clock frequency. Therefore, the valid OUTDIV2 settings are limited to 0, 3, 5, 7, 9, 11, 13, and 15. See OUTDIV1 bit description for more details.</p>
4–0 OUTDIV1	<p>Output divider for generating the core clock frequency. The divider is the value of this bit field plus 1. A value of zero disables this clock.</p> $f_{\text{SYS}} = \frac{f_{\text{VCO}}}{\text{OUTDIV1} + 1}$ <p><b>Note:</b> If the internal bus clock is enabled, the core clock frequency must be two times the internal bus clock frequency. Therefore, the valid OUTDIV1 values are limited to 0x0–0x7.</p>

### 8.2.3 PLL Status Register (PLL\_SR)

The PLL status register provides indicators for PLL operating mode, loss-of-lock, and loss-of-clock status. You must write a one to the loss-of-lock and loss-of-clock flag bits to clear them. All other bits in the PLL\_SR are read-only.

Address: 0xFC0C_0008 (PLL_SR)																Access: User read/write									
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	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	
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	MODE								
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

**Figure 8-5. PLL Status Register (PLL\_SR)****Table 8-5. PLL\_SR Field Descriptions**

Field	Description
31–10	Reserved, must be cleared.
9 LOCFF	<p>Loss-of-clock flag. If PLL_CR[LOCIRQ, LOCF] are set, LOCFF is set upon loss-of-clock detection and is subsequently used to generate the interrupt request. To clear LOCFF, assert reset or write a one to this bit. Writing zero has no effect. This bit is sticky; if clocks are restored, the bit remains set until writing a one or asserting reset.</p> <p>0 Interrupt service not requested 1 Interrupt service requested</p>

**Table 8-5. PLL\_SR Field Descriptions (continued)**

Field	Description
8 LOC	Loss-of-clock status. Indicates whether the PLL has lost its reference clock. If you read this bit at the same time the loss-of-clock condition occurs, the bit does not reflect the current loss-of-clock condition. If a loss-of-clock condition occurs which sets this bit and the clocks later return to normal, this bit is cleared. A loss-of-clock condition can only be detected if PLL_CR[LOCEN] is set. 0 Clocks are operating normally 1 Clocks have failed due to a reference failure
7	Reserved, must be cleared.
6 LOLF	Loss-of-lock flag. If PLL_CR[LOLIR, LOLEN] are set, LOLF is set upon loss of lock and is subsequently used to generate the interrupt request. To clear LOLF, assert reset or write a one to this bit. Writing 0 has no effect. This bit is sticky; if lock is reacquired, the bit remains set until writing a one or asserting reset. 0 Interrupt service not requested 1 Interrupt service requested
5 LOCKS	Sticky PLL lock status bit. Set by the lock-detect circuitry when the PLL acquires lock. If the PLL loses lock, this bit is cleared and remains cleared even after the PLL re-locks. If you read this bit when the PLL simultaneously loses lock, the bit does not reflect the current loss-of-lock condition. 0 PLL has lost lock since last reset 1 PLL has not lost lock since last reset
4 LOCK	PLL lock status bit. Indicates if the PLL has acquired lock. If PLL_CR[LOLEN] is set, PLL lock occurs when the synthesized frequency matches to within 0.75% of the programmed frequency. The PLL loses lock when a frequency deviation of greater than ~1.5% occurs. If PLL_CR[LOLEN] is cleared, PLL lock is declared following 2000 reference clock cycles after negation of reset. The LOCK bit remains set until reset is asserted. In functional mode, LOCK is asserted following 16 reference clock cycles after negation of reset. The LOCK bit remains set until reset is asserted. If you read this bit when the PLL simultaneously loses lock, the bit does not reflect the current loss-of-lock condition. 0 PLL is unlocked 1 PLL is locked
3	Reserved, must be cleared.
2–0 MODE	PLL mode setting. Read-only bit indicating the current PLL mode. 000 PLL functional mode Else Reserved

## 8.3 Functional Description

This subsection provides a functional description of the clock module.

### 8.3.1 PLL Frequency Multiplication Factor Select

The frequency multiplication factor of the PLL is defined by the feedback, reference, and output dividers specified in the PLL\_CR and PLL\_DR registers. An example equation for the core frequency is given below:

$$f_{\text{SYS}} = \frac{f_{\text{VCO}}}{\text{PLL\_DR}[\text{OUTDIV1}] + 1} \quad \text{Eqn. 8-7}$$

where  $f_{\text{sys}}$  is the clock frequency of the ColdFire core and  $f_{\text{REF}}$  is the PLL clock source as shown in Figure 8-1. The allowable range of values for OUTDIV $n$  is 1–15. The other clocks on the processor are

configurable in a similar fashion. However, there are various dependencies. See [Section 8.2.2, “PLL Divider Register \(PLL\\_DR\),”](#) for details.

The PLL\_DR[OUTDIV $n$ ] fields can be changed during normal operation or when the device is in limp mode. After a new value is written to the PLL\_DR, the PLL synchronizes the new value of the PLL\_DR with the VCO clock domain. Then, the transition from the old divider value to the new divider value takes place, such that the PLL output clocks remain glitch-free. During the adjustment to the new divider value, a PLL output clock may experience an intermediate transition while the divider values are being synchronized. Following the transition period, all output clocks begin toggling at the new divider values simultaneously. The transition from the old divider value to the new divider value takes no more than 100 ns. Because the output divider transition takes a period of time to change, the PLL\_CR may not be written back-to-back without waiting 100 ns between writes.

### 8.3.2 PLL Frequency Synthesis

The frequency synthesis of the PLL is defined by the feedback divider (P-divider) and reference divider (Q-divider) in [Equation 8-8](#):

$$f_{VCO} = f_{REF} \times \frac{P}{Q} = f_{REF} \times \left( \frac{\text{PLL\_CR}[FBKDIV] + 1}{2^{\text{PLL\_CR}[REFDIV]}} \right) \quad \text{Eqn. 8-8}$$

The allowable range of values for FBKDIV is 0–63 and REFDIV is 0–1. The following constraints must be met:

- $f_{VCO}$  must be in the range 240–500 MHz.
- $f_{REF}$  must be in the range 14–50 MHz.
- $\frac{f_{REF}}{2^{\text{PLL\_CR}[REFDIV]}}$  must be in the range 14–50 MHz.

The PLL\_CR can be modified during normal operation or while the device is in limp mode. See [Section 8.2.1, “PLL Control Register \(PLL\\_CR\),”](#) for further information about FBKDIV and REFDIV.

### 8.3.3 Lock Conditions

When PLL\_CR[LOLEN] is set, the lock-detect logic monitors the PFD reference clock and the feedback clock 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 PLL\_SR[LOCK].

The lock-detect function uses two counters that are clocked by the divided reference and feedback clocks 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 still match, the lock criteria is relaxed by one count, and the system is notified that the PLL has achieved frequency lock.

After lock is detected, the lock circuitry continues to monitor the reference and feedback clocks using the alternate count-and-compare process. If the counters do not match at any comparison time, then PLL\_SR[LOCK] 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.

When PLL\_CR[LOLEN] is cleared, PLL\_SR[LOCK] is set following a count of 2000 reference clock cycles, indicating sufficient time has elapsed for the PLL to reach a locked state. PLL\_SR[LOCK] is cleared by asserting system reset.

In limp mode, since the PLL is disabled, there is no PLL lock status.

### 8.3.4 Loss-of-Lock Detection

When the PLL achieves lock following a system reset, PLL\_SR[LOCK, LOCKS] are set. When the PLL loses lock, both bits are cleared. 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. Therefore, during the re-locking process, the system clock frequency is not well defined and may exceed the maximum system frequency. This violates the system clock timing specifications. Due to this condition, we recommend using the loss-of-lock reset functionality as described in [Section 8.3.4.1, “Loss-of-Lock Reset Request”](#).

When the PLL has re-locked, the PLL does not update the PLL\_SR[LOCKS] bit indicating lock has been lost since a system reset. This bit is sticky and must be cleared by writing a one to it before the PLL writes the register again.

#### 8.3.4.1 Loss-of-Lock Reset Request

The PLL can assert reset when a loss-of-lock condition occurs by programming the PLL\_SR[LOLRE] bit. Because the PLL\_SR[LOCK, LOCKS] bits are cleared after reset, the reset status register (RSR) must be read to determine a loss-of-lock condition occurred. See [Section 12.3.2, “Reset Status Register \(RSR\)”](#), for more information on the RSR register. To exit reset in PLL mode, the reference must be present and the PLL must acquire lock.

#### 8.3.4.2 Loss-of-Lock Interrupt Request

The PLL can request an interrupt when a loss-of-lock condition occurs by programming the PLL\_SR[LOLIRQ] bit. If this bit is set and a loss of lock is detected, PLL\_SR[LOLF] is set, requesting an interrupt to the processor. The LOLF bit is sticky and remains set until it is cleared by a system reset or by writing a one to it. LOLIRQ provides information to the lock-detect logic to let it know if an interrupt should be generated upon loss of lock.

### 8.3.5 Loss-of-Clock Detection

When PLL\_CR[LOCEN] is set, the loss-of-reference logic monitors the PFD reference clock for a loss-of-clock condition. For every rising edge of the PFD reference clock, a logic 1 is written to a register, which is reset every eight feedback clocks. This register is read just before being reset. If found to be a logic 0, a loss-of-clock condition is declared. The reset cycle of eight feedback clocks was chosen to prevent minor phase variations between the PFD reference and feedback clocks triggering loss of clock. When loss of reference is detected, PLL\_SR[LOC] is set. After the reference clock is restored, PLL\_SR[LOC] is cleared.

When PLL\_CR[LOCEN] is cleared, the loss-of-reference logic is disabled.

### 8.3.5.1 Loss-of-Clock Reset Request

When you set PLL\_SR[LOCRE, LOCEN], the PLL requests a reset when a loss-of-clock condition occurs. Since the PLL\_SR[LOC] bit is cleared after reset, the reset controller's RSR register must be read to determine that a loss-of-clock condition occurred. See [Section 12.3.2, “Reset Status Register \(RSR\)”](#), for more information on the RSR register. To exit reset in PLL mode, the reference must be present and the PLL must acquire lock. PLL\_SR[LOCRE] is ignored when PLL\_SR[LOCEN] is cleared.

### 8.3.5.2 Loss-of-Clock Interrupt Request

When you set PLL\_CR[LOCIRQ, LOCEN], the PLL requests an interrupt when a loss-of-clock condition occurs. If this bit is set and loss of lock is detected, the PLL sets PLL\_SR[LOCF] and asserts an interrupt request to the processor. The LOCF bit is sticky and remains set until it is cleared by writing a one to it or by asserting a system reset. PLL\_SR[LOCIRQ] is ignored when PLL\_SR[LOCEN] is cleared.

## 8.3.6 System Clock Modes

[Table 8-6](#) shows the clock-out to clock-in frequency relationships for the possible system clock modes.

**Table 8-6. System Clock And Reference Clock Relationship**

Clock Mode	PLL option <sup>1</sup>	Reference
Normal PLL mode	$f_{SYS} = \frac{f_{VCO}}{PLL\_DR[OUTDIV1] + 1}$	<a href="#">Section 8.1.3.1, “Normal PLL Mode with Crystal Reference”</a> and <a href="#">Section 8.1.3.2, “Normal PLL Mode with External Reference”</a>
Limp mode	$f_{SYS} = \frac{f_{REF}}{2^{MISCCR[LPDIV]}}$	<a href="#">Section 8.1.3.3, “Input Clock (Limp Mode”</a>

<sup>1</sup>  $f_{ref}$  = input reference frequency,  $f_{VCO}$  = VCO frequency  
MISCCR[LPDIV] ranges from 0 to 15

## 8.3.7 Clock Operation During Reset

This section describes the PLL reset operation. Power-on reset and normal reset are described.

### 8.3.7.1 Power-On Reset (POR)

After VDD\_PLL and the input clock are within specification, the PLL is held in reset for at least ten input clock cycles to initialize the PLL. The reset configuration signals are used to select the multiply factor of the PLL and the reset state of the PLL registers. While in reset, the PLL input clock is output to the device. After RESET de-asserts, PLL output clocks generate; however, until the PLL\_SR[LOCK] bit is set, the PLL output clock frequencies are not stable and within specification. When this bit is set, the PLL is in frequency lock.

### 8.3.7.2 External Reset

When RESET asserts, the PLL input clock outputs to the device, and the PLL does not begin acquiring lock until RESET is negated. The PLL\_SR[LOCK] bit is cleared and remains cleared while the PLL is acquiring lock.

#### CAUTION

When running in an unlocked state, the clocks the PLL generate are not guaranteed to be stable and may exceed the maximum specified frequency.

# Chapter 9

## Power Management

### 9.1 Introduction

This chapter explains the low-power operation of the device.

#### 9.1.1 Features

These features support low-power operation:

- Four operation modes: run, wait, doze, and stop
- Ability to shut down most peripherals independently
- Ability to shut down clocks to most peripherals independently
- Ability to run the device in low-frequency limp mode
- Ability to shut down the external FB\_CLK pin

### 9.2 Memory Map/Register Definition

The power management programming model consists of registers from the SCM and CCM memory space:

Table 9-1. Power Management Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
<b>Supervisor Access Only Registers<sup>1</sup></b>					
0xFC04_0013	Wakeup Control Register (WCR)	8	R/W	0x00	<a href="#">9.2.1/9-2</a>
0xFC04_002C	Peripheral Power Management Set Register 0 (PPMSR0)	8	W	0x00	<a href="#">9.2.2/9-3</a>
0xFC04_002D	Peripheral Power Management Clear Register 0 (PPMCR0)	8	W	0x00	<a href="#">9.2.3/9-4</a>
0xFC04_002E	Peripheral Power Management Set Register 1 (PPMSR1)	8	W	0x00	<a href="#">9.2.2/9-3</a>
0xFC04_002F	Peripheral Power Management Clear Register 1 (PPMCR1)	8	W	0x00	<a href="#">9.2.3/9-4</a>
0xFC04_0030	Peripheral Power Management High Register 0 (PPMHR0)	32	R/W	0x7E1E_FBFF	<a href="#">9.2.4/9-5</a>
0xFC04_0034	Peripheral Power Management Low Register 0 (PPMLR0)	32	R/W	0xFFDE_F300	<a href="#">9.2.4/9-5</a>
0xFC04_0038	Peripheral Power Management High Register 1 (PPMHR1)	32	R/W	0xFFFF_FFCF	<a href="#">9.2.4/9-5</a>
0xFC04_003C	Peripheral Power Management Low Register 1 (PPMLR1)	32	R/W	0x3F00_C0F4	<a href="#">9.2.4/9-5</a>
0xEC09_0007	Low-Power Control Register (LPCR)	8	R/W	0x00	<a href="#">9.2.5/9-9</a>
0xEC09_000E	Miscellaneous control register (MISCCR) <sup>2</sup>	16	R/W	0x183D	<a href="#">10.3.4/10-5</a>

**Table 9-1. Power Management Memory Map (continued)**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xEC09_0010	Clock divider register high (CDRH) <sup>2</sup>	16	R/W	0x0101	10.3.5/10-7
0xEC09_0012	Clock divider register low (CDRL) <sup>2</sup>	16	R/W	0x0000	10.3.6/10-8

<sup>1</sup> User access to supervisor only address locations have no effect and result in a bus error

<sup>2</sup> The MISCCR and CDR{H,L} registers are described in [Chapter 10, “Chip Configuration Module \(CCM\).](#)”

## 9.2.1 Wake-up Control Register (WCR)

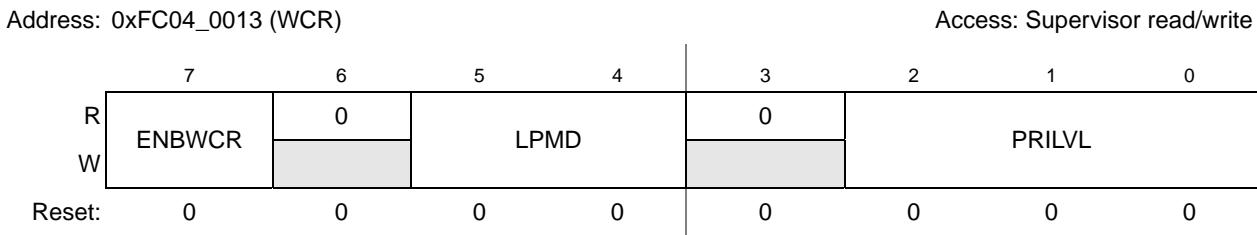
Implementation of low-power stop mode and exit from a low-power mode via an interrupt requires communication between the core and logic associated with the interrupt controller. The WCR enables entry into low-power modes and includes the interrupt level setting needed to exit a low-power mode.

### NOTE

The setting of the low-power mode select field, WCR[LPMD], determines which low-power mode the device enters when a STOP instruction is issued.

Sequence of operations generally needed to enable this functionality:

1. The WCR register is programmed, setting the ENBWCR bit and the desired interrupt priority level.
2. At the appropriate time, the processor executes the privileged STOP instruction. After the processor stops execution, it asserts a specific processor status (PST) encoding. Issuing the STOP instruction when the WCR[ENBWCR] is set causes the SCM to enter the mode specified in WCR[LPMD].
3. The low power mode control logic processes the entry into a low power mode, and the appropriate clocks (usually those related to the high-speed processor core) are disabled.
4. After entering the low-power mode, the interrupt controller enables a combinational logic path which evaluates any unmasked interrupt requests. The device waits for an event to generate an interrupt request with a priority level greater than the value programmed in WCR[PRILVL].
5. After an appropriately high interrupt request level arrives, the interrupt controller signals its presence, and the SCM responds by asserting the request to exit low-power mode.
6. The low-power mode control logic senses the request signal and re-enables the appropriate clocks.
7. With the processor clocks enabled, the core processes the pending interrupt request.

**Figure 9-1. Wake-up Control Register (WCR)**

**Table 9-2. WCR Field Descriptions**

<b>Field</b>	<b>Description</b>																
7 ENBWCR	Enable low-power mode entry. The mode entered is specified in WCR[LPMD]. 0 Low-power mode entry is disabled 1 Low-power mode entry is enabled.																
6	Reserved, must be cleared.																
5–4 LPMD	Low-power mode select. Used to select the low-power mode the chip enters after the ColdFire core executes the STOP instruction. To take effect, write these bits prior to instruction execution. The LPMD bits are readable and writable in all modes. 00 Run 01 Doze 10 Wait 11 Stop <b>Note:</b> If WCR[LPMD] is cleared, the device stops executing code upon a STOP instruction. However, no clocks are disabled.																
3	Reserved, must be cleared.																
2–0 PRILVL	Exit low-power mode interrupt priority level. This field defines the interrupt priority level to exit the low-power mode: <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th><b>PRILVL</b></th><th><b>Interrupt Level Needed to Exit Low-Power Mode</b></th></tr> </thead> <tbody> <tr> <td>000</td><td>Any interrupt request exits low-power mode</td></tr> <tr> <td>001</td><td>Interrupt request levels [2-7] exit low-power mode</td></tr> <tr> <td>010</td><td>Interrupt request levels [3-7] exit low-power mode</td></tr> <tr> <td>011</td><td>Interrupt request levels [4-7] exit low-power mode</td></tr> <tr> <td>100</td><td>Interrupt request levels [5-7] exit low-power mode</td></tr> <tr> <td>101</td><td>Interrupt request levels [6-7] exit low-power mode</td></tr> <tr> <td>11x</td><td>Interrupt request level [7] exits low-power mode</td></tr> </tbody> </table>	<b>PRILVL</b>	<b>Interrupt Level Needed to Exit Low-Power Mode</b>	000	Any interrupt request exits low-power mode	001	Interrupt request levels [2-7] exit low-power mode	010	Interrupt request levels [3-7] exit low-power mode	011	Interrupt request levels [4-7] exit low-power mode	100	Interrupt request levels [5-7] exit low-power mode	101	Interrupt request levels [6-7] exit low-power mode	11x	Interrupt request level [7] exits low-power mode
<b>PRILVL</b>	<b>Interrupt Level Needed to Exit Low-Power Mode</b>																
000	Any interrupt request exits low-power mode																
001	Interrupt request levels [2-7] exit low-power mode																
010	Interrupt request levels [3-7] exit low-power mode																
011	Interrupt request levels [4-7] exit low-power mode																
100	Interrupt request levels [5-7] exit low-power mode																
101	Interrupt request levels [6-7] exit low-power mode																
11x	Interrupt request level [7] exits low-power mode																

## 9.2.2 Peripheral Power Management Set Registers (PPMSR0, PPMSR1)

The PPMSR $n$  registers provide a simple mechanism to set a given bit in the PPM{H,L}R $n$  registers to disable the clock for a given peripheral module without needing to perform a read-modify write on the PPMR. The data value on a register write causes the corresponding bit in the PPM{H,L}R $n$  to set. The SAMCD bit provides a global set function forcing the entire contents of the PPMR to set, disabling all peripheral module clocks. Reads of these registers return all zeroes.

- PPMSR0 affects PPMHR0 and PPMLR0
- PPMSR1 affects PPMHR1 and PPMLR1

Address: 0xFC04\_002C (PPMSR0)  
0xFC04\_002E (PPMSR1)

Access: Supervisor Write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		SAMCD	SMCD					
Reset:	0	0	0	0	0	0	0	0

Figure 9-2. Peripheral Power Management Set Registers (PPMSR $n$ )

Table 9-3. PPMSR $n$  Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 SAMCD	Set all module clock disables. 0 Set only those bits specified in the SMCD field 1 Set all bits in PPMRH and PPMRL, disabling all peripheral clocks
5–0 SMCD	Set module clock disable. Set the corresponding bit in PPM{H,L}R, disabling the peripheral clock.

### 9.2.3 Peripheral Power Management Clear Register (PPMCR0, PPMCR1)

The PPMCR $n$  registers provide a simple mechanism to clear a given bit in the PPM{H,L}R $n$  registers, enabling the clock for a given peripheral module without needing to perform a read-modify write on the PPMR. The data value on a register write causes the corresponding bit in the PPM{H,L}R $n$  to clear. The CAMCD bit provides a global clear function, forcing the entire PPMR contents to clear, enabling all peripheral module clocks. Reads of these registers return all zeroes.

- PPMCR0 affects PPMHR0 and PPMLR0
- PPMCR1 affects PPMHR1 and PPMLR1

Address: 0xFC04\_002D (PPMCR0)  
0xFC04\_002F (PPMCR1)

Access: Supervisor Write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		CAMCD	CMCD					
Reset:	0	0	0	0	0	0	0	0

Figure 9-3. Peripheral Power Management Clear Registers (PPMCR $n$ )

Table 9-4. PPMCR $n$  Field Descriptions

Field	Description
7	Reserved, must be cleared.

**Table 9-4. PPMCR $n$  Field Descriptions (continued)**

Field	Description
6 CAMCD	Clear all module clock disables. 0 Clear only those bits specified in the CMCD field 1 Clear all bits in PPMRH and PPMRL, enabling all peripheral clocks
5–0 CMCD	Clear module clock disable. Clear the corresponding bit in PPMR{H,L}, enabling the peripheral clock.

## 9.2.4 Peripheral Power Management Registers (PPMHR{1,0}, PPMLR{1,0})

The PPMR registers provide a bit map for controlling the generation of the peripheral clocks for each decoded address space. Recall each peripheral module is mapped into 16 kByte slots within the memory map. The PPMR registers provide a unique control bit for each address space that defines whether the module clock for the given space is enabled or disabled.

Because the operation of the crossbar switch and the system control module (SCM) are fundamental to the operation of the device, the clocks for these modules cannot be disabled.

Using a read-modify-write to this register directly or indirectly through writes to the PPMSR and PPMCR registers to set/clear individual bits can modify the PPMR individual bits.

### CAUTION

Take extreme caution when setting PPMHR1[CD36] to disable clocking of the CCM, reset controller, and power management modules. This may disable logic to reset the chip and disable the external bus monitor or other logic contained within these blocks.

Address: 0xFC04_0030 (PPMHR0)																Access: Supervisor read/write								
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	CD63	1	1	1	CD56	CD55	CD54	CD53
W																	CD51	CD50	CD49	CD48				
Reset	0	1	1	1	1	1	1	0	0	0	0	1	1	1	1	0								
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	CD47	CD46	CD45	CD44	CD43	CD42	CD39	CD38
W																	CD37	CD36	CD35	CD34	CD33	CD32		
Reset	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	0								

**Figure 9-4. Peripheral Power Management High Register 0 (PPMHR0)****Table 9-5. PPMHR0[CD $n$ ] Assignments**

Slot Number	CD $n$	Peripheral
32	CD32	PIT 0
33	CD33	PIT 1
34	CD34	PIT 2

**Table 9-5. PPMHR0[CDn] Assignments (continued)**

<b>Slot Number</b>	<b>CDn</b>	<b>Peripheral</b>
35	CD35	PIT 3
37	CD37	Edge Port
38	CD38	ADC
39	CD39	DAC0
42	CD42	Robust RTC
43	CD43	SIM
44	CD44	USB OTG
45	CD45	USB Host
46	CD46	DDR SDRAM Controller
47	CD47	SSI0
48	CD48	PLL
49	CD49	RNG
50	CD50	SSI1
51	CD51	eSDHC
53	CD53	MACNET0
54	CD54	MACNET1
55	CD55	Ethernet Switch
56	CD56	Ethernet Switch
63	CD63	NAND Flash Controller

Address: 0xFC04\_0034 (PPMLR0)

Access: Supervisor read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CD31	CD30	CD29	CD28	CD27	CD26	CD25	CD24	CD23	CD22	0	CD20	CD19	CD18	CD17	0
W	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	0
Reset	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CD15	CD14	1	1	0	0	CD9	CD8	0	0	0	0	0	CD2	0	0
W																
Reset	1	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0

**Figure 9-5. Peripheral Power Management Low Register 0 (PPMLR0)****Table 9-6. PPMLR0[CDn] Assignments**

<b>Slot Number</b>	<b>CDn</b>	<b>Peripheral</b>
2	CD2	FlexBus
8	CD8	FlexCAN0

**Table 9-6. PPMLR0[CD $n$ ] Assignments (continued)**

Slot Number	CD $n$	Peripheral
9	CD9	FlexCAN1
14	CD14	I <sup>2</sup> C1
15	CD15	DSPI1
17	CD17	eDMA Controller
18	CD18	Interrupt Controller 0
19	CD19	Interrupt Controller 1
20	CD20	Interrupt Controller 2
22	CD22	I <sup>2</sup> C0
23	CD23	DSPI0
24	CD24	UART0
25	CD25	UART1
26	CD26	UART2
27	CD27	UART3
28	CD28	DMA Timer 0
29	CD29	DMA Timer 1
30	CD30	DMA Timer 2
31	CD31	DMA Timer 3

Address: 0xFC04\_0038 (PPMHR1)

Access: Supervisor read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
W																
Reset	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	1	1	1	1	1	1	1	1	1	1	1	CD37	CD36	1	CD34	1
W																
Reset	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	1

**Figure 9-6. Peripheral Power Management High Register 1 (PPMHR1)****Table 9-7. PPMHR1[CD $n$ ] Assignments**

Slot Number	CD $n$	Peripheral
34	CD34	mcPWM
36	CD36	CCM, Reset Controller, Power Management
37	CD37	GPIO Module

## Power Management

Address: 0xFC04_003C (PPMLR1)												Access: Supervisor read/write							
R		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16		
W		0	0	CD29	CD28	CD27	CD26	CD25	CD24	0	0	0	0	0	0	0	0		
Reset	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0		
R		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
W		CD15	CD14	0	0	0	0	0	0	CD7	CD6	CD5	CD4	0	CD2	0	0		
Reset	1	1	0	0	0	0	0	0	0	1	1	1	1	0	1	0	0		

**Figure 9-7. Peripheral Power Management Low Register 1 (PPMLR1)**

**Table 9-8. PPMLR1[CDn] Assignments**

Slot Number	$CD_n$	Peripheral
2	CD2	1-Wire
4	CD4	I <sup>2</sup> C2
5	CD5	I <sup>2</sup> C3
6	CD6	I <sup>2</sup> C4
7	CD7	I <sup>2</sup> C5
14	CD14	DSPI2
15	CD15	DSPI3
24	CD24	UART4
25	CD25	UART5
26	CD26	UART6
27	CD27	UART7
28	CD28	UART8
29	CD29	UART9

**Table 9-9. PPMHR and PPMLR Field Descriptions**

Field	Description
CDn	Module slot $n$ clock disable. 0 The clock for this module is enabled. 1 The clock for this module is disabled.

### 9.2.5 Low-Power Control Register (LPCR)

The LPCR register controls chip operation and module operation during low-power modes.

Address: 0xEC09\_0007 (LPCR)

Access: Supervisor read/write

	7	6	5	4		3	2	1	0
R	0	0	FWKUP	STPMD		0	0	0	0
W									
Reset:	0	0	0	0		0	0	0	0

Figure 9-8. Low-Power Control Register (LPCR)

Table 9-10. LPCR Field Descriptions

Field	Description																													
7–6	Reserved, should be cleared.																													
5 FWKUP	<p>Fast wake-up. Determines whether the system clocks are enabled upon wake-up from stop mode. This bit must be written before execution of the STOP instruction for it to take effect.</p> <p>0 System clocks enabled only when PLL is locked or operating normally.            1 System clocks enabled upon wake-up from stop mode, regardless of PLL lock status.</p> <p><b>Note:</b> Setting this bit is potentially dangerous and unreliable. The system may behave unpredictably when using an unlocked clock, because the clock frequency could overshoot the maximum frequency of the device.</p> <p><b>Note:</b> If FWKUP is set before entering stop mode, it should not be cleared upon wake-up from stop mode until after the PLL has actually acquired lock. Lock status may be obtained by reading PLL status register. Because the PLL never locks in limp mode, the FWKUP is ineffective. The system clocks are always enabled upon wake-up from stop mode, regardless of the value of FWKUP.</p>																													
4–3 STPMD	FB_CLK stop mode bits. Controls the operation of the clocks, PLL, and oscillator in stop mode:																													
	<table border="1"> <thead> <tr> <th>STPMD</th> <th>System Clocks</th> <th>FB_CLK</th> <th>PLL</th> <th>Oscillator</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>Disabled</td> <td>Enabled</td> <td>Enabled</td> <td>Enabled</td> </tr> <tr> <td>01</td> <td>Disabled</td> <td>Disabled</td> <td>Enabled</td> <td>Enabled</td> </tr> <tr> <td>10</td> <td>Disabled</td> <td>Disabled</td> <td>Disabled</td> <td>Enabled</td> </tr> <tr> <td>11</td> <td>Disabled</td> <td>Disabled</td> <td>Disabled</td> <td>Disabled</td> </tr> </tbody> </table>					STPMD	System Clocks	FB_CLK	PLL	Oscillator	00	Disabled	Enabled	Enabled	Enabled	01	Disabled	Disabled	Enabled	Enabled	10	Disabled	Disabled	Disabled	Enabled	11	Disabled	Disabled	Disabled	Disabled
STPMD	System Clocks	FB_CLK	PLL	Oscillator																										
00	Disabled	Enabled	Enabled	Enabled																										
01	Disabled	Disabled	Enabled	Enabled																										
10	Disabled	Disabled	Disabled	Enabled																										
11	Disabled	Disabled	Disabled	Disabled																										
2–0	Reserved, must be cleared.																													

## 9.3 Functional Description

This section discusses the functions and characteristics of the low-power modes, and how each module is affected by, or affects these modes.

### 9.3.1 Peripheral Shut Down

All peripherals, except for the SCM and crossbar switch, may have the software remove their input clocks individually to reduce power consumption. See [Section 9.2.4, “Peripheral Power Management Registers \(PPMHR{1,0}, PPMLR{1,0}\),”](#) for more information. A peripheral may be disabled at any time and remains disabled during any low-power mode of operation.

### 9.3.2 Limp mode

The device may also boot into a low-frequency limp mode, in which the PLL is bypassed and the device runs from a factor of the input clock (EXTAL). In this mode, EXTAL feeds a counter that divides the input clock by  $2^n$ , where  $n$  is the value of the programmable counter field, CDR[LPDIV]. The programmed value of the divider may be changed without glitches or otherwise negative affects to the system. While in this mode, the PLL is placed in bypass mode to reduce overall system-power consumption.

Limp mode may be entered and exited by writing to MISCCR[LIMP].

While in this mode, a 2:1 ratio maintains between the core and the primary bus clock. Because they cannot function at speeds as low as the minimum input clock frequency, the DDR SDRAM controller is not functional in limp mode.

Entering limp mode also requires a special procedure for the DDR module. As noted above the DDR controller is disabled in limp mode, so two critical steps must be followed before setting MISCCR[LIMP].

1. Transfer code execution to another memory resource. Primary options are the memory device is attached to the FlexBus boot chip select or on-chip SRAM (but not the CPU cache, as it may have to be flushed upon entering or exiting limp mode).
2. Place the DDR controller in self-refresh mode to avoid data loss while the controller is shut down.

After exiting limp mode, wait 256 clock cycles before writing to any DDR controller registers.

Since the USB controllers require a 60 MHz clock, during limp mode you must clear MISCCR[USBSRC] before entering limp mode to source their system clocks through alternate, off-chip sources (USB\_CLKIN). This also allows the USB modules to wake the device from limp mode.

### 9.3.3 Low-Power Modes

The system enters a low-power mode by executing a STOP instruction. The low-power mode the device actually enters (stop, wait, or doze) depends on the setting of the WCR[LPMD] bits. Entry into any of these modes idles the CPU with no cycles active, powers down the system, and stops all internal clocks appropriately. During stop mode, the system clock is stopped low.

A wake-up event is required to exit a low-power mode and return to run mode. Wake-up events consist of any of these conditions:

- Any type of reset
- Any valid, enabled interrupt request

Exiting from low-power mode via an interrupt request requires:

- An interrupt request whose priority is higher than the value programmed in the WCR[PRILVL].
- An interrupt request whose priority is higher than the value programmed in the interrupt priority mask (I) field of the core's status register.
- An interrupt request from a source not masked in the interrupt controller's interrupt mask register.
- An interrupt request which has been enabled at the module of the interrupt's origin.

### 9.3.3.1 Run Mode

Run mode is the normal system operating mode. Current consumption in this mode is related directly to the system clock frequency.

### 9.3.3.2 Wait Mode

Wait mode is intended to stop only the CPU and memory clocks until a wake-up event is detected. In this mode, peripherals may be programmed to continue operating and can generate interrupts, causing the CPU to exit from wait mode.

### 9.3.3.3 Doze Mode

Doze mode affects the processor in the same manner as wait mode, except that some peripherals define individual operational characteristics in doze mode. Peripherals continuing to run and having the capability of producing interrupts may cause the CPU to exit the doze mode and return to run mode. Stopped peripherals restart operation on exit from doze mode, as defined for each peripheral.

### 9.3.3.4 Stop Mode

Stop mode affects the processor the same as the wait and doze modes, except that all clocks to the system are stopped and the peripherals cease operation.

Stop mode must be entered in a controlled manner to ensure that any current operation is properly terminated. When exiting stop mode, most peripherals retain their pre-stop status and resume operation.

#### NOTE

Entering stop mode disables the DDRMC, including the refresh counter. If SDRAM is used, code is required to ensure proper entry and exit from stop mode. See [Chapter 21, “DDR1/2 SDRAM Memory Controller \(DDRM C\),”](#) for more information.

## 9.3.4 Peripheral Behavior in Low-Power Modes

The following subsections specify the operation of each module while in and when exiting low-power modes.

### 9.3.4.1 ColdFire Core

The ColdFire core disables during any low-power mode. No recovery time is required when exiting any low-power mode.

### 9.3.4.2 Internal SRAM

The SRAM is disabled only in stop mode. In wait and doze mode a non-core crossbar switch master can access SRAM. No recovery time is required when exiting any low-power mode.

### 9.3.4.3 Clock Module

In wait and doze modes, the clocks to the CPU stop and the system clocks to the peripherals enable. Each module may disable the module clocks locally at the module level, or the module clocks may be individually disabled by the PPMR registers (refer to [Section 9.2.4, “Peripheral Power Management Registers \(PPMHR{1,0}, PPMLR{1,0}\)”](#)). In stop mode, all clocks to the system stop.

There are several options for enabling or disabling the PLL or crystal oscillator in stop mode, compromising between stop mode current and wake-up recovery time. The PLL can be disabled in stop mode, but requires a wake-up period before it can relock. The oscillator can also be disabled during stop mode, but requires a wake-up period to restart.

When the PLL is enabled in stop mode (LPCR[STPMD] = 00), the external FB\_CLK signal can support systems using FB\_CLK as the clock source. See [Section 9.2.5, “Low-Power Control Register \(LPCR\),”](#) for more information about operating the PLL in stop mode.

There is also a fast wake-up option for quickly enabling the system clocks during stop recovery (LPCR[FWKUP]). This eliminates the wake-up recovery time but at the risk of sending a potentially unstable clock to the system. This is also explained in [Section 9.2.5, “Low-Power Control Register \(LPCR\).”](#)

### 9.3.4.4 Chip Configuration Module

The chip configuration module is unaffected by entry into a low-power mode, but register access is disabled. If a reset exits low-power mode, chip configuration may execute if configured to do so.

### 9.3.4.5 Reset Controller

A power-on reset (POR) always causes a chip to reset and exit from any low-power mode.

In wait and doze modes, asserting the external RESET pin for at least four clocks causes an external reset that resets the chip and exits any low-power modes.

In stop mode, the RESET pin synchronization disables and asserting the external RESET pin asynchronously generates an internal reset and exit any low-power modes. Registers lose current values and must be reconfigured from reset state if needed.

If the core watchdog timer is still enabled during wait or doze modes, a watchdog timer timeout may generate a reset to exit these low-power modes.

When the CPU is inactive, a software reset cannot generate to exit any low-power mode.

### 9.3.4.6 System Control Module (SCM)

The SCM’s core watchdog timer can bring the device out of all low-power modes except stop mode. In stop mode, all clocks stop, and the core watchdog timer does not operate.

When enabled, the core watchdog can bring the device out of low-power mode in one of two ways. Depending on the setting of the CWCR[CWRI] field, a core watchdog timeout may reset the device. Other settings of the CWRI field may enable a core watchdog interrupt and upon a watchdog timeout, this

interrupt can bring the device out of low-power mode. This system setup must meet the conditions specified in [Section 9.3.3, “Low-Power Modes,”](#) for the core watchdog interrupt to bring the part out of low-power mode.

#### **9.3.4.7 Crossbar Switch**

The crossbar switch is disabled in stop mode. It is enabled in other low power modes.

#### **9.3.4.8 GPIO Ports**

The GPIO ports are unaffected by entry into a low-power mode. These pins may impact low-power current draw if they are configured as outputs and are sourcing current to an external load. If low-power mode is exited by a reset, the state of the I/O pins reverts to their default direction settings.

#### **9.3.4.9 Interrupt Controllers (INTC0–2)**

The interrupt controller is not affected by any of the low-power modes. All logic between the input sources and generating the interrupt to the processor is combinational to allow the ability to wake up the core during low-power stop mode when all system clocks stop.

An interrupt request causes the processor to exit a low-power mode only if that interrupt’s priority level is at or above the level programmed in the interrupt priority mask field of the CPU’s status register (SR) and above the level programmed in the WCR[PRILVL]. The interrupt must also be enabled in the interrupt controller’s interrupt mask register as well as at the module from which the interrupt request would originate.

#### **9.3.4.10 Edge Port**

In wait and doze modes, the edge port continues to operate normally and may be configured to generate interrupts (either an edge transition or low level on an external pin) to exit the low-power modes.

In stop mode, no system clock is available to perform the edge detect function. Therefore, only the level detect logic is active (if configured) to allow any low level on the external interrupt pin to generate an interrupt (if enabled) to exit stop mode.

#### **9.3.4.11 eDMA Controller**

In wait and doze modes, the eDMA controller can bring the device out of a low-power mode by generating an interrupt upon completion of a transfer or upon an error condition. The completion of transfer interrupt generates when DMA interrupts are enabled by the setting of a EDMA\_INTR[INTn], and an interrupt is generated when TCDn[DONE] is set. The interrupt upon error condition is generated when EDMA\_EEIR[EEIn] is set, and an interrupt generates when any of the EDMA\_ESR bits become set.

The eDMA controller is stopped in stop mode and thus cannot cause an exit from this low-power mode.

### 9.3.4.12 FlexBus Module

In wait and doze modes, the FlexBus module continues operation but does not generate interrupts; therefore, it cannot bring a device out of a low-power mode. This module is stopped in STOP mode.

### 9.3.4.13 Ethernet Assembly

The Ethernet (ENET) assembly includes two MACNET modules and an Ethernet switch. In wait and doze modes, since the clocks to the ENET Assembly are on, the ENET is unaffected and may generate an interrupt to exit these low-power modes. Any activity on the Ethernet bus (RMII/MII interface) wakes the system and exits from the low power mode.

In stop mode, the ENET stops immediately and freezes operation, register values, state machines, and external pins. During this mode, the ENET clocks are shut down. Coming out of stop mode returns the ENET to operation from the state prior to stop mode entry. Any activity on the Ethernet bus (RMII/MII interface) wakes the system and exits from the low power mode.

#### NOTE

Only enter stop mode when there is no activity on the Ethernet bus or no packets to forward to the other port.

### 9.3.4.14 ADC

In wait mode, ADC register read/write access is disabled. In doze modes, the ADC is unaffected. In both modes the ADC may generate an interrupt to exit these low-power modes.

In stop mode, the ADC stops immediately and freeze its operation. During this mode, the ADC clocks are shut down.

### 9.3.4.15 DAC

In wait and doze modes, the DAC is unaffected and may generate an interrupt to exit these low-power modes.

In stop mode, the DAC stop immediately and freeze its operation. During this mode, the DAC clocks are shut down.

### 9.3.4.16 NAND Flash Controller (NFC)

In wait and doze modes, the NFC is unaffected and may generate an interrupt to exit these low-power modes.

In stop mode, since the system/bus clock is stopped, NFC goes into an inactive state.

### 9.3.4.17 1-Wire Interface

The 1-Wire module automatically enters low power mode when it is not communicating with a generic 1-Wire device. The main clock is gated off in low power mode. As soon as software writes to any register, the 1-Wire module exits low power mode.

In wait and doze modes, this module operates normally. In stop mode all the clocks going to the 1-Wire module are disabled.

#### **9.3.4.18 mcPWM**

Take care when using this module in certain chip operating modes. Some motors (such as 3-phase AC motors) require regular software updates for proper operation. Failure to do so could result in destroying the motor or inverter. Because of this, PWM outputs are placed in their inactive states in stop mode, and optionally under wait and debug modes. PWM outputs are reactivated (assuming they were active to begin with) when these modes are exited.

In stop mode, since peripheral and CPU clocks are stopped, the PWM outputs are inactive.

In wait and doze modes, since CPU clocks are stopped while peripheral clocks continue to run during this mode, PWM outputs are inactive as a function of the WAITEN bit.

The PWM also supports a debug mode where the PWM outputs are inactive as a function of the DEBUG bit.

#### **9.3.4.19 FlexCAN**

FlexCAN is enabled in wait mode and works normally.

Doze mode is entered when the CAN\_MCR[DOZE] bit is set and the processor is in doze mode. When in doze mode, the module shuts down the clocks to the CAN protocol interface and the message buffer management sub-modules. The module exits this mode when CAN\_MCR[DOZE] is cleared, the MCU is removed from doze mode, or when activity is detected on the CAN bus and the self wake-up mechanism is enabled.

In stop mode, the module puts itself in an inactive state and then informs the CPU that the clocks can be shut down globally. The module exits this mode when the stop mode request is removed or when activity is detected on the CAN bus and the self wake-up mechanism is enabled.

#### **9.3.4.20 Robust Real Time Clock**

In stop mode, the external clock driving EXTAL32K/XTAL32K continues to clock the RTC module. Therefore, the device can update the RTC counters, alarms, etc. while in stop mode. An RTC interrupt/wake-up can be generated while in stop mode to wake the device if the RTC alarms are triggered.

#### **9.3.4.21 Rapid GPIO**

Since RGPI0 is memory mapped device located on the processor's high-speed platform bus and works on divided processor clock, it is disabled during any low-power mode. No recovery time is required when exiting any low-power mode.

#### **9.3.4.22 Serial Boot Facility**

The clock going to serial boot facility (SBF) is disabled in stop mode. No other low power mode feature is implemented in SBF.

### 9.3.4.23 DDR SDRAM Controller (DDRMC)

SDRAM controller operation is unaffected either the wait or doze modes; however, the DDRMC is disabled by stop mode. Because the STOP mode disables all clocks to the DDRMC, the DDRMC does not generate refresh cycles.

The DDRMC supports the following low-power modes:

- Memory power down — The memory controller sets the memory devices into power-down, which reduces the overall power consumption of the system, but has the least effect of all the low power modes. In this mode, the memory controller and memory clocks are fully operational, but the CKE input bit to the memory devices is deasserted. The memory controller continues to monitor memory refresh needs and automatically brings the memory out of power-down to perform these refreshes. When a refresh is required, the CKE input bit to the memory devices is reenabled. This action brings the memory devices out of power-down. Once the refresh completes, the memory devices are returned to power-down by deasserting the CKE input bit.
- Memory power down with memory clock gating — The memory controller sets the memory devices into power-down and gates off the clock to the memory devices. Refreshes are handled as in the memory power-down mode, with the exception that gating on the memory clock is removed before asserting the CKE pin. After the refresh completes, the memory devices are returned to power-down with the clock gated. Before the memory devices are removed from power-down, the clock is gated on again.

#### NOTE

Do not use this mode for memory devices that do not support memory clock gating. Clock gating is not supported for standard DDR1 and DDR2 devices.

When set into this mode, the memory controller attempts to place the memory devices in power-down and gate off the memory clock. The memory functions unpredictably and may hang.

- Memory self refresh — In this mode, the memory controller and memory clocks are fully operational and the CKE input bit to the memory devices is deasserted. Since the memory automatically refreshes its contents, the memory controller does not need to send explicit refreshes to the memory
- Memory self refresh with memory clock gating — The memory controller sets the memory devices into self-refresh and gates off the clock to the memory devices. Before the memory devices are removed from self-refresh, the clock is gated on again.
- Memory self refresh with memory and controller clock gating — This is the deepest low-power mode of the memory controller. The memory controller sets the memory devices into self-refresh and gates off the clock to the memory devices. In addition, the clock to the memory controller and the programming parameters are gated off, except to a small portion of the DLL, which must remain active to maintain lock. Before the memory devices are removed from self-refresh, the memory controller and memory clocks are gated on.

### 9.3.4.24 eSDHC

When the clocks to the eSDHC are disabled or when the system is in a low power mode, the eSDHC can generate the following three wake-up interrupts:

- Card removal interrupt
- Card insertion interrupt
- SDIO card interrupt

The eSDHC offers a power management feature. By clearing the clock-enabled bits in the clock control register, the clocks are gated low to the eSDHC. For maximum power saving, the user can disable all the clocks to eSDHC when there is no operation in progress.

### 9.3.4.25 Cryptography Acceleration Unit (CAU)

The CAU is disabled during any low-power mode. No recovery time is required when exiting any low-power mode.

### 9.3.4.26 Random Number Generator (RNG)

Since the clocks to the RNGB are enabled, the RNGB is functional during the doze and wait modes. During the stop mode, the RNGB is not functional. The RNG entry into stop mode is delayed until it completes the current seed generation.

### 9.3.4.27 Subscriber Interface Module (SIM)

The SIM is enabled in wait mode. In doze mode SIM operation is configurable with the SIM\_RCR[DOZE] bit. If this bit is set, the SIM gates its clocks when the transmit FIFO is empty. If cleared, doze mode has no effect on the SIM.

In stop mode the SIM can be disabled completely or partially disabled. If SIM\_RCR[STOP] is set, only the SIM card baud clock runs. If cleared, all clocks are disabled.

### 9.3.4.28 USB On-the-Go Module

If the USB On-the-Go module is clocked externally, it operates normally in wait and doze. It is capable of generating an interrupt to wake up the core from the wait and doze modes. In stop mode, the USB module is disabled.

The USB block contains an automatic low power mode in which the module enters suspend mode after a 6.0 ms minimum period of inactivity. When the module receives a wake-up from the USB host, the transceiver is re-enabled for normal USB operations.

### 9.3.4.29 USB Host Module

In wait mode the clocks to the USB host module continue to run. In doze mode, the processor stops the clocks to the USB host module, but the 60 Mhz transceiver clock remains active. In doze mode, detection of resume signaling initiates a restart of the module clocks.

In stop mode, the processor stops the clock to the USB host module. In this state, the module ignores traffic on the USB and does not generate any interrupts or wake-up events. The on-chip transceiver is disabled to save power.

#### **9.3.4.30 Programmable Interrupt Timers (PIT0–3)**

In stop mode (or doze mode, if so programmed in the PCSR $n$  register), the programmable interrupt timer (PIT) ceases operation, and freezes at the current value. When exiting these modes, the PIT resumes operation from the stopped value. It is the responsibility of software to avoid erroneous operation.

When not stopped, the PIT may generate an interrupt to exit the low-power modes.

#### **9.3.4.31 DMA Timers (DTIM0–3)**

In wait and doze modes, the DMA timers may generate an interrupt to exit a low-power mode. This interrupt can generate when the DMA timer is in input capture mode or reference compare mode.

In input capture mode, where the capture enable (CE) field of the timer mode register (DTMR) has a non-zero value and the DTXMR[DMAEN] is cleared, an interrupt issues upon a captured input. In reference compare mode, where the output reference requests interrupt enable (ORRI) bit of DTMR is set and DTXMR[DMAEN] is cleared, an interrupt issues when the timer counter reaches the reference value.

DMA timer operation disables in stop mode. Upon exiting stop mode, the timer resumes operation unless stop mode was exited by reset.

#### **9.3.4.32 DMA Serial Peripheral Interface (DSPI)**

In wait mode, the DSPI module is unaffected and may generate an interrupt to exit this low-power mode.

In stop and doze modes, the DSPI first completes transfer of the current frame. It then freezes operation, register values, state machines, and external pins. During this mode, the DSPI clocks shut down. Coming out of stop mode returns the DSPI to operation from the state prior to stop mode entry.

#### **9.3.4.33 UART Modules**

In wait and doze modes, the UARTs are unaffected and may generate an interrupt to exit these low-power modes.

In stop mode, the UARTs stop immediately and freeze their operation, register values, state machines, and external pins. During this mode, the UART clocks shut down. Exiting stop mode returns the UARTs to the operation of the state prior to stop-mode entry.

#### **9.3.4.34 I<sup>2</sup>C Modules**

When the I<sup>2</sup>C modules are enabled by the setting of the I2CR[IEN] bit and the device is not in stop mode, the I<sup>2</sup>C module is operable and may generate an interrupt to bring the device out of a low-power mode.

For an interrupt to occur, the I2CR[IIE] bit must be set to enable interrupts, and the setting of the I2SR[IIF] generates the interrupt signal to the CPU and interrupt controller. The setting of I2SR[IIF] signifies the

completion of one byte transfer or the reception of a calling address matching its own specified address when in slave-receive mode.

In stop mode, the I<sup>2</sup>C module stops immediately and freezes operation, register values, and external pins. Upon exiting stop mode, the I<sup>2</sup>C resumes operation unless stop mode was exited by reset.

### 9.3.4.35 BDM

Entering halt (debug) mode via the BDM port (by asserting the external **BKPT** pin) causes the processor to exit any low-power mode.

### 9.3.4.36 JTAG

The JTAG (Joint Test Action Group) controller logic is clocked using the TCLK input and not affected by the system clock. The JTAG cannot generate an event to cause the processor to exit any low-power mode. Toggling TCLK during any low-power mode increases the system current consumption.

## 9.3.5 Summary of Peripheral State During Low-power Modes

The functionality of each of the peripherals and CPU during the various low-power modes is summarized in [Table 9-11](#). The status of each peripheral during a given mode refers to the condition the peripheral automatically assumes when the STOP instruction is executed and the WCR[LPMD] field is set for the particular low-power mode. Individual peripherals may be disabled by programming its dedicated control bits. The wake-up capability field refers to the ability of an interrupt or reset by that peripheral to force the CPU into run mode.

**Table 9-11. CPU and Peripherals in Low-Power Modes**

Module	Peripheral Status <sup>1</sup> / Wake-up Procedure					
	Wait Mode		Doze Mode		Stop Mode	
ColdFire Core	Stopped	N/A	Stopped	N/A	Stopped	N/A
SRAM	Enabled	N/A	Enabled	N/A	Stopped	N/A
Clock Module	Enabled	N/A	Enabled	N/A	Program	N/A
Power Management	Enabled	N/A	Enabled	N/A	Stopped	N/A
Chip Configuration Module	Enabled	N/A	Enabled	N/A	Stopped	N/A
Reset Controller	Enabled	Reset	Enabled	Reset	Enabled	Reset
System Control Module	Enabled	N/A	Enabled	N/A	Stopped	N/A
GPIO	Enabled	N/A	Enabled	N/A	Stopped	N/A
Interrupt controller	Enabled	Interrupt	Enabled	Interrupt	Enabled	Interrupt
Edge port	Enabled	Interrupt	Enabled	Interrupt	Stopped	Interrupt
eDMA Controller	Enabled	N/A	Enabled	N/A	Stopped	N/A
FlexBus Module	Enabled	N/A	Enabled	N/A	Stopped	N/A
DDR Controller	Enabled	N/A	Enabled	N/A	Stopped	N/A

**Table 9-11. CPU and Peripherals in Low-Power Modes (continued)**

Module	Peripheral Status <sup>1</sup> / Wake-up Procedure					
	Wait Mode		Doze Mode		Stop Mode	
Ethernet Assembly	Enabled	Interrupt	Enabled	Interrupt	Enabled	Interrupt
USB OTG	Enabled	Interrupt	Stopped	Interrupt	Stopped	N/A
USB Host	Enabled	Interrupt	Stopped	Interrupt	Stopped	N/A
SSI	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
Robust Real Time Clock	Enabled	Interrupt	Enabled	Interrupt	Enabled	Interrupt
Programmable Interrupt Timers	Enabled	Interrupt	Program	Interrupt	Stopped	N/A
DMA Timers	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
DSPI	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
UARTs	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
I <sup>2</sup> C Modules	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
RNG	Enabled	N/A	Stopped	N/A	Stopped	N/A
eSDHC	Enabled	Interrupt	Enabled	Interrupt	Stopped	Interrupt
SIM	Enabled	Interrupt	Program	Interrupt	Stopped	Interrupt
ADC	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
DAC	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
NAND Flash Controller	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
1-Wire	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
mCPWM	Program	Interrupt	Program	Interrupt	Stopped	N/A
FlexCAN	Enabled	Interrupt	Program	Interrupt	Stopped	N/A
Rapid GPIO	Stopped	N/A	Stopped	N/A	Stopped	N/A
Serial Boot Facility	Enabled	N/A	Enabled	N/A	Stopped	N/A
JTAG <sup>2</sup>	Enabled	N/A	Enabled	N/A	Enabled	N/A
BDM <sup>3</sup>	Enabled	Yes	Enabled	Yes	Enabled	Yes

<sup>1</sup> Program indicates that the peripheral function during the low-power mode is dependent on programmable bits in the peripheral register map.

<sup>2</sup> The JTAG logic is clocked by a separate TCLK clock.

<sup>3</sup> Entering halt mode via the BDM port exits any lower-power mode. Upon exit from halt mode, the previous low-power mode is re-entered, and changes made in halt mode remain in effect.





# Chapter 10

## Chip Configuration Module (CCM)

### 10.1 Introduction

The chip-configuration module (CCM) configures the device to operate in one of multiple functional modes.

#### 10.1.1 Block Diagram

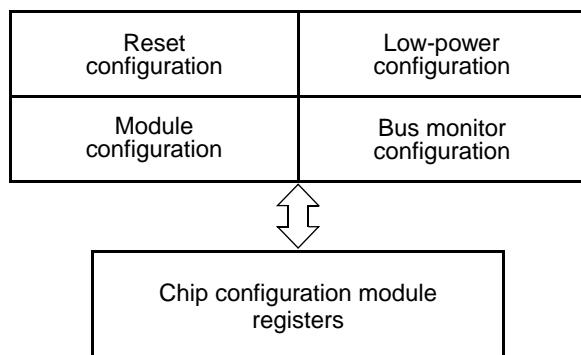


Figure 10-1. Chip-Configuration Module Block Diagram

#### 10.1.2 Features

The CCM performs these operations:

- Configures device based on chosen reset configuration options
- Selects bus-monitor configuration
- Selects low-power configuration

#### 10.1.3 Modes of Operation

The only chip operating mode available on this device is master mode. In master mode, the ColdFire core can access external memories and peripherals. The external bus consists of a 32-bit data and address buses. The available bus control signals include FB\_R/W, FB\_TS, FB\_TA, FB\_OE, FB\_TBST, and FB\_BE/BWE[3:0]. Up to six chip selects can be programmed to select and control external devices and to provide bus cycle termination.

## 10.2 External Signal Descriptions

[Table 10-1](#) provides an overview of the CCM signals.

**Table 10-1. Signal Properties**

Name	Function
BOOTMOD[1:0]	Reset configuration select
FB_AD[7:0]	Reset configuration override pins

### 10.2.1 BOOTMOD[1:0]

These signals determine the boot performed at reset. See the table below for BOOTMOD[1:0] usage.

**Table 10-2. BOOTMOD[1:0] Values**

BOOTMOD[1:0]	Meaning
00	Boot from FlexBus with defaults
01	Override defaults and RCON decides boot source (FlexBus or NAND flash)
10	Override defaults and boot from serial boot facility with optional load to and boot from RAM. If not booting from RAM, SBF_RCON decides the boot source (FlexBus or NAND flash).
11	

### 10.2.2 FB\_AD[7:0] (Reset Configuration Override)

If the external BOOTMOD[1:0] pins are driven to 01 during reset, the states of the FB\_AD[7:0] pins during reset determine the boot device, clock mode, and certain module configurations after reset. See [Table 10-17](#), for details of the function of each FB\_AD[7:0] pin during reset.

#### NOTE

The logic levels for reset configuration on FB\_AD[7:0] must be actively driven when BOOTMOD equals 01. FB\_AD[31:8] should float or be pulled high.

## 10.3 Memory Map/Register Definition

The CCM programming model consists of the registers listed in the below table.

**Table 10-3. CCM Memory Map**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
<b>Supervisor Access Only Registers<sup>1</sup></b>					
0xEC09_0004	Chip configuration register (CCR)	16	R	See Section	<a href="#">10.3.1/10-3</a>
0xEC09_0008	Reset configuration register (RCON)	16	R	0x1259	<a href="#">10.3.2/10-4</a>
0xEC09_000A	Chip identification register (CIR)	16	R	See Section	<a href="#">10.3.3/10-5</a>

**Table 10-3. CCM Memory Map (continued)**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xEC09_000E	Miscellaneous control register (MISCCR)	16	R/W	0x183D	<a href="#">10.3.4/10-5</a>
0xEC09_0010	Clock divider register high (CDRH)	16	R/W	0x0101	<a href="#">10.3.5/10-7</a>
0xEC09_0012	Clock divider register low (CDRL)	16	R/W	0x0000	<a href="#">10.3.5/10-7</a>
0xEC09_0014	USB On-the-Go controller status register (UOCSR)	16	R/W	0x0010	<a href="#">10.3.7/10-8</a>
0xEC09_0016	USB host controller status register (UHCSR)	16	R/W	0x0000	<a href="#">10.3.8/10-10</a>
0xEC09_0018	Miscellaneous control register 3 (MISCCR3)	16	R/W	0x0001	<a href="#">10.3.9/10-11</a>
0xEC09_001A	Miscellaneous control register 2 (MISCCR2)	16	R/W	0x201F	<a href="#">10.3.10/10-11</a>
0xEC09_001C	ADC trigger select register (ADCTSR)	16	R/W	0x0000	<a href="#">10.3.11/10-13</a>
0xEC09_001E	DAC trigger select register (DACTS)	16	R/W	0x0000	<a href="#">10.3.12/10-14</a>
0xEC09_0020	Serial boot facility status register (SBFSR) <sup>2</sup>	16	R	See Section	<a href="#">11.3.1/11-3</a>
0xEC09_0022	Serial boot facility control register (SBFCR) <sup>2</sup>	16	R/W	See Section	<a href="#">11.3.2/11-3</a>
0xEC09_0024	FlexBus/NAND flash arbiter control register (FNACR)	32	R/W	0x0000_0000	<a href="#">10.3.13/10-15</a>

<sup>1</sup> User access to supervisor-only address locations have no effect and result in a bus error.

<sup>2</sup> The SBFSR and SBFCR registers are described in [Chapter 11, “Serial Boot Facility \(SBF\).”](#)

## 10.3.1 Chip Configuration Register (CCR)

The CCR is a read-only register; writing to it has no effect. At reset, the CCR reflects the chosen operation of certain device functions. These functions may be set to the defaults defined by the RCON values or overridden during reset configuration using the external BOOTMOD[1:0] and the FB\_AD[7:0] pins. (See [Figure 10-3](#) for the RCON register definition.)

Address: 0xEC09_0004 (CCR)												Access: Supervisor read-only				
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BOOTMOD	0	1		PLLMULT				BOOTPS		ALE SEL	OSC MODE	PLL MODE	BOOT MEM		
W																
Reset	BOOTMOD[1:0]		0	See Note												

**Note:** Reset value depends upon chosen reset configuration. Default reset value (BOOTMOD = 00) is the value of RCON.

**Figure 10-2. Chip Configuration Register (CCR)**

**Table 10-4. CCR Field Descriptions**

<b>Field</b>	<b>Description</b>
15–14 BOOTMOD	BOOTMOD configuration. Captures the boot mode configuration from the BOOTMOD[1:0] pins. 00 Boot from FlexBus with defaults 01 Override defaults and RCON decides boot source (FlexBus or NAND flash) 1x Override defaults and boot from serial boot facility with optional load to and boot from RAM. If not booting from RAM, SBF_RCON decides the boot source (FlexBus or NAND flash).
13	Reserved, must be cleared.
12	Reserved, must be set.
11–6 PLLMULT	PLL multiplier. Reflects the multiplying factor of the PLL.  In RCON this field reflects the decoded value of the 2-bit RCON field: 00 $f_{VCO} = 10 \times f_{ref}$ 01 $f_{VCO} = 15 \times f_{ref}$ 10 $f_{VCO} = 16 \times f_{ref}$ 11 $f_{VCO} = 20 \times f_{ref}$  In SBF_RCON, this field reflects the actual SBF_RCON setting for the PLL multiplier. See <a href="#">Section 10.4.1.3, “Reset Configuration (BOOTMOD[1:0] = 1x)”</a>
5–4 BOOTPS	Boot port size. Reflects the chosen port size of the boot memory. 00 32-bit (32-bit muxed address) 01 8-bit (24-bit non-muxed address) 1x 16-bit (16-bit non-muxed address)
3 ALESEL	Reflects if the FB_ALE and FB_TS signal is present on the FB_ALE pin. 0 FB_TS 1 FB_ALE
2 OSCMODE	Oscillator clock mode. 0 Crystal oscillator mode 1 Oscillator bypass mode
1 PLLMODE	PLL mode. Reflects if the PLL is enabled upon boot. 0 Disabled 1 Enabled
0 BOOTMEM	Boot memory. Reflects the source of the boot memory 0 NAND flash 1 FlexBus

### 10.3.2 Reset Configuration Register (RCON)

At reset, the RCON register determines the default operation of certain chip functions. All default functions defined by the RCON values can be overridden only during reset configuration (see [Section 10.4.1, “Reset Configuration”](#)). RCON is a read-only register and contains the same fields as the CCR register, minus the BOOTMOD field.

Address: 0xEC09\_0008 (RCON)

Access: Supervisor read-only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	1	PLLMULT				BOOTPS			ALE SEL	OSC MODE	PLL MODE	BOOT MEM	
W					0	0	1	0	0	1	0	1	1	0	0	1
Reset	0	0	0	1	0	0	1	0	0	1	0	1	1	0	0	1

Figure 10-3. Reset Configuration Register (RCON)

### 10.3.3 Chip Identification Register (CIR)

CIR is a read-only register; writing to it has no effect.

Address: 0xEC09\_000A (CIR)

Access: Supervisor read-only

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PIN								PRN							
W																
Reset	Device Dependent												Mask Set Dependent			

Figure 10-4. Chip Identification Register (CIR)

Table 10-5. CIR Field Descriptions

Field	Description
15–6 PIN	Part identification number. Contains a unique identification number for the device. See the IDCODE[PIN] bit field description in <a href="#">Section 44.3.2, “IDCODE Register”</a> for valid encodings.
5–0 PRN	Part revision number. This number increases by one for each new full-layer mask set of this part. The revision numbers are assigned in chronological order.

### 10.3.4 Miscellaneous Control Register (MISCCR)

The MISCCR register provides various configuration options for limp mode, bus monitor, USB pull selection, and SDHC/SSI/USB clocks.

Address: 0xEC09\_000E (MISCCR)

Access: Supervisor read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PWM EXTCLK		PLL LOCK	LIMP	BME	BMT			0	SDHC SRC	SSI1 SRC	SSI0 SRC	USBH OC	USBO OC	USB PUE	USB SRC
W	0	0	0	1	1	0	0	0	0	1	1	1	1	1	0	1
Reset	0	0	0	1	1	0	0	0	0	1	1	1	1	1	0	1

Figure 10-5. Miscellaneous Control Register (MISCCR)

Table 10-6. MISCCR Field Descriptions

Field	Description
15 PWM EXTCLK	Selects the clock source for the PWM's clock input. 00 Timer 0 output (T0OUT) 01 Timer 1 output (T1OUT) 10 Timer 2 output (T2OUT) 11 Timer 3 output (T3OUT)
13 PLLLOCK	PLL lock status. Indicates the PLL is locked. 0 PLL is not locked 1 PLL is locked
12 LIMP	Limp mode enable. Selects between the PLL and the low-power clock divider as the source of all system clocks. 0 Normal operation; PLL drives system clocks. 1 Limp mode; low-power clock divider drives system clocks. <b>Note:</b> The transient behavior of the system when writing this bit cannot be predicted. When any USB wake-up event is detected, this bit is cleared, limp mode is exited, and the PLL begins the process of relocking and driving the system clocks.
11 BME	Bus monitor external enable bit. Enables the bus monitor to operate during external FlexBus cycles 0 Bus monitor disabled on external FlexBus cycles 1 Bus monitor enabled on external FlexBus cycles
10–8 BMT	Bus monitor timing field. Selects the timeout period in FlexBus clock cycles for the bus monitor: Timeout period for external bus cycles equals $2^{(16-BMT)}$ FB_CLK cycles 000 65536 001 32768 010 16384 011 8192 100 4096 101 2048 110 1024 111 512
7	Reserved, must be cleared.
6 SDHCSRC	eSDHC clock source. Selects the source of the SDHC card clock. 0 Oscillator output 1 PLL divider output
5 SSI1SRC	SSI1 clock source. Selects between the PLL and the external SSI_CLKIN pin as the source of the SSI1 baud clock. 0 SSI_CLKIN pin 1 PLL output
4 SSI0SRC	SSI0 clock source. Selects between the PLL and the external SSI_CLKIN pin as the source of the SSI0 baud clock. 0 SSI_CLKIN pin 1 PLL output
3 USBHOC	USB host VBUS over-current sense polarity. Selects the polarity of the USB host controller's VBUS over-current sense signal driven off-chip. 0 Active low 1 Active high
2 USBOC	USB On-the-Go VBUS over-current sense polarity. Selects the polarity of the USB OTG controller's VBUS over-current sense signal driven off-chip. 0 Active low 1 Active high

**Table 10-6. MISCCR Field Descriptions (continued)**

Field	Description
1 USBPUE	USB transceiver pull-up enable. Enables the on-chip USB OTG controller to drive the internal transceiver pull-up. 0 Internal transceiver pull-up is disabled. The USB_PULLUP signal triggers the external pull-up. 1 USB OTG drives the internal transceiver pull-up
0 USBSRC	USB clock source. Selects between the PLL and the external USB_CLKIN external pin as the clock source for the serial interface of the USB module. 0 USB_CLKIN pin drives USB serial interface clocks 1 PLL drives USB serial interface clocks

### 10.3.5 Clock-Divider Register High (CDRH)

The CDRH register provides clock division factors for limp mode and the SSI master clock when the PLL is used to drive the SSI clock.

Address: 0xEC09_0010 (CDRH)								Access: Supervisor read/write													
R								W													
SSI0DIV																SSI1DIV					
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1

**Figure 10-6. Clock-Divider Register High (CDRH)****Table 10-7. CDRH Field Descriptions**

Field	Description
15–8 SSI0DIV	SSI0 oversampling clock divider. Specifies the divide value that produces the SSI0 oversampling clock. This field is used only when MISCCR[SSI0SRC] is set (PLL is the source).  $\text{SSI0 Baud Clock} = \frac{f_{\text{sys}}}{\text{SSI0DIV}/2}$ <b>Eqn. 10-1</b> <b>Note:</b> A value of 0 or 1 for SSI0DIV represents a divide-by-254. SSI0DIV must not be set to any value that sets the SSI oversampling clock frequency over the bus clock frequency ( $f_{\text{sys}/2}$ ), because incorrect SSI operation could result.
7–0 SSI1DIV	SSI1 oversampling clock divider. Specifies the divide value that produces the SSI1 oversampling clock. This field is used only when MISCCR[SSI1SRC] is set (PLL is the source).  $\text{SSI1 Baud Clock} = \frac{f_{\text{sys}}}{\text{SSI1DIV}/2}$ <b>Eqn. 10-2</b> <b>Note:</b> A value of 0 or 1 for SSI1DIV represents a divide-by-254. SSI1DIV must not be set to any value that sets the SSI oversampling clock frequency over the bus clock frequency ( $f_{\text{sys}/2}$ ), because incorrect SSI operation could result.

### 10.3.6 Clock-Divider Register Low (CDRL)

The CDRL register provides the clock division factor for limp mode.

Address: 0xEC09_0012 (CDRL)																Access: Supervisor read/write							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0							
R	0	0	0	0	LPDIV				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 10-7. Clock-Divider Register Low (CDRL)

Table 10-8. CDRL Field Descriptions

Field	Description
15–12	Reserved, must be cleared.
11–8 LPDIV	<p>Low power clock divider. Specifies the divide value used to produce the system clocks during limp mode. A 2:1 ratio is maintained between the core and the internal bus. This field is used only when MISCCR[LIMP] is set.</p> $\text{System Clocks} = \frac{f_{\text{EXTAL}}}{2^{\text{LPDIV}}} \quad \text{Eqn. 10-3}$ <p><b>Note:</b> When LPDIV is cleared (divide by 1), the internal bus clock and FB_CLK does not have a 50/50 duty cycle.</p> <p><b>Note:</b> Do not change the LPDIV value from 0 to non-zero or non-zero to 0 in limp mode. Else, the output clocks glitch. To avoid this behavior, only change these settings during PLL mode.</p> <p><b>Note:</b> Wait for at least five reference clock cycles to switch from PLL to limp mode from the time where the LPDIV is modified (during PLL mode).</p>
7–0	Reserved, must be cleared.

### 10.3.7 USB On-the-Go Controller Status Register (UOCSR)

The UOCSR register controls and reflects various features of the USB OTG module. When any bit of this register generates an interrupt, that interrupt is cleared by reading the UOCSR register. The read-only bits of this register are set by the USB OTG module.

Address: 0xEC09_0014 (UOCSR)																Access: Supervisor read/write							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0							
R	0	0	DPPD	DMPD	DRV	CRG	DCR	DPPU	AVLD	BVLD	VVLD	SEND	PWR	WKUP	UOMIE	XPDE							
W					VBUS	VBUS	VBUS						FLT										
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0							

Figure 10-8. USB On-the-Go Controller Status Register (UOCSR)

**Table 10-9. UOCSR Field Descriptions**

<b>Field</b>	<b>Description</b>
15–14	Reserved, must be cleared.
13 DPPD	D+ 15 kΩ pull-down. Indicates the 15-kΩ pull-down on the OTG D+ line is active. When set, asserts an interrupt if UOMIE is set. 0 Pull-down disabled 1 Pull-down enabled
12 DMPD	D- 15 kΩ pull-down. Indicates the 15-kΩ pull-down on the OTG D- line is active. When set, asserts an interrupt if UOMIE is set. 0 Pull-down disabled 1 Pull-down enabled
11 DRVVBUS	Drive VBUS. 0 Disable the drive of 5 V power on VBUS 1 Enable the drive of 5 V power on VBUS
10 CRGVBUS	Charge VBUS. Indicates a charge resistor to pull-up VBUS is enabled. When set, asserts an interrupt if UOMIE is set. 0 Charge resistor to pull-up VBUS disabled 1 Charge resistor to pull-up VBUS enabled
9 DCRVBUS	Discharge VBUS. Indicates a discharge resistor to pull-down VBUS is enabled. When set, asserts an interrupt if UOMIE is set. 0 Discharge resistor to pull-down VBUS disabled 1 Discharge resistor to pull-down VBUS enabled
8 DPPU	D+ pull-up control. Indicates pull-up on D+ for FS-only applications is enabled. When set, asserts an interrupt if UOMIE is set. 0 D+ pull-up for FS-only applications disabled 1 D+ pull-up for FS-only applications enabled
7 AVLD	A-peripheral is valid. Indicates if the session for an A-peripheral is valid. 0 Session is not valid for an A-peripheral 1 Session is valid for an A-peripheral
6 BVLD	B-peripheral is valid. Indicates if the session for a B-peripheral is valid. 0 Session is not valid for a B-peripheral 1 Session is valid for a B-peripheral
5 VVLD	VBUS valid. Indicates if voltage on VBUS is at a valid level for operation. 0 Voltage level on VBUS is not valid for operation 1 Voltage level on VBUS is valid for operation
4 SEND	Session end. Indicates if voltage on VBUS has dropped below the session end threshold. 0 Voltage on VBUS has not dropped below the session end threshold 1 Voltage on VBUS has dropped below the session end threshold
3 PWRFLT	VBUS power fault. Indicates a power fault has occurred on VBUS (e.g. overcurrent). 0 No power fault has occurred 1 Power fault has occurred
2 WKUP	USB OTG controller wake-up event. Reflects if a wake-up event has occurred on the USB OTG controller bus. When set, asserts an interrupt if UOMIE is set. 0 No outstanding wake-up event 1 Wake-up event has occurred

**Table 10-9. UOCSR Field Descriptions (continued)**

Field	Description
1 UOMIE	USB OTG miscellaneous interrupt enable. Enables an interrupt to generate from any of the following UOCSR bits: DPPD, DMPD, CRGBUS, DCRVBUS, DPPU, and WKUP 0 Interrupt sources are disabled 1 Interrupt sources are enabled
0 XPDE	On-chip transceiver pull-down enable. 0 50 kΩ pull-downs disabled on OTG D+ and D- pins of on-chip transceiver 1 On-chip 50 kΩ pull-downs enabled on OTG D+ and D- transceiver pins of on-chip transceiver

### 10.3.8 USB Host Controller Status Register (UHCSR)

The UHCSR register controls and reflects various features of the USB host module. When any bit of this register generates an interrupt, that interrupt is cleared by reading the UHCSR register. The read-only bits of this register are set by the USB host module.

Address: 0xEC09\_0016 (UHCSR)

Access: Supervisor read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	PIC	0	0	0	0	0	0	0	0	0	0	DRV VBUS	PWR FLT	WKUP	UHMIE	XPDE
W																

**Figure 10-9. USB Host Controller Status Register (UHCSR)****Table 10-10. UHCSR Field Descriptions**

Field	Description
15–14 PIC	Port indicator. Reflects the state of the USB host controller port indicator signals.
13–5	Reserved, must be cleared.
4 DRVVBUS	Drive VBUS. 0 Disable the drive of 5 V power on VBUS 1 Enable the drive of 5 V power on VBUS
3 PWRFLT	VBUS power fault. Indicates a power fault has occurred on VBUS (e.g. overcurrent). 0 No power fault has occurred 1 Power fault has occurred
2 WKUP	USB host controller wake-up event. Reflects if a wake-up event has occurred on the USB host controller bus. When set, asserts an interrupt if UHMIE is set. 0 No outstanding wake-up event 1 Wake-up event has occurred
1 UHMIE	USB host miscellaneous interrupt enable. Enables an interrupt to generate if WKUP sets. 0 Interrupt sources are disabled 1 Interrupt sources are enabled
0 XPDE	On-chip transceiver pull-down enable. 0 50 kΩ pull-downs disabled on host D+ and D- pins of on-chip transceiver 1 On-chip 50 kΩ pull-downs enabled on host D+ and D- transceiver pins of on-chip transceiver

### 10.3.9 Miscellaneous Control Register 3 (MISCCR3)

Address: 0xEC09\_0018 (MISCCR3)

Access: Supervisor  
read/write

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

Figure 10-10. Miscellaneous Control Register 3 (MISCCR3)

Table 10-11. MISCCR2 Field Descriptions

Field	Description
15–13	Reserved, must be cleared.
12 TMRENET	Selects between the 1588 timebase counter outputs of the two MACs as the counter input to the 32-bit DMA timers. 0 MAC0 1588 timebase counter 1 MAC1 1588 timebase counter
11	Reserved, must be cleared.
10–8 ENETCLK	Selects the time-stamping clock for the Ethernet assembly. 000 Internal bus clock 001 Timer 0 in (T0IN) 010 Timer 1 in (T1IN) 011 Timer 2 in (T2IN) 100 Timer 3 in (T3IN) 101 USB_CLKIN 110 OSC_CLK 111 MII_TXCLK
7–0	Reserved, must be cleared.

### 10.3.10 Miscellaneous Control Register 2 (MISCCR2)

MISCCR2 configures the PLL mode, various clock options, ADC/DAC enables, and ULPI select.

Address: 0xEC09\_001A (MISCCR2)

Access: Supervisor read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	EXTCLK	DDR2	RGPIO	SWT	0	PLLMODE			DCC	DAC1	DAC0	ADC	ADC7	ADC3	FB	ULPI
W	BYP	CLK	HALF	SCR					BYP	SEL	SEL	EN	EN	EN	HALF	ULPI
Reset	1	0	1	0	0	0	0	0	1	0	0	1	1	1	See note	1

**Note:** Depends on SBF\_RCON[15]. Else, 1.

Figure 10-11. Miscellaneous Control Register 2 (MISCCR2)

Table 10-12. MISCCR2 Field Descriptions

Field	Description
15 EXTCLKBYP	Bypass the mcPWM's EXT_CLK input synchronizer. If the mcPWM EXT_CLK is sourced from the ENET's timestamping clock then the synchronizer should only be bypassed when the source for the timestamping clock is the internal bus clock. 0 Do not bypass input synchronizer 1 Bypass input synchronizer
14 DDR2CLK	Selects the source of the 2x clock for the DDR controller. 0 Core clock drives DDR 2x clock 1 PLL VCO drives DDR 2x clock
13 RGPIOHALF	Enables half-speed RGPIOP operation. 0 RGPIOP data registers run at core clock. Max toggle rate is $f_{sys}/2$ . 1 RGPIOP data registers run at half core clock. Max toggle rate is $f_{sys}/4$ .
12 SWTSCR	Selects the clock source for the software watchdog timer. This bit is write-once and is cleared only by a power-on reset. 0 System bus clock 1 On-chip 32-kHz RTC oscillator
11	Reserved, must be cleared.
10–8 PLLMODE	PLL mode. 000 Function mode Else Reserved <b>Note:</b> This bit should never be altered.
7 DCCBYP	Enables bypass of the DCC input clock to its output, without any duty cycle correction. 0 DCC output is a duty-cycle corrected version of its input clock 1 DCC in bypass mode. Input clock passed directly to output
6 DAC1SEL	Controls if DAC1 actively drives its analog output or tristates it for sharing with other logic. 0 DAC1 tristates its analog output 1 DAC1 actively drives its analog output
5 DAC0SEL	Controls if DAC0 actively drives its analog output or tristates it for sharing with other logic. 0 DAC0 tristates its analog output 1 DAC0 actively drives its analog output
4 ADCEN	Enables the ADC channels 6–4 and 2–0 0 Disabled 1 Enabled
3 ADC7EN	Enables ADC channel 7. 0 Disabled 1 Enabled
2 ADC3EN	Enables ADC channel 3. 0 Disabled 1 Enabled
1 FBHALF	FlexBus half clock enable 0 FlexBus runs at $f_{sys}/2$ 1 FlexBus runs at $f_{sys}/4$
0 ULPI	ULPI select bit. 0 USB OTG uses the external ULPI interface 1 USB host uses the external ULPI interface

### 10.3.11 ADC Trigger Select Register (ADCTSR)

ADCTSR selects the triggers for the ADCs.

Address: 0xEC09_001C (ADCTSR)																Access: Supervisor read/write							
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0							
W	0	0	0		ADC1CH		ADC1SRC		0	0	0		ADC0CH		ADC0SRC								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 10-12. ADC Trigger Select Register (ADCTSR)

Table 10-13. ADCTSR Field Descriptions

Field	Description
15–13	Reserved, must be cleared.
12–11 ADC1CH	ADC1 trigger source channel select. Selects a particular channel of the trigger source for ADC1. 00 Channel 0 01 Channel 1 10 Channel 2 11 Channel 3
10–8 ADC1SRC	ADC1 trigger source select. 000 PWM_A[3:0] 001 PWM_B[3:0] 010 PWM_X[3:0] 011 PWM_TRIG1[3:0] 100 PWM_TRIG0[3:0] 101 TnOUT 110 TnIN 111 mcPWM's local reload (ADC1CH = 11) or IRQ $n$ , where $n$ = 6 (ADC1CH = 10), 3 (ADC1CH = 01), 1 (ADC1CH = 00)
7–5	Reserved, must be cleared.
4–3 ADC0CH	ADC0 trigger source channel select. Selects a particular channel of the trigger source for ADC0. 00 Channel 0 01 Channel 1 10 Channel 2 11 Channel 3
2–0 ADC0SRC	ADC0 trigger source select. 000 PWM_A[3:0] 001 PWM_B[3:0] 010 PWM_X[3:0] 011 PWM_TRIG1[3:0] 100 PWM_TRIG0[3:0] 101 TnOUT 110 TnIN 111 mcPWM's local reload (ADC1CH = 11) or IRQ $n$ , where $n$ = 6 (ADC1CH = 10), 3 (ADC1CH = 01), 1 (ADC1CH = 00)

### 10.3.12 DAC Trigger Select Register (DACTSR)

DACTSR selects the triggers for the DACs.

Address: 0xEC09_001E (DACTSR)																Access: Supervisor read/write							
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0							
W	0	0	0		DAC1CH		DAC1SRC		0	0	0		DAC0CH		DAC0SRC								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							

Figure 10-13. DAC Trigger Select Register (DACTSR)

Table 10-14. DACTSR Field Descriptions

Field	Description
15–13	Reserved, must be cleared.
12–11 DAC1CH	DAC1 trigger source channel select. Selects a particular channel of the trigger source for DAC1. 00 Channel 0 01 Channel 1 10 Channel 2 11 Channel 3
10–8 DAC1SRC	DAC1 trigger source select. 000 PWM_A[3:0] 001 PWM_B[3:0] 010 PWM_X[3:0] 011 PWM_TRIG1[3:0] 100 PWM_TRIG0[3:0] 101 TnOUT 110 TnIN 111 mcPWM's local reload (ADC1CH = 11) or IRQ $n$ , where $n$ = 6 (ADC1CH = 10), 3 (ADC1CH = 01), 1 (ADC1CH = 00)
7–5	Reserved, must be cleared.
4–3 DAC0CH	DAC0 trigger source channel select. Selects a particular channel of the trigger source for DAC0. 00 Channel 0 01 Channel 1 10 Channel 2 11 Channel 3
2–0 DAC0SRC	DAC0 trigger source select. 000 PWM_A[3:0] 001 PWM_B[3:0] 010 PWM_X[3:0] 011 PWM_TRIG1[3:0] 100 PWM_TRIG0[3:0] 101 TnOUT 110 TnIN 111 mcPWM's local reload (ADC1CH = 11) or IRQ $n$ , where $n$ = 6 (ADC1CH = 10), 3 (ADC1CH = 01), 1 (ADC1CH = 00)

### 10.3.13 FlexBus/NAND Flash Arbiter Control Register (FNACR)

The FlexBus-NFC arbiter control register (FNACR) has a pair of bit fields (PCR and MCC), that specify the FlexBus post-cycle reservation period and NAND flash consecutive cycle count in internal bus clock cycles.

Address: 0xEC09_0024 (FNACR)																Access: Supervisor read/write															
R				PCR				MCC								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
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

Figure 10-14. FlexBus/NAND Flash Arbiter Control Register (FNACR)

Table 10-15. UHPCR & UOPCR Field Descriptions

Field	Description
31–28	Reserved, must be cleared.
27–24 PCR	Post-cycle reservation. Specifies the length of the FlexBus post-cycle reservation period in internal bus clock cycles. 0x0 or 0x1 No post-cycle reservation, and the arbiter returns to the idle loop as soon as the FlexBus is no longer busy. 0x2 or more Enables post-cycle reservation
23–16	Reserved, must be cleared.
15–0 MCC	Minimum consecutive cycles. Specifies the minimum amount of time in internal bus clock cycles provided by the arbiter for NAND flash activity. At least MCC internal bus clock cycles must elapse before a FlexBus request is recognized, and the NFC may, in fact, use the shared pins for more than MCC internal bus clock cycles absent such a request. A value of 0 makes the arbiter respond immediately to a FlexBus request and pause the NFC as soon as possible.

## 10.4 Functional Description

### 10.4.1 Reset Configuration

During reset, the pins for the reset override functions are immediately configured to known states.

Table 10-16 shows the states of the external pins while in reset.

Table 10-16. Reset Configuration Pin States During Reset

Pin	Pin Function	I/O	Input State
BOOTMOD[1:0]	BOOTMOD function for all modes	I	Must be driven by external logic
FB_AD[7:0]	Flexbus address/data functions (BOOTMOD ≠ 01)	I	N/A
	Reset configuration data functions during reset (BOOTMOD = 01)	I	Must be driven by external logic

### 10.4.1.1 Reset Configuration (BOOTMOD[1:0] = 00)

If the BOOTMOD pins are 00 during reset, the RCON register determines the chip configuration after reset, regardless of the states of the external address/data pins. The internal configuration signals are driven to levels specified by the RCON register's reset state for default module configuration.

### 10.4.1.2 Reset Configuration (BOOTMOD[1:0] = 01)

If the BOOTMOD pins are 01 during reset, the chip configuration after reset is determined according to the levels driven onto the FB\_AD[7:0] pins. (See [Table 10-17](#).) The internal configuration signals are driven to reflect the levels on the external configuration pins to allow for module configuration.

#### NOTE

The logic levels for reset configuration on FB\_AD[7:0] must be actively driven when BOOTMOD is 01. The FB\_AD[31:8] pins must float or be pulled high.

**Table 10-17. Parallel Configuration During Reset<sup>1</sup>**

Pin(s) Affected	Default Configuration	Override Pins in Reset <sup>2</sup>	Function
FlexBus/NFC signals	See RCON[0]	<b>FB_ADO</b>	<b>Boot memory</b>
		0	NAND flash
		1	FlexBus
(none)	See RCON[1]	<b>FB_AD1</b>	<b>PLL mode</b>
		0	Disabled
		1	Enabled
(none)	See RCON[2]	<b>FB_AD2</b>	<b>Oscillator mode</b> <b>Note:</b> Only takes affect on a POR-initiated parallel RCON fetch
		0	Crystal oscillator mode
		1	Oscillator bypass mode
FB_ALE	See RCON[3]	<b>FB_AD3</b>	<b>FB_ALE select</b>
		0	FB_TS
		1	FB_ALE
FB_AD[31:0]	See RCON[5:4]	<b>FB_AD[5:4]</b>	<b>Boot port size</b>
		00	32-bit (32-bit muxed address)
		01	8-bit (24-bit non-muxed address)
		10	16-bit (16-bit non-muxed address)
		11	16-bit (16-bit non-muxed address)

**Table 10-17. Parallel Configuration During Reset<sup>1</sup> (continued)**

<b>Pin(s) Affected</b>	<b>Default Configuration</b>	<b>Override Pins in Reset<sup>2</sup></b>	<b>Function</b>
(none)	See RCON[7:6]	<b>FB_AD[7:6]</b>	<b>PLL multiplier</b>
		00	$f_{VCO} = 10 \times f_{ref}$
		01	$f_{VCO} = 15 \times f_{ref}$
		10	$f_{VCO} = 16 \times f_{ref}$
		11	$f_{VCO} = 20 \times f_{ref}$

<sup>1</sup> Modifying the default configurations through the FB\_AD[7:0] pins is possible only if the external BOOTMOD[1:0] pins are 01 while  $\overline{RSTOUT}$  is asserted.

<sup>2</sup> The external reset override circuitry drives the address bus pins with the override values while  $\overline{RSTOUT}$  is asserted. It must stop driving the address bus pins within one FB\_CLK cycle after  $\overline{RSTOUT}$  is negated. To prevent contention with the external reset override circuitry, the reset override pins are forced to inputs during reset and do not become outputs until at least one FB\_CLK cycle after  $\overline{RSTOUT}$  is negated.

#### 10.4.1.3 Reset Configuration (BOOTMOD[1:0] = 1x)

If the BOOTMOD pins are 11 during reset, then the chip configuration after reset is determined by data obtained from external SPI memory through serial boot using the SBF\_DI, SBF\_DO,  $\overline{SBF\_CS}$ , and SBF\_CK signals. (See [Table 10-18](#).) The internal configuration signals are driven to reflect the data being received from the external SPI memory to allow for module configuration. See [Chapter 11, “Serial Boot Facility \(SBF\),”](#) for more details on serial boot.

**Table 10-18. Serial Configuration During Reset**

<b>Pin(s) Affected</b>	<b>Default Configuration</b>	<b>Override Serial RCON Bits</b>	<b>Function</b>
(none)	See RCON[5:4]	<b>SBF_RCON[31:30]</b>	<b>Boot Port Size</b>
		00	32-bit (32-bit muxed address)
		01	8-bit (24-bit non-muxed address)
		10	16-bit (16-bit non-muxed address)
		11	16-bit (16-bit non-muxed address)
FlexBus/NFC signals	See RCON[0]	<b>SBF_RCON[29]</b>	<b>Boot memory</b>
		0	NAND flash
		1	FlexBus
(none)	—	<b>SBF_RCON[28]</b>	Reserved
(none)	—	<b>SBF_RCON[27]</b>	Reserved
FB_ALE	See RCON[3]	<b>SBF_RCON[26]</b>	<b>FB_ALE select</b>
		0	$\overline{FB\_TS}$
		1	FB_ALE

Table 10-18. Serial Configuration During Reset (continued)

Pin(s) Affected	Default Configuration	Override Serial RCON Bits	Function
(none)	See RCON[2]	<b>SBF_RCON[25]</b>	<b>Oscillator mode</b> <b>Note:</b> Only takes affect on a POR-initiated serial RCON fetch
		0	Crystal oscillator mode
		1	Oscillator bypass mode
(none)	See RCON[1]	<b>SBF_RCON[24]</b>	<b>PLL mode</b>
		0	Disabled
		1	Enabled
(none)	See PLL_CR	<b>SBF_RCON[23:22]</b>	<b>PLL reference divider (lower 2 bits of PLL_CR[REFDIV])</b>
		00	1
		01	2
		10	Reserved
		11	Reserved
(none)	See PLL_CR	<b>SBF_RCON[21:16]</b>	<b>PLL reference clock multiplier</b> This value is loaded into PLL_CR[FBKDIV].
FB_CLK	1	<b>SBF_RCON[15]</b>	<b>FlexBus half clock enable</b>
		0	FlexBus runs at $f_{sys}/2$
		1	FlexBus runs at $f_{sys}/4$
(none)	See PLL_DR	<b>SBF_RCON[14:10]</b>	<b>NFC clock frequency divider</b> This value is loaded into PLL_DR[OUTDIV5].
(none)	See PLL_DR	<b>SBF_RCON[9:5]</b>	<b>Internal bus clock frequency divider</b> This value is loaded into PLL_DR[OUTDIV2].
(none)	See PLL_DR	<b>SBF_RCON[4:0]</b>	<b>Core bus clock frequency divider</b> This value is loaded into PLL_DR[OUTDIV1].

## 10.4.2 Boot Configuration

During reset configuration, the FB\_CS0 chip select pin is always configured to select an external boot device. The valid (V) bit in the CSMR0 register is ignored and FB\_CS0 is enabled after reset. FB\_CS0 is asserted for the initial boot fetch accessed from address 0x0000\_0000 for the stack pointer and address 0x0000\_0004 for the program counter (PC). It is assumed the reset vector loaded from address 0x0000\_0004 causes the processor to start executing from external memory space decoded by FB\_CS0.

### 10.4.3 Low Power Configuration

After reset, the device can be configured for operation during the low power modes using the low power control register (LPCR). For more information on this register, see [Chapter 9, “Power Management.”](#)



# Chapter 11

## Serial Boot Facility (SBF)

### 11.1 Introduction

It is nearly impossible to dedicate and very impractical to share pins for the numerous available power-up options on today's complex, highly-integrated processors. The serial boot facility (SBF), shown in [Figure 11-1](#), solves this problem by providing the user with the capability to store and load all device reset configuration data and user code from an external SPI memory. This method requires only a minimal number of I/O pins.

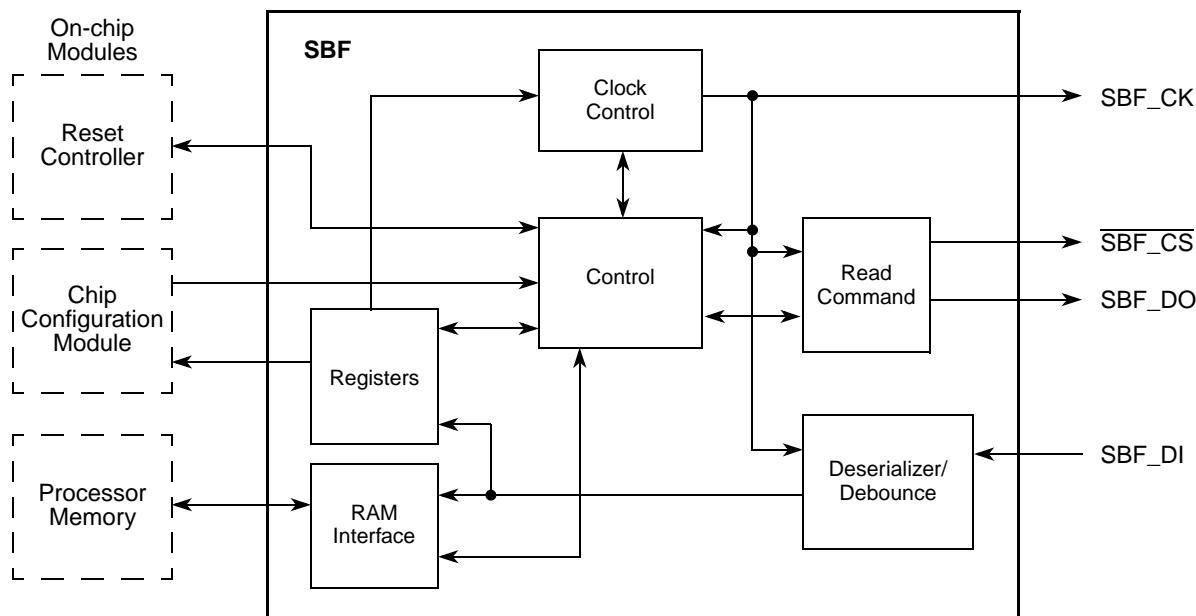


Figure 11-1. SBF Block Diagram

#### 11.1.1 Overview

The SBF interfaces to an external SPI memory to read configuration data and boot code during the processor reset sequence if BOOTMOD[1:0] equals 1x. By reading data stored in the SPI memory, the SBF adjusts the SPI memory clock frequency, configures an extended set of power-up options for the processor, and optionally loads code into the on-chip SRAM. Through interaction with the reset controller, the SBF performs these actions so that the chip is properly configured after exiting the reset state.

### 11.1.2 Features

The SBF includes these distinctive features:

- Support for many different SPI memory devices
  - EEPROM
  - Flash
  - FRAM
  - Embedded FPGA memory
- External interface maps directly to (and can be multiplexed with) the DMA serial peripheral interface (DSPI) pins
- Self-adjusting shift clock frequency for maximum throughput supported by SPI memory
- Optionally load boot code into processor's memory space

## 11.2 External Signal Description

Listed below are the SBF module external signals.

**Table 11-1. Signal Properties**

Signal	I/O	Description	Reset	Pull Up
SBF_CK	O	Shift clock. Alternate edges of this signal cause the SPI memory to accept data from and drive data to the processor	—	—
SBF_CS	O	Chip select. This signal enables the SPI memory and places it into an active state, ready to accept commands.	—	—
SBF_DI	I	Data in. The SPI memory drives and the processor accepts read data on this signal.	—	Active <sup>1</sup>
SBF_DO	O	Data out. The SBF drives the read command and address on this signal.	—	—

<sup>1</sup> Disabled by the SBF when the SPI memory begins shifting out data.

## 11.3 Memory Map/Register Definition

The SBF programming model consists of the registers listed below.

**Table 11-2. SBF Memory Map**

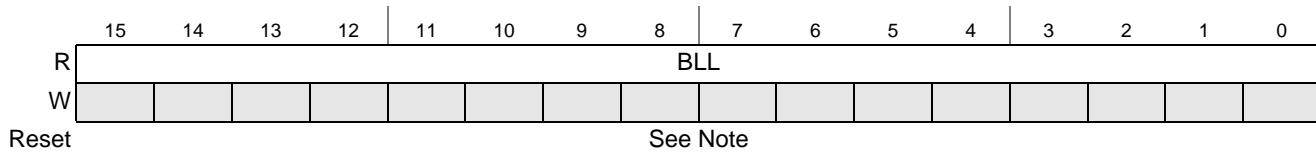
Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xEC09_0020	Serial boot facility status register (SBFSR)	16	R	See Section	<a href="#">11.3.1/11-3</a>
0xEC09_0022	Serial boot facility control register (SBFCR)	16	R/W	See Section	<a href="#">11.3.2/11-3</a>

### 11.3.1 Serial Boot Facility Status Register (SBFSR)

The read-only SBFSR register reflects the amount of boot code loaded through the external SPI memory.

Address: 0xEC09\_0020 (SBFSR)

Access: User read-only



**Note:** Reset value is user-defined (loaded from SPI memory during serial boot following any reset type)

**Figure 11-2. Serial Boot Facility Status Register (SBFSR)**

**Table 11-3. SBFSR Field Descriptions**

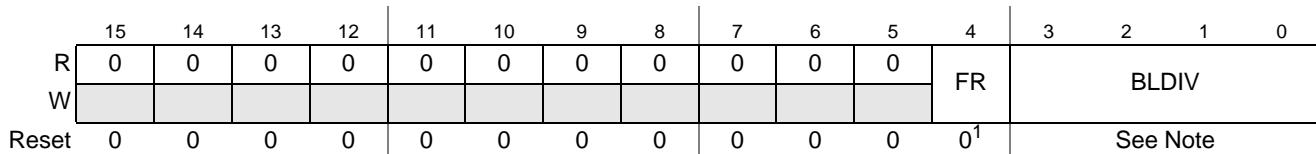
Field	Description
15–0 BLL	Boot load length. Reflects the number of longwords of boot code loaded from external SPI memory during serial boot. No boot code was loaded if BLL equals 0x0000. Otherwise, BLL plus 1 longwords were loaded.

### 11.3.2 Serial Boot Facility Control Register (SBFCR)

The read-always/write-once SBFCR register controls SBF operation following subsequent warm resets.

Address: 0xEC09\_0022 (SBFCR)

Access: User read/write-once



<sup>1</sup> Reset value is 0 and is reset only by power-on reset (remains unchanged for other reset types)

**Note:** The reset value is loaded from SPI memory during serial boot following power-on reset. It remains unchanged for other reset types. Prior to this register being loaded from SPI memory, a divisor of 67 is used to begin the serial boot sequence.

**Figure 11-3. Serial Boot Facility Control Register (SBFCR)**

**Table 11-4. SBFCR Field Descriptions**

Field	Description
15–5	Reserved, must be cleared.
4 FR	Fast read. Determines whether the SBF uses the standard READ command or flash FAST_READ command on reboot following any reset other than power-on reset. Because this register is write-once, the application must write the value for this bit in the same write that the BLDIV field is written. Any subsequent writes to this field prior to a power-on reset event terminate without effect. 0 SBF uses the standard READ command 1 SBF uses the FAST_READ command

Table 11-4. SBFCR Field Descriptions (continued)

Field	Description																																																																															
3–0 BLDIV	<p>Boot loader clock divider. Determines the SBF clock (PLL input reference clock) divisor that generates the serial shift clock output on SBF_CK. Prior to the serial boot sequence, a divisor of 67 is used.</p> <p>During the serial boot sequence, this field is loaded with the value read from the SPI memory. The application may write to this register to change the divisor for any subsequent serial boot that follows a soft-reset condition.</p> <p>Because this register is write-once, the application must write the value for this field in the same write that the FR bit is written (regardless of the value written to the FR bit). Any subsequent writes to this field prior to a power-on reset event terminate without effect.</p>																																																																															
	<table border="1"> <thead> <tr> <th rowspan="2">BLDIV</th> <th rowspan="2">Ideal Divisor</th> <th colspan="2">Shift Clock</th> </tr> <tr> <th>High Time (f<sub>ref</sub> Ticks)</th> <th>Low Time (f<sub>ref</sub> Ticks)</th> </tr> </thead> <tbody> <tr> <td>0000</td> <td>1</td> <td>Bypass</td> <td>Bypass</td> </tr> <tr> <td>0001</td> <td>2</td> <td>1</td> <td>1</td> </tr> <tr> <td>0010</td> <td>3</td> <td>2</td> <td>1</td> </tr> <tr> <td>0011</td> <td>4</td> <td>2</td> <td>2</td> </tr> <tr> <td>0100</td> <td>5</td> <td>3</td> <td>2</td> </tr> <tr> <td>0101</td> <td>7</td> <td>4</td> <td>3</td> </tr> <tr> <td>0110</td> <td>10</td> <td>5</td> <td>5</td> </tr> <tr> <td>0111</td> <td>13</td> <td>7</td> <td>6</td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th rowspan="2">BLDIV</th> <th rowspan="2">Ideal Divisor</th> <th colspan="2">Shift Clock</th> </tr> <tr> <th>High Time (f<sub>ref</sub> Ticks)</th> <th>Low Time (f<sub>ref</sub> Ticks)</th> </tr> </thead> <tbody> <tr> <td>1000</td> <td>14</td> <td>7</td> <td>7</td> </tr> <tr> <td>1001</td> <td>17</td> <td>9</td> <td>8</td> </tr> <tr> <td>1010</td> <td>25</td> <td>13</td> <td>12</td> </tr> <tr> <td>1011</td> <td>27</td> <td>14</td> <td>13</td> </tr> <tr> <td>1100</td> <td>33</td> <td>17</td> <td>16</td> </tr> <tr> <td>1101</td> <td>34</td> <td>17</td> <td>17</td> </tr> <tr> <td>1110</td> <td>50</td> <td>25</td> <td>25</td> </tr> <tr> <td>1111</td> <td>60</td> <td>30</td> <td>30</td> </tr> </tbody> </table>				BLDIV	Ideal Divisor	Shift Clock		High Time (f <sub>ref</sub> Ticks)	Low Time (f <sub>ref</sub> Ticks)	0000	1	Bypass	Bypass	0001	2	1	1	0010	3	2	1	0011	4	2	2	0100	5	3	2	0101	7	4	3	0110	10	5	5	0111	13	7	6	BLDIV	Ideal Divisor	Shift Clock		High Time (f <sub>ref</sub> Ticks)	Low Time (f <sub>ref</sub> Ticks)	1000	14	7	7	1001	17	9	8	1010	25	13	12	1011	27	14	13	1100	33	17	16	1101	34	17	17	1110	50	25	25	1111	60	30	30
BLDIV	Ideal Divisor	Shift Clock																																																																														
		High Time (f <sub>ref</sub> Ticks)	Low Time (f <sub>ref</sub> Ticks)																																																																													
0000	1	Bypass	Bypass																																																																													
0001	2	1	1																																																																													
0010	3	2	1																																																																													
0011	4	2	2																																																																													
0100	5	3	2																																																																													
0101	7	4	3																																																																													
0110	10	5	5																																																																													
0111	13	7	6																																																																													
BLDIV	Ideal Divisor	Shift Clock																																																																														
		High Time (f <sub>ref</sub> Ticks)	Low Time (f <sub>ref</sub> Ticks)																																																																													
1000	14	7	7																																																																													
1001	17	9	8																																																																													
1010	25	13	12																																																																													
1011	27	14	13																																																																													
1100	33	17	16																																																																													
1101	34	17	17																																																																													
1110	50	25	25																																																																													
1111	60	30	30																																																																													

## 11.4 Functional Description

When enabled, the SBF inserts three additional steps into the normal system boot process:

- Serial initialization and shift clock frequency adjustment
- Reset configuration and optional boot load
- Execution transfer

### 11.4.1 Serial Initialization and Shift Clock Frequency Adjustment

The following sequence is followed during a serial boot sequence:

1. The SBF is engaged when BOOTMOD[1:0] = 1x concurrent with the release of a pending source of reset (power-on, software watchdog, R<sub>RESET</sub> pin, etc.).
2. Boot-up is paused.
3. The weak internal pull-up on SBF\_DI is enabled. This allows a 1-to-0 transition to register when the SPI memory output switches from high-impedance to logic 0.
4. The SBF shifts the standard SPI memory read command (0x03) followed by repeated 0x00 address bytes to the SPI memory at f<sub>REF</sub> ÷ 60.

5. After the SPI memory accepts however many shift clock edges are necessary to respond to the READ command, it turns on its previously tri-stated output and begins driving the msb of the byte at address 0.  
Bits [7:4] of this byte must be 0000, so that the required 1-to-0 transition can be detected on SBF\_DI to synchronize the SBF state machine. If bits [7:4] of this byte are not 0000, bits[3:0] are ignored, another byte is clocked out of the SPI memory (SBF\_DO remains at logic 0), and the SBF state machine again tests for a 1-to-0 transition followed by four consecutive zero bits.
6. After the necessary 1-to-0 transition and reception of a byte with bits [7:4] equal to 0000, the SBF pauses and bits [3:0] of the received byte select a new shift clock divider according to [Table 11-4](#).
7. The weak internal pull-up on SBF\_DI is disabled.
8. The shift clock begins toggling at the new frequency, resuming the READ command already in progress.

**NOTE**

Shift clock frequency adjustment follows a power-on/hard reset only. After the new divisor is known, it is stored in the sticky SBFCR[BLDIV] field and used for subsequent soft resets. This speeds reboot for systems that do not benefit from the optional FAST\_READ on soft reset feature (e.g., the SPI memory does not support FAST\_READ, or the input reference clock does not exceed the maximum allowable frequency for the READ command).

### 11.4.2 Reset Configuration and Optional Boot Load

After the steps in [Section 11.4.1, “Serial Initialization and Shift Clock Frequency Adjustment”](#), are executed, the following is performed to load configuration data and optional boot code.

1. Next, the SBF shifts two bytes (16 bits) out of the SPI memory that indicate how many longwords, if any, are to be read during the optional boot load sequence. These bytes are software-visible in the SBFSR[BLL] field.
2. The read operation continues with one longwords (32 bits) of reset configuration data, formatted in the order presented in [Section 10.4.1.3, “Reset Configuration \(BOOTMOD\[1:0\] = 1x\)”](#).
3. At this point, the SBF determines whether or not to read boot code. If SBFSR[BLL] is non-zero, BLL plus one longwords ( $4 \times (BLL + 1)$  bytes) are consecutively loaded into the SRAM.

**NOTE**

Although the SBF permits up to 65,536 longwords (262,144 bytes) to be loaded, the maximum practical number that can be read is limited by the size of the device’s internal SRAM (16,384 longwords (65,536 bytes) for this device).

### 11.4.3 Execution Transfer

After boot load is complete or if no boot load is requested (SBFSR[BLL] = 0), the following steps complete the serial boot process:

1. The acquired configuration data is driven to the appropriate modules.

2. The system is released from reset.
3. The ColdFire processor initiates its normal reset vector fetch at address 0.
4. The actual memory that responds to the reset vector fetch depends on whether serial boot load is requested:
  - If SBFSR[BLL] is cleared, the reset vector fetch is handled by the FlexBus module or NAND flash controller, and whatever external memory is mapped at address 0, governed by the user-provided setting of RCON/CCR[BOOTPS,BOOTMEM].
  - If SBFSR[BLL] is set, the reset vector and boot code are read from the on-chip SRAM. (The SBF enables the SRAM and maps it to address 0 via the RAMBAR before control of the processor is restored to the ColdFire core.) The reset vector (initial stack pointer and program counter) should point to locations in the on-chip SRAM, so that boot code can initialize the device and load the application software from the SPI memory or via some other mechanism (e.g. a network server responding to a TFTP client).

## 11.5 Initialization Information

### 11.5.1 SPI Memory Initialization

The SBF requires that, prior to device power-up, the SPI memory is loaded with data organized according to [Table 11-5](#). See [Section 10.4.1.3, “Reset Configuration \(BOOTMOD\[1:0\] = 1x\),”](#) for the reset configuration (SBF\_RCON) data definition.

**Table 11-5. SPI Memory Organization**

Byte Address	Data Contents
0x0	{0000,BLDIV[3:0]}
0x1	BLL[7:0]
0x2	BLL[15:8]
0x3	RCON[7:0]
0x4	RCON[15:8]
0x5	RCON[23:16]
0x6	RCON[31:24]
0x7 <sup>1</sup>	CODE_BYTE_0 <sup>2</sup>
0x8 <sup>1</sup>	CODE_BYTE_1
...	...
0x6 + 4 × (BLL + 1) <sup>1</sup>	CODE_BYTE_[4 × (BLL + 1) - 1]

<sup>1</sup> This assumes SBFSR[PLL] is non-zero. If PLL is zero, the SBF does not access data at these addresses.

<sup>2</sup> Start of user code copied into the on-chip SRAM. CODE\_BYTE\_0–3 is the supervisor stack pointer (SP) when loading completes. CODE\_BYTE\_4–7 is the program counter (PC) when loading completes.

## 11.5.2 FAST\_READ Feature Initialization

Many SPI flash memories implement a FAST\_READ command that allows for a substantially higher shift-clock frequency. The SBF always uses the normal read command when coming out of a power-on/hard reset. However, when coming out of a soft reset, it is possible to use FAST\_READ because the SBF machine state is not lost.

For this reason, the SBFCR[FR] sticky bit may be set, causing the FAST\_READ command to be issued instead of the read command in the event of a soft reset. To enable the FAST\_READ feature, set SBFCR[FR] in the same write that sets the SBFCR[BLDIV] field. The value written to SBFCR[BLDIV] should correspond to the frequency the SPI memory supports in FAST\_READ mode. After a soft reset, SBFCR[BLDIV] is not overwritten with the BLDIV[3:0] value read from the SPI memory. Instead, the SBF uses the SBFCR[BLDIV] value to determine the SPI memory clock.

### NOTE

The ability to use the FAST\_READ command is limited by the SBF electrical specifications. Specifically, delays present throughout the system (including those between the SBF, the pin multiplexing logic, and the actual I/O pads) effectively limit the maximum frequency at which the SBF operates and can preclude use of the FAST\_READ feature altogether. Even when the delays within the processor itself are minimized, the actual SPI memories may have similarly untenable electrical specifications (data input setup and output valid times).



# Chapter 12

## Reset Controller Module

### 12.1 Introduction

The reset controller determines the cause of reset, asserts the appropriate reset signals to the system, and keeps a history of what caused the reset.

#### 12.1.1 Block Diagram

Figure 12-1 illustrates the reset controller and is explained in these:

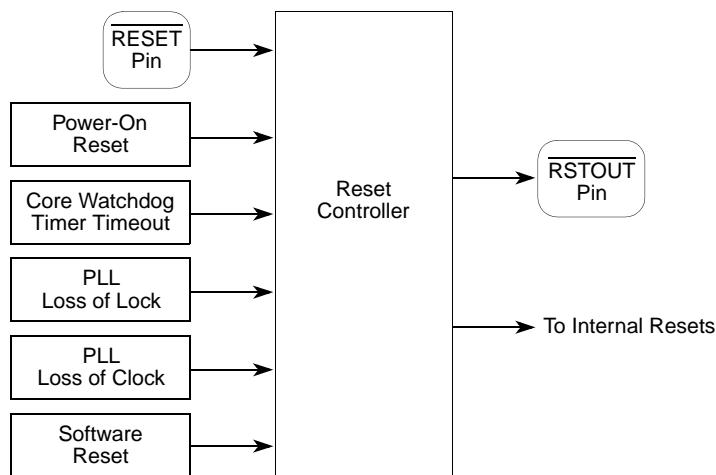


Figure 12-1. Reset Controller Block Diagram

#### 12.1.2 Features

Module features include the following:

- Six sources of reset:
  - External reset
  - Power-on reset (POR)
  - Watchdog timer
  - Phase locked-loop (PLL) loss of lock
  - Phase locked-loop (PLL) loss of reference clock
  - Software
- Software-assertable  $\overline{\text{RSTOUT}}$  pin independent of chip-reset state
- Software-readable status flags indicating the cause of the last reset

## 12.2 External Signal Description

**Table 12-1** provides a summary of the reset-controller signal properties. The signals are described in the following paragraphs.

**Table 12-1. Reset Controller Signal Properties**

Name	I/O	Pull-up	Input Hysteresis	Input Synchronization
RESET	I	Active	Y	Y <sup>1</sup>
RSTOUT	O	—	—	—

<sup>1</sup> RESET is always synchronized except when in low-power stop mode.

### 12.2.1 **RESET**

Asserting the external **RESET** for at least four rising FB\_CLK edges (if MISCCR2[FBHALF] is cleared; eight FB\_CLK edges if FBHALF is set) causes the external reset request to be recognized and latched.

### 12.2.2 **RSTOUT**

This active-low output signal is driven low when the internal reset controller resets the device. It may take up to six FB\_CLK edges after **RESET** assertion for **RSTOUT** to assert, due to an internal synchronizer on **RESET**. When **RSTOUT** is active, the user can drive override options on the data bus. See [Chapter 10, “Chip Configuration Module \(CCM\),”](#) for more details on these override options.

## 12.3 Memory Map/Register Definition

See [Table 12-2](#) for the memory map and the following sections for register descriptions.

**Table 12-2. Reset Controller Memory Map**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xEC09_0000	Reset Control Register (RCR)	8	R/W	0x00	<a href="#">12.3.1/12-3</a>
0xEC09_0001	Reset Status Register (RSR)	8	R	See Section	<a href="#">12.3.2/12-3</a>

### 12.3.1 Reset Control Register (RCR)

The RCR allows software control for requesting a reset and for independently asserting the external RSTOUT pin.

Address: 0xEC09_0000 (RCR)								Access: User read/write	
		7	6	5	4	3	2	1	0
R	SOFTRST	FRCRSTOUT	0	0	0	0	0	0	0
W									
Reset:	0	0	0	0	0	0	0	0	0

Figure 12-2. Reset Control Register (RCR)

Table 12-3. RCR Field Descriptions

Field	Description
7 SOFTRST	Allows software to request a reset. The reset caused by setting this bit clears this bit. 0 No software reset request 1 Software reset request
6 FRCRSTOUT	Allows software to assert or negate the external <u>RSTOUT</u> pin. 0 Negate <u>RSTOUT</u> pin 1 Assert <u>RSTOUT</u> pin <b>Note:</b> External logic driving reset configuration data during reset needs to be considered when asserting the <u>RSTOUT</u> pin by setting FRCRSTOUT.
5–0	Reserved, must be cleared.

### 12.3.2 Reset Status Register (RSR)

The RSR contains a status bit for every reset source. When reset is entered, the cause of the reset condition is latched, along with a value of 0 for the other reset sources that were not pending at the time of the reset condition. These values are then reflected in RSR. One or more status bits may be set at the same time. The cause of any subsequent reset is also recorded in the register, overwriting status from the previous reset condition.

RSR can be read at any time. Writing to RSR has no effect.

Address: 0xEC09_0001 (RSR)								Access: User read-only	
		7	6	5	4	3	2	1	0
R	0	0	SOFT	LOC	POR	EXT	WDR CORE	LOL	
W									
Reset:	0	0	Reset Dependent	Reset Dependent	Reset Dependent	Reset Dependent	Reset Dependent	Reset Dependent	Reset Dependent

Figure 12-3. Reset Status Register (RSR)

**Table 12-4. RSR Field Descriptions**

Field	Description
7–6	Reserved, must be cleared.
5 SOFT	Software reset flag. Indicates the software caused the last reset. 0 Last reset not caused by software 1 Last reset caused by software
4 LOC	Loss-of-clock reset flag. Indicates PLL loss of reference clock caused the last reset. 0 Last reset not caused by PLL loss-of-clock 1 Last reset caused by PLL loss-of-clock <b>Note:</b> This bit is not set if a loss-of-clock occurs during stop mode.
3 POR	Power-on reset flag. Indicates power-on reset caused the last reset. 0 Last reset not caused by power-on reset 1 Last reset caused by power-on reset
2 EXT	External reset flag. Indicates that the last reset was caused by an external device or circuitry asserting the external <b>RESET</b> pin. 0 Last reset not caused by external reset 1 Last reset caused by external reset
1 WDRCORE	Core watchdog timer reset flag. Indicates the core watchdog timer timeout caused the last reset. 0 Last reset not caused by watchdog timer timeout 1 Last reset caused by watchdog timer timeout
0 LOL	Loss-of-lock reset flag. Indicates the last reset state was caused by a PLL loss of lock. 0 Last reset not caused by loss of lock 1 Last reset caused by a loss of lock <b>Note:</b> This bit is not set if a loss-of-lock occurs during stop mode.

## 12.4 Functional Description

### 12.4.1 Reset Sources

Table 12-5 defines the reset sources and the signals driven by the reset controller.

**Table 12-5. Reset Source Summary**

Source	Type	Description
Power on	Asynchronous	Power-on reset at the time of power-up
External <b>RESET</b> pin (not stop mode)	Synchronous	Asserting <b>RESET</b> for at least four rising FB_CLK edges (if MISCCR2[FBHALF] is cleared; eight FB_CLK edges if FBHALF is set) causes the device to recognize and latch the external reset request.
External <b>RESET</b> pin (during stop mode)	Asynchronous	Asserting <b>RESET</b> in stop mode causes an external reset to be recognized asynchronously.
Watchdog timer	Synchronous	A watchdog timer timeout causes the timer reset request to be recognized and latched.
PLL loss of clock	Asynchronous	When the PLL detects a loss of reference clock, the system is reset if programmed to do so. This event also results in a loss of lock for the PLL and a loss of lock reset is also generated.

**Table 12-5. Reset Source Summary (continued)**

<b>Source</b>	<b>Type</b>	<b>Description</b>
PLL loss of lock	Asynchronous	When the PLL loses lock, the system is reset if programmed to do so. In limp and stop modes, this source of reset is gated off.
Software	Synchronous	A software reset occurs when the RCR[SOFRST] bit is set.

To protect data integrity, a synchronous reset source is not acted upon by the reset control logic until the end of the current bus cycle. Reset is then asserted on the next rising edge of the system clock after the cycle is terminated. Internal byte, word, or longword writes are guaranteed to complete without data corruption when a synchronous reset occurs. External writes, including longword writes to 16-bit ports, are also guaranteed to complete.

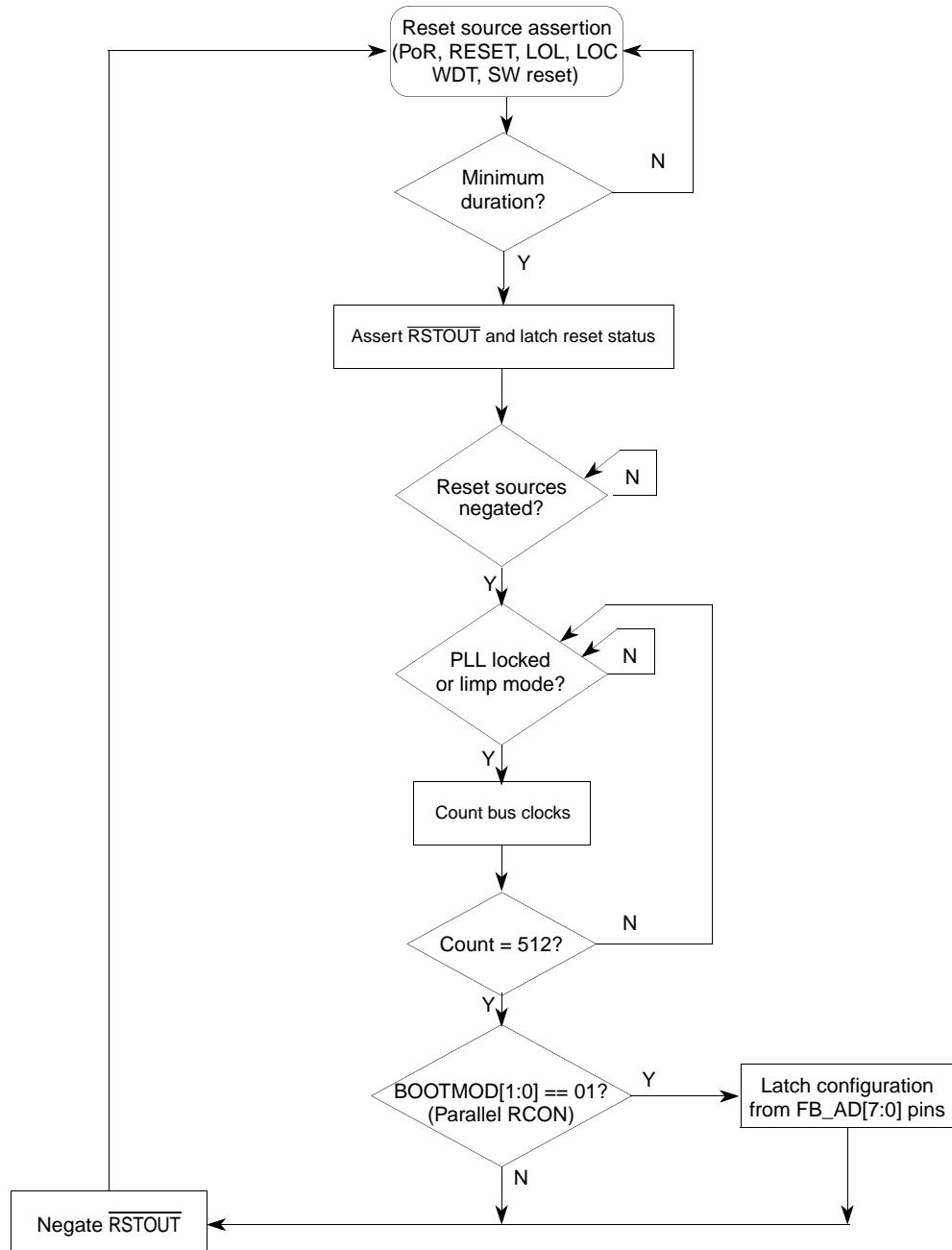
Asynchronous reset sources usually indicate a catastrophic failure. Therefore, the reset control logic does not wait for the current bus cycle to complete. Reset is immediately asserted to the system.

### 12.4.2 Reset Control Flow

The reset logic control flow depends on if the serial boot facility is enabled through the BOOTMOD pins during reset.

### 12.4.2.1 Non-SBF Reset Sequence

The non-SBF reset sequence is shown in Figure 12-4.



**Figure 12-4. Non-SBF Reset Sequence**

When a reset condition occurs, the global resets and  $\overline{\text{RESET}}$  output are asserted and the reset status is updated in relevant registers in reset controller. During an external reset, the  $\overline{\text{RESET}}$  pin must be asserted for more than four bus clock cycles for it to be recognized.  $\overline{\text{RSTOUT}}$  might take a further two clock cycles before asserting. If  $\text{BOOTMOD} = 01$ , the RCON values on the designated pins are observed, but not

latched at this point. This means that a change in the RCON pins are reflected in the reset configuration logic.

The next step is to wait for the source of reset to negate. For a reset source which resets the PLL too, the logic waits for a power-on-reset or loss-of-lock, the next step is to wait for the PLL to lock, whereas for a loss-of-clock, the system waits for reference clock and then lock to be regained. After that happens, the reset controller waits for 512 bus clock cycles, and then latches the RCON values (whether default or overridden, depending on BOOTMOD) samples the RCON pin. If the PLL loses lock during this 512 count, the logic waits for the PLL to lock and the count is started again. If it is active (low), the chip configuration override values on the data pins are latched. Thereafter the global resets and RSTOUT are de-asserted and the system is brought out of reset. Note that up until this point the system continues to be in reset, even though the original reset source may have negated.

For an external reset, the sequence has an additional step of waiting for RESET to negate. The watchdog timer timeout reset and software reset flows are similar.

While waiting for the PLL to lock, or during the 512 bus clock cycle wait, assertion of any of the reset sources results in the sequence starting again. The reset sources which are recognized during these two stages are:

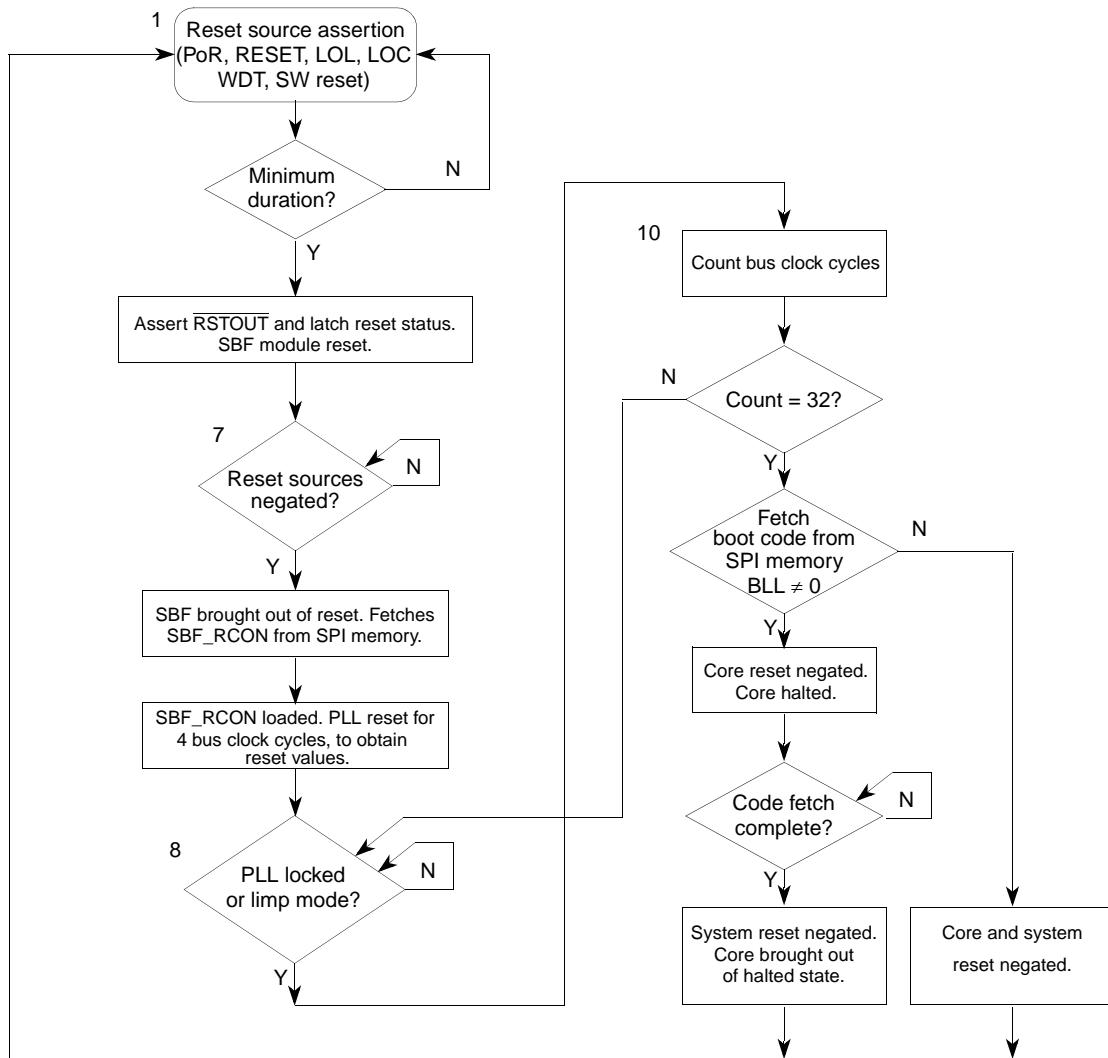
1. PLL lock stage — POR, RESET, PLL loss of clock
2. 512 FB\_CLK wait stage – POR, RESET, PLL loss of lock, PLL loss of clock.

#### **NOTE**

The watchdog timeout and software reset cannot interrupt this sequence, since the registers implementing their logic are in a reset state during the reset sequence and their associated resets aren't be asserted during this period.

### 12.4.2.2 SBF Reset Sequence

The SBF reset sequence proceeds differently and is shown in [Figure 12-4](#). The 512 bus clock cycle count from the non-SBF reset sequence is held at 32 for this sequence.



**Figure 12-5. SBF Reset Sequence**

The reset sequence for the SBF-enabled case recognizes the same reset sources as the non-SBF case. The initial part of the sequence is same, with the same constraints applicable as earlier, on recognition of a reset source.

With negation of the reset source, while the global resets are kept asserted, the SBF is brought out of reset. The SBF starts fetching the SBF\_RCON words from the external, SPI memory and when finished, signals to the CCM that SBF\_RCON is valid.

At this point, SBF\_RCON is loaded and the PLL is reset for a short duration. This is done to ensure that if not in limp mode, the PLL obtains the reset configuration values.

After the PLL locks or if in limp mode, a bus clock counter is started. After completing a count of 32 bus clock cycles, the reset to the core (CPU) is negated. If the PLL loses lock at any time during the count, the counter is reset and started only when the PLL regains lock. At this point, the boot load length (BLL) is checked to see if code fetch from the serial memory is required.

- If BLL is non-zero, the core is halted and the internal SRAM is loaded with code from the serial memory. When this boot-code loading completes, the reset to the rest of the system is negated and the core is brought out of its halted state.
- If BLL is zero, the resets to the core and the rest of the system are lifted at the same time.



# Chapter 13

## System Control Module (SCM)

### 13.1 Introduction

This system control module (SCM) provides a software core watchdog timer and generic access error information for the processor core.

#### 13.1.1 Overview

The SCM's core watchdog timer (CWT) provides a means of preventing system lockup due to uncontrolled software loops via a special software service sequence. If periodic software servicing action does not occur, the CWT times out with a programmed response (system reset or interrupt) to allow recovery or corrective action to be taken.

Fault access reporting is also available within the SCM. The user can use these registers during the resulting interrupt service routine and perform an appropriate recovery.

#### 13.1.2 Features

The SCM includes these distinctive features:

- System control registers
  - Core watchdog control register (CWCR) for watchdog timer control
  - Core watchdog service register (CWSR) to service watchdog timer
  - Watchdog can be clocked by the internal bus clock or the 32 kHz RTC clock
  - SCM interrupt status register (SCMISR) to service a bus fault or watchdog interrupt
  - Bus monitor timeout register (BMT)
- Core fault reporting registers

### 13.2 Memory Map/Register Definition

The memory map for the SCM registers is shown in [Table 13-1](#).

Attempted accesses to reserved addresses result in a bus error, while attempted writes to read-only registers are ignored and do not terminate with an error. Unless noted otherwise, writes to the programming model

must match the size of the register, e.g., an 8-bit register supports only 8-bit writes, etc. Attempted writes of a different size than the register width produce a bus error and no change to the targeted register.

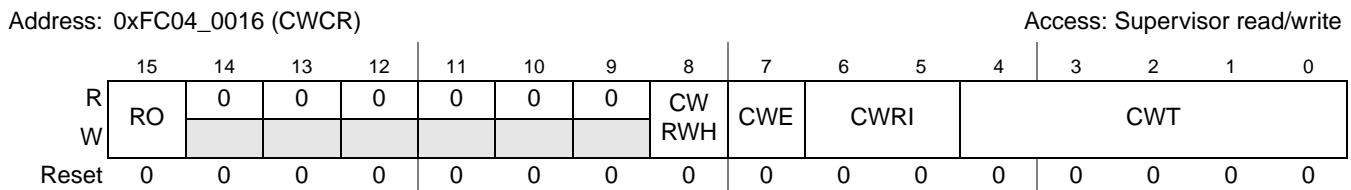
**Table 13-1. SCM Memory Map**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
<b>Supervisor Access Only Registers</b>					
0xFC04_0013	Wakeup control register (WCR) <sup>1</sup>	8	R/W	0x00	9.2.1/9-2
0xFC04_0016	Core watchdog control register (CWCR)	16	R/W	0x0000	13.2.1/13-2
0xFC04_001B	Core watchdog service register (CWSR)	8	R/W	Undefined	13.2.2/13-3
0xFC04_001F	SCM interrupt status register (SCMISR)	8	R/W	0x00	13.2.3/13-4
0xFC04_0024	Burst configuration register (BCR)	32	R/W	0x0000_0000	13.2.4/13-5
0xFC04_0070	Core fault address register (CFADR)	32	R	Undefined	13.2.5/13-5
0xFC04_0075	Core fault interrupt enable register (CFIER)	8	R/W	0x00	13.2.6/13-6
0xFC04_0076	Core fault location register (CFLOC)	8	R	Undefined	13.2.7/13-6
0xFC04_0077	Core fault attributes register (CFATR)	8	R	Undefined	13.2.8/13-7
0xFC04_007C	Core fault data register (CFDTR)	32	R	Undefined	13.2.9/13-7

<sup>1</sup> The WCR register is described in [Chapter 9, “Power Management.”](#)

### 13.2.1 Core Watchdog Control Register (CWCR)

The CWCR controls the software watchdog timer, time-out periods, and software watchdog timer interrupt. The register can be read or written at any time. At system reset, the software watchdog timer is disabled.

**Figure 13-1. Core Watchdog Control Register (CWCR)****Table 13-2. CWCR Field Descriptions**

Field	Description
15 RO	Read-only control bit. 0 CWCR can be read or written. 1 CWCR is read-only. A system reset is required to clear this register. The setting of this bit is intended to prevent accidental writes of the CWCR from changing the defined core watchdog configuration.
14–9	Reserved, must be cleared.

**Table 13-2. CWCR Field Descriptions (continued)**

Field	Description
8 CWRWH	Core watchdog run while halted. 0 Core watchdog timer stops counting if the core is halted. 1 Core watchdog timer continues to count even while the core is halted.
7 CWE	Core watchdog timer enable. 0 CWT is disabled. 1 CWT is enabled.
6–5 CWRI	Core watchdog reset/interrupt. 00 If a time-out occurs, the CWT generates an interrupt to the core. Refer to <a href="#">Chapter 17, “Interrupt Controller Modules,”</a> for details on setting its priority level. 01 The first time-out generates an interrupt to the processor, and if not serviced, a second time-out generates a system reset and sets the RSR[WDRCORE] flag in the reset controller. 10 If a time-out occurs, the CWT generates a system reset and RSR[WDRCORE] in the reset controller is set. 11 The CWT functions in a window mode of operation. For this mode, the servicing of the CWSR must occur during the last 25% of the time-out period. Any writes to the CWSR during the first 75% of the time-out period generate an immediate system reset. Likewise, if the CWSR is not serviced during the last 25% of the time-out period, a system reset is generated. For any type of reset response, the RSR[WDRCORE] flag is set.
4–0 CWT	Core watchdog time-out period. Selects the time-out period for the CWT. At reset, this field is cleared selecting the minimum time-out period, but the CWT is disabled because CWCR[CWE] is cleared at reset.  If CWCR[CWT] is $n$ , the time-out period equals $2^n$ system clock cycles. However, if $n$ is less than 8, the time-out period is forced to $2^8$ . 0x00 $2^8$ ... 0x08 $2^8$ 0x09 $2^9$ ... 0x1F $2^{31}$

### 13.2.2 Core Watchdog Service Register (CWSR)

The software watchdog service sequence must be performed using the CWSR as a data register to prevent a CWT time-out. The service sequence requires two writes to this data register: a write of 0x55 followed by a write of 0xAA. Both writes must be performed in this order prior to the CWT time-out, but any number of instructions can be executed between the two writes. If the CWT has already timed out, writing to this register has no effect in negating the CWT interrupt or reset. [Figure 13-2](#) illustrates the CWSR. At system reset, the contents of CWSR are uninitialized.

#### NOTE

If the CWT is enabled and has not timed out, then any write of a data value other than 0x55 or 0xAA causes an immediate system reset, regardless of the value in the CWCR[CWRI] field.

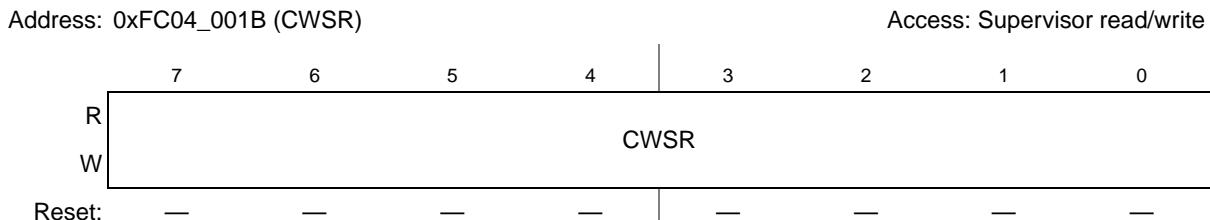


Figure 13-2. Core Watchdog Service Register (CWSR)

### 13.2.3 SCM Interrupt Status Register (SCMISR)

For certain values in the CWCR[CWRI] field, the CWT generates an interrupt response to a time-out. For these configurations, the SCMISR provides a program visible interrupt request from the watchdog timer. During the interrupt service routine which handles this interrupt, the source must be explicitly cleared by writing a 0x01 to the SCMISR.

The SCMISR also indicates system bus fault errors. An interrupt is sent only to the interrupt controller when the CFIER[ECFEI] bit is set. The SCMISR[CFEI] bit flags fault errors independent of the CFIER[ECFEI] setting. Therefore, if CFEI is set prior to setting ECFEI, an interrupt is requested immediately after ECFEI is set.

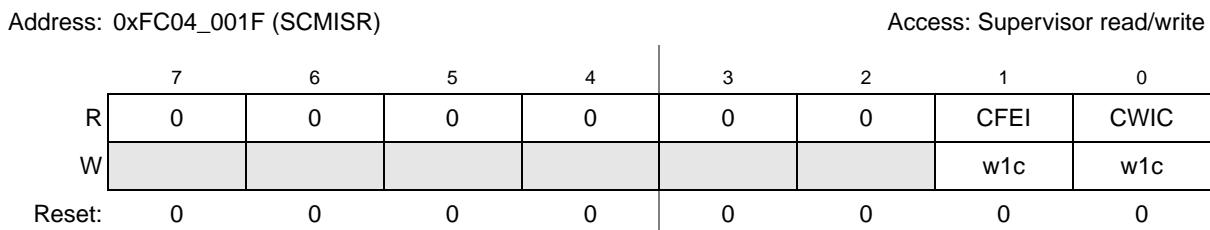


Figure 13-3. SCM Interrupt Status Register (SCMISR)

Table 13-3. SCMISR Field Descriptions

Field	Description
7–2	Reserved, must be cleared.
1 CFEI	Core fault error interrupt flag. Indicates if a bus fault has occurred. Writing a 1 clears this bit and negates the interrupt request. Writing a 0 has no effect. 0 No bus error. 1 A bus error has occurred. The faulting address, attributes (and possibly write data) are captured in the CFADR, CFATR, and CFDTR registers. The error interrupt is enabled only if CFLOC[ECFEI] is set. <b>Note:</b> This bit reports core faults regardless of the setting of CFIER[ECFEI]. Therefore, if the error interrupt is disabled and a core fault occurs, this bit is set. Then, if the error interrupt is subsequently enabled, an interrupt is immediately requested. To prevent an undesired interrupt, clear the captured error by writing one to CFEI before enabling the interrupt.
0 CWIC	Core watchdog interrupt flag. Indicates whether an CWT interrupt has occurred. Writing a 1 clears this bit and negates the interrupt request. Writing a 0 has no effect. 0 No CWT interrupt has occurred. 1 CWT interrupt has occurred.

### **13.2.4 Burst Configuration Register (BCR)**

The BCR register enables or disables the eSDHC, USB host, and USB On-the-Go modules for bursting to/from the crossbar switch slave modules. There is an enable field for the slaves, and either direction (read and write) is supported via the GBR and GBW bits.

**Figure 13-4. Burst Configuration Register (BCR)**

#### **Table 13-4. BCR Field Descriptions**

Field	Description
31–10	Reserved, must be cleared.
9 GBR	<p>Global burst enable for reads. Allows bursts to happen on read transactions from the crossbar switch slaves to the eSDHC, USB host, and USB On-the-Go modules.</p> <p>0 Read bursts are disabled 1 Read bursts are enabled</p> <p><b>Note:</b> If GBR and GBW are cleared, then SBE is ignored.</p>
8 GBW	<p>Global burst enable for writes. Allows bursts to happen on write transactions to the crossbar switch slaves from the eSDHC, USB host, and USB On-the-Go modules.</p> <p>0 Write bursts are disabled 1 Write bursts are enabled</p> <p><b>Note:</b> If GBR and GBW are cleared, then SBE is ignored.</p>
7–0 SBE	<p>Slave burst enable. Allows bursts to happen to/from the crossbar switch slaves. The only valid settings for this field are 0x00 or 0xFF.</p> <p>0x00 Bursts disabled 0xFF Bursts enabled. The GBR and GBW bits determine the burst direction. If neither is set, then this bit has no effect. Else Reserved</p>

### 13.2.5 Core Fault Address Register (CFADR)

The CFADR is a read-only register indicating the address of the last core access terminated with an error response.

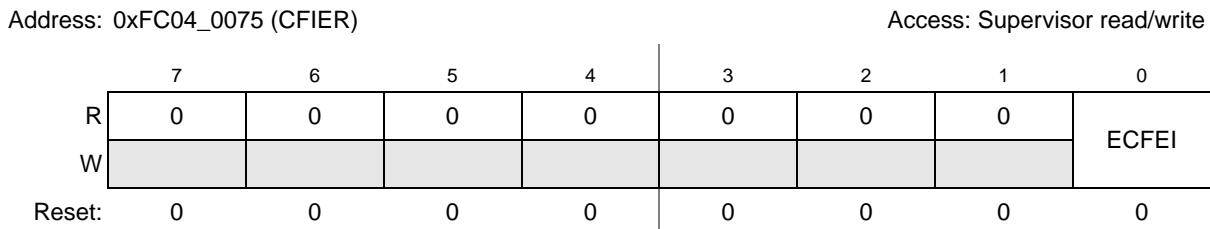
**Figure 13-5. Core Fault Address Register (CFADR)**

**Table 13-5. CFADR Field Descriptions**

Field	Description
31–0 ADDR	Indicates the faulting address of the last core access terminated with an error response.

### 13.2.6 Core Fault Interrupt Enable Register (CFIER)

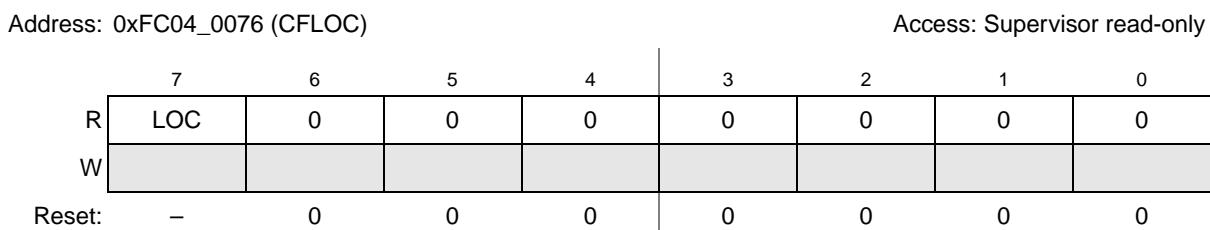
The CFIER register enables the system bus-error interrupt. See [Chapter 17, “Interrupt Controller Modules,”](#) for more information of the interrupt controller.

**Figure 13-6. Core Fault Interrupt Enable Register (CFIER)****Table 13-6. CFIER Field Descriptions**

Field	Description
7–1	Reserved, must be cleared.
0 ECFEI	Enable core fault error interrupt. 0 Do not generate an error interrupt on a faulted system bus cycle. 1 Generate an error interrupt to the interrupt controller on a faulted system bus cycle.

### 13.2.7 Core Fault Location Register (CFLOC)

The read-only CFLOC register indicates the location of the last captured fault.

**Figure 13-7. Core Fault Location Register (CFLOC)****Table 13-7. CFLOC Field Descriptions**

Field	Description
7 LOC	The location of the last captured fault. 0 Error occurred on a data reference 1 Error occurred on an instruction fetch
6–0	Reserved, must be cleared.

### 13.2.8 Core Fault Attributes Register (CFATR)

The read-only CFATR register captures the processor's attributes of the last faulted core access to the system bus.

**Figure 13-8. Core Fault Attributes Register (CFATR)**

**Table 13-8. CFATR Field Descriptions**

Field	Description
7 WRITE	Indicates the direction of the last faulted core access. 0 Core read access. 1 Core write access.
6–4 SIZE	Indicates the size of the last faulted core access. 000 8-bit core access. 001 16-bit core access. 010 32-bit core access. Else Reserved.
3 CACHE	Indicates if last faulted core access was cacheable. 0 Non-cacheable 1 Cacheable
2	Reserved, must be cleared.
1 MODE	Indicates the mode the device was in during the last faulted core access. 0 User mode 1 Supervisor mode
0 TYPE	Defines the type of last faulted core access. 0 Instruction 1 Data

### **13.2.9 Core Fault Data Register (CFDTR)**

The CFDTR is a read-only register for capturing the data associated with the last faulted processor write data access from the device's internal bus. The CFDTR is valid only for faulted internal bus-write accesses, CFLOC[LOC] is cleared.

**Figure 13-9. Core Fault Data Register (CFDTR)**

**Table 13-9. CFDTR Field Descriptions**

Field	Description
31–0 CFDTR	Contains data associated with the faulting access of the last internal bus write access. Contains the data value taken directly from the write data bus.

## 13.3 Functional Description

### 13.3.1 Core Watchdog Timer

The core watchdog timer (CWT) prevents system lockup if the software becomes trapped in a loop with no controlled exit or if a bus transaction becomes hung. The core watchdog timer can be enabled through CWCR[CWE]; it is disabled at reset. If enabled, the CWT requires the periodic execution of a core watchdog servicing sequence. If this periodic servicing action does not occur, the timer expires and, depending on the setting of CWCR[CWRI], different events may occur:

- An interrupt may be generated to the core.
- An immediate system reset.
- Upon the first time-out, a watchdog timer interrupt is asserted. If this time-out condition is not serviced before a second time-out occurs, the CWT asserts a system reset. This configuration supports a more graceful response to watchdog time-outs.
- In addition to these three basic modes of operation, the CWT also supports a windowed mode of operation. In this mode, the time-out period is divided into four equal segments and the entire service sequence of the CWT must occur during the last segment (last 25% of the time-out period). If the timer is serviced anytime (any write to the CWSR register) in the first 75% of the time-out period, an immediate system reset occurs.

To prevent the core watchdog timer from interrupting or resetting, the CWSR register must be serviced by performing the following sequence:

1. Write 0x55 to CWSR.
2. Write 0xAA to CWSR.

Both writes must occur in order before the time-out, but any number of instructions can execute between the two writes. This allows interrupts and exceptions to occur, if necessary, between the two writes.

#### NOTE

If the CWT is enabled and has not timed out, any write of a data value other than 0x55 or 0xAA causes an immediate system reset, regardless of the value in the CWCR[CWRI] field.

The timer value is constantly compared with the time-out period specified by CWCR[CWT], and any write to the CWCR register resets the watchdog timer. In addition, a write-once control bit in the CWCR sets the CWCR to read-only to prevent accidental updates to this control register from changing the desired system configuration. After this bit, CWCR[RO], is set, a system reset is required to clear it.

For certain values in the CWCR[CWRI] field, the CWT generates an interrupt response to a time-out. For these configurations, the SCMISR register provides a program visible interrupt request from the watchdog

timer. During the interrupt service routine which handles this interrupt, the source must be explicitly cleared by writing a 0x01 to the SCMISR.

### 13.3.1.1 Watchdog Timer Clock Source

The watchdog timer clock source is configurable via the write-once MISCCR2[SWTSCR] field in the CCM. You can choose the system bus clock or the 32 kHz RTC clock. Using the RTC clock allows the SWT to operate during battery standby mode.

See [Section 10.3.10, “Miscellaneous Control Register 2 \(MISCCR2\)”](#) for more information.

### 13.3.2 Core Data Fault Recovery Registers

To aid in recovery from certain types of access errors, the SCM module supports a number of registers that capture access address, attribute, and data information on bus cycles terminated with an error response. These registers can then be read during the resulting exception service routine and the appropriate recovery performed.

The details on the core fault recovery registers are provided in the above sections. It is important to note these registers are used to capture fault recovery information on any processor-initiated system bus cycle terminated with an error.



# Chapter 14

## Crossbar Switch (XBS)

### 14.1 Overview

This section provides information on the layout, configuration, and programming of the crossbar switch. The crossbar switch connects the bus masters and bus slaves using a crossbar switch structure. This structure allows bus masters to access different bus slaves simultaneously with no interference while providing arbitration among the bus masters when they access the same slave. A variety of bus arbitration methods and attributes may be programmed on a slave by slave basis.

The MCF5441x devices have up to eight masters and six slaves (8Mx6S) connected to the crossbar switch. The eight masters are the ColdFire core/serial boot facility, eDMA controller, USB host, USB OTG modules, eSDHC, NAND flash controller, and the Ethernet assembly. The slaves are DDR SDRAM controller, FlexBus, SRAM controller backdoor, and the two peripheral bus controllers.

Figure 14-1 is a block diagram of the MCF5441x family bus architecture showing the crossbar switch configuration.

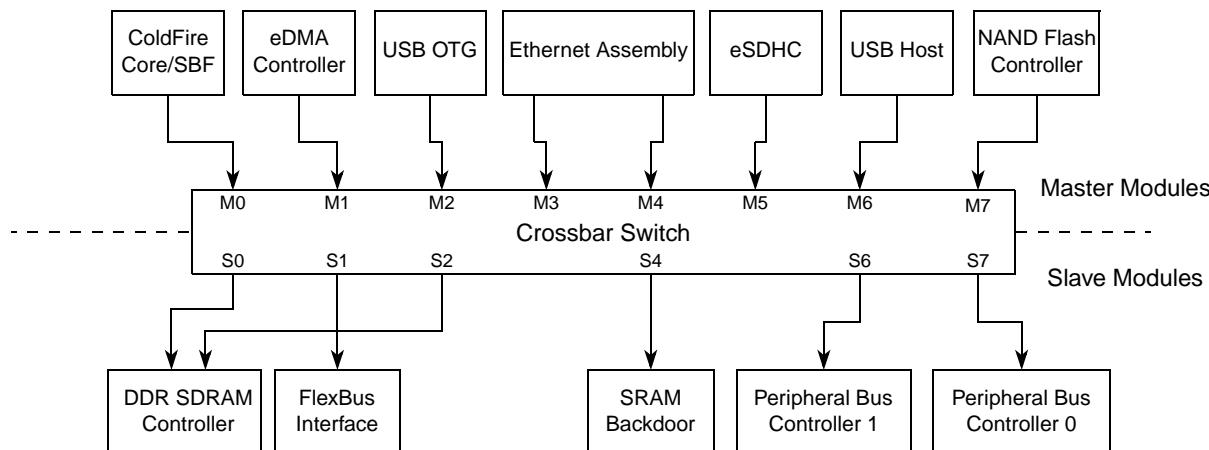


Figure 14-1. Bus Architecture Block Diagram

The modules are assigned to the numbered ports on the crossbar switch in the below table.

**Table 14-1. Crossbar Switch Master/Slave Assignments**

<b>Master Modules</b>	
<b>Crossbar Port</b>	<b>Module</b>
Master 0 (M0)	ColdFire core/Serial boot facility
Master 1 (M1)	eDMA controller
Master 2 (M2)	USB On-the-Go
Master 3 (M3)	Ethernet assembly
Master 4 (M4)	Ethernet assembly
Master 5 (M5)	eSDHC
Master 6 (M6)	USB host
Master 7 (M7)	NAND flash controller
<b>Slave Modules</b>	
<b>Crossbar Port</b>	<b>Module</b>
Slave 0 (S0)	DDR SDRAM Controller
Slave 1 (S1)	FlexBus
Slave 2 (S2)	DDR SDRAM Controller
Slave 4 (S4)	Internal SRAM Backdoor
Slave 6 (S6)	Peripheral Bus Controller 1 <sup>2</sup>
Slave 7 (S7)	Peripheral Bus Controller 0 <sup>2</sup>

<sup>2</sup> See the memory map section within [Chapter 1, “Overview”](#) for a list of which modules are connected to which peripheral bus controller.

The system memory map differs in the lower address range (0x0000\_0000–0x3FFF\_FFFF) depending on the boot method you choose:

- Booting from FlexBus — FlexBus accesses are valid for the entire range
- Booting from the NAND flash controller — maps the flash controller’s base (normally at 0xFC0F\_C000) to 0x0000\_0000. After booting, clear the NFC’s boot mode configuration flag to disable this remapping.
- Serial boot — maps the internal SRAM base to 0x0000\_0000 so it can serve as the boot device

**Table 14-2. System Memory Map per Boot Mode**

<b>Address Range<sup>1</sup></b>	<b>FlexBus</b>	<b>Flash Controller</b>	<b>Serial Boot</b>
0x0000_0000 0x0000_FFFF	FlexBus	NAND flash controller <sup>2</sup>	Internal SRAM <sup>2</sup>
0x0001_0000 0x00FF_FFFF			
0x1000_0000  0x3FFF_FFFF		FlexBus	FlexBus
0x4000_0000  0x7FFF_FFFF		SDRAM controller	
0x8000_0000 0x8FFF_0000		Internal SRAM backdoor	
0x9000_0000  0xBFFF_FFFF		Reserved	
0xC000_0000 0xDFFF_FFFF		FlexBus	
0xE000_0000 0xFFFF_FFFF		Peripheral bus controller 1 <sup>2</sup>	
0xF000_0000 0xFFFF_FFFF		Peripheral bus controller 0 <sup>2</sup>	

<sup>1</sup> See the various peripheral chapters for their memory maps. Any unused space by these peripherals within this memory range is reserved and must not be accessed.

<sup>2</sup> See the memory map section within [Chapter 1, “Overview”](#) for a list of which modules are connected to each peripheral bus controller.

**NOTE**

This memory map provides two disjoint regions mapped to the FlexBus controller to support glueless connections to external memories (e.g., flash and SRAM) and a second space with one (or more) unique chip-selects that can be used for non-cacheable, non-memory devices (addresses 0xC000\_0000–0xDFFF\_FFFF). Additionally, this mapping is easily maps into the ColdFire access control registers, which provide a coarse association between memory addresses and their attributes (e.g., cacheable, non-cacheable). For this device, one possible configuration defines the default memory attribute as non-cacheable, and one ACR then identifies cacheable addresses, e.g., ADDR[31] equals 0 identifies the cacheable space.

## 14.2 Features

The crossbar switch includes these distinctive features:

- Symmetric crossbar bus switch implementation
  - Allows concurrent accesses from different masters to different slaves
  - Slave arbitration attributes configured on a slave by slave basis
- 32 bits wide and supports byte, word (2 byte), longword (4 byte), and 16 byte burst transfers
- Operates at a 1-to-1 clock frequency with the bus masters

## 14.3 Modes of Operation

The crossbar switch supports two arbitration modes (fixed or round-robin), which may be set on a slave by slave basis. Slaves configured for fixed arbitration mode have a unique arbitration level assigned to each bus master.

In fixed priority mode, the highest priority active master accessing a particular slave is granted the master bus path to that slave. A higher priority master blocks access to a given slave from a lower priority master if the higher priority master continuously requests that slave. See [Section 14.5.1.1, “Fixed-Priority Operation.”](#)

In round-robin arbitration, active masters accessing a particular slave are initially granted the slave based on their master port number. Master priority is then modified in a wrap-around manner to give all masters fair access to the slave. See [Section 14.5.1.2, “Round-Robin Priority Operation.”](#)

## 14.4 Memory Map / Register Definition

Two registers reside in each slave port of the crossbar switch. Read- and write-transfers require two bus clock cycles. The registers can only be read from and written to in supervisor mode. Additionally, these registers can only be read from or written to by 32-bit accesses.

A bus error response is returned if an unimplemented location is accessed within the crossbar switch. See [Section 13.2.3, “SCM Interrupt Status Register \(SCMISR\).”](#)

The slave registers also feature a bit that, when set, prevents the registers from being written. The registers remain readable, but future write attempts have no effect on the registers and are terminated with a bus error response to the master initiating the write. The core, for example, takes a bus error interrupt.

**Table 14-3** shows the memory map for the crossbar switch program-visible registers.

**Table 14-3. XBS Memory Map**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC00_4000	Priority Register Slave 0 (XBS_PRS0)	32	R/W	0x7654_3210	<a href="#">14.4.1/14-5</a>
0xFC00_4010	Control Register Slave 0 (XBS_CRS0)	32	R/W	0x0000_0000	<a href="#">14.4.2/14-7</a>
0xFC00_4100	Priority Register Slave 1 (XBS_PRS1)	32	R/W	0x7654_3210	<a href="#">14.4.1/14-5</a>
0xFC00_4110	Control Register Slave 1 (XBS_CRS1)	32	R/W	0x0000_0000	<a href="#">14.4.2/14-7</a>
0xFC00_4200	Priority Register Slave 2 (XBS_PRS2)	32	R/W	0x7654_3210	<a href="#">14.4.1/14-5</a>
0xFC00_4210	Control Register Slave 2 (XBS_CRS2)	32	R/W	0x0000_0000	<a href="#">14.4.2/14-7</a>
0xFC00_4400	Priority Register Slave 4 (XBS_PRS4)	32	R/W	0x7654_3210	<a href="#">14.4.1/14-5</a>
0xFC00_4410	Control Register Slave 4 (XBS_CRS4)	32	R/W	0x0000_0000	<a href="#">14.4.2/14-7</a>
0xFC00_4600	Priority Register Slave 6 (XBS_PRS6)	32	R/W	0x7654_3210	<a href="#">14.4.1/14-5</a>
0xFC00_4610	Control Register Slave 6 (XBS_CRS6)	32	R/W	0x0000_0000	<a href="#">14.4.2/14-7</a>
0xFC00_4700	Priority Register Slave 7 (XBS_PRS7)	32	R/W	0x7654_3210	<a href="#">14.4.1/14-5</a>
0xFC00_4710	Control Register Slave 7 (XBS_CRS7)	32	R/W	0x0000_0000	<a href="#">14.4.2/14-7</a>

#### 14.4.1 XBS Priority Registers (XBS\_PRSn)

The priority registers (XBS\_PRSn) set the priority of each master port on a per slave port basis and reside in each slave port. The priority register can be accessed only with 32-bit accesses. After the XBS\_CRSn[RO] bit is set, the XBS\_PRSn register can only be read; attempts to write to it have no effect on XBS\_PRSn and result in a bus-error response to the master initiating the write.

Additionally, no two available master ports may be programmed with the same priority level, including reserved masters. Attempts to program two or more masters with the same priority level result in a bus-error response (see [Section 13.2.3, “SCM Interrupt Status Register \(SCMISR\)”](#)) and the XBS\_PRSn is not updated.

## Crossbar Switch (XBS)

Address: 0xFC00\_4000 (XBS\_PRS0)  
 0xFC00\_4100 (XBS\_PRS1)  
 0xFC00\_4200 (XBS\_PRS2)  
 0xFC00\_4400 (XBS\_PRS4)  
 0xFC00\_4600 (XBS\_PRS6)  
 0xFC00\_4700 (XBS\_PRS7)

Access: Supervisor read/write

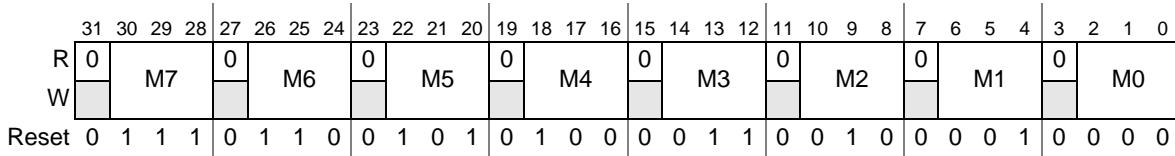


Figure 14-2. XBS Priority Registers Slave n (XBS\_PRSn)

Table 14-4. XBS\_PRSn Field Descriptions

Field	Description
31	Reserved, must be cleared.
30–28 M7	Master 7 (NAND flash controller) priority. Sets the arbitration priority for this port on the associated slave port. 000 This master has level 1 (highest) priority when accessing the slave port. 001 This master has level 2 priority when accessing the slave port. 010 This master has level 3 priority when accessing the slave port. 011 This master has level 4 priority when accessing the slave port. 100 This master has level 5 priority when accessing the slave port. 101 This master has level 6 priority when accessing the slave port. 110 This master has level 7 priority when accessing the slave port. 111 This master has level 8 (lowest) priority when accessing the slave port.
27	Reserved, must be cleared.
26–24 M6	Master 6 (USB Host) priority. See M7 description.
23	Reserved, must be cleared.
22–20 M5	Master 5 (eSDHC) priority. See M7 description.
19	Reserved, must be cleared.
18–16 M4	Master 4 (Ethernet assembly) priority. See M7 description.
15	Reserved, must be cleared.
14–12 M3	Master 3 (Ethernet assembly) priority. See M7 description.
11	Reserved, must be cleared.
10–8 M2	Master 2 (USB OTG) priority. See M7 description.
7	Reserved, must be cleared.
6–4 M1	Master 1 (eDMA) priority. See M7 description.

**Table 14-4. XBS\_PRS<sub>n</sub> Field Descriptions (continued)**

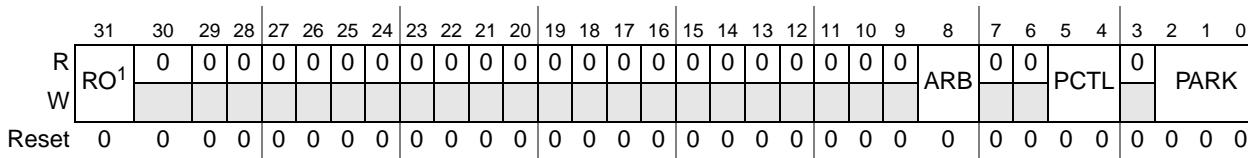
Field	Description
3	Reserved, must be cleared.
2–0 M0	Master 0 (ColdFire core/serial boot) priority. See M7 description.

## 14.4.2 XBS Control Registers (XBS\_CRS<sub>n</sub>)

The XBS control registers (XBS\_CRS<sub>n</sub>) control several features of each slave port and must be accessed using 32-bit accesses. After XBS\_CRS<sub>n</sub>[RO] is set, the XBS\_CRS<sub>n</sub> can only be read; attempts to write to it have no effect and result in an error response.

Address: 0xFC00\_4010 (XBS\_PRS0)  
 0xFC00\_4110 (XBS\_PRS1)  
 0xFC00\_4210 (XBS\_PRS2)  
 0xFC00\_4410 (XBS\_PRS4)  
 0xFC00\_4610 (XBS\_PRS6)  
 0xFC00\_4710 (XBS\_PRS7)

Access: Supervisor read/write



<sup>1</sup> After this bit is set, only a hardware reset clears it.

**Figure 14-3. XBS Control Registers Slave n (XBS\_CRS<sub>n</sub>)****Table 14-5. XBS\_CRS<sub>n</sub> Field Descriptions**

Field	Description
31 RO	Read only. Forces both of the slave port's registers (XBS_CRS <sub>n</sub> and XBS_PRS <sub>n</sub> ) to be read-only. After set, only a hardware reset clears it. 0 Both of the slave port's registers are writeable. 1 Both of the slave port's registers are read-only and cannot be written (attempted writes have no effect on the registers and result in a bus error response).
30–9	Reserved, must be cleared.
8 ARB	Arbitration Mode. Selects the arbitration policy for the slave port. 0 Fixed priority 1 Round robin (rotating) priority
7–6	Reserved, must be cleared.

**Table 14-5. XBS\_CRS<sub>n</sub> Field Descriptions (continued)**

Field	Description
5–4 PCTL	Parking control. Determines the slave port's parking control. The low-power park feature results in an overall power savings if the slave port is not saturated; however, this forces an extra latency clock when any master tries to access the slave port while not in use because it is not parked on any master. 00 When no master makes a request, the arbiter parks the slave port on the master port defined by the PARK bit field. 01 When no master makes a request, the arbiter parks the slave port on the last master to be in control of the slave port. 10 When no master makes a request, the slave port is not parked on a master and the arbiter drives all outputs to a constant safe state. 11 Reserved.
3	Reserved, must be cleared.
2–0 PARK	Park. Determines which master port the current slave port parks on when no masters are actively making requests and the PCTL bits are cleared. 000 Park on master port M0 (ColdFire Core) 001 Park on master port M1 (eDMA Controller) 010 Park on master port M2 (USB OTG) 011 Park on master port M3 (Ethernet assembly) 100 Park on master port M4 (Ethernet assembly) 101 Park on master port M5 (eSDHC) 110 Park on master port M6 (USB Host) 111 Park on master port M7 (NAND flash controller)

## 14.5 Functional Description

### 14.5.1 Arbitration

The crossbar switch supports two arbitration schemes: a simple fixed-priority comparison algorithm and a simple round-robin fairness algorithm. The arbitration scheme is independently programmable for each slave port.

#### 14.5.1.1 Fixed-Priority Operation

When operating in fixed-priority mode, each master is assigned a unique priority level in the XBS\_PRS<sub>n</sub> (priority registers). If two masters request access to a slave port, the master with the highest priority in the selected priority register gains control over the slave port.

When a master makes a request to a slave port, the slave port checks if the new requesting master's priority level is higher than that of the master that currently has control over the slave port (unless the slave port is in a parked state). The slave port does an arbitration check at every bus transfer boundary makes certain 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 new requesting master is granted control over the slave port at the next clock edge. The exception to this rule is if the master that currently has control over the slave port is running a fixed length burst transfer or a locked transfer. In this case, the new requesting master must wait until the end of the burst transfer or locked transfer before it is granted control of the slave port.

If the new requesting master's priority level is lower than the master that currently has control of the slave port, the new requesting master is forced to wait until the current master runs one of the following cycles:

- An IDLE cycle
- A non-IDLE cycle to a location other than the current slave port.

#### **14.5.1.2 Round-Robin Priority Operation**

When operating in round-robin mode, each master is assigned a relative priority based on the master port number. This priority is based on how far ahead the master port number of the requesting master is to the master port number of the current bus master for this slave. Master port numbers are compared modulo the total number of bus masters, i.e. take the requesting master port number minus the current bus master's port number modulo the total number of bus masters. The master port with the highest priority based on this comparison is granted control over the slave port at the next bus transfer boundary.

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 at the next transfer boundary.

Parking may continue to be used in a round-robin mode, but does not affect the round-robin pointer unless the parked master actually performs a transfer. Handoff occurs to the next master in line after one cycle of arbitration. If the slave port is put into low-power park mode, the round-robin pointer is reset to point at master port 0, giving it the highest priority.

#### **14.5.1.3 Priority Assignment**

Each master port needs to be assigned a unique 3-bit priority level. If an attempt is made to program multiple master ports with the same priority level within the priority registers (XBS\_PRSn) the crossbar switch responds with a bus error (refer to [Section 13.2.3, “SCM Interrupt Status Register \(SCMISR\)”](#)) and the registers are not updated.

### **14.6 Initialization/Application Information**

No initialization is required by or for the crossbar switch. Hardware reset ensures all the register bits used by the crossbar switch are properly initialized to a valid state. Settings and priorities should be programmed to achieve maximum system performance.



# **Chapter 15**

## **Pin-Multiplexing and Control**

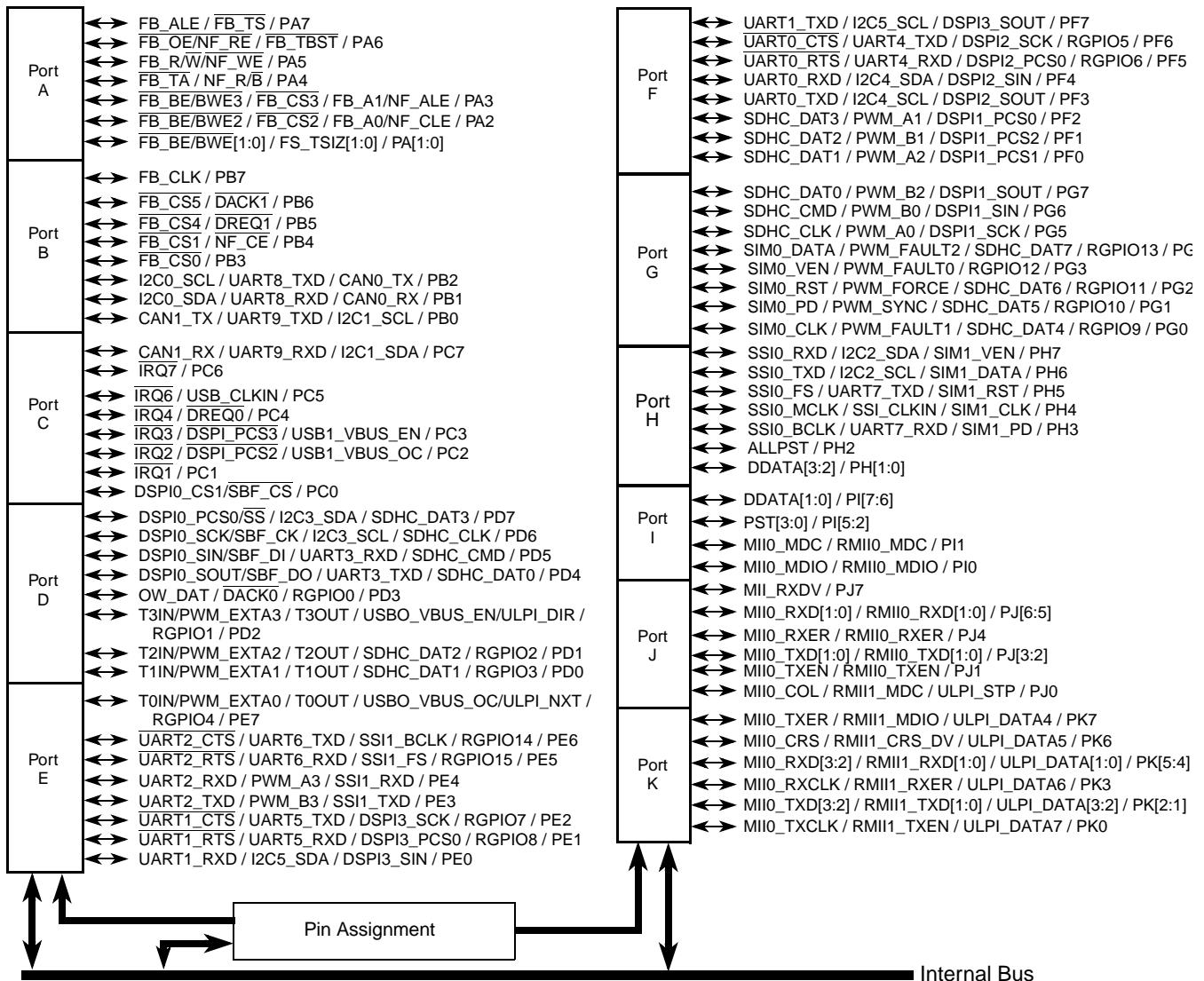
### **15.1 Introduction**

Many of the pins associated with the device may be used for several different functions. Their primary functions are to provide external interfaces to access off-chip resources. When not used for their primary function, many of the pins may be used as general-purpose digital I/O (GPIO) pins. In some cases, the pin function is set by the operating mode, and the alternate pin functions are not supported.

Each GPIO port has registers that configure, monitor, and control the port pins. [Figure 15-1](#) is a block diagram of the device ports.

This chapter also includes registers for controlling the drive strengths, hysteresis, and slew rates of the external pins.

## Pin-Multiplexing and Control



**Figure 15-1. Pin Multiplexing and Control Block Diagram**

### 15.1.1 Overview

The external pin-muxing and control module configures various external pins, including those used for:

- External bus accesses
- External device selection
- eSDHC
- SIM ports
- Ethernet data and control
- CAN
- I<sup>2</sup>C
- DSPI

- UART
- Edge ports
- 32-bit DMA timers

### 15.1.2 Features

The module includes these distinctive features:

- Control of primary function use
  - On all supported GPIO ports
- General purpose I/O support for all ports
  - Registers for storing output pin data
  - Registers for controlling pin data direction
  - Registers for reading current pin state
  - Registers for setting and clearing output pin data registers
- Control of functional pad drive strengths
- Slew rate control
- Hysteresis control
- Pull-up/down control

## 15.2 External Signal Description

The external pins that are controllable by this module are listed in the below table under the GPIO column.

### NOTE

In this table and throughout this document a single signal within a group is designated without square brackets (i.e., FB\_AD23), while designations for multiple signals within a group use brackets (i.e., FB\_AD[23:21]) and is meant to include all signals within the two bracketed numbers when these numbers are separated by a colon.

### NOTE

The primary functionality of a pin is not necessarily its default functionality. Most pins that are muxed with GPIO default to their GPIO functionality. See [Table 15-1](#) for a list of the exceptions.

**Table 15-1. Special-case default signal functionality**

Pin	Default signal
FB_CLK, FB_OE, FB_R/W, FB_BE/BWE[1:0], FB_CS[5:4]	FB_CLK, FB_OE, FB_R/W, FB_BE/BWE[1:0], FB_CS[5:4]
FB_ALE	FB_ALE or FB_TS (depending on RCON[3])
FB_BE/BWE3	Boot from NFC, NF_ALE. Otherwise, FB_BE/BWE3.
FB_BE/BWE2	Boot from NFC, NF_CLE. Otherwise, FB_BE/BWE2.
FB_CS1	Boot from NFC, NFC_CE. Otherwise, GPIO.
FB_CS0	Boot from FlexBus, FB_CS0. Otherwise, GPIO.
FB_TA	Boot from NFC, NFC_R/B. Otherwise, FB_TA.
ALLPST, PST[3:0], DDATA[3:0]	ALLPST, PST[3:0], DDATA[3:0]

**Table 15-2. MCF5441x Signal information and muxing**

Signal name	GPIO	Alternate 1	Alternate 2	Pullup (U) <sup>1</sup> Pulldown (D)	Direction <sup>2</sup>	Voltage domain	Pad type <sup>3</sup>	196 MAPBGA	256 MAPBGA
<b>Reset</b>									
RESET	—	—	—	U	I	EVDD	ssr	K14	K15
RSTOUT	—	—	—	—	O	EVDD	msr	L13	L16
<b>Clock</b>									
EXTAL/ RMII_REF_CLK	—	—	—	—	I <sup>5</sup>	EVDD	ae	G14	G16
XTAL	—	—	—	—	O	EVDD	ae	H14	H16
<b>Mode selection</b>									
BOOTMOD[1:0]	—	—	—	—	I	EVDD	msr	G5,H5	K5, L5
<b>FlexBus</b>									
FB_AD[31:24]/ NFC_IO[15:8] <sup>6</sup>	—	—	—	—	I/O	FBVDD	fsr	A10, A9, B9, C9, A8, B8, C8, A7	B9, C8, A9, B8, D8, A8, D7, B7

**Table 15-2. MCF5441x Signal information and muxing (continued)**

Signal name	GPIO	Alternate 1	Alternate 2	Pullup (U) <sup>1</sup> Pulldown (D)	Direction <sup>2</sup>	Voltage domain	Pad type <sup>3</sup>	196 MAPBGA <sup>4</sup>	256 MAPBGA
FB_AD[23:16]/ NFC_IO[7:0] <sup>6</sup>	—	—	—	—	I/O	FBVDD	fsr	B7, C7, C6, B6, A6, A5, B5, A4	C7, A7, D6, A6, B6, D5, C6, A5
FB_AD[15:10]	—	—	—	— <sup>7</sup>	I/O	FBVDD	fsr	C5, A3, B4, C4, B3, A2	B5, A4, A3, D4, B4, C5
FB_AD[9:8]	—	—	—	U <sup>8</sup>	I/O	FBVDD	fsr	B2, C3	C4, B3
FB_AD[7:0]	—	—	—	—	I/O	FBVDD	fsr	D4, B1, C2, D3, C1, D2, E3, D1	C3, E4, D3, E3, A2, B2, C2, F3
FB_ALE	PA7	FB_TS	—	—	O	FBVDD	fsr	E2	D2
FB_OE/ NFC_RE	PA6	FB_TBST/ NFC_RE	—	—	O	FBVDD	fsr	H1	F1
FB_R/W/ NFC_WE	PA5	—	—	—	O	FBVDD	fsr	H2	G2
FB_TA	PA4	—	NFC_R/B	U <sup>9</sup>	O	FBVDD	fsr	H3	H3
FB_BE/BWE3	PA3	FB_CS3	FB_A1/ NFC_ALE <sup>10</sup>	—	O	FBVDD	fsr	F3	C1
FB_BE/BWE2	PA2	FB_CS2	FB_A0/ NFC_CLE <sup>11</sup>	—	O	FBVDD	fsr	E1	E2
FB_BE/BWE[1:0]	PA[1:0]	FB_TSI[1:0]	—	—	O	FBVDD	fsr	F2, F1	D1, F4
FB_CLK	PB7	—	—	—	O	FBVDD	fsr	G1	G1
FB_CS5	PB6	DACK1	—	—	O	FBVDD	fsr	—	F2
FB_CS4	PB5	DREQ1	—	—	O	FBVDD	fsr	—	B1
FB_CS1	PB4	—	NFC_CE	—	O	FBVDD	fsr	G3	E1
FB_CS0	PB3	—	—	—	O	FBVDD	fsr	G2	G3
<b>I<sup>2</sup>C 0</b>									
I2C0_SCL	PB2	UART8_TXD	CAN0_TX	—	I/O	EVDD	ssr	H12	G15
I2C0_SDA	PB1	UART8_RXD	CAN0_RX	—	I/O	EVDD	ssr	G12	G14
<b>FlexCAN 1</b>									
CAN1_TX	PB0	UART9_TXD	I2C1_SCL	—	I/O	EVDD	ssr	—	D14
CAN1_RX	PC7	UART9_RXD	I2C1_SDA	—	I/O	EVDD	ssr	—	D15
<b>SDRAM controller</b>									
SD_A14	—	—	—	—	O	SDVDD	st_dec_ap	—	P6

**Table 15-2. MCF5441x Signal information and muxing (continued)**

Signal name	GPIO	Alternate 1	Alternate 2	Pullup (U) <sup>1</sup> Pulldown (D)	Direction <sup>2</sup>	Voltage domain	Pad type <sup>3</sup>	196 MAPBGA <sup>4</sup>	256 MAPBGA
SD_A[13:0]	—	—	—	—	O	SDVDD	st_dec_ap	P3, M1, M3, L2, L1, N4, M2, P2, L3, L4, N1, N2, K1, N3, A2, A1, A0	R4, R1, R3, N4, P3, T4, R2, T2, N3, P5, P4, N5, P2, T3
SD_BA[2:0]	—	—	—	—	O	SDVDD	st_dec_ap	M6, L5, P4	P7, N6, R5
<u>SD_CAS</u>	—	—	—	—	O	SDVDD	st_dec_ap	L6	N8
SD_CKE	—	—	—	—	O	SDVDD	st_dec_ap	N6	R7
SD_CLK	—	—	—	—	O	SDVDD	st_ck	P6	T5
<u>SD_CLK</u>	—	—	—	—	O	SDVDD	st_ck	P7	T6
<u>SD_CS</u>	—	—	—	—	O	SDVDD	st_dec_ap	M5	N7
SD_D[7:0]	—	—	—	—	I/O	SDVDD	st_odi	P11, M10, N10, M9, P10, M8, N8, M7	T12, R11, T11, R10, N9, T10, P9, R9
SD_DM	—	—	—	—	O	SDVDD	st_odi	P7	T7
SD_DQS	—	—	—	—	I/O	SDVDD	st_dqs	P8	T8
<u>SD_DQS</u>	—	—	—	—	I/O	SDVDD	st_dqs	P9	T9
SD_ODT	—	—	—	—	O	SDVDD	st_dec_ap	P5	P8
<u>SD_RAS</u>	—	—	—	—	O	SDVDD	st_dec_ap	M4	R6
<u>SD_WE</u>	—	—	—	—	O	SDVDD	st_dec_ap	N5	R8
SD_VREF	—	—	—	—	—	SDVDD	st_vref	N9	P10
SD_VTT	—	—	—	—	—	SDVDD	st_vtt	L7	N10
<b>External interrupts port</b>									
<u>IRQ7</u>	PC6	—	—	—	I	EVDD	ssr	F12	F12
<u>IRQ6</u>	PC5	—	USB_CLKIN <sup>12</sup>	—	I	EVDD	ssr	—	N1
<u>IRQ4</u>	PC4	<u>DREQ0</u>	—	—	I	EVDD	ssr	E11	F14
<u>IRQ3</u>	PC3	DSPI0_PCS3	USBH_VBUS_EN	—	I	EVDD	ssr	—	M1
<u>IRQ2</u>	PC2	DSPI0_PCS2	USBH_VBUS_OC	— <sup>13</sup>	I	EVDD	ssr	—	M2
<u>IRQ1</u>	PC1	—	—	—	I	EVDD	ssr	E13	F13

**Table 15-2. MCF5441x Signal information and muxing (continued)**

Signal name	GPIO	Alternate 1	Alternate 2	Pullup (U) <sup>1</sup> Pulldown (D) <sup>2</sup>	Direction <sup>2</sup>	Voltage domain	Pad type <sup>3</sup>	196 MAPBGA <sup>4</sup>	256 MAPBGA
<b>USB On-the-Go</b>									
USBO_DM	—	—	—	—	I/O	VDD_USB0	ae	B13	A14
USBO_DP	—	—	—	—	I/O	VDD_USB0	ae	A13	B14
<b>USB host</b>									
USBH_DM	—	—	—	—	I/O	VDD_USBH	ae	—	A15
USBH_DP	—	—	—	—	I/O	VDD_USBH	ae	—	B15
<b>ADC</b>									
ADC_IN7/ DAC1_OUT	—	—	—	—	I	VDDA_DAC_ADC	ae	—	K3
ADC_IN[6:4]	—	—	—	—	I	VDDA_ADC	ae	—	H2, J3, G4
ADC_IN3/ DAC0_OUT	—	—	—	—	I	VDDA_DAC_ADC	ae	—	K4
ADC_IN[2:0]	—	—	—	—	I	VDDA_ADC	ae	—	J2, J1, H1
<b>Real time clock</b>									
RTC_EXTAL	—	—	—	—	I <sup>5</sup>	VSTBY	ae	B14	B16
RTC_XTAL	—	—	—	—	O	VSTBY	ae	C14	C16
<b>DSPI/SBF<sup>14</sup></b>									
DSPI0_PCS1/ SBF_CS	PC0	—	—	—	I/O	EVDD	msr	K3	L1
DSPI0_PCS0/SS	PD7	I2C3_SDA	SDHC_DAT3	—	I/O	EVDD	msr	J1	K2
DSPI0_SCK/ SBF_CK	PD6	I2C3_SCL	SDHC_CLK	—	I/O	EVDD	msr	J3	L2
DSPI0_SIN/ SBF_DI	PD5	UART3_RXD	SDHC_CMD	U <sup>15</sup>	I	EVDD	msr	K2	L3
DSPI0_SOUT/ SBF_DO	PD4	UART3_TXD	SDHC_DAT0	—	O	EVDD	msr	J2	K1
<b>One wire</b>									
OW_DAT	GPIO0/PD3	DACK0	—	—	I/O	EVDD	ssr	M11	N11

**Table 15-2. MCF5441x Signal information and muxing (continued)**

Signal name	GPIO	Alternate 1	Alternate 2	Pullup (U) <sup>1</sup> Pulldown (D)	Direction <sup>2</sup>	Voltage domain	Pad type <sup>3</sup>	196 MAPBGA <sup>4</sup>	256 MAPBGA
<b>DMA timers</b>									
T3IN/PWM_EXTA3	GPIO1/PD2	T3OUT	USBO_VBUS_EN/ ULPI_DIR <sup>16</sup>	—	I	EVDD	msr	G13	G13
T2IN/PWM_EXTA2	GPIO2/PD1	T2OUT	SDHC_DAT2	—	I	EVDD	msr	J12	H14
T1IN/PWM_EXTA1	GPIO3/PD0	T1OUT	SDHC_DAT1	—	I	EVDD	msr	H13	H13
T0IN/PWM_EXTA0	GPIO4/PE7	T0OUT	USBO_VBUS_OC/ ULPI_NXT <sup>17</sup>	— <sup>18</sup>	I	EVDD	msr	J13	H15
<b>UART 2</b>									
UART2_CTS	GPIO14/PE6	UART6_TXD	SSI1_BCLK	—	I	EVDD	msr	—	M4
UART2_RTS	GPIO15/PE5	UART6_RXD	SSI1_FS	—	O	EVDD	msr	—	M3
UART2_RXD	PE4	PWM_A3	SSI1_RXD	—	I	EVDD	msr	—	P1
UART2_TXD	PE3	PWM_B3	SSI1_TXD	—	I/O <sup>19</sup>	EVDD	msr	—	N2
<b>UART 1</b>									
UART1_CTS	GPIO7/PE2	UART5_TXD	DSPI3_SCK	—	I	EVDD	msr	D11	C10
UART1_RTS	GPIO8/PE1	UART5_RXD	DSPI3_PCS0	—	O	EVDD	msr	C12	D10
UART1_RXD	PE0	I2C5_SDA	DSPI3_SIN	—	I	EVDD	msr	B10	C9
UART1_TXD	PF7	I2C5_SCL	DSPI3_SOUT	—	I/O <sup>19</sup>	EVDD	msr	C10	D9
<b>UART 0</b>									
UART0_CTS	GPIO5/PF6	UART4_TXD	DSPI2_SCK	—	I	EVDD	msr	E12	E13
UART0_RTS	GPIO6/PF5	UART4_RXD	DSPI2_PCS0	—	O	EVDD	msr	D12	B11
UART0_RXD	PF4	I2C4_SDA	DSPI2_SIN	—	I	EVDD	msr	C11	B10
UART0_TXD	PF3	I2C4_SCL	DSPI2_SOUT	—	I/O <sup>19</sup>	EVDD	msr	B11	D11
<b>Enhanced secure digital host controller</b>									
SDHC_DAT3	PF2	PWM_A1	DSPI1_PCS0	—	I/O	EVDD	msr	—	B13
SDHC_DAT2	PF1	PWM_B1	DSPI1_PCS2	—	I/O	EVDD	msr	—	E14
SDHC_DAT1	PF0	PWM_A2	DSPI1_PCS1	—	I/O	EVDD	msr	—	D12
SDHC_DAT0	PG7	PWM_B2	DSPI1_SOUT	—	I/O	EVDD	msr	—	B12
SDHC_CMD	PG6	PWM_B0	DSPI1_SIN	—	I/O	EVDD	msr	—	C11
SDHC_CLK	PG5	PWM_A0	DSPI1_SCK	—	O	EVDD	msr	—	A10

**Table 15-2. MCF5441x Signal information and muxing (continued)**

Signal name	GPIO	Alternate 1	Alternate 2	Pullup (U) <sup>1</sup> Pulldown (D)	Direction <sup>2</sup>	Voltage domain	Pad type <sup>3</sup>	196 MAPBGA <sup>4</sup>	256 MAPBGA
<b>Smart card interface 0</b>									
SIM0_DATA	GPIO13/PG4	PWM_FAULT2	SDHC_DAT7	—	I/O	EVDD	msr	—	E12
SIM0_VEN	GPIO12/PG3	PWM_FAULT0	—	—	O	EVDD	msr	—	D13
SIM0_RST	GPIO11/PG2	PWM_FORCE	SDHC_DAT6	—	O	EVDD	msr	—	C15
SIM0_PD	GPIO10/PG1	PWM_SYNC	SDHC_DAT5	—	I	EVDD	msr	—	C14
SIM0_CLK	GPIO9/PG0	PWM_FAULT1	SDHC_DAT4	—	O	EVDD	msr	—	A11
<b>Synchronous serial interface 0</b>									
SSI0_RXD	PH7	I2C2_SDA	SIM1_VEN	—	I	EVDD	msr	B12	C12
SSI0_TXD	PH6	I2C2_SCL	SIM1_DATA	—	O	EVDD	msr	A11	C13
SSI0_FS	PH5	UART7_TXD	SIM1_RST	—	I/O	EVDD	msr	C13	E15
SSI0_MCLK	PH4	SSI_CLKIN	SIM1_CLK	—	O	EVDD	msr	A12	A12
SSI0_BCLK	PH3	UART7_RXD	SIM1_PD	—	I/O	EVDD	msr	D13	A13
<b>Ethernet subsystem</b>									
MII0_MDC	PI1	RMII0_MDC <sup>20</sup>	—	—	O	EVDD	fsr	N14	P16
MII0_MDIO	PI0	RMII0_MDIO <sup>20</sup>	—	—	I/O	EVDD	fsr	M14	N16
MII0_RXDV	PJ7	RMII0_CRS_DV <sup>20</sup>	—	—	I	EVDD	fsr	M13	P14
MII0_RXD[1:0]	PJ[6:5]	RMII0_RXD[1:0] <sup>20</sup>	—	—	I	EVDD	fsr	P13, N13	R15, T15
MII0_RXER	PJ4	RMII0_RXER <sup>20</sup>	—	—	I	EVDD	fsr	M12	N14
MII0_TXD[1:0]	PJ[3:2]	RMII0_TXD[1:0] <sup>20</sup>	—	—	O	EVDD	fsr	P12, N12	R13, P13
MII0_TXEN	PJ1	RMII0_TXEN <sup>20</sup>	—	D <sup>21</sup>	O	EVDD	fsr	N11	P12
MII0_COL	PJ0	RMII1_MDC	ULPI_STP	—	I	EVDD	fsr	—	R12
MII0_TXER	PK7	RMII1_MDIO	ULPI_DATA4	—	O	EVDD	fsr	—	R14
MII0_CRS	PK6	RMII1_CRS_DV	ULPI_DATA5	—	I	EVDD	fsr	—	P11
MII0_RXD[3:2]	PK[5:4]	RMII1_RXD[1:0]	ULPI_DATA[1:0]	—	I	EVDD	fsr	—	P15, N13
MII0_RXCLK	PK3	RMII1_RXER	ULPI_DATA6	—	I	EVDD	fsr	—	M14
MII0_TXD[3:2]	PK[2:1]	RMII1_TXD[1:0]	ULPI_DATA[3:2]	—	O	EVDD	fsr	—	T13, N12
MII0_TXCLK	PK0	RMII1_TXEN	ULPI_DATA7	D <sup>21</sup>	I	EVDD	fsr	—	T14
<b>BDM/JTAG</b>									
ALLPST <sup>22</sup>	PH2	—	—	—	O	EVDD	fsr	K12	—

**Table 15-2. MCF5441x Signal information and muxing (continued)**

Signal name	GPIO	Alternate 1	Alternate 2	Pullup (U) <sup>1</sup> Pulldown (D)	Direction <sup>2</sup>	Voltage domain	Pad type <sup>3</sup>	196 MAPBGA <sup>4</sup>	256 MAPBGA
DDATA[3:2]	PH[1:0]	—	—	—	O	EVDD	fsr	—	L15, M13
DDATA[1:0]	PI[7:6]	—	—	—	O	EVDD	fsr	—	M15, L14
PST[3:0]	PI[5:2]	—	—	—	O	EVDD	fsr	—	J13, J16, J15, J14
JTAG_EN	—	—	—	D	I	EVDD	msr	L9	N15
PSTCLK	—	TCLK <sup>23</sup>	—	—	I	EVDD	fsr	L14	M16
DSI	—	TDI <sup>23</sup>	—	U	I	EVDD	msr	L11	L13
DSO	—	TDO <sup>23</sup>	—	—	O	EVDD	msr	L12	K14
<u>BKPT</u>	—	TMS <sup>23</sup>	—	U	I	EVDD	msr	K13	K16
DSCLK	—	<u>TRST</u> <sup>23</sup>	—	U	I	EVDD	msr	L10	K13
<b>Test</b> (this signal must be grounded)									
TEST	—	—	—	D	I	EVDD	ssr	L8	R16
<b>Power supplies</b>									
IVDD	—	—	—	—	—	—	—	D9, D10, E9, E10, F9, F10, G10	E9-E11, F9-F11
EVDD	—	—	—	—	—	—	—	F4-F7, G6, G7, H6, H7, J5, J6	H8, J7-J10, K6-K11, L6
FB_VDD	—	—	—	—	—	—	—	D5-D7, E4-E7	E5-E7, F5, F6, G5
SD_VDD	—	—	—	—	—	—	—	J8-J10, K7-K10	M7-M12
VDD_OSC_A_PLL	—	—	—	—	—	—	vddint	F13	F15
VSS_OSC_A_PLL	—	—	—	—	—	—	vddint	F14	F16
VDD_USB0	—	—	—	—	—	—	vdde	F11	G12
VDD_USBH	—	—	—	—	—	—	vdde	—	H12
VDDA_ADC	—	—	—	—	—	—	—	—	H4
VSSA_ADC	—	—	—	—	—	—	vssint	—	H5
VDDA_DAC_ADC	—	—	—	—	—	—	vddint	—	J4
VSSA_DAC_ADC	—	—	—	—	—	—	vssint	—	J5

**Table 15-2. MCF5441x Signal information and muxing (continued)**

Signal name	GPIO	Alternate 1	Alternate 2	Pullup (U) <sup>1</sup> Pulldown (D)	Direction <sup>2</sup>	Voltage domain	Pad type <sup>3</sup>	196 MAPBGA <sup>4</sup>	256 MAPBGA
VSTBY <sup>24</sup>	—	—	—	—	—	—	vddint	E14	E16
VSS	—	—	—	—	—	—	—	A1, A14, D8, D14, E8, F8, G4, G8, G9, G11, H4, H8-11, J4, J7, J11, J14, K4-K6, K11, P1, P14	A1, A16, D16, E8, F7, F8, G6-G11, H6, H7, H9-H11, J6, J11, J12, K12, L4, L7-L12, M5, M6, T1, T16

<sup>1</sup> All pins available with GPIO contain a configurable pull-up/down. This column indicates the pull devices that are enabled automatically at reset. Pull-ups are generally only enabled on pins with their primary function, except as noted.

<sup>2</sup> Refers to pin's primary function.

<sup>3</sup> For details on the available slew rates of the various pad types see section "Output Pad Loading and Slew Rate" of the *MCF5441x Data Sheet* or section "Slew Rate Control Registers (SRCR\_x)" in chapter "Pin-Multiplexing and Control" of the *MCF5441x Reference Manual*.

<sup>4</sup> This is tentative information — as of September 21, 2010 the 196 MAPBGA ball map has not yet been finalized.

<sup>5</sup> Enabled as input only in oscillator bypass mode (internal crystal oscillator is disabled).

<sup>6</sup> These pins are time-division multiplexed between the FlexBus and NFC. An arbitration mechanism determines which module drives these pins at any point in time.

<sup>7</sup> An internal pulldown circuit is enabled during system reset for FB\_AD[10].

<sup>8</sup> An internal pullup circuit is enabled when the system is in reset state.

<sup>9</sup> Configurable pull that is enabled and pulled up after reset.

<sup>10</sup> When configured for FB\_A1, this pin is time-division multiplexed between the FlexBus and NFC. An arbitration mechanism determines which module drives the pin at any point in time. When not configured as FB\_A1, NFC\_ALE cannot be used.

<sup>11</sup> When configured for FB\_A0, this pin is time-division multiplexed between the FlexBus and NFC. An arbitration mechanism determines which module drives the pin at any point in time. When not configured as FB\_A0, NFC\_CLE cannot be used.

<sup>12</sup> Since USB\_CLKIN is a clock signal, it must be dedicated to the USB system. Do not implement this pin as dual-use.

<sup>13</sup> When Alternate 2 is selected, then internal pullup/pulldown control will come from the MISCCR[3] register of CIM.

<sup>14</sup> When booting from serial boot flash, the SBF function is enabled automatically. After the SBF function completes its reset sequence, the signals are returned to GPIO functionality.

<sup>15</sup> Automatic pull-up when SBF controls the pin during reset only. Configurable pull when UART, DSPI, or SDHC control the pin.

<sup>16</sup> If ULPI is enabled, ULPI\_DIR is available as the Alternate 2 function. If ULPI is disabled, USBO\_VBUS\_EN is available.

<sup>17</sup> If ULPI is enabled, ULPI\_NXT is available as the Alternate 2 function. If ULPI is disabled, USBO\_VBUS\_OC is available.

<sup>18</sup> When Alternate 2 is selected, then internal pullup/pulldown control will come from the MISCCR[2] register of CIM.

<sup>19</sup> UART<sub>x</sub>\_TXD pad can act as RXD(input) pad when UART One Wire mode is enabled.

<sup>20</sup> These RMII functions are selected by the mode chosen by the MAC-NET, not by the pin-multiplexing and control (GPIO) module.

<sup>21</sup> Configurable pull that is enabled and pulled down after reset.

<sup>22</sup> The ALLPST signal is available only on the 196 MAPBGA package and allows limited debug trace functionality compared to the 256 MAPBGA package.

<sup>23</sup> If JTAG\_EN is asserted, these pins default to Alternate 1 (JTAG) functionality. The GPIO module is not responsible for assigning these pins.

<sup>24</sup> VSTBY is for optional standby lithium battery. If not used, connect to EVDD.

## Pin-Multiplexing and Control

Refer to the [Chapter 2, “Signal Descriptions,”](#) for more detailed descriptions of these pins and other pins not controlled by this module. The function of most of the pins (primary function, GPIO, etc.) is determined by the pin assignment registers (PAR\_x).

As shown in [Table 15-3](#), there are several cases where a function is available on more than one pin. While it is possible to enable the function on more than one pin simultaneously, avoid this for input functions to prevent unexpected behavior. All multiple-pin functions are listed in [Table 15-3](#).

**Table 15-3. Multiple-Pin Functions**

Function	Associated Pins
SDHC_CLK	DSPI0_SCK/SBF_CK, SDHC_CLK
SDHC_CMD	DSPI0_SIN/SBF_DI, SDHC_CMD
SDHC_DAT3	DSPI0_PCS0/SS, SDHC_DAT3
SDHC_DAT2	T2IN, SDHC_DAT2
SDHC_DAT1	T1IN, SDHC_DAT1
SDHC_DAT0	DSPI0_SOUT/SBF_DO, SDHC_DAT0

## 15.3 Memory Map/Register Definition

[Table 15-4](#) summarizes all the registers in the pin multiplexing and control address space.

**Table 15-4. Pin Multiplexing and Control Memory Map**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xEC09_4000–0xEC09_400A	Port output data registers (PODR_A – K)	8	R/W	0xFF	<a href="#">15.3.1/15-14</a>
0xEC09_400C–0xEC09_4016	Port data direction registers (PDDR_A – K)	8	R/W	0x00	<a href="#">15.3.2/15-15</a>
0xEC09_4018–0xEC09_4022	Port pin data/set data registers (PPDSDR_A – K)	8	R/W	See Section	<a href="#">15.3.3/15-15</a>
0xEC09_4024–0xEC09_402E	Port clear output data registers (PCLRR_A – K)	8	W	0x00	<a href="#">15.3.4/15-16</a>
0xEC09_4030–0xEC09_4044	Pull control registers (PCR_A – K)	16	R/W	See Section	<a href="#">15.3.5/15-17</a>
Pin Assignment Registers					
0xEC09_4048	PAR_FBCTL	8	R/W	See Section	<a href="#">15.3.6.1/15-18</a>
0xEC09_4049	PAR_BE	8	R/W	0xFF	<a href="#">15.3.6.2/15-18</a>
0xEC09_404A	PAR_CS	8	R/W	See Section	<a href="#">15.3.6.3/15-19</a>
0xEC09_404B	PAR_CANI2C	8	R/W	0x00	<a href="#">15.3.6.4/15-19</a>
0xEC09_404C	PAR_IRQ0H	8	R/W	0x00	<a href="#">15.3.6.5/15-19</a>
0xEC09_404D	PAR_IRQ0L	8	R/W	0x00	<a href="#">15.3.6.5/15-19</a>

**Table 15-4. Pin Multiplexing and Control Memory Map (continued)**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xEC09_404E	PAR_DSPIOWH	8	R/W	0x00	<a href="#">15.3.6.6/15-20</a>
0xEC09_404F	PAR_DSPIOWL	8	R/W	0x00	<a href="#">15.3.6.6/15-20</a>
0xEC09_4050	PAR_TIMER	8	R/W	0x00	<a href="#">15.3.6.7/15-20</a>
0xEC09_4051	PAR_UART2	8	R/W	0x00	<a href="#">15.3.6.8/15-20</a>
0xEC09_4052	PAR_UART1	8	R/W	0x55	<a href="#">15.3.6.8/15-20</a>
0xEC09_4053	PAR_UART0	8	R/W	0x00	<a href="#">15.3.6.8/15-20</a>
0xEC09_4054	PAR_SDHCH	8	R/W	0x00	<a href="#">15.3.6.9/15-21</a>
0xEC09_4055	PAR_SDHCL	8	R/W	0x00	<a href="#">15.3.6.9/15-21</a>
0xEC09_4056	PAR_SIMPOH	8	R/W	0x00	<a href="#">15.3.6.10/15-21</a>
0xEC09_4057	PAR_SIMPOL	8	R/W	0x00	<a href="#">15.3.6.10/15-21</a>
0xEC09_4058	PAR_SSI0H	8	R/W	0x00	<a href="#">15.3.6.11/15-22</a>
0xEC09_4059	PAR_SSI0L	8	R/W	0x00	<a href="#">15.3.6.11/15-22</a>
0xEC09_405A	PAR_DEBUGH1	8	R/W	0x55	<a href="#">15.3.6.12/15-22</a>
0xEC09_405B	PAR_DEBUGH0	8	R/W	0x55	<a href="#">15.3.6.12/15-22</a>
0xEC09_405C	PAR_DEBUGL	8	R/W	0x01	<a href="#">15.3.6.12/15-22</a>
0xEC09_405E	PAR_FEC	8	R/W	0x0D	<a href="#">15.3.6.13/15-23</a>
<b>Mode Select Control Registers</b>					
0xEC09_4060	MSCR_SDRAMC	8	R/W	0x03	<a href="#">15.3.7/15-24</a>
<b>Slew Rate Control Registers</b>					
0xEC09_4064	SRCR_FB1	8	R/W	See Section	<a href="#">15.3.8/15-25</a>
0xEC09_4065	SRCR_FB2	8	R/W	See Section	<a href="#">15.3.8/15-25</a>
0xEC09_4066	SRCR_FB3	8	R/W	See Section	<a href="#">15.3.8/15-25</a>
0xEC09_4067	SRCR_FB4	8	R/W	See Section	<a href="#">15.3.8/15-25</a>
0xEC09_4068	SRCR_DSPIOW	8	R/W	0x03	<a href="#">15.3.8/15-25</a>
0xEC09_4069	SRCR_CANI2C	8	R/W	0x00	<a href="#">15.3.8/15-25</a>
0xEC09_406A	SRCR_IRQ0	8	R/W	0x00	<a href="#">15.3.8/15-25</a>
0xEC09_406B	SRCR_TIMER	8	R/W	0x00	<a href="#">15.3.8/15-25</a>
0xEC09_406C	SRCR_UART	8	R/W	0x00	<a href="#">15.3.8/15-25</a>
0xEC09_406D	SRCR_FEC	8	R/W	0x00	<a href="#">15.3.8/15-25</a>
0xEC09_406E	SRCR_SDHC	8	R/W	0x00	<a href="#">15.3.8/15-25</a>
0xEC09_406F	SRCR_SIMPO	8	R/W	0x00	<a href="#">15.3.8/15-25</a>
0xEC09_4070	SRCR_SSI0	8	R/W	0x00	<a href="#">15.3.8/15-25</a>

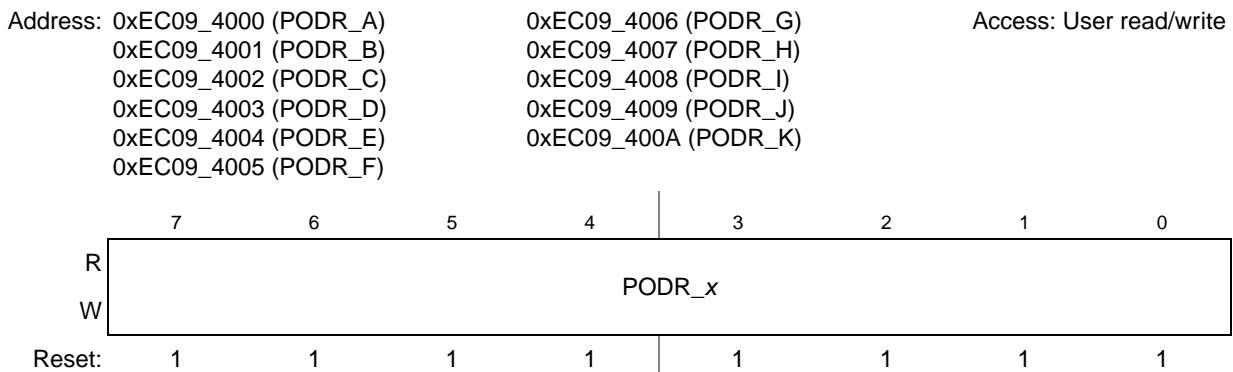
**Table 15-4. Pin Multiplexing and Control Memory Map (continued)**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
<b>Miscellaneous UART Registers</b>					
0xEC09_4074	RTS polarity control register (URTS_POL)	16	R/W	0x0000	<a href="#">15.3.9/15-29</a>
0xEC09_4076	CTS polarity control register (UCTS_POL)	16	R/W	0x0000	<a href="#">15.3.9/15-29</a>
0xEC09_4078	Transmitter wired-or mode control register (UTXD_WOM)	16	R/W	0x0000	<a href="#">15.3.10/15-29</a>
0xEC09_407C	Receiver wired-or mode control register (URXD_WOM)	32	R/W	0x0000_0000	<a href="#">15.3.10/15-29</a>
<b>Hysteresis Control Registers</b>					
0xEC09_4080	Hysteresis control register 1 (HCR1)	32	R/W	0x0000_0000	<a href="#">15.3.9/15-29</a>
0xEC09_4084	Hysteresis control register 2 (HCR0)	32	R/W	0x0000_0000	<a href="#">15.3.9/15-29</a>

### 15.3.1 Port Output Data Registers (PODR\_x)

The PODR\_x registers store the data to be driven on the corresponding port pins when the pins are configured for general purpose output. The PODR\_x registers are read/write. At reset, all PODR\_x bits are set.

Reading a PODR\_x register returns the current values in the register, not the port pin values. To set bits in a PODR\_x register, set the PODR\_x bits, or set the corresponding bits in PPDSDR\_x. To clear bits in a PODR\_x register, clear the PODR\_x bits, or clear the corresponding bits in PCLR\_R\_x.

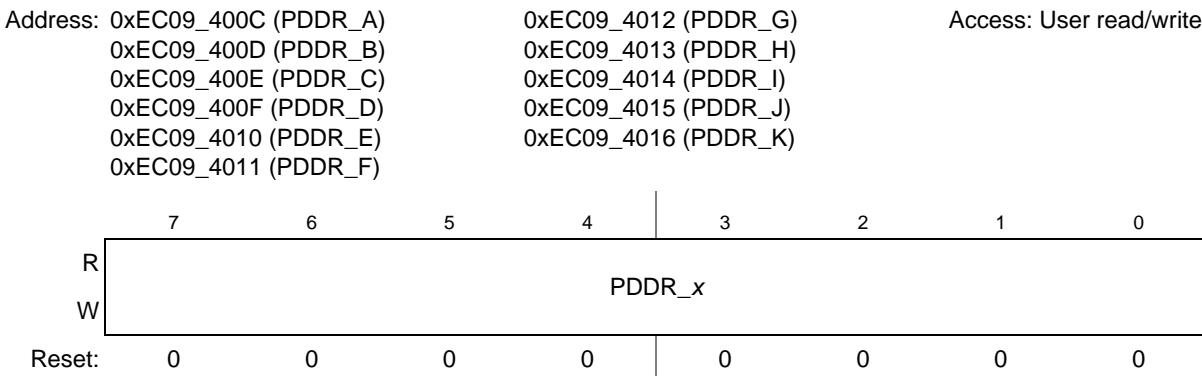
**Figure 15-2. Port x Output Data Registers (PODR\_x)****Table 15-5. PODR\_x Field Descriptions**

Field	Description
7–0 PODR_x	Port x output data bits. 0 Drives 0 when the port x pin is general purpose output 1 Drives 1 when the port x pin is general purpose output

### 15.3.2 Port Data Direction Registers (PDDR\_x)

The PDDRs control the direction of the port pin drivers when the pins are configured for GPIO. The PDDR<sub>x</sub> registers are each eight bits wide.

The PDDRs are read/write. At reset, all bits in the PDDRs are cleared. Setting any bit in a PDDR<sub>x</sub> register configures the corresponding port pin as an output. Clearing any bit in a PDDR<sub>x</sub> register configures the corresponding pin as an input.



**Figure 15-3. Port x Data Direction Registers (PDDR\_x)**

**Table 15-6. PDDR\_x Field Descriptions**

Field	Description
7–0 PDDR <sub>x</sub>	Port x output data direction bits. 1 Port x pin configured as output 0 Port x pin configured as input

### 15.3.3 Port Pin Data/Set Data Registers (PPDSDR\_x)

The PPDSDR registers reflect the current pin states and control the setting of output pins when the pin is configured for GPIO. The PPDSDR<sub>x</sub> registers are each eight bits wide.

The PPDSDR<sub>x</sub> registers are read/write. At reset, the bits in the PPDSDR<sub>x</sub> registers are set to the current pin states. Reading a PPDSDR<sub>x</sub> register returns the current state of the port *x* pins. Setting a PPDSDR<sub>x</sub> register sets the corresponding bits in the PODR<sub>x</sub> register. Writing 0s has no effect.

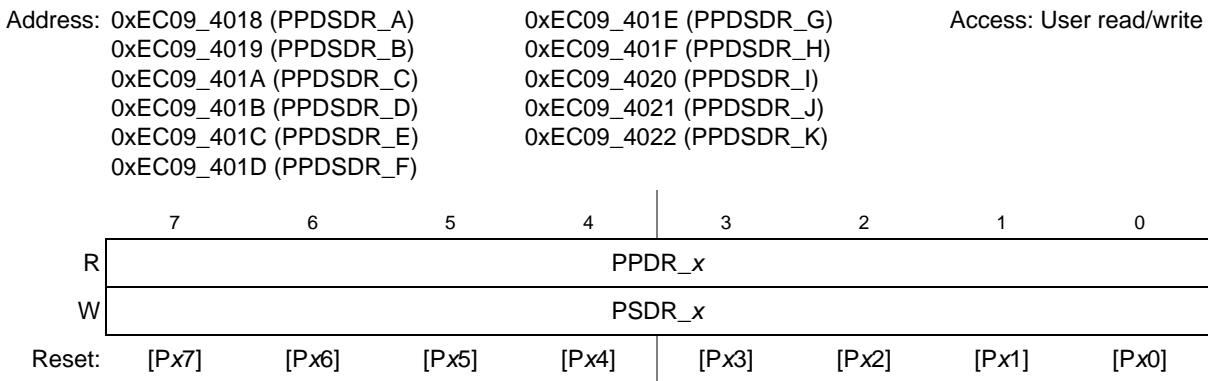


Figure 15-4. Port x Pin Data/Set Data Registers (PPDSDR\_x)

Table 15-7. PPDSDR\_x Field Descriptions

Field	Description
7–0 PPDR_x (read)	Port x pin data bits. 0 Port x pin state is 0 1 Port x pin state is 1
7–0 PSDR_x (write)	Port x set data bits. 0 No effect. 1 Set corresponding PODR_x bit.

### 15.3.4 Port Clear Output Data Registers (PCLRR\_x)

Clearing a PCLRR\_x register clears the corresponding bits in the PODR\_x register. Setting it has no effect. Reading the PCLRR\_x register returns 0s. The PCLRR\_x registers are each eight bits wide.

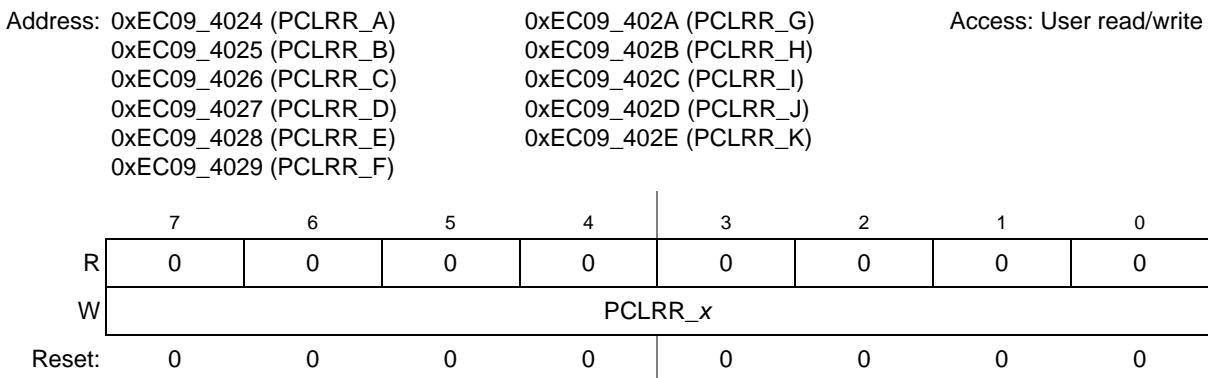


Figure 15-5. Port x Clear Output Data Registers (PCLRR\_x)

Table 15-8. PCLRR\_x Field Descriptions

Field	Description
7–0 PCLRR_x	Port x clear data bits. 0 Clears corresponding PODR_x bit 1 No effect

### 15.3.5 Pull Control Registers (PCR\_x)

Most pins on the device have configurable pull-ups/downs. The pull control registers select their direction and enable them.

## **NOTE**

The reset values for PCR\_J[PUE1] and PCR\_K[PUE0] are 1. This ensures that the TXEN pins are pulled low after reset, which disables the Ethernet PHY from transmitting unknown data after reset.

The reset values for PCR\_A[PUE4, PUS4] are 1. This ensures that after reset, a pull-up is enabled on FB\_TA/NF\_R/B.

PCR\_C[PUS2] and PCR\_E[PUS7] do not determine the direction of the pull when the USBn\_VBUS\_OC signals control these pins. See [Chapter 10, “Chip Configuration Module \(CCM\)](#), for a description of the MISCCR register which contains the pull direction controls (polarity select) for the USB controllers.

Address: 0xEC09_4030 (PCR_A)	0xEC09_403C (PCR_G)	Access: User read/write	
0xEC09_4032 (PCR_B)	0xEC09_403E (PCR_H)		
0xEC09_4034 (PCR_C)	0xEC09_4040 (PCR_I)		
0xEC09_4036 (PCR_D)	0xEC09_4042 (PCR_J)		
0xEC09_4038 (PCR_E)	0xEC09_4044 (PCR_K)		
0xEC09_403A (PCR_F)			
15      14      13      12	11      10      9      8	7      6      5      4	3      2      1      0
R W	PUE7 PUS7 PUE6 PUS6 PUE5 PUS5 PUE4 PUS4 PUE3 PUS3 PUE2 PUS2 PUE1 PUS1 PUE0 PUS0		
Reset:	0 0 0 0	0 0 0 0	0 0 0 0

**Figure 15-6. Pull Control Registers (PCR\_x)**

**Table 15-9. PCR x Field Descriptions**

Field	Description
15, 13, 11, 9, 7, 5, 3, 1 PUEn	Enables the pull for the corresponding port pin. 0 Pull disabled 1 Pull enabled
14, 12, 10, 8, 6, 4, 2, 0 PUSn	Selects the direction of the pull on corresponding pin when PUEn is set. This bit is ignored if PUEn is cleared. 0 Pull-down 1 Pull-up

### **15.3.6 Pin Assignment Registers (PAR\_x)**

The pin assignment registers control which functions are currently active on the external pins. All pin assignment registers are read/write. The bit fields in the PAR\_ $x$  registers are one or two bits wide, except for PAR\_FEC. The encodings are shown in the tables below and correspond to the headings in [Table 15-2](#).

**Table 15-10. PAR\_x Settings (Two Bit Field)**

Value	Function
00	GPIO
01	Alternate 2 Function
10	Alternate 1 Function
11	Primary Function

**Table 15-11. PAR\_x Settings (One Bit Field)**

Value	Function
0	GPIO
1	Primary Function

**NOTE**

In many cases, not all four functions are available on a pin. Do not configure the PARs for unavailable functions. Refer to [Figure 15-1](#) for the list of signals available on each pin.

**15.3.6.1 FlexBus Control Pin Assignment Register (PAR\_FBCTL)**

Address: 0xEC09\_4048 (PAR\_FBCTL)

Access: User read/write

	7	6	5	4		3	2	1	0
R					PAR_ALE	PAR_OE	PAR_FBCLK	PAR_RW	PAR_TA
W									

Reset: 1 See note #1 1 1 1 1 1 See note #2

<sup>1</sup> Depends on RCON[3] (FB\_AD3)<sup>2</sup> Depends on boot source. If booting from NAND flash, resets to 01. If booting from FlexBus, resets to 11.**Figure 15-7. FlexBus Control Pin Assignment Register (PAR\_FBCTL)****15.3.6.2 Byte Enables Pin Assignment Register (PAR\_BE)**

Address: 0xEC09\_4049 (PAR\_BE)

Access: User read/write

	7	6	5	4		3	2	1	0
R					PAR_BE3	PAR_BE2	PAR_BE1	PAR_BE0	
W									

Reset: See note See note 1 1 1 1

**Note:** Depends on boot source. If booting from NAND flash, resets to 01. If booting from FlexBus, resets to 11.

**Figure 15-8. Byte Enable Pin Assignment Register (PAR\_BE)**

### 15.3.6.3 Chip Selects Pin Assignment Register (PAR\_CS)

Address: 0xEC09_404A (PAR_CS)								Access: User read/write							
R				W				R				W			
7	6	5	4	3	2	1	0	PAR_CS5	PAR_CS4	PAR_CS1	0	PAR_CS0			
Reset:	1		1			See note #1		0		0		See note #2			

<sup>1</sup> Depends on boot source. If booting from NAND flash, resets to 01. Otherwise, resets to 00.

<sup>2</sup> Depends on boot source. If booting from FlexBus, resets to 1. Otherwise, resets to 0.

Figure 15-9. Chip Select Pin Assignment Register (PAR\_CS)

### 15.3.6.4 CAN1 and I<sup>2</sup>C0 Pin Assignment Registers (PAR\_CANI2C)

Address: 0xEC09_404B (PAR_CANI2C)								Access: User read/write							
R				W				R				W			
7	6	5	4	3	2	1	0	PAR_I2C0_SCL	PAR_I2C0_SDA	PAR_CAN1_TX	PAR_CAN1_RX				
Reset:	0	0	0	0	0	0	0					0	0	0	0

Figure 15-10. CAN1 and I<sup>2</sup>C0 Pin Assignment Register (PAR\_CANI2C)

### 15.3.6.5 Edge Port 0 Pin Assignment Registers (PAR\_IRQ0H & PAR\_IRQ0L)

The PAR\_IRQ0 registers control the functions of the edge port pins.

Address: 0xEC09_404C (PAR_IRQ0H)								Access: User read/write							
R				W				R				W			
7	6	5	4	3	2	1	0	0	0	0	0	PAR_IRQ07	PAR_IRQ04	PAR_IRQ01	
Reset:	0	0	0	0	0	0	0								

Figure 15-11. IRQ0 Pin Assignment High (PAR\_IRQ0H)

Address: 0xEC09_404D (PAR_IRQ0L)								Access: User read/write							
R				W				R				W			
7	6	5	4	3	2	1	0	0	0	0	0	PAR_IRQ06	PAR_IRQ03	PAR_IRQ02	
Reset:	0	0	0	0	0	0	0								

Figure 15-12. IRQ0 Pin Assignment Low (PAR\_IRQ0L)

### 15.3.6.6 DSPI0 and One-Wire Pin Assignment Registers (PAR\_DSPIOWH & PAR\_DSPIOWL)

The PAR\_DSPIOWH/L registers control the functions of the DSPI and one-wire pins.

Address: 0xEC09_404E (PAR_DSPIOWH)								Access: User read/write			
R				W							
7	6	5	4					3	2	1	0
PAR_SIN				PAR_SOUT				PAR_SCK			
Reset:	0	0	0	0	0	0	0	0	0	0	0

Figure 15-13. DSPI0 & One-Wire Pin Assignment High (PAR\_DSPIOWH)

Address: 0xEC09_404F (PAR_DSPIOWL)								Access: User read/write			
R				W							
7	6	5	4					3	2	1	0
PAR_PCS1				PAR_OWDAT				0	0	0	0
Reset:	0	0	0	0	0	0	0	0	0	0	0

Figure 15-14. DSPI0 & One-Wire Pin Assignment Low (PAR\_DSPIOWL)

### 15.3.6.7 Timer Pin Assignment Registers (PAR\_TIMER)

Address: 0xEC09_4050 (PAR_TIMER)								Access: User read/write			
R				W							
7	6	5	4					3	2	1	0
PAR_T3IN				PAR_T2IN				PAR_T1IN			
Reset:	0	0	0	0	0	0	0	0	0	0	0

Figure 15-15. Timer Pin Assignment (PAR\_TIMER)

### 15.3.6.8 UARTn Pin Assignment Registers (PAR\_UARTn)

The PAR\_UARTn registers control the functions of the UART2–0 pins.

Address: 0xEC09_4051 (PAR_UART2)								Access: User read/write			
0xEC09_4052 (PAR_UART1)											
0xEC09_4053 (PAR_UART0)											
7	6	5	4					3	2	1	0
R	PAR_CTS	PAR_RTS						PAR_RXD	PAR_TXD		
W	0	0	0	0	0	0	0	0	0	0	0

Figure 15-16. UARTn Pin Assignment (PAR\_UART2–0)

### 15.3.6.9 eSDHC Pin Assignment Registers (PAR\_SDHCH & PAR\_SDHCL)

The PAR\_SDHCH/L registers control the functions of the eSDHC pins.

Address: 0xEC09_4054 (PAR_SDHCH)								Access: User read/write			
	7	6	5	4		3	2	1	0		
R	PAR_DATA3		PAR_DATA2			PAR_DATA1		PAR_DATA0			
W	0	0	0	0	0	0	0	0	0	0	0

Reset: 0 0 0 0 0 0 0 0 0 0 0 0

Figure 15-17. SDHC Pin Assignment High (PAR\_SDHCH)

Address: 0xEC09_4055 (PAR_SDHCL)								Access: User read/write			
	7	6	5	4		3	2	1	0		
R	0	0	0	0		PAR_CMD		PAR_CLK			
W	0	0	0	0	0	0	0	0	0	0	0

Reset: 0 0 0 0 0 0 0 0 0 0 0 0

Figure 15-18. SDHC Pin Assignment High (PAR\_SDHCL)

### 15.3.6.10 SIM Port 0 Pin Assignment Registers (PAR\_SIMP0H & PAR\_SIMP0L)

The PAR\_SIMP0H/L registers control the functions of the SIM port 1 pins.

Address: 0xEC09_4056 (PAR_SIMP0H)								Access: User read/write			
	7	6	5	4		3	2	1	0		
R	PAR_DATA		PAR_VEN			PAR_RST		PAR_PD			
W	0	0	0	0	0	0	0	0	0	0	0

Reset: 0 0 0 0 0 0 0 0 0 0 0 0

Figure 15-19. SIM Port 0 Pin Assignment High (PAR\_SIMP0H)

Address: 0xEC09_4057 (PAR_SIMP0L)								Access: User read/write			
	7	6	5	4		3	2	1	0		
R	0	0	0	0		0	0	PAR_CLK			
W	0	0	0	0	0	0	0	0	0	0	0

Reset: 0 0 0 0 0 0 0 0 0 0 0 0

Figure 15-20. SIM Port 0 Pin Assignment Low (PAR\_SIMP0L)

### 15.3.6.11 SSI0 Pin Assignment Registers (PAR\_SSI0H & PAR\_SSI0L)

The PAR\_SSI0H/L registers control the functions of the SSI0 pins.

Address: 0xEC09_4058 (PAR_SSI0H)								Access: User read/write							
R				W				R				W			
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
R	PAR_RXD			PAR_TXD				R	PAR_FS			PAR_MCLK			
W								W							
Reset:	0	0	0	0	0	0	0	Reset:	0	0	0	0	0	0	0

Figure 15-21. SSI0 Pin Assignment High (PAR\_SSI0H)

Address: 0xEC09_4059 (PAR_SSI0L)								Access: User read/write							
R				W				R				W			
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	R	PAR_BCLK						
W								W							
Reset:	0	0	0	0	0	0	0	Reset:	0	0	0	0	0	0	0

Figure 15-22. SSI0 Pin Assignment Low (PAR\_SSI0L)

### 15.3.6.12 Debug Pin Assignment Registers (PAR\_DEBUGH1, PAR\_DEBUGH0 & PAR\_DEBUGL)

The PAR\_DEBUGx registers control the functions of the debug pins.

Address: 0xEC09_405A (PAR_DEBUGH1)								Access: User read/write							
R				W				R				W			
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
R	0	PAR_DDATA3		0	PAR_DDATA2			R	0	PAR_DDATA1		0	PAR_DDATA0		
W								W							
Reset:	0	1	0	1	0	1	0	Reset:	0	1	0	0	1	0	1

Figure 15-23. Debug Pin Assignment High 1 (PAR\_DEBUGH1)

Address: 0xEC09_405B (PAR_DEBUGH0)								Access: User read/write							
R				W				R				W			
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
R	0	PAR_PST3		0	PAR_PST2			R	0	PAR_PST1		0	PAR_PST0		
W								W							
Reset:	0	1	0	1	0	1	0	Reset:	0	1	0	0	1	0	1

Figure 15-24. Debug Pin Assignment High 0 (PAR\_DEBUGH0)

Address: 0xEC09\_405C (PAR\_DEBUGL) Access: User read/write

	7	6	5	4	3	2	1	0	
R	0	0	0	0	0	0	0	0	PAR_ ALLPST
W									
Reset:	0	0	0	0	0	0	0	0	1

Figure 15-25. Debug Pin Assignment Low (PAR\_DEBUGL)

### 15.3.6.13 FEC Pin Assignment Registers (PAR\_FEC)

The PAR\_FEC register configures the device's pins for the various FEC functions, ULPI, or GPIO functions. This register resets only after a power-on-reset.

Address: 0xEC09\_405E (PAR\_FEC) Access: User read/write

	7	6	5	4	3	2	1	0	
R	0	0	0	0					PAR_FEC
W									
Reset:	0	0	0	0	0	1	1	0	1

Figure 15-26. FEC Pin Assignment (PAR\_FEC)

**Table 15-12. PAR\_FEC Field Descriptions**

Field	Description		
7-3	Reserved, must be cleared.		
2-0 PAR_FEC	PAR_FEC	Mode	Description
	0000	MII full mode	All 18 MII0 pins function as MII0. <b>Note:</b> RMII_MODE bit for MAC 0 must be cleared.
	0001	MII non-full mode	Same as MII full mode, except MII0_MDC and MII0_MDIO pins function as GPIO. <b>Note:</b> RMII_MODE bit for MAC 0 must be cleared.
	0010	—	Reserved
	0011	RMII0 & 1 full mode	All 18 MII0 pins function as their respective RMII0 and RMII1 signals. <b>Note:</b> RMII_MODE bits for MAC0 and MAC 1 must be set.
	0100	RMII0 & 1 non-full mode	Same as RMII full mode, but MII0_MDC, MII0_MDIO, MII0_COL and MII0_TXER function as GPIO. <b>Note:</b> RMII_MODE bits for MAC 0 and MAC 1 must be set.
	0101	RMII0 full mode RMII1 non-full mode	Same as RMII full mode, but MII0_COL and MII0_TXER pins function as GPIO. <b>Note:</b> RMII_MODE bits for MAC 0 and MAC 1 must be set.
	0110	RMII0 non-full mode RMII1 full mode	Same as RMII full mode, except MII0_MDC and MII0_MDIO as GPIO. <b>Note:</b> RMII_MODE bits for MAC 0 and MAC 1 must be set.
	0111	RMII0 full mode	All MII0 pins that have RMII0 functionality function as RMII0. Other nine MII0 pins function as GPIO. <b>Note:</b> RMII_MODE bit for MAC 0 must be set.
	1000	RMII0 full mode ULPI mode	All MII0 pins that have RMII0 functionality function as RMII 0. Other nine MII0 pins function as ULPI signals. <b>Note:</b> RMII_MODE bit for MAC 0 must be set.
	1001	RMII0 non-MI mode	Same as RMII0 full mode, except MII0_MDC and MII0_MDIO function as GPIO. <b>Note:</b> RMII_MODE bit for MAC 0 must be set.
	1010	RMII non-MI mode ULPI mode	Same as RMII0 full & ULPI mode except that MII0_MDC and MII0_MDIO function as GPIO. <b>Note:</b> RMII_MODE bit for MAC 0 must be set.
	1011	RMII1 full mode	All MII0 pins that have RMII1 functionality function as RMII 1. Other 9 MII0 pins function as GPIO. <b>Note:</b> RMII_MODE bit for MAC 1 must be set.
	1100	RMII1 non-MI mode	Same as RMII1 full mode except that MII0_COL and MII0_TXER function as GPIO. <b>Note:</b> RMII_MODE bit for MAC 1 must be set.
	1101	All GPIO	All MII0 pins function as GPIO.
	1110	—	Reserved
	1111	—	Reserved

### 15.3.7 SDRAM Mode Select Control Registers (MSCR\_SDRAMC)

The MSCR\_SDRAMC register selects the drive strength of the following SDRAM pins:

- SD\_A[14:0], SD\_BA[2:0], SD\_CAS, SD\_CKE, SD\_CLK, SD\_CLK, SD\_CS, SD\_D[7:0], SD\_DQS, SD\_DQS, SD\_ODT, SD\_RAS, and SD\_WE

Address: 0xEC09\_4060 (MSCR\_SDRAMC)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0		
W							MSC	

Reset: 0 0 0 0 0 0 1 1

**Figure 15-27. SDRAMC Mode Select Control Register (MSCR\_SDRAMC)****Table 15-13. MSCR\_SDRAMC Field Descriptions**

Field	Description
7–2	Reserved, must be cleared.
1–0 MSC	Drive strength mode. 00 Half strength 1.8V DDR2 01 Full strength 1.8V DDR2 10 Reserved 11 2.5V DDR1

### 15.3.8 Slew Rate Control Registers (SRCR\_x)

The SRCR\_x registers control the slew rate of the some pins on the device. See [Table 15-15](#) for a list of the affected pins.

#### NOTE

To allow the I/O interfaces to run at their maximum frequency, set their respective SRE\_x bits to 11.

Address: 0xEC09\_4064 (SRCR\_FB1)  
0xEC09\_4065 (SRCR\_FB2)  
0xEC09\_4066 (SRCR\_FB3)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0		
W							SRE_FBn	

Reset: 0 0 0 0 0 0 1 1

**Figure 15-28. Flexbus Slew Rate Control Registers (SRCR\_FB0–3)**

Address: 0xEC09\_4067 (SRCR\_FB4)

Access: User read/write

	7	6	5	4	3	2	1	0
R	0	0	0	0				
W					SRE_FB5		SRE_FB4	

Reset: 0 0 0 0 1 See Note 1 See Note

**Note:** Reset state is the value of CCR[LOAD].

**Figure 15-29. Flexbus Slew Rate Control Registers (SRCR\_FB4)**

## Pin-Multiplexing and Control

Address: 0xEC09\_4068 (SRCR\_DSPIOW) Access: User read/write

	7	6	5	4		3	2	1	0
R	0	0	SRE_OWDAT			0	0	SRE_DSPI0	
W									

Reset: 0 0 0 0 0 0 0 0 1 1

Figure 15-30. DSPI0 & One-Wire Slew Rate Control Register (SRCR\_DSPIOW)

Address: 0xEC09\_4069 (SRCR\_CANI2C) Access: User read/write

	7	6	5	4		3	2	1	0
R	0	0	0	0		SRE_CAN1		SRE_I2C0	
W						0	0	0	0

Reset: 0 0 0 0 0 0 0 0 0 0

Figure 15-31. CAN & I<sup>2</sup>C Slew Rate Control Register (SRCR\_CANI2C)

Address: 0xEC09\_406A (SRCR\_IRQ0) Access: User read/write

	7	6	5	4		3	2	1	0
R	0	0	0	0		0	0	SRE_IRQ0	
W						0	0	0	0

Reset: 0 0 0 0 0 0 0 0 0 0

Figure 15-32. Edge Port 0 Slew Rate Control Register (SRCR\_IRQ0)

Address: 0xEC09\_406B (SRCR\_TIMER) Access: User read/write

	7	6	5	4		3	2	1	0
R	SRE_TIMER3		SRE_TIMER2		SRE_TIMER1		SRE_TIMER0		
W						0	0	0	0

Reset: 0 0 0 0 0 0 0 0 0 0

Figure 15-33. Timer Slew Rate Control Register (SRCR\_TIMER)

Address: 0xEC09\_406C (SRCR\_UART) Access: User read/write

	7	6	5	4		3	2	1	0
R	0	0	SRE_UART2		SRE_UART1		SRE_UART0		
W						0	0	0	0

Reset: 0 0 0 0 0 0 0 0 0 0

Figure 15-34. UART Slew Rate Control Register (SRCR\_UART)

Address: 0xEC09\_406D (SRCR\_FEC)

Access: User read/write

	7	6	5	4		3	2	1	0
R	0	0	0	0		SRE_RMII0		SRE_RMII1	
W									
Reset:	0	0	0	0		0	0	0	0

**Note:** This register is reset only after a power-on-reset.**Figure 15-35. FEC Slew Rate Control Register (SRCR\_FEC)**

Address: 0xEC09\_406E (SRCR\_SDHC)

Access: User read/write

	7	6	5	4		3	2	1	0
R	0	0	0	0		0	0		SRE_SDHC
W									
Reset:	0	0	0	0		0	0	0	0

**Figure 15-36. eSDHC Slew Rate Control Register (SRCR\_SDHC)**

Address: 0xEC09\_406F (SRCR\_SIM0)

Access: User read/write

	7	6	5	4		3	2	1	0
R	0	0	0	0		0	0		SRE_SIM0
W									
Reset:	0	0	0	0		0	0	0	0

**Figure 15-37. SIM0 Slew Rate Control Register (SRCR\_SIM0)**

Address: 0xEC09\_4070 (SRCR\_SSI0)

Access: User read/write

	7	6	5	4		3	2	1	0
R	0	0	0	0		0	0		SRE_SSI0
W									
Reset:	0	0	0	0		0	0	0	0

**Figure 15-38. SSI0 Slew Rate Control Register (SRCR\_SSI0)****Table 15-14. SRCR\_x Field Descriptions**

Field	Description
SRE_x	Slew rate control. 00 Lowest slew rate 01 Low slew rate 10 High slew rate 11 Highest slew rate

**Note:** See above figures for bit field positions.

**Table 15-15. SR<sub>R</sub><sub>C</sub><sub>R</sub>\_x Pins Affected**

SR <sub>R</sub> <sub>C</sub> <sub>R</sub> _x	Signals Affected
SRE_FB1	FB_BE/BWE[1:0], FB_CS[5,4,0], FB_ALE, FB_CLK (dedicated FlexBus clock and control signals)
SRE_FB2	FB_BE/BWE[3:2], FB_CS1, FB_OE, FB_R/W, FB_TA (shared FlexBus/NFC signals)
SRE_FB3	FB_AD[15:0]
SRE_FB4	FB_AD[23:16]
SRE_FB5	FB_AD[31:24]
SRE_DSPIO	DSPI0_PCS[1:0], DSPI0_SIN, DSPI0_SOUT, DSPI0_SCK
SRE_OWDAT	OW_DAT
SRE_CAN1	CAN1_TX, CAN1_RX
SRE_I2C0	I2C0_SDA, I2C_SCL
SRE_IRQ0	IRQ0[7,6,4:1]
SRE_TIMER3	T3IN
SRE_TIMER2	T2IN
SRE_TIMER1	T1IN
SRE_TIMER0	T0IN
SRE_UART2	UART2_TXD, UART2_RXD, UART2_CTS, UART2_RTS
SRE_UART1	UART1_TXD, UART1_RXD, UART1_CTS, UART1_RTS
SRE_UART0	UART0_TXD, UART0_RXD, UART0_CTS, UART0_RTS
SRE_SDHC	SDHC_CLK, SDHC_CMD, SDHC_DAT[3:0]
SRE_SIM0	SIM0_CLK, SIM0_PD, SIM0_RST, SIM0_VEN, SIM0_DAT
SRE_SSI0	SSI0_BCLK, SSI0_FS, SSI0_MCLK, SSI0_RXD, SSI0_TXD
SRE_RMII0	MII0_COL, MII0_TXER, MII0_CRS, MII0_RXD[3:2], MII0_RXCLK, MII0_TXD[3:2], MII0_TXCLK
SRE_RMII1	MII0_MDC, MII0_MDIO, MII0_RXDV, MII0_RXD[1:0], MII0_RXER, MII0_TXD[1:0], MII0_TXEN

### 15.3.9 UART RTS and CTS Polarity Control Register (URTS\_POL & UCTS\_POL)

U<sub>x</sub>\_POL controls the polarity of the UARTs' RTS and CTS signals.

Address: 0xEC09\_4074 (URTS\_POL)  
0xEC09\_4076 (UCTS\_POL)

Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	U2 INV	U1 INV	U0 INV
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-39. UART RTS and CTS Polarity Control Register (URTS\_POL & UCTS\_POL)

Table 15-16. URTS\_POL & UCTS\_POL Field Descriptions

Field	Description
15–3	Reserved, must be cleared.
2–0 UnINV	UART <sub>n</sub> inverter enable 0 Disable; corresponding RTS/CTS signal is asserted low 1 Enable; corresponding RTS/CTS signal is asserted high

### 15.3.10 UART Transmitter & Receiver Wired-Or Mode Control Registers (UTXD\_WOM & URXD\_WOM)

U<sub>x</sub>\_WOM controls the wired-or mode of the UARTs' TXD and RXD signals.

Address: 0xEC09\_4078 (UTXD\_WOM)

Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	U9 WOM	U8 WOM	U7 WOM	U6 WOM	U5 WOM	U4 WOM	U3 WOM	U2 WOM	U1 WOM	U0 WOM
W																
Reset:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-40. UART TXD Wired-Or Mode Control Register (UTXD\_WOM)

Table 15-17. UTXD\_WOM Field Descriptions

Field	Description
15–10	Reserved, must be cleared.
9–0 UnWOM	UART <sub>n</sub> _TXD wired-or enable. 0 Disable; corresponding UART <sub>n</sub> _TXD not in wired-or mode. 1 Enable; corresponding UART <sub>n</sub> _TXD in wired-or mode.

Address: 0xEC09\_407C (URXD\_WOM) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	U9	U8		
W													WOM	WOM		
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	U7 WOM		U6 WOM		U5 WOM		U4 WOM		U3 WOM		U2 WOM		U1 WOM		U0 WOM	
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 15-41. UART RXD Wired-Or Mode Control Register (URXD\_WOM)

Table 15-18. UTXD\_WOM Field Descriptions

Field	Description
31–20	Reserved, must be cleared.
19–0 UnWOM	UART $n$ _RXD wired-or control. 00 Disconnect UART $n$ _RXD from UART $n$ _TXD 01 Connect UART $n$ _RXD to UART $n$ _TXD 1x Automatic single-wire mode. <u>UART<math>n</math>_RTS</u> controls when UART $n$ _RXD is connected to UART $n$ _TXD

## 15.4 Hysteresis Control Registers (HCR0–1)

Some of the pins on the device have hysteresis controls. The bit field names in the following figures correspond to the GPIO signal names in [Table 15-2](#).

Address: 0xEC09\_4080 (HCR1) Access: User 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	PG[4:0]				PF[7:3]				PE[6:0]				PD[7:3]				PC[7:1]				PB[2:0]												
W	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	0

Figure 15-42. Hysteresis Control Register 1 (HCR1)

Address: 0xEC09\_4084 (HCR0) Access: User 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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PK	PK	PD[2:0]	PE	PH[7:3]							
W																																	

Figure 15-43. Hysteresis Control Register 0 (HCR0)

**Table 15-19. HCR0–1 Field Descriptions**

Field	Description
P <sub>xn</sub>	Hysteresis enable 0 Disable 1 Enable

## 15.5 Functional Description

### 15.5.1 Overview

Initial pin function is determined during reset configuration. The pin assignment registers select among various primary functions and general purpose I/O after reset. Most pins are configured as GPIO by default. The notable exceptions are external bus control pins, address/data pins, chip select, and debug pins. These pins are configured for their primary functions after reset, allowing access to external boot memory and allowing dynamic debug information to appear on the pins after reset.

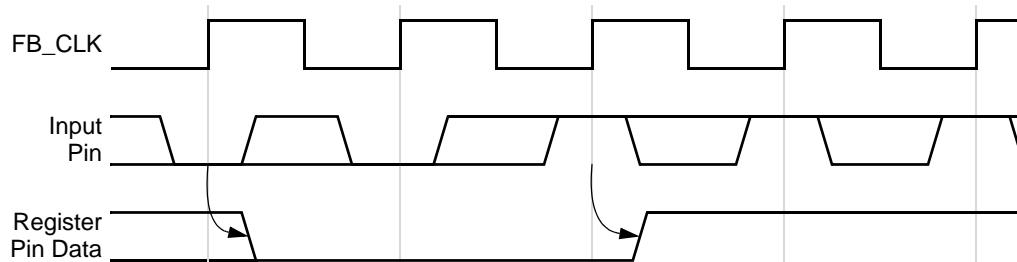
Every GPIO pin is individually configurable as an input or output via a data direction register (PDDR<sub>-x</sub>). Every GPIO port has an output data register (PODR<sub>-x</sub>) and a pin data register (PPDSDR<sub>-x</sub>) to monitor and control the state of its pins. Data written to a PODR<sub>-x</sub> register is stored and then driven to the corresponding port *x* pins configured as outputs.

Reading a PODR<sub>-x</sub> register returns the current state of the register regardless of the state of the corresponding pins. Reading a PPDSDR<sub>-x</sub> register returns the current state of the corresponding pins when configured as GPIO, regardless of whether the pins are inputs or outputs.

Every GPIO port has a PPDSDR<sub>-x</sub> register and a clear register (PCLR<sub>-x</sub>) for setting or clearing individual bits in the PODR<sub>-x</sub> register.

### 15.5.2 Port Digital I/O Timing

Input data on all pins configured as general purpose input is synchronized to the rising edge of FB\_CLK, as shown in [Figure 15-44](#).

**Figure 15-44. General Purpose Input Timing**

Data written to the PODR<sub>-x</sub> register of any pin configured as a general purpose output is immediately driven to its respective pin, as shown in [Figure 15-45](#).

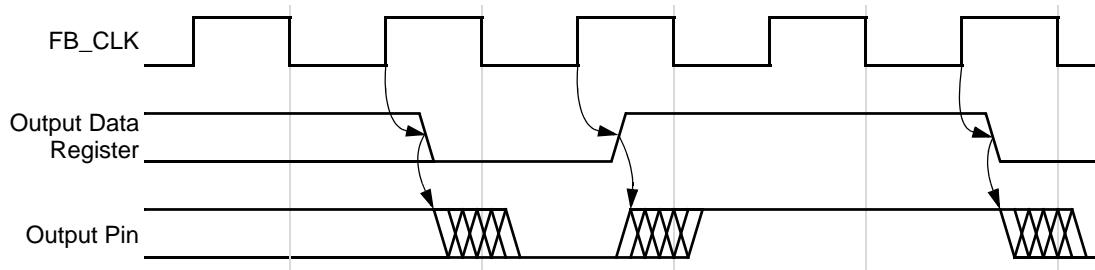


Figure 15-45. General Purpose Output Timing

## 15.6 Initialization/Application Information

The initialization for the pin multiplexing and control is done during reset configuration. All registers are reset to a predetermined state. Refer to [Section 15.3, “Memory Map/Register Definition,”](#) for more details on reset and initialization.

# Chapter 16

## Rapid GPIO (RGPIO)

### 16.1 Introduction

The Rapid GPIO (RGPIO) module provides a 16-bit general-purpose I/O module directly connected to the processor's high-speed 32-bit local bus. This connection plus support for single-cycle, zero wait-state data transfers allows the RGPIO module to provide improved pin performance when compared to more traditional GPIO modules located on the internal slave peripheral bus.

Many of the pins associated with a device may be used for several different functions. Their primary functions are to provide external interfaces to access off-chip resources. When not used for their primary function, many of the pins may be used as general-purpose digital I/O (GPIO) pins. The definition of the exact pin functions and the affected signals is specific to each device. Every GPIO port, including the RGPIO module, has registers that configure, monitor, and control the port pins.

#### 16.1.1 Overview

The RGPIO module provides 16-bits of high-speed GPIO functionality, mapped to the processor's bus. The key features of this module include:

- 16 bits of high-speed GPIO functionality connected to the processor's local 32-bit bus
- Memory-mapped device connected to the ColdFire core's local bus
  - Support for all access sizes: byte, word, and longword
  - All reads and writes complete in a single data phase cycle for zero wait-state response
- Data bits can be accessed directly or via alternate addresses to provide set, clear, and toggle functions
  - Alternate addresses allow set, clear, toggle functions using simple store operations without the need for read-modify-write references
- Unique data direction and pin enable control registers
- Package pin toggle rates typically 1.5–3.5x faster than comparable pin mapped onto peripheral bus

A simplified block diagram of the RGPIO module is shown in [Figure 16-1](#). The details of the pin muxing and pad logic are device-specific.

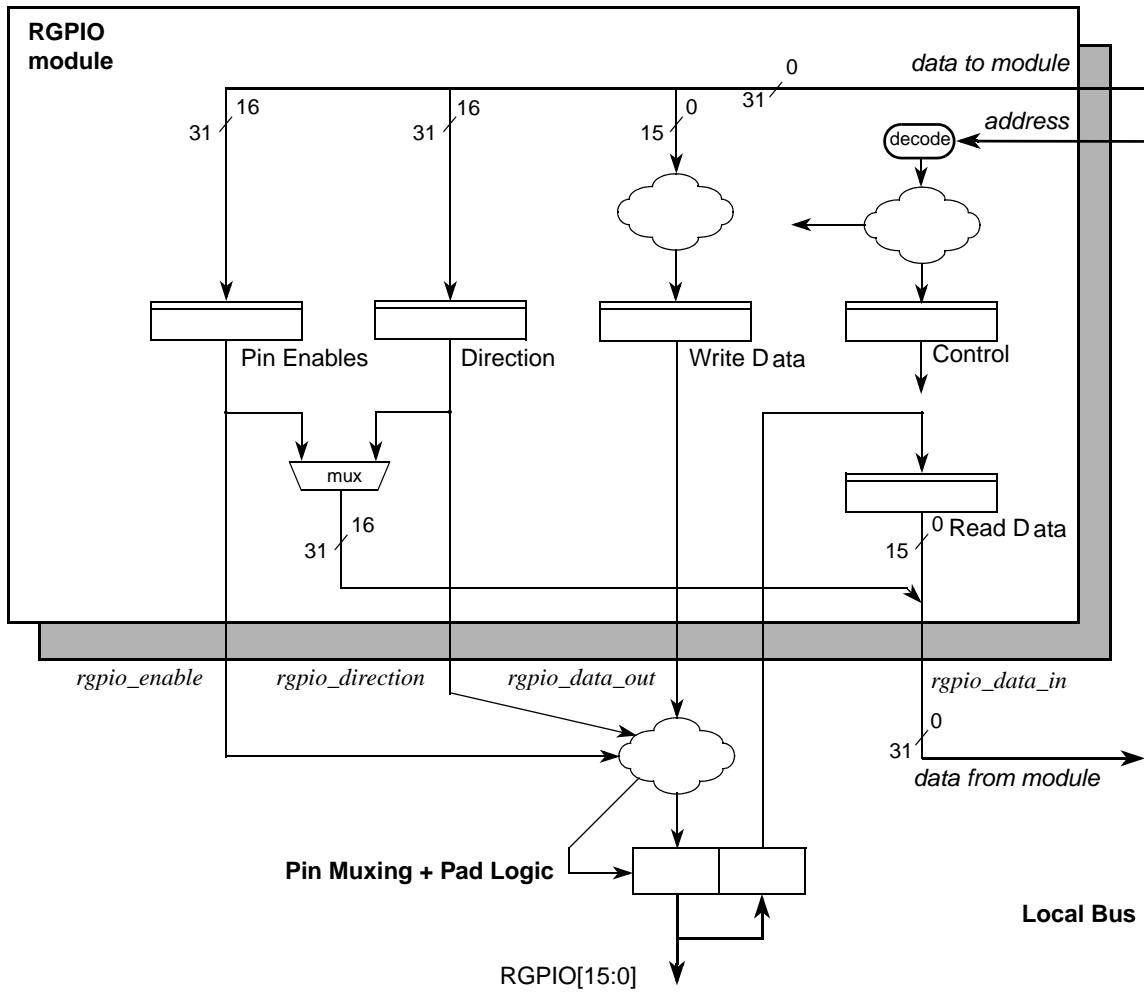


Figure 16-1. RGPIO Block Diagram

### 16.1.2 Features

The major features of the RGPIO module providing 16 bits of high-speed general-purpose input/output are:

- Small memory-mapped device connected to the processor's local bus
  - All memory references complete in a single cycle to provide zero wait-state responses
  - Located in processor's high-speed clock domain
    - Option to operate at the processor clock or half the processor clock frequency by a register located in the CCM. See [Chapter 10, “Chip Configuration Module \(CCM\)”,](#) for details.
- Simple programming model
  - Four 16-bit registers, mapped as three program-visible locations
    - Register for pin enables
    - Register for controlling the pin data direction
    - Register for storing output pin data

- Register for reading current pin state
- The two data registers (read, write) are mapped to a single program-visible location
- Alternate addresses to perform data set, clear, and toggle functions using simple writes
- Separate read and write programming model views enable simplified driver software
- Support for any access size (byte, word, or longword)

### 16.1.3 Modes of Operation

The RGPIO module does not support any special modes of operation. As a memory-mapped device located on the processor's high-speed local bus, it responds based strictly on memory address and does not consider the operating mode (supervisor, user) of its references.

## 16.2 External Signal Description

### 16.2.1 Overview

As shown in [Figure 16-1](#), the RGPIO module's interface to external logic is indirect via the device pin-muxing and pad logic. For a list of the associated RGPIO input/output signals, see [Table 16-1](#).

**Table 16-1. RGPIO Module External I/O Signals**

Signal Name	Type	Description
GPIO[15:0]	I/O	RGPIO Data Input/Output

### 16.2.2 Detailed Signal Descriptions

[Table 16-2](#) provides descriptions of the RGPIO module's input and output signals.

**Table 16-2. RGPIO Detailed Signal Descriptions**

Signal	I/O	Description				
GPIO[15:0]	I/O	<p>Data Input/Output. When configured as an input, the state of this signal is reflected in the read data register. When configured as an output, this signal is the output of the write data register.</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 15%; padding: 5px;"><b>State Meaning</b></td> <td style="width: 85%; padding: 5px;">           Asserted—            Input: Indicates the GPIO pin was sampled as a logic high at the time of the read.            Output: Indicates a properly-enabled GPIO output pin is to be driven high.            Negated—            Input: Indicates the GPIO pin was sampled as a logic low at the time of the read.            Output: Indicates a properly-enabled GPIO output pin is to be driven low.         </td> </tr> <tr> <td style="padding: 5px;"><b>Timing</b></td> <td style="padding: 5px;">           Assertion/Negation—            Input: Anytime. The input signal is sampled at the rising-edge of the processor's high-speed clock on the data phase cycle of a read transfer of this register.            Output: Occurs at the rising-edge of the processor's high-speed clock on the data phase cycle of a write transfer to this register. This output is asynchronously cleared by system reset.         </td> </tr> </table>	<b>State Meaning</b>	Asserted— Input: Indicates the GPIO pin was sampled as a logic high at the time of the read. Output: Indicates a properly-enabled GPIO output pin is to be driven high. Negated— Input: Indicates the GPIO pin was sampled as a logic low at the time of the read. Output: Indicates a properly-enabled GPIO output pin is to be driven low.	<b>Timing</b>	Assertion/Negation— Input: Anytime. The input signal is sampled at the rising-edge of the processor's high-speed clock on the data phase cycle of a read transfer of this register. Output: Occurs at the rising-edge of the processor's high-speed clock on the data phase cycle of a write transfer to this register. This output is asynchronously cleared by system reset.
<b>State Meaning</b>	Asserted— Input: Indicates the GPIO pin was sampled as a logic high at the time of the read. Output: Indicates a properly-enabled GPIO output pin is to be driven high. Negated— Input: Indicates the GPIO pin was sampled as a logic low at the time of the read. Output: Indicates a properly-enabled GPIO output pin is to be driven low.					
<b>Timing</b>	Assertion/Negation— Input: Anytime. The input signal is sampled at the rising-edge of the processor's high-speed clock on the data phase cycle of a read transfer of this register. Output: Occurs at the rising-edge of the processor's high-speed clock on the data phase cycle of a write transfer to this register. This output is asynchronously cleared by system reset.					

## 16.3 Memory Map/Register Definition

The GPIO module provides a compact 16-byte programming model based at a system memory address of 0x8C00\_0000 (noted as GPIO\_BASE throughout the chapter). The GPIOBAR register configures access to the GPIO module. Program this register by using the privileged MOVEC instruction with an Rc address of 0x009. As previously noted, the programming model views are different between reads and writes as this enables simplified software for manipulation of the GPIO pins.

Additionally, the GPIO programming model is defined with a 32-bit organization. The basic size of each program-visible register is 16 bits, but the programming model may be referenced using byte (8-bit), word (16-bit) or longword (32-bit) accesses. Performance is typically maximized using 32-bit accesses.

### NOTE

Writes to the two-byte fields at GPIO\_BASE + 0x8 and GPIO\_BASE + 0xC are allowed, but do not affect any program-visible register within the GPIO module.

**Table 16-3. GPIO Write Memory Map**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0x8C00_0000	GPIO Data Direction Register (GPIO_DIR)	16	W	0x0000	<a href="#">16.3.2/16-5</a>
0x8C00_0002	GPIO Write Data Register (GPIO_DATA)	16	W	0x0000	<a href="#">16.3.3/16-6</a>
0x8C00_0004	GPIO Pin Enable Register (GPIO_ENB)	16	W	0x0000	<a href="#">16.3.4/16-6</a>
0x8C00_0006	GPIO Write Data Clear Register (GPIO_CLR)	16	W	N/A	<a href="#">16.3.5/16-7</a>
0x8C00_000A	GPIO Write Data Set Register (GPIO_SET)	16	W	N/A	<a href="#">16.3.6/16-7</a>
0x8C00_000E	GPIO Write Data Toggle Register (GPIO_TOG)	16	W	N/A	<a href="#">16.3.7/16-8</a>

**Table 16-4. GPIO Read Memory Map**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0x8C00_0000	GPIO Data Direction Register (GPIO_DIR)	16	R	0x0000	<a href="#">16.3.2/16-5</a>
0x8C00_0002	GPIO Write Data Register (GPIO_DATA)	16	R	0x0000	<a href="#">16.3.3/16-6</a>
0x8C00_0004	GPIO Pin Enable Register (GPIO_ENB)	16	R	0x0000	<a href="#">16.3.4/16-6</a>
0x8C00_0006	GPIO Write Data Register (GPIO_DATA)	16	R	0x0000	<a href="#">16.3.3/16-6</a>
0x8C00_0008	GPIO Data Direction Register (GPIO_DIR)	16	R	0x0000	<a href="#">16.3.2/16-5</a>
0x8C00_000A	GPIO Write Data Register (GPIO_DATA)	16	R	0x0000	<a href="#">16.3.3/16-6</a>
0x8C00_000C	GPIO Data Direction Register (GPIO_DIR)	16	R	0x0000	<a href="#">16.3.2/16-5</a>
0x8C00_000E	GPIO Write Data Register (GPIO_DATA)	16	R	0x0000	<a href="#">16.3.3/16-6</a>

### 16.3.1 RGPIOBAR Register

Rc: 0x009 Access: Supervisor write-only  
Debug 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	1	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	1	1	0	1	V	
W																													UD			
Reset	1	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	1	1	0	0		

Figure 16-2. RGPIOBAR Register

Table 16-5.

Field	Description	
31–2	Reserved, must be set to 0b1000_1100_0000_0000_0000_0011_01.	
1 UD	User data address space mask. 0 Allows access to the RGPIOBAR register. 1 Disables the address space from the RGPIOBAR register. If a reference using this address space is made, it is inhibited from accessing the RGPIOBAR register, and is processed like any other memory reference.	
0 V	Valid. 0 Processor accesses of the RGPIOBAR register are masked. 1 Processor accesses of the RGPIOBAR register are enabled	

### 16.3.2 RGPIODIR Register

The read/write RGPIODIR register defines whether a properly-enabled RGPIODIR pin is configured as an input or output:

- Setting any bit in RGPIODIR configures a properly-enabled RGPIODIR port pin as an output
- Clearing any bit in RGPIODIR configures a properly-enabled RGPIODIR port pin as an input

At reset, all bits in the RGPIODIR are cleared.

Offset: RGPIODIR = RGPIOBASE + 0x0 (RGPIODIR)  
Access: Read/write  
RGPIODIR = RGPIOBASE + 0x8 Read-only  
RGPIODIR = RGPIOBASE + 0xC Read-only

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

Figure 16-3. RGPIODIR Register

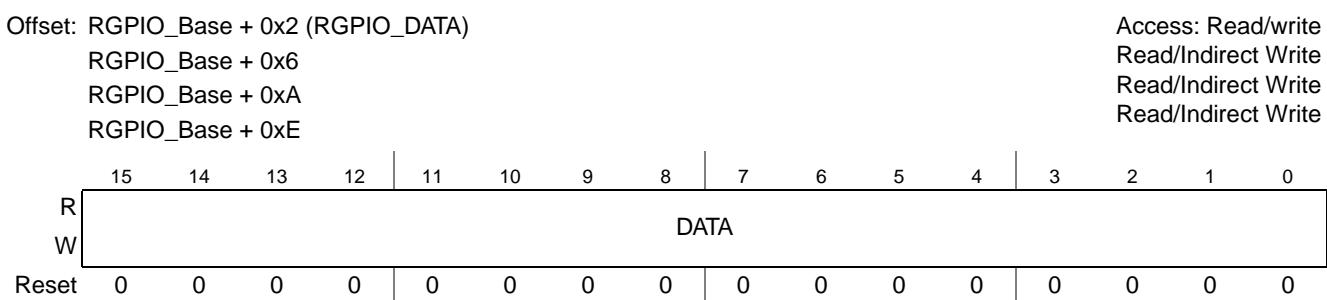
Table 16-6. RGPIODIR Field Descriptions

Field	Description
15–0 DIR	Data direction. 0 A properly-enabled RGPIODIR pin is configured as an input 1 A properly-enabled RGPIODIR pin is configured as an output

### 16.3.3 RGPIODATA (RGPIO\_DATA)

The RGPIODATA register specifies the write data for a properly-enabled RGPIODATA output pin or the sampled read data value for a properly-enabled input pin. An attempted read of the RGPIODATA register returns undefined data for disabled pins, since the data value is dependent on the device-level pin muxing and pad implementation. The RGPIODATA register is read/write. At reset, all bits in the RGPIODATA registers are cleared.

To set bits in a RGPIODATA register, directly set the RGPIODATA bits or set the corresponding bits in the RGPIOSET register. To clear bits in the RGPIODATA register, directly clear the RGPIODATA bits, or clear the corresponding bits in the RGPIOCLR register. Setting a bit in the RGPIOTOG register inverts (toggles) the state of the corresponding bit in the RGPIODATA register.



**Figure 16-4. RGPIODATA Register (RGPIODATA)**

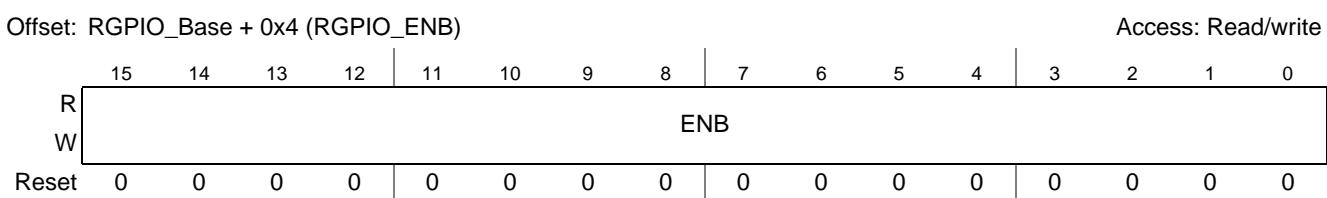
**Table 16-7. RGPIODATA Field Descriptions**

Field	Description
15–0 DATA	RGPIODATA. 0 A properly-enabled RGPIODATA output pin is driven with a logic 0, or a properly-enabled RGPIODATA input pin was read as a logic 0 1 A properly-enabled RGPIODATA output pin is driven with a logic 1, or a properly-enabled RGPIODATA input pin was read as a logic 1

### 16.3.4 RGPIOPINENABLE (RGPIO\_ENB)

The RGPIOENB register configures the corresponding package pin as a RGPIOPINENABLE pin instead of the normal GPIO pin mapped onto the peripheral bus.

The RGPIOENB register is read/write. At reset, all bits in the RGPIOENB are cleared, disabling the RGPIOPINENABLE functionality.



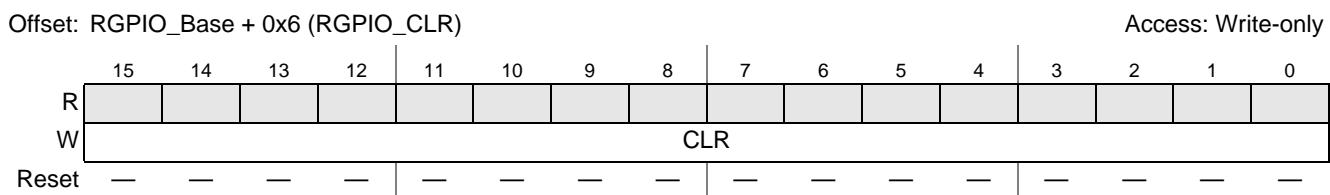
**Figure 16-5. RGPIOENB Register (RGPIOENB)**

**Table 16-8. GPIO\_ENB Field Descriptions**

Field	Description
15–0 ENB	Enable pin for GPIO 0 The corresponding package pin is configured for use as a normal GPIO pin, not a GPIO 1 The corresponding package pin is configured for use as a GPIO pin

### 16.3.5 GPIO Clear Data (GPIO\_CLR)

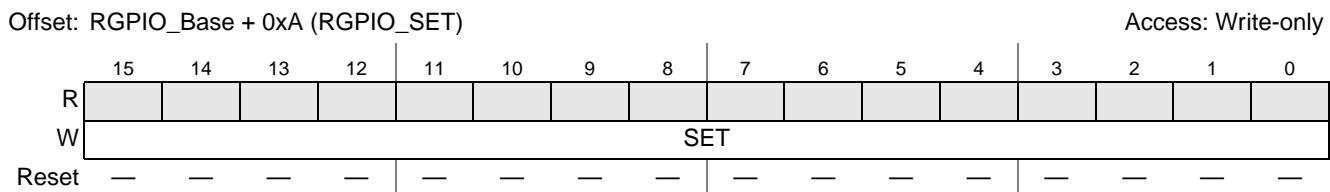
The GPIO\_CLR register provides a mechanism to clear specific bits in the GPIO\_DATA by performing a simple write. Clearing a bit in GPIO\_CLR clears the corresponding bit in the GPIO\_DATA register. Setting it has no effect. The GPIO\_CLR register is write-only; reads of this address return the GPIO\_DATA register.

**Figure 16-6. GPIO Clear Data Register (GPIO\_CLR)****Table 16-9. GPIO\_CLR Field Descriptions**

Field	Description
15–0 CLR	Clear data bit 0 Clears the corresponding bit in the GPIO_DATA register 1 No effect

### 16.3.6 GPIO Set Data (GPIO\_SET)

The GPIO\_SET register provides a mechanism to set specific bits in the GPIO\_DATA register by performing a simple write. Setting a bit in GPIO\_SET asserts the corresponding bit in the GPIO\_DATA register. Clearing it has no effect. The GPIO\_SET register is write-only; reads of this address return the GPIO\_DATA register.

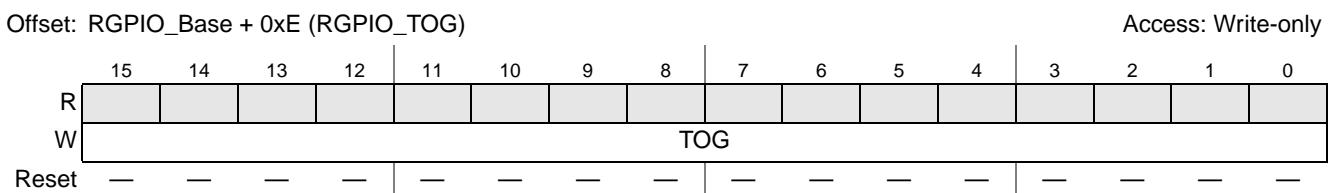
**Figure 16-7. GPIO Set Data Register (GPIO\_SET)**

**Table 16-10. GPIO\_SET Field Descriptions**

Field	Description
15–0 SET	Set data bit 0 No effect 1 Sets the corresponding bit in the GPIO_DATA register

### 16.3.7 GPIO Toggle Data (GPIO\_TOG)

The GPIO\_TOG register provides a mechanism to invert (toggle) specific bits in the GPIO\_DATA register by performing a simple write. Setting a bit in GPIO\_TOG inverts the corresponding bit in the GPIO\_DATA register. Clearing it has no effect. The GPIO\_TOG register is write-only; reads of this address return the GPIO\_DATA register.

**Figure 16-8. GPIO Toggle Data Register (GPIO\_TOG)****Table 16-11. GPIO\_TOG Field Descriptions**

Field	Description
15–0 TOG	Toggle data. 0 No effect 1 Inverts the corresponding bit in GPIO_DATA

## 16.4 Functional Description

The GPIO module is a relatively-simple design with its behavior controlled by the program-visible registers defined within its programming model.

The GPIO module is connected to the processor's local two-stage pipelined bus with the stages of the ColdFire core's operand execution pipeline (OEP) mapped directly onto the bus. This structure allows the processor access to the GPIO module for single-cycle pipelined reads and writes with a zero wait-state response (as viewed in the system bus data phase stage).

## 16.5 Initialization Information

The reset state of the GPIO module disables the entire 16-bit data port. Prior to using the GPIO port, software typically:

- Enables the appropriate pins in GPIO\_ENB
- Configures the pin direction in GPIO\_DIR
- Defines the contents of the data register (GPIO\_DATA)

## 16.6 Application Information

This section examines the relative performance of the RGPIO output pins for two simple applications

- The processor executes a loop to toggle an output pin for a specific number of cycles, producing a square-wave output
- The processor transmits a 16-bit message using a three-pin SPI-like interface with a serial clock, serial chip select, and serial data bit.

In both applications, the relative speed of the GPIO output is presented as a function of the location of the output bit (RGPIO versus peripheral bus GPIO).

### 16.6.1 Application 1: Simple Square-Wave Generation

In this example, several different instruction loops are executed, each generating a square-wave output with a 50% duty cycle. For this analysis, the executed code is mapped into the processor's RAM. The following instruction loops were studied:

- **BCHG\_LOOP** — In this loop, a bit change instruction was executed using the GPIO data byte as the operand. This instruction performs a read-modify-write operation and inverts the addressed bit. A pulse counter is decremented until the appropriate number of square-wave pulses have been generated.
- **SET+CLR\_LOOP** — For this construct, two store instructions are executed: one to set the GPIO data pin and another to clear it. Single-cycle NOP instructions (the tpf opcode) are included to maintain the 50% duty cycle of the generated square wave. The pulse counter is decremented until the appropriate number of square-wave pulse have been generated.

The square-wave output frequency was measured and the relative performance results are presented in [Table 16-12](#). The relative performance is stated as a fraction of the processor's operating frequency, defined as  $f$  MHz. The performance of the BCHG loop operating on a GPIO output is selected as the reference.

**Table 16-12. Square-Wave Output Performance**

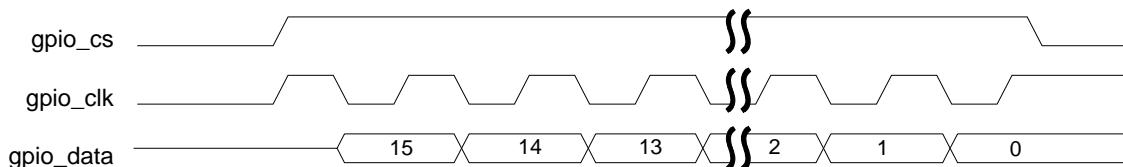
Loop	Peripheral Bus-mapped GPIO			RGPIO		
	Sq-Wave Frequency	Frequency @ CPU $f = 50$ MHz	Relative Speed	Sq-Wave Frequency	Frequency @ CPU $f = 50$ MHz	Relative Speed
bchg	$(1/24) \times f$ MHz	2.083 MHz	1.00x	$(1/14) \times f$ MHz	3.571 MHz	1.71x
set+clr (+toggle)	$(1/12) \times f$ MHz	4.167 MHz	2.00x	$(1/8) \times f$ MHz	6.250 MHz	3.00x

#### NOTE

The square-wave frequency is measured from rising-edge to rising-edge, where the output wave has a 50% duty cycle.

## 16.6.2 Application 2: 16-bit Message Transmission using SPI Protocol

In this second example, a 16-bit message is transmitted using three programmable output pins. The output pins include a serial clock, an active-high chip select, and the serial data bit. The software is configured to sample the serial data bit at the rising-edge of the clock with the data sent in a most-significant to least-significant bit order. The resulting 3-bit output is shown in [Figure 16-9](#).



**Figure 16-9. GPIO SPI Example Timing Diagram**

For this example, the processing of the SPI message is considerably more complex than the generation of a simple square wave of the previous example. The code snippet used to extract the data bit from the message and build the required GPIO data register writes is shown in [Figure 16-10](#).

```
# subtest: send a 16-bit message via a SPI interface using a RGPIO

# the SPI protocol uses a 3-bit value: clock, chip-select, data
# the data is centered around the rising-edge of the clock

    align 16
send_16b_spi_message_rgpio:
    lea    -12(%sp),%sp          # allocate stack space
00510: 4fef ffff4             movm.l &0x8c,(%sp)      # save d2,d3,d7
00514: 48d7 008c              mov.w  RAM_BASE+message2,%d2  # get 16-bit message
00518: 3439 0080 0582         movq.l &15,%d3        # static shift count
0051e: 760f                   movq.l &16,%d7        # message bit length
00520: 7e10                   mov.l   &GPIO_DATA+1,%a0  # pointer to low-order data byte
00522: 207c 00c0 0003         mov.l   &0xffff,%d0      # data value for _ENB and _DIR regs
00528: 203c 0000 ffff         mov.w  %d0,-3(%a0)    # set GPIO_DIR register
0052e: 3140 fffd              mov.w  %d0,1(%a0)    # set GPIO_ENB register
00532: 3140 0001              align 4

00536: 223c 0001 0000         mov.l  &0x10000,%d1     # d1[17:16] = {clk, cs}
0053c: 2001                   mov.l  %d1,%d0        # copy into temp reg
0053e: e6a8                   lsr.l  %d3,%d0        # align in d0[2:0]
00540: 5880                   addq.l &4,%d0        # set clk = 1
00542: 1080                   mov.b  %d0,(%a0)    # initialize data
00544: 6002                   bra.b  L%1
                                align 4

L%1:
00548: 3202                   mov.w  %d2,%d1        # d1[17:15] = {clk, cs, data}
0054a: 2001                   mov.l  %d1,%d0        # copy into temp reg
0054c: e6a8                   lsr.l  %d3,%d0        # align in d0[2:0]
0054e: 1080                   mov.b  %d0,(%a0)    # transmit data with clk = 0
00550: 5880                   addq.l &4,%d0        # force clk = 1
00552: e38a                   lsl.l  &1,%d2        # d2[15] = new message data bit
00554: 51fc                   tpf
00556: 51fc                   tpf
00558: 51fc                   tpf
0055a: 51fc                   tpf
```

```

0055c: 1080      mov.b   %d0,(%a0)          # transmit data with clk = 1
0055e: 5387      subq.l  &1,%d7            # decrement loop counter
00560: 66e6      bne.b   L%1

00562: c0bc 0000 ffff5      and.l   &0xffff5,%d0    # negate chip-select
00568: 1080      mov.b   %d0,(%a0)          # update gpio

0056a: 4cd7 008c      movm.l  (%sp),&0x8c    # restore d2,d3,d7
0056e: 4fef 000c      lea     12(%sp),%sp    # deallocate stack space
00572: 4e75      rts


```

**Figure 16-10. GPIO SPI Code Example**

The resulting SPI performance, as measured in the effective Mbps transmission rate for the 16-bit message, is shown in [Table 16-13](#).

**Table 16-13. Emulated SPI Performance using GPIO Outputs**

Peripheral Bus-mapped GPIO		RGPIO	
SPI Speed @ CPU f = 50 MHz	Relative Speed	SPI Speed @ CPU f = 50 MHz	Relative Speed
2.063 Mbps	1.00x	3.809 Mbps	1.29x



# Chapter 17

## Interrupt Controller Modules

### 17.1 Introduction

This section details the functionality of the interrupt controllers (INTC0, INTC1, INTC2). The general features of the interrupt controller block include:

- 192 fully-programmable interrupt sources. Not all possible interrupt source locations are used on this device
- Each of the sources has a unique interrupt control register (ICR0 $n$ , ICR1 $n$ , ICR2 $n$ ) to define the software-assigned levels
- Unique vector number for each interrupt source
- Ability to mask any individual interrupt source, plus global mask-all capability
- Supports hardware and software interrupt acknowledge cycles
- Wake-up signal from low-power stop modes

The 64, fully-programmable interrupt sources for the three interrupt controllers manage the complete set of interrupt sources from all of the modules on the device. This section describes how the interrupt sources are mapped to the interrupt controller logic and how interrupts are serviced.

#### 17.1.1 68 K/ColdFire Interrupt Architecture Overview

Before continuing with the specifics of the interrupt controllers, a brief review of the interrupt architecture of the 68K/ColdFire family is appropriate.

The interrupt architecture of ColdFire is exactly the same as the M68000 family, where there is a 3-bit encoded interrupt priority level sent from the interrupt controller to the core, providing 7 levels of interrupt requests. Level 7 represents the highest priority interrupt level, while level 1 is the lowest priority. The processor samples for active interrupt requests once-per-instruction by comparing the encoded priority level against a 3-bit interrupt mask value (I) contained in bits 10:8 of the machine's status register (SR). If the priority level is greater than the SR[I] field at the sample point, the processor suspends normal instruction execution and initiates interrupt exception processing. Level 7 interrupts are treated as non-maskable and edge-sensitive within the processor, while levels 1-6 are treated as level-sensitive and may be masked depending on the value of the SR[I] field. For correct operation, the ColdFire device requires that, after asserted, the interrupt source remain asserted until explicitly disabled by the interrupt service routine.

During the interrupt exception processing, the CPU enters supervisor mode, disables trace mode, and then fetches an 8-bit vector from the interrupt controller. This byte-sized operand fetch is known as the interrupt acknowledge (IACK) cycle with the ColdFire implementation using a special memory-mapped address

space within the interrupt controller. The fetched data provides an index into the exception vector table that contains 256 addresses, each pointing to the beginning of a specific exception service routine. In particular, vectors 64 - 255 of the exception vector table are reserved for user interrupt service routines. The first 64 exception vectors are reserved for the processor to manage reset, error conditions (access, address), arithmetic faults, system calls, etc. After the interrupt vector number has been retrieved, the processor continues by creating a stack frame in memory. For ColdFire, all exception stack frames are 2 longwords in length, and contain 32 bits of vector and status register data, along with the 32-bit program counter value of the instruction that was interrupted (see [Section 3.3.3.1, “Exception Stack Frame Definition,”](#) for more information on the stack frame format). After the exception stack frame is stored in memory, the processor accesses the 32-bit pointer from the exception vector table using the vector number as the offset, and then jumps to that address to begin execution of the service routine. After the status register is stored in the exception stack frame, the SR[I] mask field is set to the level of the interrupt being acknowledged, effectively masking that level and all lower values while in the service routine.

The processing of the interrupt acknowledge cycle is fundamentally different than previous 68K/ColdFire cores. In this approach, all IACK cycles are directly managed by the interrupt controller, so the requesting peripheral device is not accessed during the IACK. As a result, the interrupt request must be explicitly cleared in the peripheral during the interrupt service routine. For more information, see [Section 17.3.1.3, “Interrupt Vector Determination.”](#)

ColdFire processors guarantee that the first instruction of the service routine is executed before sampling for interrupts is resumed. By making this initial instruction a load of the SR, interrupts can be safely disabled, if required.

For more information on exception processing, see the *ColdFire Programmer’s Reference Manual* at <http://www.freescale.com/coldfire>.

## 17.2 Memory Map/Register Definition

The register programming model for the interrupt controllers is memory-mapped to a 256-byte space. In the following discussion, there are a number of program-visible registers greater than 32 bits in size. For these control fields, the physical register is partitioned into two 32-bit values: a register high (the upper longword) and a register low (the lower longword). The nomenclature <reg\_name>H and <reg\_name>L is used to reference these values.

The registers and their locations are defined in [Table 17-2](#). The base addresses for the interrupt controllers are listed below.

**Table 17-1. Interrupt Controller Base Addresses**

Interrupt Controller Number	Base Address
INTC0	0xFC04_8000
INTC1	0xFC04_C000
INTC2	0xFC05_0000
Global IACK Registers Space <sup>1</sup>	0xFC05_4000

<sup>1</sup> This address space only contains the global SWIACK and global L1ACK-L7IACK registers. See [Section 17.2.10, “Software and Level 1–7 IACK Registers \(SWIACK \$n\$ , L1ACK \$n\$ –L7IACK \$n\$ \)](#) for more information

**Table 17-2. Interrupt Controller Memory Map**

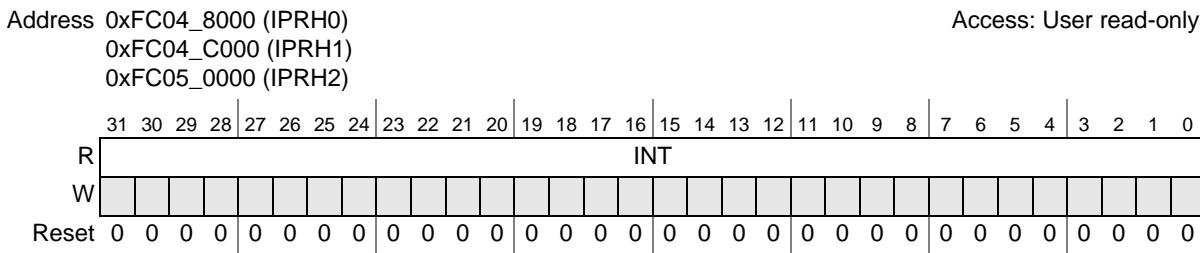
Address	Register	Width (bits)	Access	Reset Value	Section/ Page
<b>Interrupt Controller 0</b>					
0xFC04_8000	Interrupt Pending Register High (IPRH0)	32	R	0x0000_0000	<a href="#">17.2.1/17-4</a>
0xFC04_8004	Interrupt Pending Register Low (IPRL0)	32	R	0x0000_0000	<a href="#">17.2.1/17-4</a>
0xFC04_8008	Interrupt Mask Register High (IMRH0)	32	R/W	0xFFFF_FFFF	<a href="#">17.2.2/17-5</a>
0xFC04_800C	Interrupt Mask Register Low (IMRL0)	32	R/W	0xFFFF_FFFF	<a href="#">17.2.2/17-5</a>
0xFC04_8010	Interrupt Force Register High (INTFRCH0)	32	R/W	0x0000_0000	<a href="#">17.2.3/17-7</a>
0xFC04_8014	Interrupt Force Register Low (INTFRCL0)	32	R/W	0x0000_0000	<a href="#">17.2.3/17-7</a>
0xFC04_801A	Interrupt Configuration Register (ICONFIG)	16	R/W	0x0000	<a href="#">17.2.4/17-8</a>
0xFC04_801C	Set Interrupt Mask (SIMR0)	8	W	0x00	<a href="#">17.2.5/17-8</a>
0xFC04_801D	Clear Interrupt Mask (CIMR0)	8	W	0x00	<a href="#">17.2.6/17-9</a>
0xFC04_801E	Current Level Mask (CLMASK)	8	R/W	0x0F	<a href="#">17.2.7/17-10</a>
0xFC04_801F	Saved Level Mask (SLMASK)	8	R/W	0x0F	<a href="#">17.2.8/17-11</a>
0xFC04_8040 + $n$ ( $n=0:63$ )	Interrupt Control Registers (ICR0 $n$ )	8	R/W	0x00	<a href="#">17.2.9/17-12</a>
0xFC04_80E0	Software Interrupt Acknowledge (SWIACK0)	8	R	0x00	<a href="#">17.2.10/17-19</a>
0xFC04_80E0 + $4n$ ( $n=1:7$ )	Level $n$ Interrupt Acknowledge Registers (LnIACK0)	8	R	0x18	<a href="#">17.2.10/17-19</a>
<b>Interrupt Controller 1</b>					
0xFC04_C000	Interrupt Pending Register High (IPRH1)	32	R	0x0000_0000	<a href="#">17.2.1/17-4</a>
0xFC04_C004	Interrupt Pending Register Low (IPRL1)	32	R	0x0000_0000	<a href="#">17.2.1/17-4</a>
0xFC04_C008	Interrupt Mask Register High (IMRH1)	32	R/W	0xFFFF_FFFF	<a href="#">17.2.2/17-5</a>
0xFC04_C00C	Interrupt Mask Register Low (IMRL1)	32	R/W	0xFFFF_FFFF	<a href="#">17.2.2/17-5</a>
0xFC04_C010	Interrupt Force Register High (INTFRCH1)	32	R/W	0x0000_0000	<a href="#">17.2.3/17-7</a>
0xFC04_C014	Interrupt Force Register Low (INTFRCL1)	32	R/W	0x0000_0000	<a href="#">17.2.3/17-7</a>
0xFC04_C01C	Set Interrupt Mask (SIMR1)	8	W	0x00	<a href="#">17.2.5/17-8</a>
0xFC04_C01D	Clear Interrupt Mask (CIMR1)	8	W	0x00	<a href="#">17.2.5/17-8</a>
0xFC04_C040 + $n$ ( $n=1:63$ )	Interrupt Control Registers (ICR1 $n$ )	8	R/W	0x00	<a href="#">17.2.9/17-12</a>
0xFC04_C0E0	Software Interrupt Acknowledge (SWIACK1)	8	R	0x00	<a href="#">17.2.10/17-19</a>

**Table 17-2. Interrupt Controller Memory Map (continued)**

Address	Register	Width (bits)	Access	Reset Value	Section/ Page
0xFC04_C0E0 + 4n (n=1:7)	Level <i>n</i> Interrupt Acknowledge Registers (LnIACK1)	8	R	0x18	<a href="#">17.2.10/17-19</a>
<b>Interrupt Controller 3</b>					
0xFC05_0000	Interrupt Pending Register High (IPRH2)	32	R	0x0000_0000	<a href="#">17.2.1/17-4</a>
0xFC05_0004	Interrupt Pending Register Low (IPRL2)	32	R	0x0000_0000	<a href="#">17.2.1/17-4</a>
0xFC05_0008	Interrupt Mask Register High (IMRH2)	32	R/W	0xFFFF_FFFF	<a href="#">17.2.2/17-5</a>
0xFC05_000C	Interrupt Mask Register Low (IMRL2)	32	R/W	0xFFFF_FFFF	<a href="#">17.2.2/17-5</a>
0xFC05_0010	Interrupt Force Register High (INTFRCH2)	32	R/W	0x0000_0000	<a href="#">17.2.3/17-7</a>
0xFC05_0014	Interrupt Force Register Low (INTFRCL2)	32	R/W	0x0000_0000	<a href="#">17.2.3/17-7</a>
0xFC05_001C	Set Interrupt Mask (SIMR2)	8	W	0x00	<a href="#">17.2.5/17-8</a>
0xFC05_001D	Clear Interrupt Mask (CIMR2)	8	W	0x00	<a href="#">17.2.5/17-8</a>
0xFC05_0040 + <i>n</i> (n=1:63)	Interrupt Control Registers (ICR2 <i>n</i> )	8	R/W	0x00	<a href="#">17.2.9/17-12</a>
0xFC05_00E0	Software Interrupt Acknowledge (SWIACK2)	8	R	0x00	<a href="#">17.2.10/17-19</a>
0xFC05_00E0 + 4 <i>n</i> (n=1:7)	Level <i>n</i> Interrupt Acknowledge Registers (LnIACK2)	8	R	0x18	<a href="#">17.2.10/17-19</a>
<b>Global IACK Registers</b>					
0xFC05_40E0	Global Software Interrupt Acknowledge (GSWIACK)	8	R	0x00	<a href="#">17.2.10/17-19</a>
0xFC05_40E0 + 4 <i>n</i> (n=1:7)	Global Level <i>n</i> Interrupt Acknowledge Registers (GLnIACK)	8	R	0x18	<a href="#">17.2.10/17-19</a>

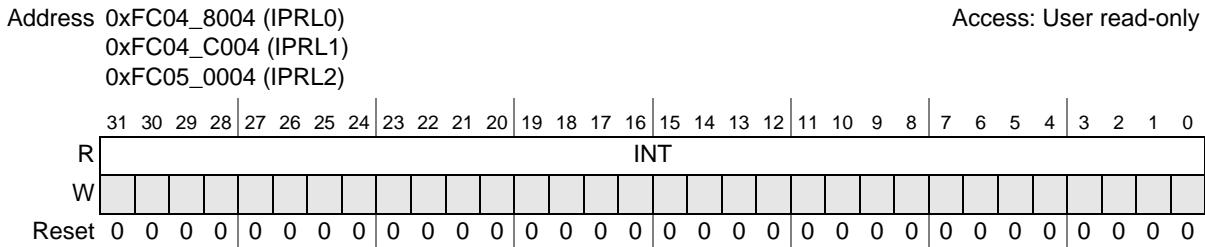
## 17.2.1 Interrupt Pending Registers (IPRH*n*, IPRL*n*)

The IPRH*n* and IPRL*n* registers, [Figure 17-1](#) and [Figure 17-2](#), are each 32 bits in size, and provide a bit map for each interrupt request to indicate if there is an active request (1 equals active request, 0 equals no request) for the given source. The interrupt mask register state does not affect the IPR*n*. The IPR*n* is cleared by reset and is a read-only register, so any attempted write to this register is ignored.

**Figure 17-1. Interrupt Pending Register High (IPRH*n*)**

**Table 17-3. IPRH $n$  Field Descriptions**

Field	Description
31–0 INT	Interrupt pending. Each bit corresponds to an interrupt source. The corresponding IMRH $n$ bit determines whether an interrupt condition can generate an interrupt. At every system clock, the IPRH $n$ samples the signal generated by the interrupting source. The corresponding IPRH $n$ bit reflects the state of the interrupt signal even if the corresponding IMRH $n$ bit is set. 0 The corresponding interrupt source does not have an interrupt pending 1 The corresponding interrupt source has an interrupt pending

**Figure 17-2. Interrupt Pending Register Low (IPRL $n$ )****Table 17-4. IPRL $n$  Field Descriptions**

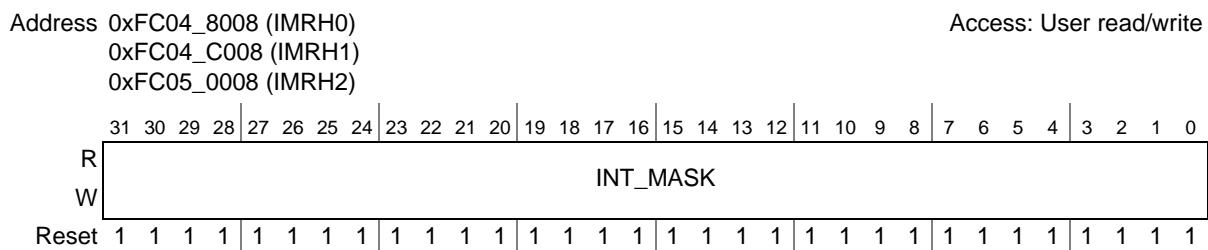
Field	Description
31–0 INT	Interrupt Pending. Each bit corresponds to an interrupt source. The corresponding IMRL $n$ bit determines whether an interrupt condition can generate an interrupt. At every system clock, the IPRL $n$ samples the signal generated by the interrupting source. The corresponding IPRL $n$ bit reflects the state of the interrupt signal even if the corresponding IMRL $n$ bit is set. 0 The corresponding interrupt source does not have an interrupt pending 1 The corresponding interrupt source has an interrupt pending

## 17.2.2 Interrupt Mask Register (IMRH $n$ , IMRL $n$ )

The IMRH $n$  and IMRL $n$  registers are each 32 bits in size and provide a bit map for each interrupt to allow the request to be disabled (1 equals disable the request, 0 equals enable the request). The IMRL register is used for masking interrupt sources 0 to 31, while the IMRH register is used for masking interrupts 32 to 63. The IMR $n$  is set to all ones by reset, disabling all interrupt requests. The IMR $n$  can be read and written.

## **NOTE**

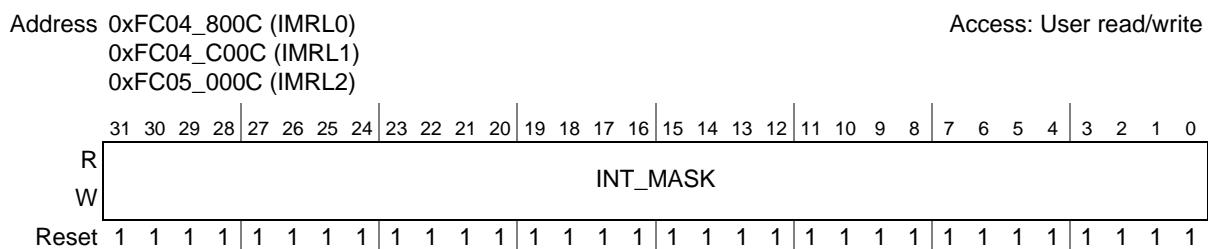
A spurious interrupt may occur if an interrupt source is being masked in the interrupt controller mask register (IMR) or a module's interrupt mask register while the interrupt mask in the status register (SR[I]) is set to a value lower than the interrupt's level. This is because by the time the status register acknowledges this interrupt, the interrupt has been masked. A spurious interrupt is generated because the CPU cannot determine the interrupt source. To avoid this situation for interrupts sources with levels 1-6, first write a higher level interrupt mask to the status register, before setting the mask in the IMR or the module's interrupt mask register. After the mask is set, return the interrupt mask in the status register to its previous value. Because level 7 interrupts cannot be disabled in the status register prior to masking, use of the IMR or module interrupt mask registers to disable level 7 interrupts is not recommended.



**Figure 17-3. Interrupt Mask Register High (IMRHn)**

**Table 17-5. IMRHN Field Descriptions**

Field	Description
31–0 INT_MASK	<p>Interrupt mask. Each bit corresponds to an interrupt source. The corresponding IMRH<math>n</math> bit determines whether an interrupt condition can generate an interrupt. The corresponding IPRH<math>n</math> bit reflects the state of the interrupt signal even if the corresponding IMRH<math>n</math> bit is set.</p> <ul style="list-style-type: none"> <li>0 The corresponding interrupt source is not masked</li> <li>1 The corresponding interrupt source is masked</li> </ul>



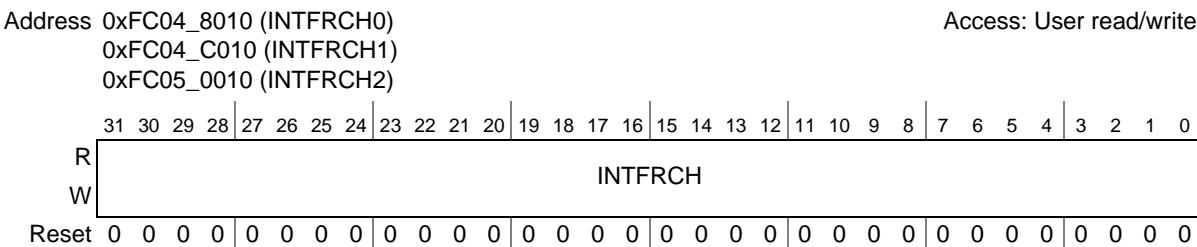
**Figure 17-4. Interrupt Mask Register Low (IMRLn)**

**Table 17-6. IMRLn Field Descriptions**

Field	Description
31–0 INT_MASK	<p>Interrupt mask. Each bit corresponds to an interrupt source. The corresponding IMRL<math>n</math> bit determines whether an interrupt condition can generate an interrupt. The corresponding IPRL<math>n</math> bit reflects the state of the interrupt signal even if the corresponding IMRL<math>n</math> bit is set.</p> <p>0 The corresponding interrupt source is not masked 1 The corresponding interrupt source is masked</p>

### 17.2.3 Interrupt Force Registers (INTFRCH $n$ , INTFRCL $n$ )

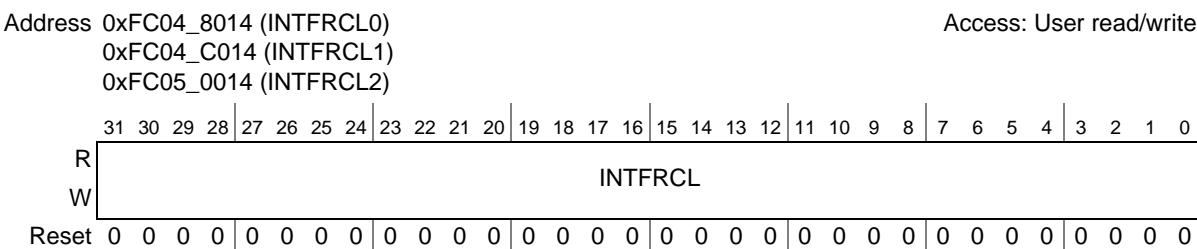
The INTFRCH $n$  and INTFRCL $n$  registers are each 32 bits in size and provide a mechanism to allow software generation of interrupts for each possible source for functional or debug purposes. The system design may reserve one or more sources to allow software to self-schedule interrupts by forcing one or more of these bits (set to force request, clear to negate request) in the appropriate INTFRC $n$  register. The INTFRCL $n$  register forces interrupts for sources 0 to 31, while the INTFRCH $n$  register forces interrupts for sources 32 to 63. The assertion of an interrupt request via the interrupt force register is not affected by the interrupt mask register. The INTFRC $n$  registers are cleared by reset.



**Figure 17-5. Interrupt Force Register High (INTFRCH $n$ )**

**Table 17-7. INTFRCH*n* Field Descriptions**

Field	Description
31-0 INTFRCH	Interrupt force. Allows software generation of interrupts for each possible source for functional or debug purposes.



**Figure 17-6. Interrupt Force Register Low (INTFRCLn)**

**Table 17-8. INTFRCL $n$  Field Descriptions**

Field	Description
31–0 INTFRCL	Interrupt force. Allows software generation of interrupts for each possible source for functional or debug purposes. 0 No interrupt forced on corresponding interrupt source 1 Force an interrupt on the corresponding source

## 17.2.4 Interrupt Configuration Register (ICONFIG)

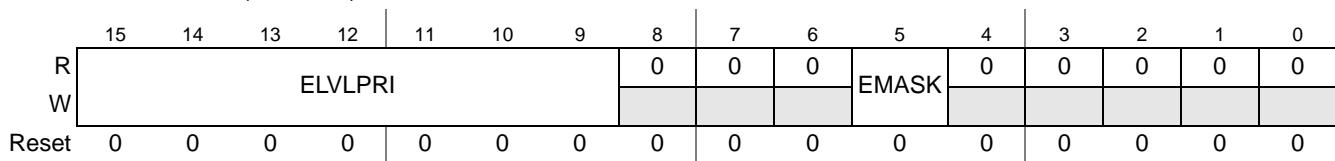
This 16-bit register defines the operating configuration for the interrupt controller module.

### NOTE

Only one copy of this register exists among the 3 interrupt controller modules. All reads and writes to this register must be made to the INTC0 memory space.

Address: 0xFC04\_801A (ICONFIG)

Access: User read/write

**Figure 17-7. Interrupt Configuration Register (ICONFIG)****Table 17-9. ICONFIG Field Descriptions**

Field	Description
15–9 ELVLPRI	Enable core's priority elevation on priority levels. Each ELVLPRI[7:1] bit corresponds to the available priority levels 1 – 7. If set, the assertion of the corresponding level- $n$ request to the core causes the processor's bus master priority to be temporarily elevated in the device's crossbar switch arbitration logic. The processor's bus master arbitration priority remains elevated until the level- $n$ request is negated. If round-robin arbitration is enabled, this bit has no effect. If cleared, the assertion of a level- $n$ request does not affect the processor's bus master priority.
8–6	Reserved, must be cleared.
5 EMASK	If set, the interrupt controller automatically loads the level of an interrupt request into the CLMASK (current level mask) when the acknowledge is performed. At the exact same cycle, the value of the current interrupt level mask is saved in the SLMASK (saved level mask) register. This feature can be used to support software-managed nested interrupts, and is intended to complement the interrupt masking functions supported in the ColdFire processor. The value of SLMASK register should be read from the interrupt controller and saved in the interrupt stack frame in memory, and restored near the service routine's exit. If cleared, the INTC does not perform any automatic masking of interrupt levels. The state of this bit does not affect the ColdFire processor's interrupt masking logic in any manner.
4–0	Reserved, must be cleared.

## 17.2.5 Set Interrupt Mask Register (SIMR $n$ )

The SIMR $n$  register provides a simple mechanism to set a given bit in the IMR $n$  registers to mask the corresponding interrupt request. The value written to the SIMR field causes the corresponding bit in the IMR $n$  register to be set. The SIMR $n$ [SALL] bit provides a global set function, forcing the entire contents

of IMR $n$  to be set, thus masking all interrupts. Reads of this register return all zeroes. This register is provided so interrupt service routines can easily mask the given interrupt request without the need to perform a read-modify-write sequence on the IMR $n$  register.

Address:	0xFC04_801C (SIMR0)	Access:	User write-only					
	0xFC04_C01C (SIMR1)							
	0xFC05_001C (SIMR2)							
R	7 0	6 0	5 0	4 0	3 0	2 0	1 0	0 0
W	SALL							
Reset:	0	0	0	0	0	0	0	0

Figure 17-8. Set Interrupt Mask Register (SIMR $n$ )

Table 17-10. SIMR $n$  Field Descriptions

Field	Description
7	Reserved, must be cleared.
6 SALL	Set all bits in the IMR $n$ register, masking all interrupt requests. 0 Only set those bits specified in the SIMR field. 1 Set all bits in IMR $n$ register. The SIMR field is ignored.
5–0 SIMR	Set the corresponding bit in the IMR $n$ register, masking the interrupt request.

## 17.2.6 Clear Interrupt Mask Register (CIMR $n$ )

The CIMR $n$  register provides a simple mechanism to clear a given bit in the IMR $n$  registers to enable the corresponding interrupt request. The value written to the CIMR field causes the corresponding bit in the IMR $n$  register to be cleared. The CIMR $n$ [CALL] bit provides a global clear function, forcing the entire contents of IMR $n$  to be cleared, thus enabling all interrupts. Reads of this register return all zeroes. This register is provided so interrupt service routines can easily enable the given interrupt request without the need to perform a read-modify-write sequence on the IMR $n$  register.

In the event of a simultaneous write to the CIMR $n$  and SIMR $n$ , the SIMR $n$  has priority and the resulting function would be a set of the interrupt mask register.

Address:	0xFC04_801D (CIMR0)	Access:	User write-only					
	0xFC04_C01D (CIMR1)							
	0xFC05_001D (CIMR2)							
R	7 0	6 0	5 0	4 0	3 0	2 0	1 0	0 0
W	CALL							
Reset:	0	0	0	0	0	0	0	0

Figure 17-9. Clear Interrupt Mask Register (CIMR $n$ )

**Table 17-11. CIMR $n$  Field Descriptions**

Field	Description
7	Reserved, must be cleared.
6 CALL	Clear all bits in the IMR $n$ register, enabling all interrupt requests. 0 Only set those bits specified in the CIMR field. 1 Clear all bits in IMR $n$ register. The CIMR field is ignored.
5–0 CIMR	Clear the corresponding bit in the IMR $n$ register, enabling the interrupt request.

### 17.2.7 Current Level Mask Register (CLMASK)

The CLMASK register is provided so the interrupt controller can optionally automatically manage masking of interrupt requests based on the programmed priority level. If enabled by ICONFIG[EMASK] bit being set, an interrupt acknowledge read cycle returns a vector number identifying the physical request source, and the CLMASK register is loaded with the level number associated with the request. After the CLMASK register is updated, then all interrupt requests with level numbers equal to or less than this value are masked by the controller and are not allowed to cause the assertion of the interrupt signal to the processor core. As the CLMASK register is updated during the IACK cycle read, the former value is saved in the SLMASK register.

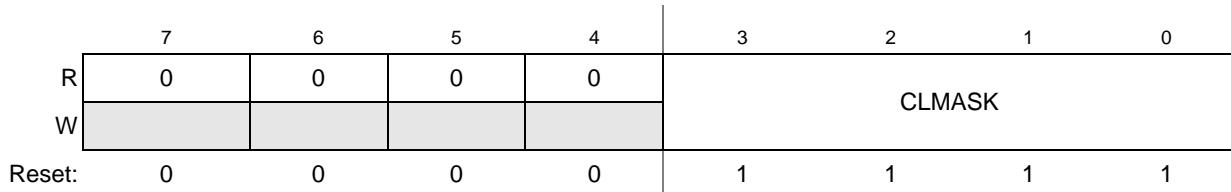
Typically, after a level- $n$  interrupt request is managed, the service routine restores the saved level mask value into the current level mask register to re-enable the lower priority requests. In addition, an interrupt service routine can explicitly load this register with a lower priority value to query for any pending interrupts via software interrupt acknowledge cycles.

#### NOTE

Only one copy of this register exists among the 3 interrupt controller modules. All reads and writes to this register must be made to the INTC0 memory space.

Address: 0xFC04\_801E (CLMASK)

Access: User read/write

**Figure 17-10. Current Level Mask Register (CLMASK)**

**Table 17-12. CLMASK Field Descriptions**

Field	Description
7–4	Reserved, must be cleared.
3–0 CLMASK	<p>Current level mask. Defines the level mask, where only interrupt levels greater than the current value are processed by the controller</p> <ul style="list-style-type: none"> <li>0000 Level 1 – 7 requests are processed.</li> <li>0001 Level 2 – 7 requests are processed.</li> <li>0010 Level 3 – 7 requests are processed.</li> <li>0011 Level 4 – 7 requests are processed.</li> <li>0100 Level 5 – 7 requests are processed.</li> <li>0101 Level 6 – 7 requests are processed.</li> <li>0110 Level 7 requests are processed.</li> <li>0111 All requests are masked.</li> <li>1000 – 1110 Reserved.</li> <li>1111 Level 1 – 7 requests are processed.</li> </ul>

### 17.2.8 Saved Level Mask Register (SLMASK)

The SLMASK register is provided so the interrupt controller can optionally automatically manage masking of interrupt requests based on the programmed priority level. If enabled by ICONFIG[EMASK] bit being set, an interrupt acknowledge read cycle returns a vector number identifying the physical request source, and the CLMASK register is loaded with the level number associated with the request. After the CLMASK register is updated, then all interrupt requests with level numbers equal to or less than this value are masked by the controller and are not allowed to cause the assertion of the interrupt signal to the processor core. As the CLMASK register is updated during the IACK cycle read, the former value is saved in the SLMASK register.

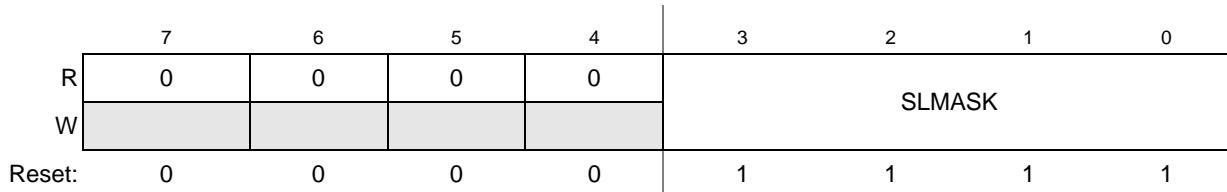
Typically, after a level-*n* interrupt request is managed, the service routine restores the saved level mask value into the current level mask register to re-enable the lower priority requests.

#### NOTE

Only one copy of this register exists among the three interrupt controller modules. All reads and writes to this register must be made to the INTC0 memory space.

Address: 0xFC04\_801F (SLMASK)

Access: User read/write

**Figure 17-11. Saved Level Mask Register (SLMASK)**

**Table 17-13. SLMASK Field Descriptions**

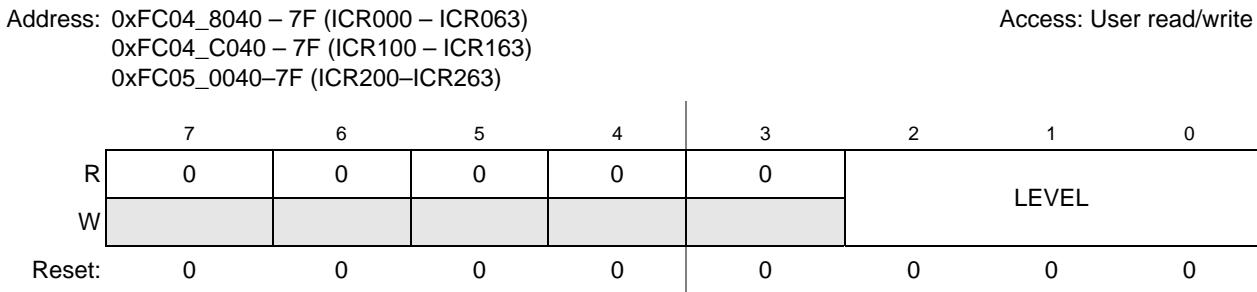
Field	Description
7–4	Reserved, must be cleared.
3–0 SLMASK	Saved level mask. Defines the saved level mask. See the CLMASK field definition for more information on the specific values.

### 17.2.9 Interrupt Control Register (ICR0n, ICR1n, ICR2n ( $n = 00, 01, 02, \dots, 63$ ))

Each ICR register specifies the interrupt level (1–7) for the corresponding interrupt source. These registers are cleared by reset and should be programmed with the appropriate levels before interrupts are enabled.

When multiple interrupt requests are programmed to the same level number, they are processed in a descending request number order. As an example, if requests 63, 62, 2, and 1 are programmed to a common level, request 63 is processed first, then request 62, then request 2, and finally request 1.

This definition allows software maximum flexibility in grouping interrupt request sources within any given priority level. The priority level in the ICRs directly corresponds to the interrupt level supported by the ColdFire processor.

**Figure 17-12. Interrupt Control Registers (ICR0n, ICR1n, ICR2n)****Table 17-14. ICRn Field Descriptions**

Field	Description
7–3	Reserved, must be cleared.
2–0 LEVEL	Interrupt level. Indicates the interrupt level assigned to each interrupt input. A level of 0 effectively disables the interrupt request, while a level 7 interrupt is given the highest priority. If interrupt masking is enabled (ICONFIG[EMASK] = 1), the acknowledgement of a level-n request forces the controller to automatically mask all interrupt requests of level-n and lower.

#### 17.2.9.1 Interrupt Sources

list the interrupt sources for each interrupt request line for INTC0, INTC1, and INTC2.

**Table 17-15. Interrupt Source Assignment For INTC0**

Source	Module	Flag	Source Description	Flag Clearing Mechanism
0			Not Used	
1	EPORT 0	EPFR0[EPF1]	Edge port 0 flag 1	Write 1 to EPF1
2		EPFR0[EPF2]	Edge port 0 flag 2	Write 1 to EPF2
3		EPFR0[EPF3]	Edge port 0 flag 3	Write 1 to EPF3
4		EPFR0[EPF4]	Edge port 0 flag 4	Write 1 to EPF4
5		EPFR0[EPF5]	Edge port 0 flag 5	Write 1 to EPF5
6		EPFR0[EPF6]	Edge port 0 flag 6	Write 1 to EPF6
7		EPFR0[EPF7]	Edge port 0 flag 7	Write 1 to EPF7
8	DMA	EDMA_INTR[INT00]	DMA Channel 0 transfer complete	Write EDMA_CINTR[CINT] = 0
9		EDMA_INTR[INT01]	DMA Channel 1 transfer complete	Write EDMA_CINTR[CINT] = 1
10		EDMA_INTR[INT02]	DMA Channel 2 transfer complete	Write EDMA_CINTR[CINT] = 2
11		EDMA_INTR[INT03]	DMA Channel 3 transfer complete	Write EDMA_CINTR[CINT] = 3
12		EDMA_INTR[INT04]	DMA Channel 4 transfer complete	Write EDMA_CINTR[CINT] = 4
13		EDMA_INTR[INT05]	DMA Channel 5 transfer complete	Write EDMA_CINTR[CINT] = 5
14		EDMA_INTR[INT06]	DMA Channel 6 transfer complete	Write EDMA_CINTR[CINT] = 6
15		EDMA_INTR[INT07]	DMA Channel 7 transfer complete	Write EDMA_CINTR[CINT] = 7
16		EDMA_INTR[INT08]	DMA Channel 8 transfer complete	Write EDMA_CINTR[CINT] = 8
17		EDMA_INTR[INT09]	DMA Channel 9 transfer complete	Write EDMA_CINTR[CINT] = 9
18		EDMA_INTR[INT10]	DMA Channel 10 transfer complete	Write EDMA_CINTR[CINT] = 10
19		EDMA_INTR[INT11]	DMA Channel 11 transfer complete	Write EDMA_CINTR[CINT] = 11
20		EDMA_INTR[INT12]	DMA Channel 12 transfer complete	Write EDMA_CINTR[CINT] = 12
21		EDMA_INTR[INT13]	DMA Channel 13 transfer complete	Write EDMA_CINTR[CINT] = 13
22		EDMA_INTR[INT14]	DMA Channel 14 transfer complete	Write EDMA_CINTR[CINT] = 14
23		EDMA_INTR[INT15]	DMA Channel 15 transfer complete	Write EDMA_CINTR[CINT] = 15
24		EDMA_ERR[ERRn]	DMA Error Interrupt	Write EDMA_CERR[CERR] = n
25	SCM	SCMIR[CWIC]	Core Watchdog Timeout	Write 1 to SCMISR[CWIC]
26	UART0	UISR0 register	UART0 Interrupt Request	Automatically cleared
27	UART1	UISR1 register	UART1 Interrupt Request	Automatically cleared
28	UART2	UISR2 register	UART2 Interrupt Request	Automatically cleared
29	UART3	UISR3 register	UART3 Interrupt Request	Automatically cleared
30	I <sup>2</sup> C0	I20SR[IIF]	I <sup>2</sup> C0 Interrupt	Write 0 to I20SR[IIF]
31	DSPI0	DSPI0_SR register	DSPI0 OR'd interrupt	Write 1 to appropriate DSPI0_SR bit
32	DTIM0	DTER0 register	Timer 0 interrupt	Write 1 to appropriate DTER0 bit

**Table 17-15. Interrupt Source Assignment For INTC0 (continued)**

<b>Source</b>	<b>Module</b>	<b>Flag</b>	<b>Source Description</b>	<b>Flag Clearing Mechanism</b>
33	DTIM1	DTER1 register	Timer 1 interrupt	Write 1 to appropriate DTER1 bit
34	DTIM2	DTER2 register	Timer 2 interrupt	Write 1 to appropriate DTER2 bit
35	DTIM3	DTER3 register	Timer 3 interrupt	Write 1 to appropriate DTER3 bit
36	MAC-NET0	ENET0_EIR[TXF]	Transmit frame interrupt	Write 1 to ENET0_EIR[TXF]
37		ENET0_EIR[TXB]	Transmit buffer interrupt	Write 1 to ENET0_EIR[TXB]
38		ENET0_EIR[UN]	Transmit FIFO underrun	Write 1 to ENET0_EIR[UN]
39		ENET0_EIR[RL]	Collision retry limit	Write 1 to ENET0_EIR[RL]
40		ENET0_EIR[RXF]	Receive frame interrupt	Write 1 to ENET0_EIR[RXF]
41		ENET0_EIR[RXB]	Receive buffer interrupt	Write 1 to ENET0_EIR[RXB]
42		ENET0_EIR[MII]	MII interrupt	Write 1 to ENET0_EIR[MII]
43		ENET0_EIR[LC]	Late collision	Write 1 to ENET0_EIR[LC]
44		Not used		
45		ENET0_EIR[GRA]	Graceful stop complete	Write 1 to ENET0_EIR[GRA]
46	MAC-NET1	ENET0_EIR[EBERR]	Ethernet bus error	Write 1 to ENET0_EIR[EBERR]
47		ENET0_EIR[BABT]	Babbling transmit error	Write 1 to ENET0_EIR[BABT]
48		ENET0_EIR[BABR]	Babbling receive error	Write 1 to ENET0_EIR[BABR]
49		ENET1_EIR[TXF]	Transmit frame interrupt	Write 1 to ENET1_EIR[TXF]
50		ENET1_EIR[TXB]	Transmit buffer interrupt	Write 1 to ENET1_EIR[TXB]
51		ENET1_EIR[UN]	Transmit FIFO underrun	Write 1 to ENET1_EIR[UN]
52		ENET1_EIR[RL]	Collision retry limit	Write 1 to ENET1_EIR[RL]
53		ENET1_EIR[RXF]	Receive frame interrupt	Write 1 to ENET1_EIR[RXF]
54		ENET1_EIR[RXB]	Receive buffer interrupt	Write 1 to ENET1_EIR[RXB]
55		ENET1_EIR[MII]	MII interrupt	Write 1 to ENET1_EIR[MII]
56		ENET1_EIR[LC]	Late collision	Write 1 to ENET1_EIR[LC]
57		Not used		
58	SCM	ENET1_EIR[GRA]	Graceful stop complete	Write 1 to ENET1_EIR[GRA]
59		ENET1_EIR[EBERR]	Ethernet bus error	Write 1 to ENET1_EIR[EBERR]
60		ENET1_EIR[BABT]	Babbling transmit error	Write 1 to ENET1_EIR[BABT]
61		ENET1_EIR[BABR]	Babbling receive error	Write 1 to ENET1_EIR[BABR]
62	SCM	SCMIR[CFEI]	Core bus error interrupt	Write 1 to SCMIR[CFEI]
63	1-Wire	OW_ISR	1-Wire interrupt	Reading OW_ISR

**Table 17-16. Interrupt Source Assignment for INTC1**

<b>Source</b>	<b>Module</b>	<b>Flag</b>	<b>Source Description</b>	<b>Flag Clearing Mechanism</b>
0	Flex CAN0	IFLAG0[BUF <sub>n</sub> l]	Logical OR of CAN0's MB requests	Write 1 to IFLAG0[BUF <sub>n</sub> l] after reading as 1
1		ERRSTAT0 [BOFFINT]	Bus-off interrupt	Write 1 to ERRSTAT1[BOFFINT]
2		Not used		
3		ERRSTAT0[TXWRN or RXWRN]	Error interrupt	Write 1 to ERRSTAT0[TXWRN or RXWRN]
4	Flex CAN1	IFLAG1[BUF <sub>n</sub> l]	Logical OR of CAN1's MB requests	Write 1 to IFLAG1[BUF <sub>n</sub> l] after reading as 1
5		ERRSTAT1 [BOFFINT]	Bus-off interrupt	Write 1 to ERRSTAT1[BOFFINT]
6		Not used		
7		ERRSTAT1[TXWRN or RXWRN]	Error interrupt	Write 1 to ERRSTAT1[TXWRN or RXWRN]
8	DMA 16–31	EDMA_INTR[INT16]	DMA Channel 16 transfer complete	Write EDMA_CINTR[CINT] = 16
9		EDMA_INTR[INT17]	DMA Channel 17 transfer complete	Write EDMA_CINTR[CINT] = 17
10		EDMA_INTR[INT18]	DMA Channel 18 transfer complete	Write EDMA_CINTR[CINT] = 18
11		EDMA_INTR[INT19]	DMA Channel 19 transfer complete	Write EDMA_CINTR[CINT] = 19
12		EDMA_INTR[INT20]	DMA Channel 20 transfer complete	Write EDMA_CINTR[CINT] = 20
13		EDMA_INTR[INT21]	DMA Channel 21 transfer complete	Write EDMA_CINTR[CINT] = 21
14		EDMA_INTR[INT22]	DMA Channel 22 transfer complete	Write EDMA_CINTR[CINT] = 22
15		EDMA_INTR[INT23]	DMA Channel 23 transfer complete	Write EDMA_CINTR[CINT] = 23
16		EDMA_INTR[INT24]	DMA Channel 24 transfer complete	Write EDMA_CINTR[CINT] = 24
17		EDMA_INTR[INT25]	DMA Channel 25 transfer complete	Write EDMA_CINTR[CINT] = 25
18		EDMA_INTR[INT26]	DMA Channel 26 transfer complete	Write EDMA_CINTR[CINT] = 26
19		EDMA_INTR[INT27]	DMA Channel 27 transfer complete	Write EDMA_CINTR[CINT] = 27
20		EDMA_INTR[INT28]	DMA Channel 28 transfer complete	Write EDMA_CINTR[CINT] = 28
21		EDMA_INTR[INT29]	DMA Channel 29 transfer complete	Write EDMA_CINTR[CINT] = 29
22		EDMA_INTR[INT30]	DMA Channel 30 transfer complete	Write EDMA_CINTR[CINT] = 30
23		EDMA_INTR[INT31]	DMA Channel 31 transfer complete	Write EDMA_CINTR[CINT] = 31

**Table 17-16. Interrupt Source Assignment for INTC1 (continued)**

<b>Source</b>	<b>Module</b>	<b>Flag</b>	<b>Source Description</b>	<b>Flag Clearing Mechanism</b>
24	DMA 32–47	EDMA_INTR[INT32]	DMA Channel 32 transfer complete	Write EDMA_CINTR[CINT] = 32
25		EDMA_INTR[INT33]	DMA Channel 33 transfer complete	Write EDMA_CINTR[CINT] = 33
26		EDMA_INTR[INT34]	DMA Channel 34 transfer complete	Write EDMA_CINTR[CINT] = 34
27		EDMA_INTR[INT35]	DMA Channel 35 transfer complete	Write EDMA_CINTR[CINT] = 35
28		EDMA_INTR[INT36]	DMA Channel 36 transfer complete	Write EDMA_CINTR[CINT] = 36
29		EDMA_INTR[INT37]	DMA Channel 37 transfer complete	Write EDMA_CINTR[CINT] = 37
30		EDMA_INTR[INT38]	DMA Channel 38 transfer complete	Write EDMA_CINTR[CINT] = 38
31		EDMA_INTR[INT39]	DMA Channel 39 transfer complete	Write EDMA_CINTR[CINT] = 39
32		EDMA_INTR[INT40]	DMA Channel 40 transfer complete	Write EDMA_CINTR[CINT] = 40
33		EDMA_INTR[INT41]	DMA Channel 41 transfer complete	Write EDMA_CINTR[CINT] = 41
34		EDMA_INTR[INT42]	DMA Channel 42 transfer complete	Write EDMA_CINTR[CINT] = 42
35		EDMA_INTR[INT43]	DMA Channel 43 transfer complete	Write EDMA_CINTR[CINT] = 43
36		EDMA_INTR[INT44]	DMA Channel 44 transfer complete	Write EDMA_CINTR[CINT] = 44
37		EDMA_INTR[INT45]	DMA Channel 45 transfer complete	Write EDMA_CINTR[CINT] = 45
38		EDMA_INTR[INT46]	DMA Channel 46 transfer complete	Write EDMA_CINTR[CINT] = 46
39		EDMA_INTR[INT47]	DMA Channel 47 transfer complete	Write EDMA_CINTR[CINT] = 47
40	DMA 48–55	EDMA_INTR[INT48]	DMA Channel 48 transfer complete	Write EDMA_CINTR[CINT] = 48
41		EDMA_INTR[INT49]	DMA Channel 49 transfer complete	Write EDMA_CINTR[CINT] = 49
42		EDMA_INTR[INT50]	DMA Channel 50 transfer complete	Write EDMA_CINTR[CINT] = 50
43		EDMA_INTR[INT51]	DMA Channel 51 transfer complete	Write EDMA_CINTR[CINT] = 51
44		EDMA_INTR[INT52]	DMA Channel 52 transfer complete	Write EDMA_CINTR[CINT] = 52
45		EDMA_INTR[INT53]	DMA Channel 53 transfer complete	Write EDMA_CINTR[CINT] = 53
46		EDMA_INTR[INT54]	DMA Channel 54 transfer complete	Write EDMA_CINTR[CINT] = 54
47		EDMA_INTR[INT55]	DMA Channel 55 transfer complete	Write EDMA_CINTR[CINT] = 55
48	UART4	UISR4 register	UART4 Interrupt Request	Automatically cleared
49	UART5	UISR5 register	UART5 Interrupt Request	Automatically cleared
50	UART6	UISR6 register	UART6 Interrupt Request	Automatically cleared
51	UART7	UISR7 register	UART7 Interrupt Request	Automatically cleared
52	UART8	UISR8 register	UART8 Interrupt Request	Automatically cleared
53	UART9	UISR9 register	UART9 Interrupt Request	Automatically cleared
54	DSPI1	DSPI1_SR register	DSPI1 OR'd interrupt	Write 1 to appropriate DSPI1_SR bit
55	DSPI2	DSPI2_SR register	DSPI2 OR'd interrupt	Write 1 to appropriate DSPI2_SR bit
56	DSPI3	DSPI3_SR register	DSPI3 OR'd interrupt	Write 1 to appropriate DSPI3_SR bit
57	I <sup>2</sup> C1	I2CSR[IIF]	I <sup>2</sup> C1 Interrupt	Write 0 to I2CSR[IIF]

**Table 17-16. Interrupt Source Assignment for INTC1 (continued)**

<b>Source</b>	<b>Module</b>	<b>Flag</b>	<b>Source Description</b>	<b>Flag Clearing Mechanism</b>
58	I <sup>2</sup> C2	I22SR[IIF]	I <sup>2</sup> C2 Interrupt	Write 0 to I22SR[IIF]
59	I <sup>2</sup> C3	I23SR[IIF]	I <sup>2</sup> C3 Interrupt	Write 0 to I23SR[IIF]
60	I <sup>2</sup> C4	I24SR[IIF]	I <sup>2</sup> C4 Interrupt	Write 0 to I24SR[IIF]
61	I <sup>2</sup> C5	I25SR[IIF]	I <sup>2</sup> C5 Interrupt	Write 0 to I25SR[IIF]
62			Not used	
63			Not used	

**Table 17-17. Interrupt Source Assignment for INTC2**

<b>Source</b>	<b>Module</b>	<b>Flag</b>	<b>Source Description</b>	<b>Flag Clearing Mechanism</b>
0	DMA 56–63	EDMA_INTR [INT56–63]	DMA Channel 55–63 OR'd transfer complete	Write EDMA_CINTR[CINT] = 56–63 to clear appropriate channel
1	mcPWM	PWM_SM0SR[CFxn]	Submodule 0 OR'd input capture	Write 1 to appropriate PWM_SM0SR[CFxn]
2		PWM_SM1SR[CFxn]	Submodule 1 OR'd input capture	Write 1 to appropriate PWM_SM1SR[CFxn]
3		PWM_SM2SR[CFxn]	Submodule 2 OR'd input capture	Write 1 to appropriate PWM_SM2SR[CFxn]
4		PWM_SM3SR[CFxn]	Submodule 3 OR'd input capture	Write 1 to appropriate PWM_SM3SR[CFxn]
5		PWM_SM0SR[RF]	Submodule 0 reload	Write 1 to PWM_SM0SR[RF]
6		PWM_SM1SR[RF]	Submodule 1 reload	Write 1 to PWM_SM1SR[RF]
7		PWM_SM2SR[RF]	Submodule 2 reload	Write 1 to PWM_SM2SR[RF]
8		PWM_SM3SR[RF]	Submodule 3 reload	Write 1 to PWM_SM3SR[RF]
9		PWM_FSR[FFLAG]	OR'd fault input interrupt	Write 1 to appropriate PWM_FSR[FFLAG]
10		PWM_SMnSR[REF]	OR'd reload error interrupt	Write 1 to appropriate PWM_SMnSR[REF]
11	PLL	PLL_SR[LOCF]	Loss of clock interrupt	Write 1 to LOCF
12		PLL_SR[LOLF]	Loss of lock interrupt	Write 1 to LOLF
13	PIT0	PCSR0[PIF]	PIT interrupt flag	Write 1 to PIF or write PMR
14	PIT1	PCSR1[PIF]	PIT interrupt flag	Write 1 to PIF or write PMR
15	PIT2	PCSR2[PIF]	PIT interrupt flag	Write 1 to PIF or write PMR
16	PIT3	PCSR3[PIF]	PIT interrupt flag	Write 1 to PIF or write PMR
17	USB OTG	USB_STS	USB OTG interrupt	Write 1 to corresponding bit in the USB_STS
18	USB Host	USB_STS	USB host interrupt	Write 1 to corresponding bit in the USB_STS
19	mcPWM	PWM_SM0SR[CMPF]	Submodule 0 compare	Write 1 to appropriate PWM_SM0SR[CMPF]
20		PWM_SM1SR[CMPF]	Submodule 1 compare	Write 1 to appropriate PWM_SM1SR[CMPF]
21		PWM_SM2SR[CMPF]	Submodule 2 compare	Write 1 to appropriate PWM_SM2SR[CMPF]
22		PWM_SM3SR[CMPF]	Submodule 3 compare	Write 1 to appropriate PWM_SM3SR[CMPF]

**Table 17-17. Interrupt Source Assignment for INTC2 (continued)**

<b>Source</b>	<b>Module</b>	<b>Flag</b>	<b>Source Description</b>	<b>Flag Clearing Mechanism</b>
23	SSI0	SSI0_ISR	SSI0 interrupt	Various, see chapter for details.
24	SSI1	SSI1_ISR	SSI1 interrupt	Various, see chapter for details.
25	Flash	NFC_ISR	NAND flash controller interrupt	Write 1 to appropriate bit in NFC_ISR
26	Robust RTC	RTC_ISR	RTC interrupt	Write 1 to corresponding bit in the RTC_ISR
27	CCM	UOCSR	USB status Interrupt	Read UOCSR.
28	RNG	EI	RNG interrupt flag	Write 1 to RNGCR[CI]
29	SIM	SIM_TSR or SIM_RSR	SIM data interrupt (SIM_TSR)	Various, see chapter for details
30			SIM general interrupt (SIM_RSR)	Various, see chapter for details
31	SDHC	IRQSTAT	SDHC OR'd interrupt	Write 1 to corresponding bit in IRQSTAT
32	ADC	ADC_SR[EOSI0]	ADC subconverter A done	Write 1 to ADC_SR[EOSI0]
33		ADC_SR[EOSI1]	ADC subconverter B done	Write 1 to ADC_SR[EOSI1]
34		ADC_LSR or ADC_ZCSR	Zero-crossing/limit-crossed	Write 1 to appropriate bit in ADC_LSR or ADC_ZCSR
35			Not used	
36	SDRAM	DDR_CR27 [INTSTATUS]	SDRAM OR'd interrupt	Write 1 to corresponding bit in DDR_CR25[INTACK]
37			Not used	
38	L2 Ethernet Switch	ESW_ISR[EBERR]	Bus error interrupt	Write 1 to ESW_ISR[EBERR]
39		ESW_ISR[RXB]	Receive buffer interrupt	Write 1 to ESW_ISR[RXB]
40		ESW_ISR[RXF]	Receive frame interrupt	Write 1 to ESW_ISR[RXF]
41		ESW_ISR[TXB]	Transmit buffer interrupt	Write 1 to ESW_ISR[TXB]
42		ESW_ISR[TXF]	Transmit frame interrupt	Write 1 to ESW_ISR[TXF]
43		ESW_ISR[QM]	Low amount of free memory	Write 1 to ESW_ISR[QM]
44		ESW_ISR[OD0]	Port 0 output discard	Write 1 to ESW_ISR[OD0]
45		ESW_ISR[OD1]	Port 1 output discard	Write 1 to ESW_ISR[OD1]
46		ESW_ISR[OD2]	Port 2 output discard	Write 1 to ESW_ISR[OD2]
47		ESW_ISR[LRN]	Learning record available	Write 1 to ESW_ISR[LRN]
48	MAC-NET0	ENET0_EIR [TS_AVAIL]	Timestamp available	Write 1 to ENET0_EIR[TS_AVAIL]
49		ENET0_EIR [WAKEUP]	Wake from sleep	Write 1 to ENET0_EIR[WAKEUP]
50		ENET0_EIR[PLR]	Payload receive error	Write 1 to ENET0_EIR[PLR]
51–54			Not used	

**Table 17-17. Interrupt Source Assignment for INTC2 (continued)**

Source	Module	Flag	Source Description	Flag Clearing Mechanism
55	MAC-NET1	ENET1_EIR [TS_AVAIL]	Timestamp available	Write 1 to ENET1_EIR[TS_AVAIL]
56		ENET1_EIR [WAKEUP]	Wake from sleep	Write 1 to ENET1_EIR[WAKEUP]
57		ENET1_EIR[PLR]	Payload receive error	Write 1 to ENET1_EIR[PLR]
58–63		Not used		

### 17.2.10 Software and Level 1–7 IACK Registers (**SWIACK $n$ , L $n$ IACK $n$ –L7IACK $n$** )

The eight IACK registers (per interrupt controller) can be explicitly addressed via the CPU, or implicitly addressed via a processor-generated interrupt acknowledge cycle during exception processing. In either case, the interrupt controller’s actions are very similar.

First, consider an IACK cycle to a specific level: a level- $n$  IACK. When this type of IACK arrives in the interrupt controller, the controller examines all the currently-active level  $n$  interrupt requests, determines the highest priority within the level, and then responds with the unique vector number corresponding to that specific interrupt source. The vector number is supplied as the data for the byte-sized IACK read cycle. In addition to providing the vector number, the interrupt controller also loads the level into the CLMASK register, where it may be retrieved later.

This interrupt controller design also supports the concept of a software IACK. A software IACK allows an interrupt service routine to determine if there are other pending interrupts so that the overhead associated with interrupt exception processing (including machine state save/restore functions) can be minimized. In general, the software IACK is performed near the end of an interrupt service routine, and if there are additional active interrupt sources, the current interrupt service routine (ISR) passes control to the appropriate service routine, but without taking another interrupt exception.

When the interrupt controller receives a software IACK read, it returns the vector number associated with the highest unmasked interrupt source for that interrupt controller. If there are no active sources, the interrupt controller returns an all-zero vector as the operand for the SWIACK register. A read from the L $n$ IACK registers when there are no active requests returns a value of 24 (0x18), signaling a spurious interrupt.

In addition to the software IACK registers in each interrupt controller, there are global software IACK registers. A read from the global SWIACK (GSWIACK) returns the vector number for the highest level and priority unmasked interrupt source from all interrupt controllers. A read from one of the global L $n$ IACK (GL $n$ IACK) registers returns the vector for the highest priority unmasked interrupt within a level for all interrupt controllers.

Address: 0xFC04_80E0 (SWIACK0)	Access: User read-only																										
0xFC04_80E0+4n (LnIACK0) n=1:7																											
0xFC04_C0E0 (SWIACK1)																											
0xFC04_C0E0+4n (LnIACK1) n=1:7																											
0xFC05_40E0 (GSWIACK)																											
0xFC05_40E0+4n (GLnIACK) n=1:7																											
<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 10%;">R</th> <th style="width: 10%;">7</th> <th style="width: 10%;">6</th> <th style="width: 10%;">5</th> <th style="width: 10%;">4</th> <th colspan="4" style="width: 40%;">VECTOR</th> <th style="width: 10%;">3</th> <th style="width: 10%;">2</th> <th style="width: 10%;">1</th> <th style="width: 10%;">0</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">W</td> <td></td> </tr> </tbody> </table>		R	7	6	5	4	VECTOR				3	2	1	0	W												
R	7	6	5	4	VECTOR				3	2	1	0															
W																											
Reset (SWIACKn):	0	0	0	0	0	0	0	0	0	0	0	0															
Reset (LnIACKn):	0	0	0	1	1	0	0	0	0	0	0	0															

Figure 17-13. Software and Level  $n$  IACK Registers (SWIACK $n$ , L1IACK $n$  – L7IACK $n$ )Table 17-18. SWIACK $n$  and LxIACK $n$  Field Descriptions

Field	Description
7–0 VECTOR	Vector number. A read from the SWIACK register returns the vector number associated with the highest priority pending interrupt source. A read from one of the LnIACK registers returns the highest priority unmasked interrupt source within the level. A write to any IACK register causes an error termination.

## 17.3 Functional Description

### 17.3.1 Interrupt Controller Theory of Operation

To support the interrupt architecture of the 68K/ColdFire programming model, the 64 interrupt sources are organized as 7 levels, with an arbitrary number of requests programmed to each level. The priority structure within a single interrupt level depends on the interrupt source number assignments (see [Section 17.2.9.1, “Interrupt Sources”](#)). The higher numbered interrupt source has priority over the lower numbered interrupt source. See the below table for an example.

Table 17-19. Example Interrupt Priority Within a Level

Interrupt Source	ICR[2:0]	Priority
40	011	Highest
22	011	
8	011	
2	011	Lowest

The level is fully programmable for all sources. The 3-bit level is defined in the interrupt control register (ICR0 $n$ , ICR1 $n$ , ICR2 $n$ ).

The operation of the interrupt controller can be broadly partitioned into three activities:

- Recognition
- Prioritization
- Vector determination during IACK

### 17.3.1.1 Interrupt Recognition

The interrupt controller continuously examines the request sources ( $IPR_n$ ) and the interrupt mask register ( $IMR_n$ ) to determine if there are active requests. This is the recognition phase. The interrupt force register ( $INTFRC_n$ ) also factors into the generation of an active request.

### 17.3.1.2 Interrupt Prioritization

As an active request is detected, it is translated into the programmed interrupt level. Next, the appropriate level masking is performed if this feature is enabled. The level of the active request must be greater than the current mask level before it is signaled in the processor. The resulting unmasked decoded priority level is driven out of the interrupt controller. The decoded priority levels from the interrupt controllers are logically summed together, and the highest enabled interrupt request is sent to the processor core during this prioritization phase.

### 17.3.1.3 Interrupt Vector Determination

After the core has sampled for pending interrupts and begun interrupt exception processing, it generates an interrupt acknowledge cycle (IACK). The IACK transfer is treated as a memory-mapped byte read by the processor, and routed to the appropriate interrupt controller. Next, the interrupt controller extracts the level being acknowledged from address bits[4:2], and then determines the highest unmasked level for the type of interrupt being acknowledged, and returns the 8-bit interrupt vector for that request to complete the cycle. The 8-bit interrupt vector is formed using the following algorithm:

```
For INTC0,           vector_number = 64 + interrupt source number
For INTC1,           vector_number = 128 + interrupt source number
For INTC2,           vector_number = 192 + interrupt source number
```

Recall vector\_numbers 0-63 are reserved for the ColdFire processor and its internal exceptions. Thus, the following mapping of bit positions to vector numbers applies for INTC0:

```
if interrupt source 0 is active and acknowledged, then vector_number = 64
if interrupt source 1 is active and acknowledged, then vector_number = 65
if interrupt source 2 is active and acknowledged, then vector_number = 66
...
if interrupt source 63 is active and acknowledged, then vector_number = 127
```

The net effect is a fixed mapping between the bit position within the source to the actual interrupt vector number.

If there is no active interrupt source for the given level, a special spurious interrupt vector (vector\_number equals 24) is returned and it is the responsibility of the service routine to manage this error situation.

This protocol implies the interrupting peripheral is not accessed during the acknowledge cycle because the interrupt controller completely services the acknowledge. This means the interrupt source must be

explicitly disabled in the interrupt service routine. This design provides unique vector capability for all interrupt requests, regardless of the complexity of the peripheral device.

In some applications, it is expected that the hardware masking of interrupt levels by the interrupt controller is enabled. This masking capability can be used with the processor's masking logic to form a dual-mask capability. In this operation mode, the IACK read cycle also causes the current interrupt level mask to be saved in the SLMASK register, and the new level being acknowledged loaded into the CLMASK register. This operation then automatically masks the new level (and all lower levels) while in the service routine. Generally, as the service routine completes execution, and the initiating request source has been negated, the saved mask level is restored into the current mask level to re-enable the lower priority levels.

Finally, the vector number returned during the IACK cycle provides the association with the request and the physical interrupt signal. The CLMASK and SLMASK registers are all loaded (if properly enabled) during the interrupt acknowledge read cycle.

### **17.3.2 Prioritization Between Interrupt Controllers**

The interrupt controllers have a fixed priority, where INTC0 has the highest priority, and INTC2 has the lowest priority. If all interrupt controllers have active interrupts at the same level, then the INTC0 interrupt is serviced first. If INTC1 has an active interrupt with a higher level than the highest INTC0 interrupt, then the INTC1 interrupt is serviced first.

### **17.3.3 Low-Power Wake-up Operation**

The system control module (SCM) contains an 8-bit low-power control register (LPCR) to control the low-power stop mode. This register must be explicitly programmed by software to enter low-power mode. It also contains a wake-up control register (WCR) sets the priority level of the interrupt necessary to bring the device out of the specified low-power mode. Refer to [Chapter 9, “Power Management,”](#) for definitions of the LPCR and WCR registers, as well as more information on low-power modes.

Each interrupt controller provides a special combinatorial logic path to provide a special wake-up signal to exit from the low-power stop mode. This special mode of operation works as follows:

1. The WCR register is programmed, setting the ENBWCR bit and the desired interrupt priority level.
2. At the appropriate time, the processor executes the privileged STOP instruction. After the processor has stopped execution, it asserts a specific processor status (PST) encoding. Issuing the STOP instruction when the WCR[ENBWCR] bit is set causes the SCM to enter the mode specified in LPCR[LPMD].
3. The entry into a low-power mode is processed by the low-power mode control logic, and the appropriate clocks (usually those related to the high-speed processor core) are disabled.
4. After entering the low-power mode, the interrupt controller enables a combinational logic path which evaluates any unmasked interrupt requests. The device waits for an event to generate a level 7 interrupt request or an interrupt request with a priority level greater than the value programmed in WCR[PRILVL].
5. After an appropriately high interrupt request level arrives, the interrupt controller signals its presence, and the SCM responds by asserting the request to exit low-power mode.

6. The low-power mode control logic senses the request signal and re-enables the appropriate clocks.
7. With the processor clocks enabled, the core processes the pending interrupt request.

For more information, see [Section 9.2.1, “Wake-up Control Register \(WCR\)”](#).

## 17.4 Initialization/Application Information

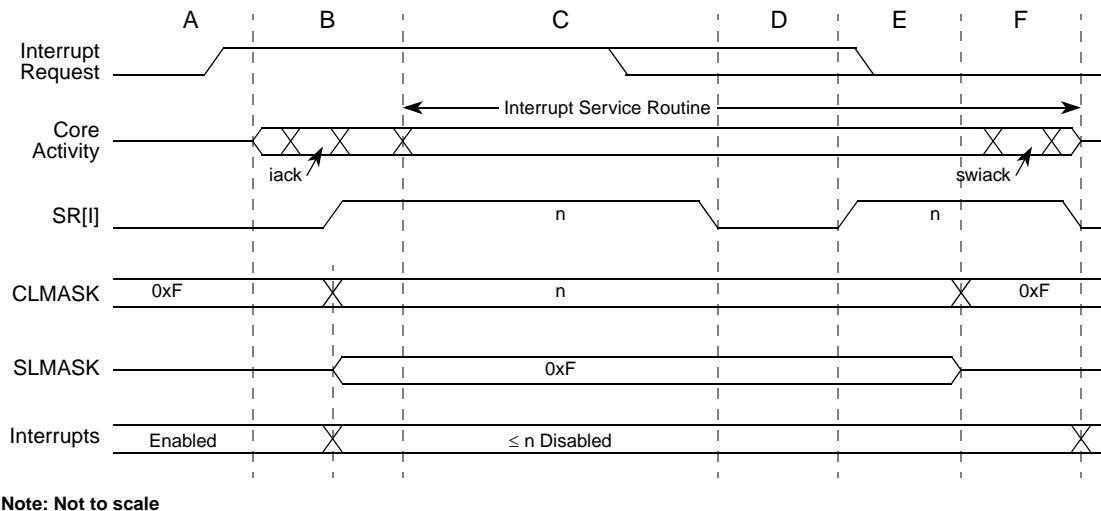
The interrupt controller’s reset state has all requests masked via the IMR. Before any interrupt requests are enabled, the following steps must be taken:

1. Set the ICONFIG register to the desired system configuration.
2. Program the  $ICR_n$  registers with the appropriate interrupt levels.
3. The reset value for the level mask registers (CLMASK and SLMASK) is 0xF (no levels masked). Typically, these registers do not need to be modified before interrupts are enabled.
4. Load the appropriate interrupt vector tables and interrupt service routines into memory.
5. Enable the interrupt requests, by clearing the appropriate bits in the IMR and lowering the interrupt mask level in the core’s status register ( $SR[I]$ ) to an appropriate level.

### 17.4.1 Interrupt Service Routines

This section focuses on the interaction of the interrupt masking functionality with the service routine.

[Figure 17-14](#) presents a timing diagram showing various phases during the execution of an interrupt service routine with the controller level masking functionality enabled. The time scale in this diagram is not meant to be accurate.



**Figure 17-14. Interrupt Service Routine and Masking**

Consider the events depicted in each segment (A – F) of the above diagram.

In A, an interrupt request is asserted, which is then signalled to the core.

As B begins, the interrupt request is recognized, and the core begins interrupt exception processing. During the core’s exception processing, the IACK cycle performs and the interrupt controller returns the

appropriate vector number. As the interrupt acknowledge read performs, the vector number returns to the core. The contents of the CLMASK register load into the SLMASK register, and the CLMASK register updates to the level of the acknowledge interrupt. Additionally, the processor raises the interrupt mask in the status register (SR[I]) to match the level of the acknowledged request. At the end of the core's exception processing, control passes to the interrupt service routine (ISR), shown as the beginning of segment C.

During C, the initial portion of the ISR executes. Near the end of this segment, the ISR accesses the peripheral to negate the interrupt request source. At the conclusion of segment C, the SR[I] field can be lowered to re-enable interrupts with a priority greater than the original request.

The bulk of the interrupt service routine executes in segment D, with interrupts enabled. Near the end of the service routine, the SR[I] field is again raised to the original acknowledged level, preparing to perform the context switch.

At the end of segment E, the original value in the saved level mask (SLMASK) is restored in the current level mask (CLMASK). Optionally, the service routine can directly load the CLMASK register with any value with pending interrupt requests of certain levels need to be examined.

In segment F, the interrupt service routine completes execution. During this period of time, it is possible to access the interrupt controller with a software IACK to see if there are any pending properly-enabled requests. Checking for any pending interrupt requests at this time provides ability to initiate processing of another interrupt without the need to return from the original and incur the overhead of another interrupt exception.

At the conclusion of segment G, the processor core returns to the original interrupted task or a different task ready to execute.

Obviously, there are many variations to the managing of the SR[I] and the CLMASK values to create a flexible, responsive system for managing interrupt requests within the device.

# Chapter 18

## Edge Port Module (EPORT)

### 18.1 Introduction

The edge port module (EPORT) has up to eight interrupt pins,  $\overline{\text{IRQ}7}$  –  $\overline{\text{IRQ}0}$ . Each pin can be configured individually as a level-sensitive interrupt pin or an edge-detecting interrupt pin (rising edge, falling edge, or both).

#### NOTE

Not all EPORT signals may be output from the device. See [Chapter 2, “Signal Descriptions,”](#) to determine which signals are available.

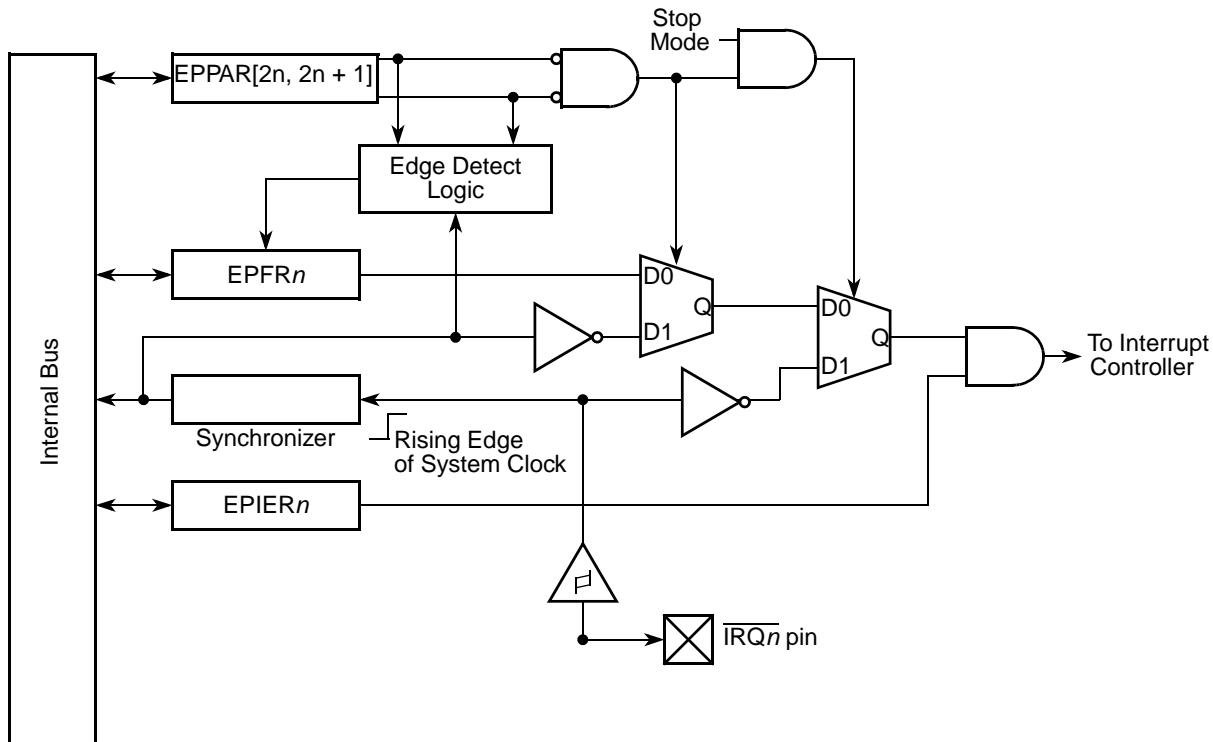


Figure 18-1. EPORT Block Diagram

#### NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to [Chapter 15, “Pin-Multiplexing and Control”](#)) prior to configuring the edge-port module.

## 18.2 Low-Power Mode Operation

This section describes the operation of the EPORT module in low-power modes. For more information on low-power modes, see [Chapter 9, “Power Management”](#). Table 18-1 shows EPORT-module operation in low-power modes and describes how this module may exit each mode.

### NOTE

The wakeup control register (WCR) in the system control module specifies the interrupt level at or above what is needed to bring the device out of a low-power mode.

**Table 18-1. Edge Port Module Operation in Low-Power Modes**

Low-power Mode	EPORT Operation	Mode Exit
Wait	Normal	Any $\overline{IRQ_n}$ interrupt at or above level in WCR
Doze	Normal	Any $\overline{IRQ_n}$ interrupt at or above level in WCR
Stop	Level-sensing only	Any $\overline{IRQ_n}$ interrupt set for level-sensing at or above level in WCR. See note below.

In wait and doze modes, the EPORT module continues to operate as it does in run mode. It may be configured to exit the low-power modes by generating an interrupt request on a selected edge or a low level on an external pin. In stop mode, no clocks are available to perform the edge-detect function. Only the level-detect logic is active (if configured) to allow any low level on the external interrupt pin to generate an interrupt (if enabled) to exit stop mode.

### NOTE

In stop mode, the input pin synchronizer is bypassed for the level-detect logic because no clocks are available.

## 18.3 Signal Descriptions

The values used in the edge/level detect logic are synchronized to the rising edge of FB\_CLK. These pins use Schmitt-triggered input buffers with built-in hysteresis designed to decrease the probability of generating false, edge-triggered interrupts for slow rising and falling input signals.

## 18.4 Memory Map/Register Definition

This subsection describes the memory map and register structure. Refer to [Table 18-2](#) for a description of the EPORT memory map.

### NOTE

Longword accesses to any of the edge-port registers result in a bus error. Only byte and word accesses are allowed.

**Table 18-2. Edge Port Module Memory Map**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
<b>Supervisor Access Only Registers<sup>1</sup></b>					
0xFC09_0000	EPORT Pin Assignment Register (EPPAR)	16	R/W	0x0000	<a href="#">18.4.1/18-3</a>
0xFC09_0003	EPORT Interrupt Enable Register (EPIER)	8	R/W	0x00	<a href="#">18.4.2/18-4</a>
<b>Supervisor/User Access Registers</b>					
0xFC09_0006	EPORT Flag Register (EPFR)	8	R/W	0x00	<a href="#">18.4.3/18-4</a>

<sup>1</sup> User access to supervisor-only address locations have no effect and result in a bus error.

### 18.4.1 EPORT Pin Assignment Register (EPPAR)

The EPORT pin assignment register (EPPAR) controls the function of each pin individually.

Address: 0xFC09_0000 (EPPAR)								Access: Supervisor read/write									
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	W	EPPA7	EPPA6	EPPA5	EPPA4	EPPA3	EPPA2	EPPA1	EPPA0								
Reset		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

**Figure 18-2. EPORT Pin Assignment Register (EPPAR)****Table 18-3. EPPAR Field Descriptions**

Field	Description
15–0 EPPAn	EPORT pin assignment select fields. The read/write EPPAn fields configure EPORT pins for level detection and rising and/or falling edge detection. Pins configured as level-sensitive are active-low (logic 0 on the external pin represents a valid interrupt request). Level-sensitive interrupt inputs are not latched. To guarantee that a level-sensitive interrupt request is acknowledged, the interrupt source must keep the signal asserted until acknowledged by software. Level sensitivity must be selected to bring the device out of stop mode with an $\overline{IRQ}_n$ interrupt. Pins configured as edge-triggered are latched and need not remain asserted for interrupt generation. Interrupt requests generated in the EPORT module can be masked by the interrupt controller module. Reset clears the EPPAn fields. 00 Pin $\overline{IRQ}_n$ level-sensitive 01 Pin $\overline{IRQ}_n$ rising edge triggered 10 Pin $\overline{IRQ}_n$ falling edge triggered 11 Pin $\overline{IRQ}_n$ falling edge and rising edge triggered

## 18.4.2 Edge Port Interrupt Enable Register (EPIER)

The EPORT interrupt enable register (EPIER) enables interrupt requests for each pin individually.

Address: 0xFC09_0003 (EPIER)								Access: User read/write							
	7	6	5	4		3	2	1	0						
R	EPIE7	EPIE6	EPIE5	EPIE4		EPIE3	EPIE2	EPIE1	EPIE0						
W	0	0	0	0		0	0	0	0						

Figure 18-3. EPORT Port Interrupt Enable Register (EPIER)

Table 18-4. EPIER Field Descriptions

Field	Description
7–0 EPIEn	Edge port interrupt enable bits enable EPORT interrupt requests. If a bit in EPIER is set, EPORT generates an interrupt request when: <ul style="list-style-type: none"> <li>The corresponding bit in the EPORT flag register (EPFR) is set or later becomes set</li> <li>The corresponding pin level is low and the pin is configured for level-sensitive operation</li> </ul> Clearing a bit in EPIER negates any interrupt request from the corresponding EPORT pin. Reset clears EPIE7–EPIE0. 0 Interrupt requests from corresponding EPORT pin disabled 1 Interrupt requests from corresponding EPORT pin enabled

## 18.4.3 Edge Port Flag Register (EPFR)

The EPORT flag register (EPFR) individually latches EPORT edge events.

Address: 0xFC09_0006 (EPFR)								Access: User read/write							
	7	6	5	4		3	2	1	0						
R	EPF7	EPF6	EPF5	EPF4		EPF3	EPF2	EPF1	EPF0						
W	w1c	w1c	w1c	w1c		w1c	w1c	w1c	w1c						

Figure 18-4. EPORT Port Flag Register (EPFR)

Table 18-5. EPFR Field Descriptions

Field	Description
7–0 EPFn	Edge port flag bits. When an EPORT pin is configured for edge triggering, its corresponding read/write bit in EPFR indicates that the selected edge has been detected. Reset clears EPF7 – EPF0. Bits in this register are set when the selected edge is detected on the corresponding pin. A bit remains set until cleared by writing a 1 to it. Writing 0 has no effect. If a pin is configured as level-sensitive (EPPARn = 00), pin transitions do not affect this register. 0 Selected edge for IRQn pin not detected 1 Selected edge for IRQn pin detected

# Chapter 19

## Enhanced Direct Memory Access (eDMA)

### 19.1 Overview

The enhanced direct memory access (eDMA) controller is a second-generation module capable of performing complex data transfers with minimal intervention from a host processor. The hardware microarchitecture includes a DMA engine that performs source- and destination-address calculations, and the actual data-movement operations, along with local memory containing transfer control descriptors for each channel.

#### 19.1.1 Block Diagram

Figure 19-1 is a block diagram of the eDMA module.

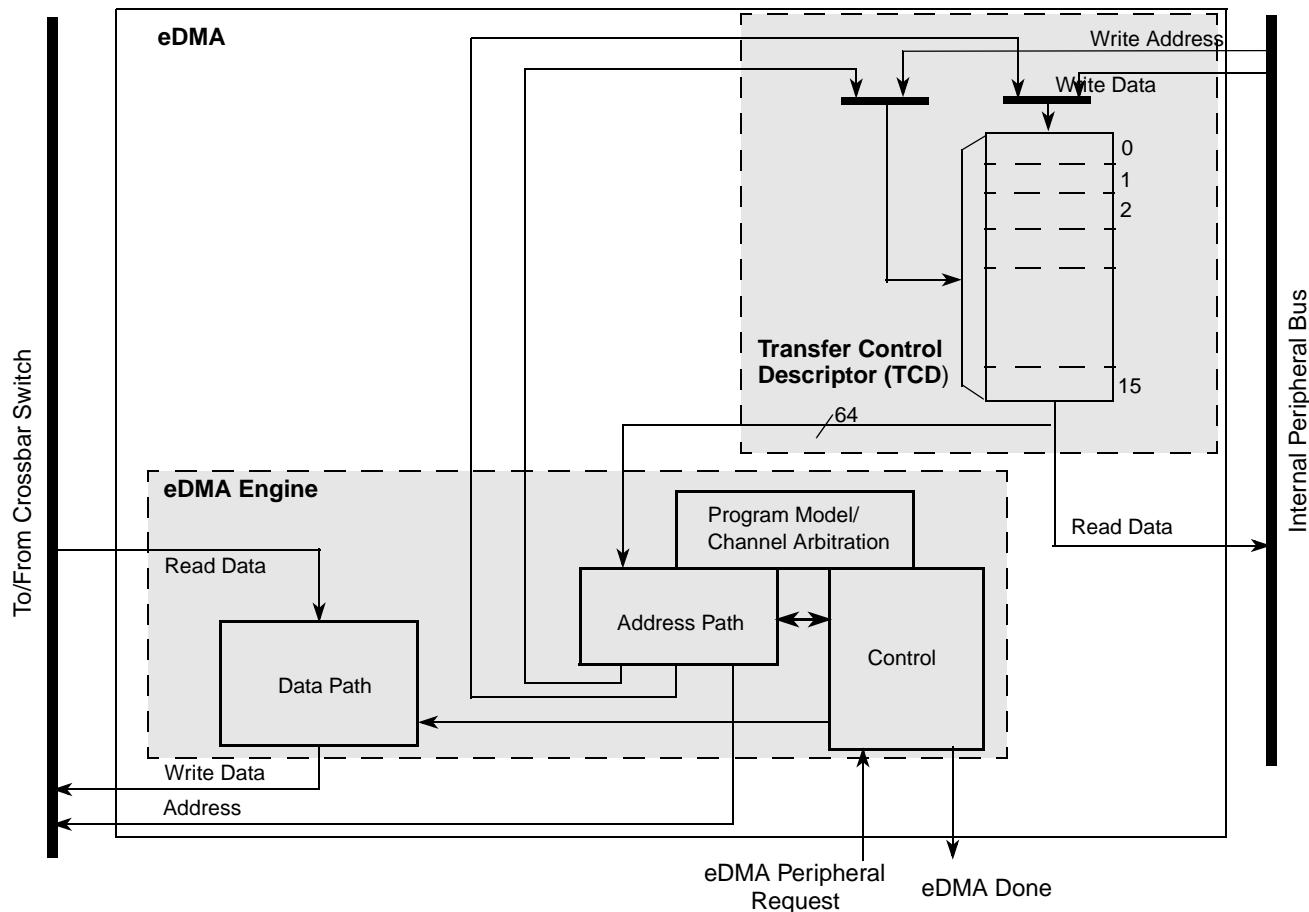


Figure 19-1. eDMA Block Diagram

## 19.1.2 Features

The eDMA is a highly-programmable data-transfer engine optimized to minimize the required intervention from the host processor. It is intended for use in applications where the data size to be transferred is statically known and not defined within the data packet itself. The eDMA module features:

- All data movement via dual-address transfers: read from source, write to destination
  - Programmable source and destination addresses and transfer size, plus support for enhanced addressing modes
- 64-channel implementation that performs complex data transfers with minimal intervention from a host processor
  - Internal data buffer, used as temporary storage to support 16-byte burst transfers
  - Connections to the crossbar switch for bus mastering the data movement
- Transfer control descriptor (TCD) organized to support two-deep, nested transfer operations
  - 32-byte TCD stored in local memory for each channel
  - 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 continual transfers
  - Peripheral-paced hardware requests (one per channel)
- Support for fixed-priority and round-robin channel arbitration
- Channel completion reported via optional interrupt requests
  - One interrupt per channel, optionally asserted at completion of major iteration count
  - Error terminations are optionally enabled per channel and logically summed together to form one error interrupt to the interrupt controller
- Optional support for scatter/gather DMA processing
- Support for complex data structures
- Support to cancel transfers via hardware or software

Throughout this chapter, *n* is used to reference the channel number.

## 19.2 Modes of Operation

### 19.2.1 Normal Mode

In normal mode, the eDMA transfers 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.

A service request initiates a transfer of a specific number of bytes (NBYTES) as specified in the transfer control descriptor (TCD). The minor loop is the sequence of read-write operations that transfers these NBYTES per service request. A major loop is the number of minor loop iterations defining a task.

## 19.2.2 Debug Mode

In debug mode, the eDMA stops transferring data. If debug mode is entered during the transfer of a data block described by a minor loop in the current active channel's TCD, the eDMA continues operation until completion of the minor loop.

## 19.3 External Signal Description

This section describes the external signals of the eDMA controller.

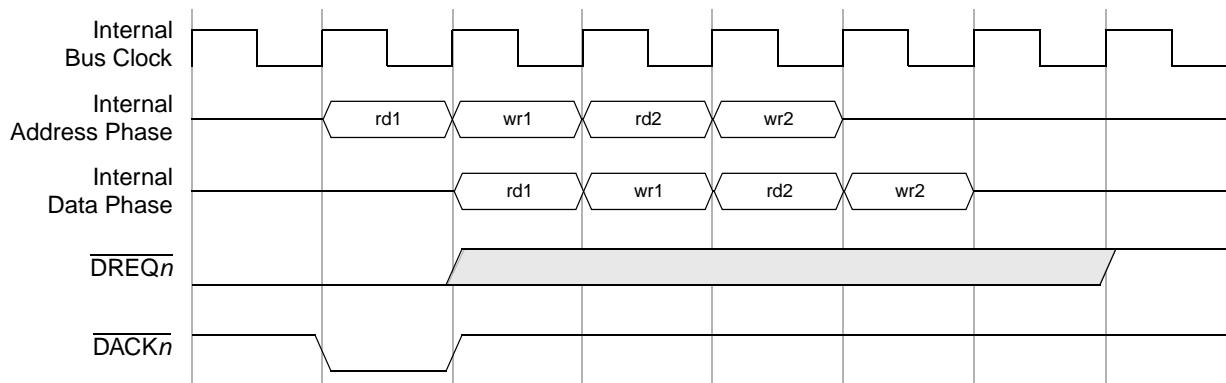
**Table 19-1. External Signal List**

Signal Name	I/O	Description
$\overline{\text{DREQ}0}$	I	Provides external requests from peripherals needing DMA service. When asserted, the device is requesting service. This request pin is tied to DMA channel 0.
$\overline{\text{DACK}0}$	O	Indicates when the external DMA request has been acknowledged.

### 19.3.1 External Signal Timing

Asserting the external DMA request signal,  $\overline{\text{DREQ}n}$ , initiates a service request for that channel. It must remain asserted until the corresponding  $\overline{\text{DACK}n}$  signal indicates the channel's data transfer has started. The  $\overline{\text{DACK}n}$  output is asserted for one cycle during the address phase of the channel's first internal read access.

- When no further requests are needed, the  $\overline{\text{DREQ}n}$  signal must negate after the  $\overline{\text{DACK}n}$  assertion and on or before the second cycle following the data phase of the last internal bus write (see [Figure 19-2](#)).
- If another service request is needed,  $\overline{\text{DREQ}n}$  may simply remain asserted.
- To request continuous service,  $\overline{\text{DREQ}n}$  may remain continuously asserted.



**Figure 19-2.  $\overline{\text{DREQ}n}$  and  $\overline{\text{DACK}n}$  Timing**

After a service request has been initiated, it cannot be canceled. Removing a service request after it has been asserted may result in one of three actions depending on the DMA engine's status:

- The request is never recognized because another channel is executing.

- The request is considered spurious and discarded, because the request is removed during arbitration for next channel selection.
- The channel is selected by arbitration and begins execution.

## 19.4 Memory Map/Register Definition

The eDMA's programming model is partitioned into two regions: the first region defines a number of registers providing control functions, while the second region corresponds to the local transfer control descriptor memory.

Some registers are implemented as two 32-bit registers, and include an H and L suffix, signaling the high and low portions of the control function. [Table 19-2](#) is a 32-bit view of the eDMA's memory map.

Reading reserved bits in a register return the value of zero and writes to reserved bits in a register are ignored. Reading or writing to a reserved memory location generates a bus error.

**Table 19-2. eDMA Controller Memory Map**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC04_4000	eDMA Control Register (EDMA_CR)	32	R/W	0x0000_0000	<a href="#">19.4.1/19-5</a>
0xFC04_4004	eDMA Error Status Register (EDMA_ES)	32	R	0x0000_0000	<a href="#">19.4.2/19-7</a>
0xFC04_4008	eDMA Enable Request High Register (EDMA_ERQH, Channels 63-32)	32	R/W	0x0000_0000	<a href="#">19.4.3/19-10</a>
0xFC04_400E	eDMA Enable Request Low Register (EDMA_ERQL, Channels 31-00)	16	R/W	0x0000	<a href="#">19.4.3/19-10</a>
0xFC04_4010	eDMA Enable Error Interrupt High Register (EDMA_EEIH, Channels 63-32)	32	R/W	0x0000_0000	<a href="#">19.4.4/19-13</a>
0xFC04_4016	eDMA Enable Error Interrupt Low Register (EDMA_EEIL, Channels 31-00)	16	R/W	0x0000	<a href="#">19.4.4/19-13</a>
0xFC04_4018	eDMA Set Enable Request (EDMA_SERQ)	8	W	0x00	<a href="#">19.4.5/19-14</a>
0xFC04_4019	eDMA Clear Enable Request (EDMA_CERQ)	8	W	0x00	<a href="#">19.4.6/19-15</a>
0xFC04_401A	eDMA Set Enable Error Interrupt Register (EDMA_SEEI)	8	W	0x00	<a href="#">19.4.7/19-16</a>
0xFC04_401B	eDMA Clear Enable Error Interrupt Register (EDMA_CEEI)	8	W	0x00	<a href="#">19.4.8/19-16</a>
0xFC04_401C	eDMA Clear Interrupt Request Register (EDMA_CINT)	8	W	0x00	<a href="#">19.4.9/19-17</a>
0xFC04_401D	eDMA Clear Error Register (EDMA_CERR)	8	W	0x00	<a href="#">19.4.10/19-18</a>
0xFC04_401E	eDMA Set START Bit Register (EDMA_SSRT)	8	W	0x00	<a href="#">19.4.11/19-18</a>
0xFC04_401F	eDMA Clear DONE Status Bit Register (EDMA_CDNE)	8	W	0x00	<a href="#">19.4.12/19-19</a>
0xFC04_4020	eDMA Interrupt Request High Register (EDMA_INTH, Channels 63-32)	32	R/W	0x0000_0000	<a href="#">19.4.13/19-20</a>
0xFC04_4026	eDMA Interrupt Request Low Register (EDMA_INTL, Channels 31-00)	32	R/W	0x0000	<a href="#">19.4.13/19-20</a>
0xFC04_4028	eDMA Error High Register (EDMA_ERRH, Channels 63-32)	32	R/W	0x0000_0000	<a href="#">19.4.14/19-21</a>

**Table 19-2. eDMA Controller Memory Map (continued)**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC04_402E	eDMA Error Low Register (EDMA_ERRL, Channels 31-00)	32	R/W	0x0000	<a href="#">19.4.14/19-21</a>
0xFC04_4030	eDMA Hardware Request Status High (EDMA_RSH, Channels 63-32)	32	R/W	0x0000	<a href="#">19.4.15/19-22</a>
0xFC04_4034	eDMA Hardware Request Status Low (EDMA_RSL, Channels 31-00)	32	R/W	0x0000	<a href="#">19.4.15/19-22</a>
0xFC04_4100 + hex( <i>n</i> )	eDMA Channel <i>n</i> Priority Register (DCHPRIn) for <i>n</i> = 0 – 63	8	R/W	See Section	<a href="#">19.4.16/19-23</a>
0xFC04_5000 + hex(32× <i>n</i> )	Transfer Control Descriptor (TCDn) for <i>n</i> = 0 – 63	256	R/W	See Section	<a href="#">19.4.17/19-24</a>

### 19.4.1 eDMA Control Register (EDMA\_CR)

The EDMA\_CR defines the basic operating configuration of the eDMA. The eDMA arbitrates channel service requests in four groups (0, 1, 2, 3) of 16 channels each. Group 3 contains channels 63-48; group 2 contains channels 47-32; group 1 contains channels 31-16; and group 0 contains channels 15-0.

Arbitration within a group can be configured to use a fixed-priority or a round-robin scheme. In fixed-priority arbitration, the highest priority channel requesting service is selected to execute. The channel priority registers assign the priorities (see [Section 19.4.16, “eDMA Channel n Priority Registers \(DCHPRIn\)”](#)). In round-robin arbitration mode, the channel priorities are ignored, and channels within each group are cycled through without regard to priority.

#### NOTE

For proper operation, writes to the EDMA\_CR register must only be performed when the DMA channels are inactive (TCR*n*\_CSR[ACTIVE] bits are cleared).

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\_CR). All group priorities must have unique values prior to any channel service requests occur, otherwise a configuration error will be reported. In group round robin mode, the group priorities are ignored and the groups are cycled through without regard to priority.

## Enhanced Direct Memory Access (eDMA)

Address: 0xFC04\_4000 (EDMA\_CR) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CX	ECX
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	GRP3PRI	GRP2PRI	GRP1PRI	GRP0PRI	EMLM	CLM	HALT	HOE	ERGA	ERCA	EDBG	0				
W					0	1	0	0	0	0	0	0	0	0	0	0
Reset	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0

Figure 19-3. eDMA Control Register (EDMA\_CR)

Table 19-3. EDMA\_CR Field Descriptions

Field	Description
31–17	Reserved, must be cleared.
17 CX	Cancel transfer 0 Normal operation 1 Cancel the remaining data transfer. Stop the executing channel and force the minor loop to finish. 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.
16 ECX	Error cancel transfer 0 Normal operation 1 Cancel the remaining data transfer in the same fashion as the CX bit. Stop the executing channel and force the minor loop to finish. The cancel takes effect after the last write of the current read/write sequence. The ECX bit clears itself after the cancel is honored. In addition to cancelling the transfer, ECX treats the cancel as an error condition; thus updating the EDMA_ES register and generating an optional error interrupt.
15–14 GRP3PRI	Channel group 3 priority. Group 3 priority level when fixed priority group arbitration is enabled.
13–12 GRP2PRI	Channel group 2 priority. Group 2 priority level when fixed priority group arbitration is enabled.
11–10 GRP1PRI	Channel group 1 priority. Group 1 priority level when fixed priority group arbitration is enabled.
9–8 GRP0PRI	Channel group 0 priority. Group 0 priority level when fixed priority group arbitration is enabled.
7 EMLM	Enable minor loop mapping. 0 Disabled. TCDn.word2 is defined as a 32-bit NBYTES field. 1 Enabled. TCDn.word2 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.
6 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 activates 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.

**Table 19-3. EDMA\_CR Field Descriptions (continued)**

Field	Description
5 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 this bit is cleared.
4 HOE	Halt on error. 0 Normal operation 1 Any error causes the HALT bit to set. Subsequently, all service requests are ignored until the HALT bit is cleared.
3 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.
2 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.
1 EDBG	Enable debug. 0 When in debug mode the DMA continues to operate. 1 When in debug mode, the eDMA stalls the start of a new channel. Executing channels are allowed to complete. Channel execution resumes when the system exits debug mode or the EDBG bit is cleared.
0	Reserved, must be cleared.

#### 19.4.2 eDMA Error Status Register (EDMA\_ES)

The EDMA\_ES provides information concerning 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 reported when the starting source or destination address, source or destination offsets, minor loop byte count, or the transfer size represent an inconsistent state. Each of these possible causes are detailed in the below list:

- The addresses and offsets must be aligned on 0-modulo-transfer-size boundaries
- 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 caused by any two channel priorities being equal within a group, or any group priority levels being equal among the groups. All channel priority levels within a group must be unique and all group priority levels among the groups must be unique when fixed arbitration mode is enabled.
- If a scatter/gather operation is enabled upon 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 upon channel completion, a configuration error is reported when the link is attempted if the TCD $n$ \_CITER[E\_LINK] bit does not equal the TCD $n$ \_BITER[E\_LINK] bit.

If enabled, all configuration error conditions, except the scatter/gather and minor-loop link errors, report as the channel activates and asserts an error interrupt request. A scatter/gather configuration error is reported when the scatter/gather operation begins at major loop completion when properly enabled. 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 stopped and the appropriate bus error flag set. In this case, the state of the channel's transfer control descriptor is updated by the eDMA engine with the current source address, destination address and current iteration count at the point of the fault. When a system-bus error occurs, the channel terminates after the read or write transaction (which is already pipelined after errant access) has completed. If a bus error occurs on the last read prior to beginning the write sequence, the write executes 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 executes before the channel terminates due to the destination bus error.

A transfer may be cancelled by software with the EDMA\_CR[CX] bit. When a cancel transfer request is recognized, the DMA 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 EDMA\_ES register is updated with the cancelled channel number and ECX is set. The TCD of a cancelled channel contains the source and destination addresses of the last transfer saved in the TCD. If the channel needs to be restarted, you must re-initialize the TCD since the aforementioned fields no longer represent the original parameters. When a transfer is cancelled by the error cancel transfer mechanism, the channel number is loaded into DMA\_ES[ERRCHN] and ECX and VLD are set. In addition, an error interrupt may be generated if enabled.

The occurrence of any error causes the eDMA engine to stop the active channel immediately, and the appropriate channel bit in the eDMA error register is asserted. At the same time, the details of the error condition are loaded into the EDMA\_ES. 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 eDMA engine continues operating 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 executes and terminate with the same error condition.

Address: 0xFC04\_4004 (EDMA\_ES)

Access: User read-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
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 19-4. eDMA Error Status Register (EDMA\_ES)

Table 19-4. EDMA\_ES Field Descriptions

Field	Description
31 VLD	Logical OR of all EDMA_ERRH and EDMA_ERRL status bits 0 No EDMA_ERR bits are set 1 At least one EDMA_ERR bit is set indicating a valid error exists that has not been cleared
30–17	Reserved, must be cleared.
16 ECX	Transfer cancelled 0 No cancelled transfers 1 The last recorded entry was a cancelled transfer by the error cancel transfer input
15 GPE	Group priority error 0 No group priority error 1 The last recorded error was a configuration error among the group priorities. All group priorities are not unique.
14 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. Channel priorities within a group are not unique.
13–8 ERRCHN	Error channel number or cancelled channel number. The channel number of the last recorded error (excluding GPE and CPE errors) or last recorded error cancelled transfer.
7 SAE	Source address error. 0 No source address configuration error. 1 The last recorded error was a configuration error detected in the TCDn_SADDR field. TCDn_SADDR is inconsistent with TCDn_ATTR[SSIZE]
6 SOE	Source offset error. 0 No source offset configuration error. 1 The last recorded error was a configuration error detected in the TCDn_SOFF field. TCDn_SOFF is inconsistent with TCDn_ATTR[SSIZE].
5 DAE	Destination address error. 0 No destination address configuration error. 1 The last recorded error was a configuration error detected in the TCDn_DADDR field. TCDn_DADDR is inconsistent with TCDn_ATTR[DSIZE].
4 DOE	Destination offset error. 0 No destination offset configuration error. 1 The last recorded error was a configuration error detected in the TCDn_DOFF field. TCDn_DOFF is inconsistent with TCDn_ATTR[DSIZE].

**Table 19-4. EDMA\_ES Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>
3 NCE	NBYTES/CITER configuration error. 0 No NBYTES/CITER configuration error. 1 The last recorded error was a configuration error detected in the TCD <sub>n</sub> _NBYTES or TCD <sub>n</sub> _CITER fields. <ul style="list-style-type: none"><li>• TCD<sub>n</sub>_NBYTES is not a multiple of TCD<sub>n</sub>_ATTR[SSIZE] and TCD<sub>n</sub>_ATTR[DSIZE], or</li><li>• TCD<sub>n</sub>_CITER[CITER] is equal to zero, or</li><li>• TCD<sub>n</sub>_CITER[E_LINK] is not equal to TCD<sub>n</sub>_BITER[E_LINK].</li></ul>
2 SGE	Scatter/gather configuration error. 0 No scatter/gather configuration error. 1 The last recorded error was a configuration error detected in the TCD <sub>n</sub> _DLAST_SGA field. This field is checked at the beginning of a scatter/gather operation after major loop completion if TCD <sub>n</sub> _CSR[E_SG] is enabled. TCD <sub>n</sub> _DLAST_SGA is not on a 32 byte boundary.
1 SBE	Source bus error. 0 No source bus error. 1 The last recorded error was a bus error on a source read.
0 DBE	Destination bus error. 0 No destination bus error. 1 The last recorded error was a bus error on a destination write.

### 19.4.3 eDMA Enable Request Registers (EDMA\_ERQH, EDMA\_ERQL)

The EDMA\_ERQ{H,L} registers provide a bit map for the 64 implemented channels to enable the request signal for each channel. EDMA\_ERQH supports channels 63-32, while EDMA\_EQRL covers channels 31-00. The state of any given channel enable is directly affected by writes to this register; it is also affected by writes to the EDMA\_SERQ and EDMA\_CERQ. The EDMA\_{S,C}ERQR are provided so the request enable for a single channel can easily be modified without needing to perform a read-modify-write sequence to the EDMA\_ERQ{H,L}.

DMA request input signals 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 affect a channel service request made explicitly through software or a linked channel request.

Address: 0xFC04_4008 (EDMA_ERQH)																Access: User read/write								
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	ERQ							
W	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	ERQ							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ERQ							
W	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	ERQ							
Reset	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. eDMA Enable Request High Register (EDMA\_ERQH)**

Address: 0xFC04_400C (EDMA_ERQ)																Access: User read/write							
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16							
R	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							
W	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16							
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	ERQ 15	ERQ 14	ERQ 13	ERQ 12	ERQ 11	ERQ 10	ERQ 9	ERQ 8	ERQ 7	ERQ 6	ERQ 5	ERQ 4	ERQ 3	ERQ 2	ERQ 1	ERQ 0							
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							

Figure 19-6. eDMA Enable Request Low Register (EDMA\_ERQL)

Table 19-5. EDMA\_ERQH, EDMA\_ERQL Field Descriptions

Field	Description
31–0 ERQ $n$	Enable DMA Request $n$ . 0 The DMA request signal for channel $n$ is disabled. 1 The DMA request signal for channel $n$ is enabled.

The assignments between the DMA requests from the peripherals to the channels of the eDMA are shown in [Table 19-6](#).

Table 19-6. DMA Request Summary for eDMA

Channel	Source	Description
0	DREQ0	External DMA request 0
1	DREQ1	External DMA request 1
2	UISR0[FFULL/RXRDY]	UART0 Receive
3	UISR0[TXRDY]	UART0 Transmit
4	UISR1[FFULL/RXRDY]	UART1 Receive
5	UISR1[TXRDY]	UART1 Transmit
6	UISR2[FFULL/RXRDY]	UART2 Receive
7	UISR2[TXRDY]	UART2 Transmit
8	DTER0[CAP] or DTER0[REF]	Timer 0
9	DTER1[CAP] or DTER1[REF]	Timer 1
10	DTER2[CAP] or DTER2[REF]	Timer 2
11	DTER3[CAP] or DTER3[REF]	Timer 3
12	DSPI0_SR[RFDF]	DSPI0 Receive
13	DSPI0_SR[TFFF]	DSPI0 Transmit
14	DSPI1_SR[RFDF]	DSPI1 Receive
15	DSPI1_SR[TFFF]	DSPI1 Transmit
16	UISR3[FFULL/RXRDY]	UART3 Receive

**Table 19-6. DMA Request Summary for eDMA (continued)**

<b>Channel</b>	<b>Source</b>	<b>Description</b>
17	UISR3[TXRDY]	UART3 Transmit
18	UISR4[FFULL/RXRDY]	UART4 Receive
19	UISR4[TXRDY]	UART4 Transmit
20	UISR5[FFULL/RXRDY]	UART5 Receive
21	UISR5[TXRDY]	UART5 Transmit
22	UISR6[FFULL/RXRDY]	UART6 Receive
23	UISR6[TXRDY]	UART6 Transmit
24	I2SR0[IIF]	I <sup>2</sup> C0
25	I2SR1[IIF]	I <sup>2</sup> C1
26	I2SR2[IIF]	I <sup>2</sup> C2
27	I2SR3[IIF]	I <sup>2</sup> C3
28	DSPI2_SR[RFDF]	DSPI2 Receive
29	DSPI2_SR[TFFF]	DSPI2 Transmit
30	Activated explicitly by setting the TCD $n$ _CSR[START] bit	Available for software
31	Activated explicitly by setting the TCD $n$ _CSR[START] bit	Available for software
32	UISR7[FFULL/RXRDY]	UART7 Receive
33	UISR7[TXRDY]	UART7 Transmit
34	UISR8[FFULL/RXRDY]	UART8 Receive
35	UISR8[TXRDY]	UART8 Transmit
36	UISR9[FFULL/RXRDY]	UART9 Receive
37	UISR9[TXRDY]	UART9 Transmit
38	OW_ISR	1-Wire Request
39	—	Reserved
40	I2SR4[IIF]	I <sup>2</sup> C4
41	I2SR5[IIF]	I <sup>2</sup> C5
42	Activated explicitly by setting the TCD $n$ _CSR[START] bit	Available for software
43	Activated explicitly by setting the TCD $n$ _CSR[START] bit	Available for software
44	DSPI3_SR[RFDF]	DSPI3 Receive
45	DSPI3_SR[TFFF]	DSPI3 Transmit
46	Activated explicitly by setting the TCD $n$ _CSR[START] bit	Available for software

**Table 19-6. DMA Request Summary for eDMA (continued)**

<b>Channel</b>	<b>Source</b>	<b>Description</b>
47	Activated explicitly by setting the TCD $n$ _CSR[START] bit	Available for software
48	SSI0_SR[RFF0]	SSI0 Receive 0
49	SSI0_SR[RFF1]	SSI0 Receive 1
50	SSI0_SR[TFE0]	SSI0 Transmit 0
51	SSI0_SR[TFE1]	SSI0 Transmit 1
52	SSI1_SR[RFF0]	SSI1 Receive 0
53	SSI1_SR[RFF1]	SSI1 Receive 1
54	SSI1_SR[TFE0]	SSI1 Transmit 0
55	SSI1_SR[TFE1]	SSI1 Transmit 1
56	Depends on PWM_SM $n$ DMAEN[CAPTDE]	mcPWM Capture
57	PWM_SM $n$ SR[RF]	mcPWM Value
58	—	Reserved
59	SDHC_SR[BRRDY, BWRDY]	eSDHC
60	ADC_SR	ADC0
61	ADC_SR	ADC1
62	DAC0_SR	DAC0
63	DAC1_SR	DAC1

As a given channel completes the processing of its major iteration count, a flag in the transfer control descriptor that affect the ending state of the EDMA\_ERQ bit for that channel. If the TCD $n$ \_CSR[D\_REQ] bit is set, the corresponding EDMA\_ERQ bit is cleared, disabling the DMA request. If the D\_REQ bit clears, the state of the EDMA\_ERQ bit is unaffected.

#### 19.4.4 eDMA Enable Error Interrupt Registers (EDMA\_EEIH, EDMA\_EEIL)

The EDMA\_EEI{H,L} registers provide a bit map for the 64 channels to enable the error interrupt signal for each channel. EDMA\_EEIH supports channels 63-32, while EDMA\_EEIL covers channels 31-00. The state of any given channel's error interrupt enable is directly affected by writes to this register; it is also affected by writes to the EDMA\_SEEI and EDMA\_CEEI. The EDMA\_{S,C}EEIR are provided so the error interrupt enable for a single channel can easily be modified without the need to perform a read-modify-write sequence to the EDMA\_EEI{H,L} registers.

The DMA error indicator and the error interrupt enable flag must be asserted before an error interrupt request for a given channel is asserted to the interrupt controller.

## Enhanced Direct Memory Access (eDMA)

Address: 0xFC04_4010 (EDMA_EEIH)																Access: User read/write							
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16						
R	EEI63	EEI62	EEI61	EEI60	EEI59	EEI58	EEI57	EEI56	EEI55	EEI54	EEI53	EEI52	EEI51	EEI50	EEI49	EEI48							
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
Reset		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
R	EEI47	EEI46	EEI45	EEI44	EEI43	EEI42	EEI41	EEI40	EEI39	EEI38	EEI37	EEI36	EEI35	EEI34	EEI33	EEI32							
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							

Figure 19-7. eDMA Enable Error Interrupt High Register (EDMA\_EEIH)

Address: 0xFC04_4014 (EDNA_EEIL)																Access: User read/write							
		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16						
R	EEI31	EEI30	EEI29	EEI28	EEI27	EEI26	EEI25	EEI24	EEI23	EEI22	EEI21	EEI20	EEI19	EEI18	EEI17	EEI16							
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
Reset		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0						
R	EEI15	EEI14	EEI13	EEI12	EEI11	EEI10	EEI9	EEI8	EEI7	EEI6	EEI5	EEI4	EEI3	EEI2	EEI1	EEI0							
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							

Figure 19-8. eDMA Enable Error Interrupt Low Register (EDMA\_EEIL)

Table 19-7. EDMA\_EEIH, EDMA\_EEIL Field Descriptions

Field	Description
31–0 EEIn	Enable error interrupt <i>n</i> . 0 The error signal for channel <i>n</i> does not generate an error interrupt. 1 The assertion of the error signal for channel <i>n</i> generates an error interrupt request.

## 19.4.5 eDMA Set Enable Request Register (EDMA\_SERQ)

The EDMA\_SERQ provides a simple memory-mapped mechanism to set a given bit in the EDMA\_ERQ{H,L} to enable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA\_ERQ{H,L} to be set. Setting the SAER bit provides a global set function, forcing the entire contents of EDMA\_ERQ{H,L} to be set. If bit 7 is set the command is ignored. This allows you to write multiple-byte registers as a 32-bit word. Reads of this register return all zeroes.

Address: 0xFC04\_4018 (EDMA\_SERQ)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W	NOP	SAER	SERQ					
Reset	0	0	0	0	0	0	0	0

Figure 19-9. eDMA Set Enable Request Register (EDMA\_SERQ)

Table 19-8. EDMA\_SERQ Field Descriptions

Field	Description
7 NOP	0 Normal operation 1 No operation, ignore bits 6–0 of this register
6 SAER	Set all enable requests. 0 Set only those EDMA_ERQ{H,L} bits specified in the SERQ field. 1 Set all bits in EDMA_ERQ{H,L}.
5–0 SERQ	Set enable request. Sets the corresponding bit in EDMA_ERQ{H,L}.

#### 19.4.6 eDMA Clear Enable Request Register (EDMA\_CERQ)

The EDMA\_CERQ provides a simple memory-mapped mechanism to clear a given bit in the EDMA\_ERQ{H,L} to disable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA\_ERQ{H,L} to be cleared. Setting the CAER bit provides a global clear function, forcing the entire contents of the EDMA\_ERQ{H,L} to be cleared, disabling all DMA request inputs. If bit 7 is set, the command is ignored. This allows you to write multiple-byte registers as a 32-bit word. Reads of this register return all zeroes.

Address: 0xFC04\_4019 (EDMA\_CERQ)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W	NOP	CAER	CERQ					
Reset	0	0	0	0	0	0	0	0

Figure 19-10. eDMA Clear Enable Request Register (EDMA\_CERQ)

Table 19-9. EDMA\_CERQ Field Descriptions

Field	Description
7 NOP	0 Normal operation 1 No operation, ignore bits 6–0 of this register

**Table 19-9. EDMA\_CERQ Field Descriptions (continued)**

Field	Description
6 CAER	Clear all enable requests. 0 Clear only those EDMA_ERQ{H,L} bits specified in the CERQ field. 1 Clear all bits in EDMA_ERQ{H,L}.
5–0 CERQ	Clear enable request. Clears the corresponding bit in EDMA_ERQ{H,L}.

### 19.4.7 eDMA Set Enable Error Interrupt Register (EDMA\_SEEI)

The EDMA\_SEEI provides a simple memory-mapped mechanism to set a given bit in the EDMA\_EEI{H,L} to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA\_EEI{H,L} to be set. Setting the SAEE bit provides a global set function, forcing the entire EDMA\_EEI{H,L} contents to be set. If bit 7 is set, the command is ignored. This allows you to write multiple-byte registers as a 32-bit word. Reads of this register return all zeroes.

Address: 0xFC04\_401A (EDMA\_SEEI)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W	NOP	SAEE			SEEI			
Reset	0	0	0	0	0	0	0	0

**Figure 19-11. eDMA Set Enable Error Interrupt Register (EDMA\_SEEI)****Table 19-10. EDMA\_SEEI Field Descriptions**

Field	Description
7 NOP	0 Normal operation 1 No operation, ignore bits 6–0 of this register
6 SAEE	Sets all enable error interrupts. 0 Set only those EDMA_EEI{H,L} bits specified in the SEEI field. 1 Sets all bits in EDMA_EEI{H,L}.
5–0 SEEI	Set enable error interrupt. Sets the corresponding bit in EDMA_EEI{H,L}.

### 19.4.8 eDMA Clear Enable Error Interrupt Register (EDMA\_CEEI)

The EDMA\_CEEI provides a simple memory-mapped mechanism to clear a given bit in the EDMA\_EEI{H,L} to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA\_EEI{H,L} to be cleared. Setting the CAEE bit provides a global clear function, forcing the EDMA\_EEI{H,L} contents to be cleared, disabling all DMA request inputs. If bit 7 is set, the command is ignored. This allows you to write multiple-byte registers as a 32-bit word. Reads of this register return all zeroes.

Address: 0xFC04\_401B (EDNA\_CEEI)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W	NOP	CAEE	CEEI					
Reset	0	0	0	0	0	0	0	0

Figure 19-12. eDMA Clear Enable Error Interrupt Register (EDMA\_CEEI)

Table 19-11. EDMA\_CEEI Field Descriptions

Field	Description
7 NOP	0 Normal operation 1 No operation, ignore bits 6–0 of this register
6 CAEE	Clear all enable error interrupts. 0 Clear only those EDMA_EEI{H,L} bits specified in the CEEI field. 1 Clear all bits in EDMA_EEI{H,L}.
5–0 CEEI	Clear enable error interrupt. Clears the corresponding bit in EDMA_EEI{H,L}.

### 19.4.9 eDMA Clear Interrupt Request Register (EDMA\_CINT)

The EDMA\_CINT provides a simple, memory-mapped mechanism to clear a given bit in the EDMA\_INT{H,L} to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the EDMA\_INT{H,L} to be cleared. Setting the CAIR bit provides a global clear function, forcing the entire contents of the EDMA\_INT{H,L} to be cleared, disabling all DMA interrupt requests. If bit 7 is set, the command is ignored. This allows you to write multiple-byte registers as a 32-bit word. Reads of this register return all zeroes.

Address: 0xFC04\_401C (EDMA\_CINT)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W	NOP	CAIR	CINT					
Reset	0	0	0	0	0	0	0	0

Figure 19-13. eDMA Clear Interrupt Request (EDMA\_CINT)

Table 19-12. EDMA\_CINT Field Descriptions

Field	Description
7 NOP	0 Normal operation 1 No operation, ignore bits 6–0 of this register

**Table 19-12. EDMA\_CINT Field Descriptions (continued)**

Field	Description
6 CAIR	Clear all interrupt requests. 0 Clear only those EDMA_INT{H,L} bits specified in the CINT field. 1 Clear all bits in EDMA_INT{H,L}.
5–0 CINT	Clear interrupt request. Clears the corresponding bit in EDMA_INT{H,L}.

### 19.4.10 eDMA Clear Error Register (EDMA\_CERR)

The EDMA\_CERR provides a simple memory-mapped mechanism to clear a given bit in the EDMA\_ERR{H,L} to disable the error condition flag for a given channel. The given value on a register write causes the corresponding bit in the EDMA\_ERR{H,L} to be cleared. Setting the CAEI bit provides a global clear function, forcing the EDMA\_ERR{H,L} contents to be cleared, clearing all channel error indicators. If bit 7 is set, the command is ignored. This allows you to write multiple-byte registers as a 32-bit word. Reads of this register return all zeroes.

Address: 0xFC04\_401D (EDMA\_CERR)

Access: User write-only

R	7	6	5	4	3	2	1	0
	0	0	0	0	0	0	0	0
W	NOP	CAEI			CERR			
Reset	0	0	0	0	0	0	0	0

**Figure 19-14. eDMA Clear Error Register (EDMA\_CERR)****Table 19-13. EDMA\_CERR Field Descriptions**

Field	Description
7 NOP	0 Normal operation 1 No operation, ignore bits 6–0 of this register
6 CAEI	Clear all error indicators. 0 Clear only those EDMA_ERR{H,L} bits specified in the CERR field. 1 Clear all bits in EDMA_ERR{H,L}.
5–0 CERR	Clear error indicator. Clears the corresponding bit in EDMA_ERR{H,L}.

### 19.4.11 eDMA Set START Bit Register (EDMA\_SSRT)

The EDMA\_SSRT provides a simple 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 the SAST bit provides a global set function, forcing all START bits to be set. If bit 7 is set, the command is ignored. This allows you to write multiple-byte registers as a 32-bit word. Reads of this register return all zeroes.

Address: 0xFC04\_401E (EDMA\_SSRT)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W	NOP	SAST			SSRT			
Reset	0	0	0	0	0	0	0	0

Figure 19-15. eDMA Set START Bit Register (EDMA\_SSRT)

Table 19-14. EDMA\_SSRT Field Descriptions

Field	Description
7 NOP	0 Normal operation 1 No operation, ignore bits 6–0 of this register
6 SAST	Set all START bits (activates all channels). 0 Set only those TCDn_CSR[START] bits specified in the SSRT field. 1 Set all bits in TCDn_CSR[START].
5–0 SSRT	Set START bit. Sets the corresponding bit in TCDn_CSR[START].

### 19.4.12 eDMA Clear DONE Status Bit Register (EDMA\_CDNE)

The EDMA\_CDNE provides a simple 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 the CADN bit provides a global clear function, forcing all DONE bits to be cleared. If bit 7 is set, the command is ignored. This allows you to write multiple-byte registers as a 32-bit word. Reads of this register return all zeroes.

Address: 0xFC04\_401F (EDMA\_CDNE)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0		
W	NOP	CADN			CDNE			
Reset	0	0	0	0	0	0	0	0

Figure 19-16. eDMA Clear DONE Status Bit Register (EDMA\_CDNE)

Table 19-15. EDMA\_CDNE Field Descriptions

Field	Description
7 NOP	0 Normal operation 1 No operation, ignore bits 6–0 of this register
6 CADN	Clears all DONE bits. 0 Clears only those TCDn_CSR[DONE] bits specified in the CDNE field. 1 Clears all bits in TCDn_CSR[DONE]
5–0 CDNE	Clear DONE bit. Clears the corresponding bit in TCDn_CSR[DONE].

### 19.4.13 eDMA Interrupt Request Registers (EDMA\_INTH, EDMA\_INTL)

The EDMA\_INT{H,L} provide a bit map for the 64 channels signaling the presence of an interrupt request for each channel. Depending on the appropriate bit setting in the transfer-control descriptions, the eDMA engine generates an interrupt a data transfer completion. The outputs of this register are directly routed to the interrupt controller (INTC). During the interrupt-service routine associated with any given channel, it is the software's responsibility to clear the appropriate bit, negating the interrupt request. Typically, a write to the EDMA\_CINT 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\_CINT. On writes to the EDMA\_INT, a 1 in any bit position clears the corresponding channel's interrupt request. A zero in any bit position has no affect on the corresponding channel's current interrupt status. The EDMA\_CINT is provided so the interrupt request for a single channel can easily be cleared without the need to perform a read-modify-write sequence to the EDMA\_INT{H,L}.

Offset 0xFC04_4020 (EDMA_INTH)																Access: User read/write							
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16							
R	INT63	INT62	INT61	INT60	INT59	INT58	INT57	INT56	INT55	INT54	INT53	INT52	INT51	INT50	INT49	INT48							
W	w1c																						
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	INT47	INT46	INT45	INT44	INT43	INT42	INT41	INT40	INT39	INT38	INT37	INT36	INT35	INT34	INT33	INT32							
W	w1c																						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							

Figure 19-17. eDMA Interrupt Request High Register (EDMA\_INTH)

Address: 0xFC04_4024 (EDMA_INTL)																Access: User read/write							
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16							
R	INT31	INT30	INT29	INT28	INT27	INT26	INT25	INT24	INT23	INT22	INT21	INT20	INT19	INT18	INT17	INT16							
W	w1c																						
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	INT15	INT14	INT13	INT12	INT11	INT10	INT9	INT8	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0							
W	w1c																						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							

Figure 19-18. eDMA Interrupt Request Low Register (EDMA\_INTL)

Table 19-16. EDMA\_INTH, EDMA\_INTL Field Descriptions

Field	Description
31–0 INTn	eDMA interrupt request n 0 The interrupt request for channel n is cleared. 1 The interrupt request for channel n is active.

### 19.4.14 eDMA Error Registers (EDMA\_ERRH, EDMA\_ERRL)

The EDMA\_ERR{H,L} provide a bit map for the 64 channels, signaling the presence of an error for each channel. EDMA\_ERRH supports channels 63-32, while EDMA\_ERRL covers channels 31-00. The eDMA 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\_EEI, then logically summed across groups of 16, 32 and 64 channels to form several group error interrupt requests which is then routed to the interrupt controller. During the execution of the interrupt-service routine associated with any DMA errors, it is software's responsibility to clear the appropriate bit, negating the error-interrupt request. Typically, a write to the EDMA\_CERR in the interrupt-service routine is used for this purpose. The normal DMA 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 because a non-zero value indicates the presence of a channel error regardless of the state of the EDMA\_EEI. 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\_CERR. On writes to the EDMA\_ERR, a one in any bit position clears the corresponding channel's error status. A zero in any bit position has no affect on the corresponding channel's current error status. The EDMA\_CERR is provided so the error indicator for a single channel can easily be cleared.

Address: 0xFC04\_4028 (EDMA\_ERRH)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	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
W	w1c															
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	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
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-19. eDMA Error High Register (EDMA\_ERRH)

Address: 0xFC04\_402C (EDMA\_ERRL)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	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
W	w1c															
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	ERR 15	ERR 14	ERR 13	ERR 12	ERR 11	ERR 10	ERR 9	ERR 8	ERR 7	ERR 6	ERR 5	ERR 4	ERR 3	ERR 2	ERR 1	ERR 0
W	w1c															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 19-20. eDMA Error Low Register (EDMA\_ERRL)

**Table 19-17. EDMA\_ERRH, EDMA\_ERRL Field Descriptions**

Field	Description
31–0 ERR <sub>n</sub>	eDMA Error <i>n</i> . 0 An error in channel <i>n</i> has not occurred. 1 An error in channel <i>n</i> has occurred.

### 19.4.15 eDMA Hardware Request Status Registers (EDMA\_HRSH, EDMA\_HRSL)

The EDMA\_HRS{H,L} provide a bit map for the DMA channels, signaling the presence of an error for each channel. EDMA\_HRSH supports channels 63–32, while EDMA\_HRSL covers channels 31–00. The hardware request status bits reflect the current state of the register and qualified (via the EDMA\_ERQ{H,L} fields) DMA request signals as seen by the DMA's arbitration logic. This view into the hardware request signals may be used for debug purposes.

Address: 0xFC04_4030 (EDMA_HRSH)																Access: User read/write				
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	HRS	HRS	HRS	HRS
W	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	63	62	61	60
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	HRS	HRS	HRS	HRS
W	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	47	46	45	44
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 19-21. eDMA Hardware Request Status High Register (EDMA\_HRSH)**

Address: 0xFC04_4034 (EDMA_HRSL)																Access: User read/write				
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	HRS	HRS	HRS	HRS
W	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	31	30	29	28
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	HRS	HRS	HRS	HRS
W	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	w1c	w1c	w1c	w1c

**Figure 19-22. eDMA Hardware Request Status Low Register (EDMA\_HRSL)**

**Table 19-18. EDMA\_HRSH, EDMA\_HRSL Field Descriptions**

Field	Description
31–0 HRS <sub>n</sub>	eDMA hardware request status <i>n</i> . 0 A hardware service request for channel <i>n</i> is not present 1 A hardware service request for channel <i>n</i> is present <b>Note:</b> These bits reflect the state of the request as seen by the arbitration logic. Therefore, this status is affected by the EDMA_ERQ{H,L} bits.

### 19.4.16 eDMA Channel *n* Priority Registers (DCHPRI<sub>n</sub>)

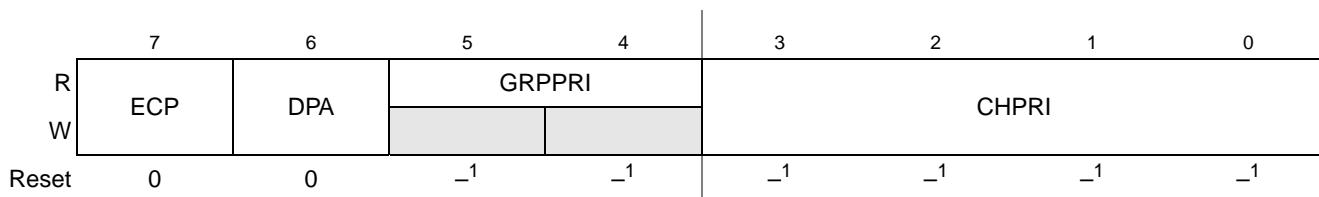
When the fixed-priority channel arbitration mode is enabled (EDMA\_CR[ERCA] = 0), the contents of these registers define the unique priorities associated with each channel within a group. The channel priorities are evaluated by numeric value; for example, 0 is the lowest priority, 1 is the next higher priority, then 2, 3, etc. Software must program the channel priorities with 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 DCHPRI<sub>n</sub> 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 DCHPRI<sub>n</sub> registers. The group priority is assigned in the EDMA\_CR. See [Section 19.4.1, “eDMA Control Register \(EDMA\\_CR\)”](#) for the EDMA\_CR definition.

Channel preemption is enabled on a per-channel basis by setting the DCHPRI<sub>n</sub>[ECP] bit. Channel preemption allows the executing channel’s data transfers to temporarily suspend 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 is requesting 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 preempt another channel can be disabled by setting DCHPRI<sub>n</sub>[DPA]. When a channel’s preempt 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.

Address: 0xFC04\_4100 + *n*, where *n* = 0 – 63 (DCHPRI<sub>n</sub>)

Access: User read/write



<sup>1</sup> Reset value for the group and channel priority fields, GRPPRI and CHPRI, is equal to the corresponding channel number for each priority register, i.e., DCHPRI31[GRPPRI] = 0b01 and DCHPRI31[CHPRI] equals 0b1111.

**Figure 19-23. eDMA Channel *n* Priority Register (DCHPRI<sub>n</sub>)**

**Table 19-19. DCHPRI<sub>n</sub> Field Descriptions**

Field	Description
7 ECP	Enable channel preemption. 0 Channel <i>n</i> cannot be suspended by a higher priority channel's service request. 1 Channel <i>n</i> can be temporarily suspended by the service request of a higher priority channel.
6 DPA	Disable preempt ability. 0 Channel <i>n</i> can suspend a lower priority channel. 1 Channel <i>n</i> cannot suspend any channel, regardless of channel priority.
5–4 GRPPRI	Channel <i>n</i> 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.
3–0 CHPRI	Channel <i>n</i> arbitration priority. Channel priority when fixed-priority arbitration is enabled.

### 19.4.17 Transfer Control Descriptors (TCD*n*)

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. Each TCD*n* definition is presented as 11 registers of 16 or 32 bits. [Table 19-20](#) is a register list of the basic TCD structure.

**Table 19-20. TCD*n* Memory Structure**

eDMA Offset	TCD <i>n</i> Register Name	Abbreviation	Width (bits)
0xFC04_5000 + (0x20 × <i>n</i> )	Source Address	TCD <i>n</i> _SADDR	32
0xFC04_5004 + (0x20 × <i>n</i> )	Transfer Attributes	TCD <i>n</i> _ATTR	16
0xFC04_5006 + (0x20 × <i>n</i> )	Signed Source Address Offset	TCD <i>n</i> _SOFF	16
0xFC04_5008 + (0x20 × <i>n</i> )	Signed Minor Loop Offset	TCD <i>n</i> _MLOFF	24
0xFC04_500B + (0x20 × <i>n</i> )	Minor Byte Count	TCD <i>n</i> _NBYTES	8
0xFC04_500C + (0x20 × <i>n</i> )	Last Source Address Adjustment	TCD <i>n</i> _SLAST	32
0xFC04_5010 + (0x20 × <i>n</i> )	Destination Address	TCD <i>n</i> _DADDR	32
0xFC04_5014 + (0x20 × <i>n</i> )	Current Minor Loop Link, Major Loop Count	TCD <i>n</i> _CITER	16
0xFC04_5016 + (0x20 × <i>n</i> )	Signed Destination Address Offset	TCD <i>n</i> _DOFF	16
0xFC04_5018 + (0x20 × <i>n</i> )	Last Destination Address Adjustment/Scatter Gather Address	TCD <i>n</i> _DLAST_SGA	32
0xFC04_501C + (0x20 × <i>n</i> )	Beginning Minor Loop Link, Major Loop Count	TCD <i>n</i> _BITER	16
0xFC04_501E + (0x20 × <i>n</i> )	Control and Status	TCD <i>n</i> _CSR	16

The following figures and tables define the fields of the TCD $n$  structure:

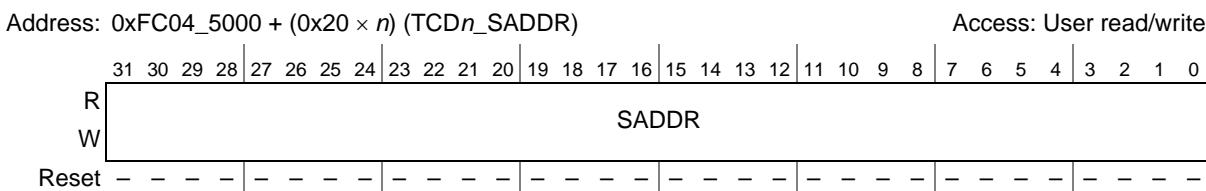


Figure 19-24. TCD $n$  Source Address (TCD $n$ \_SADDR)

Table 19-21. TCD $n$ \_SADDR Field Descriptions

Field	Description
31–0 SADDR	Source address. Memory address pointing to the source data.

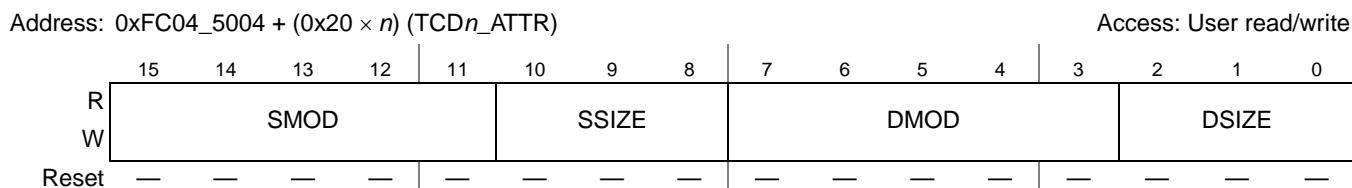


Figure 19-25. TCD $n$  Transfer Attributes (TCD $n$ \_ATTR)

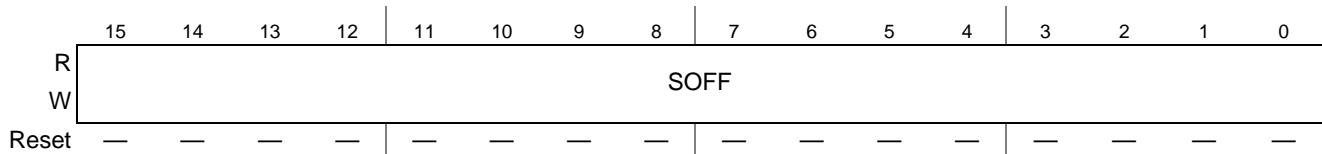
Table 19-22. TCD $n$ \_ATTR Field Descriptions

Field	Description
15–11 SMOD	Source address modulo. 0 Source address modulo feature is disabled. non-0 This value defines a specific address range 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 implement a circular data queue easily. 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 allowed to change. For a 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.
10–8 SSIZE	Source data transfer size. 000 8-bit 001 16-bit 010 32-bit 100 16-byte Else Reserved The attempted use of a Reserved encoding causes a configuration error.
7–3 DMOD	Destination address modulo. See the SMOD definition.
2–0 DSIZE	Destination data transfer size. See the SSIZE definition.

## **Enhanced Direct Memory Access (eDMA)**

Address: 0xFC04\_5006 + (0x20 × n) (TCDn\_SOFF)

## Access: User read/write



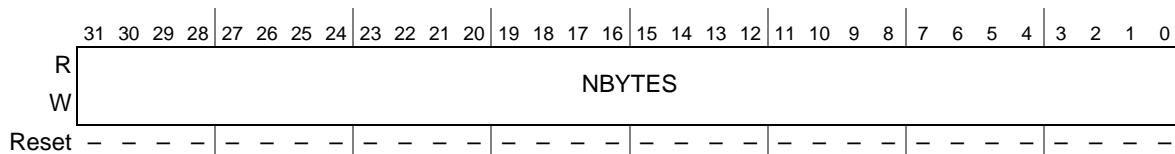
**Figure 19-26. TCDn Signed Source Address Offset (TCDn\_SOFF)**

**Table 19-23. TCDn\_SOFF Field Descriptions**

Field	Description
15–0 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.

If minor loop mapping is disabled (EDMA\_CR[EMLM] = 0), TCD word 2 is defined as in Figure 19-27.

Address: 0xFC04\_5008 + (0x20 × n) (TCDn\_NBYTES) Access: User read/write



**Figure 19-27. TCDn Minor Byte Count (TCDn\_NBYTES)**

**Table 19-24. TCDn\_NBYTES Field Descriptions**

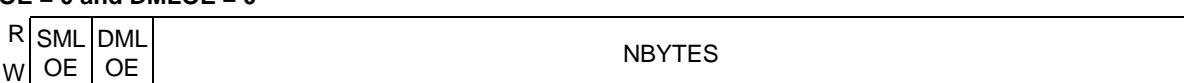
Field	Description
31–0 NBYTES	<p>Minor byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel activates, the appropriate TCD contents load into the eDMA engine, and the appropriate reads and writes perform until the minor byte transfer count has transferred. This is an indivisible operation and cannot be halted. (Although, it may be stalled by using the bandwidth control field, or via preemption.) After the minor count is exhausted, the SADDR and DADDR values are written back into the TCD memory, the major iteration count is decremented and restored to the TCD memory. If the major iteration count is completed, additional processing is performed.</p> <p><b>Note:</b> An NBYTES value of 0x0000_0000 is interpreted as a 4 GB transfer.</p>

If minor loop mapping is enabled (EDMA\_CR[EMLM] = 1), TCD word 2 is defined as in Figure 19-28.

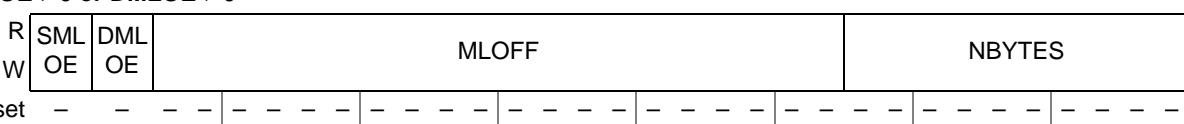
Address: 0xFC04\_5008 + (0x20 ×  $n$ ) (TCD $n$ \_NBYTES) Access: User read/write

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

If SMI QE = 0 and DMI QE = 0



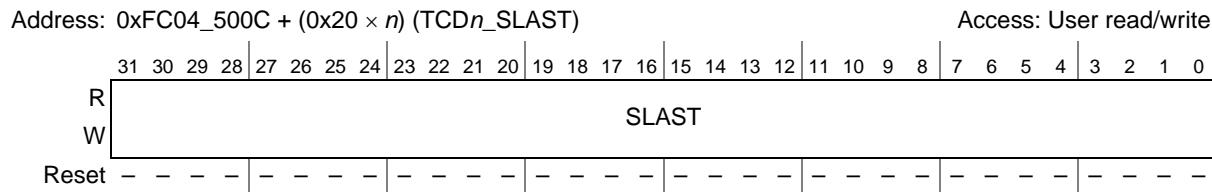
If  $SMLOE \neq 0$  or  $DMLOE \neq 0$



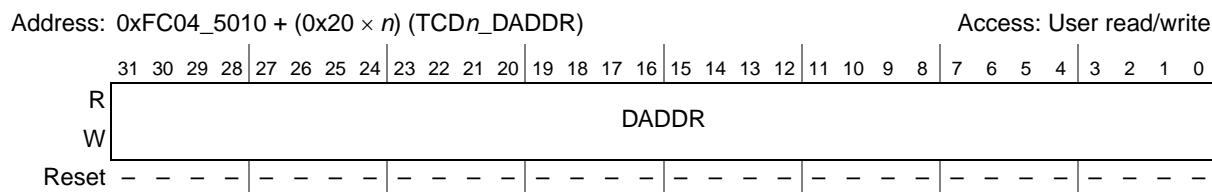
**Figure 19-28. TCDn Minor Byte Count (TCDn\_NBYTES)**

**Table 19-25. TCD $n$ \_NBYTES Field Descriptions**

Field	Description
31 SMLOE	Source minor loop offset enable. 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
30 DMLOE	Destination minor loop offset enable. 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
29–10 Mloff	If either SMLOE or DMLOE is 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 completes.
29–0 or 9–0 NBYTES	If both SMLOE and DMLOE are cleared, this field is 30 bits wide. If not, it is 10 bits wide.  Minor byte transfer count. Number of bytes to be transferred in each service request of the channel. As a channel activates, the appropriate TCD contents load into the eDMA engine, and the appropriate reads and writes perform until the minor byte transfer count has transferred. This is an indivisible operation and cannot be halted. (Although, it may be stalled by using the bandwidth control field, or via preemption.) After the minor count is exhausted, the SADDR and DADDR values are written back into the TCD memory, the major iteration count is decremented and restored to the TCD memory. If the major iteration count is completed, additional processing is performed.

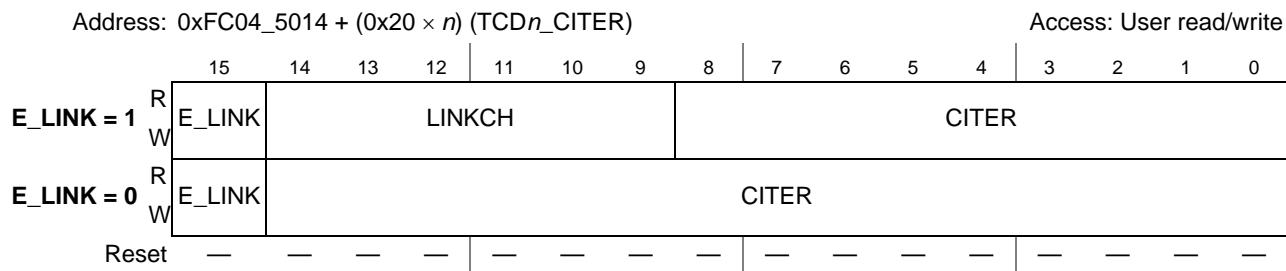
**Figure 19-29. TCD $n$  Source Last Address Adjustment (TCD $n$ \_SLAST)****Table 19-26. TCD $n$ \_SLAST Field Descriptions**

Field	Description
31–0 SLAST	Last source address adjustment. Adjustment value added to the source address at the completion of the 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.

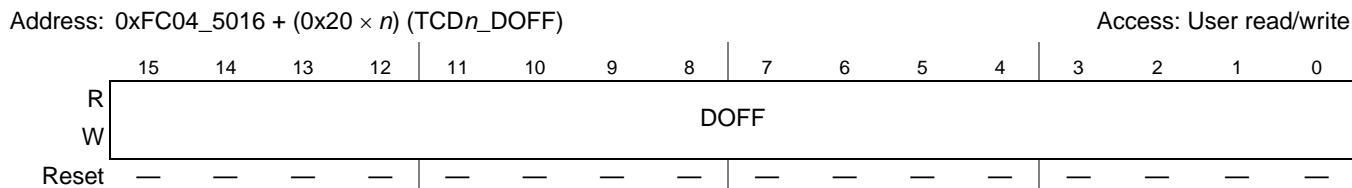
**Figure 19-30. TCD $n$  Destination Address (TCD $n$ \_DADDR)**

**Table 19-27. TCD $n$ \_DADDR Field Descriptions**

Field	Description
31–0 DADDR	Destination address. Memory address pointing to the destination data.

**Figure 19-31. TCD $n$  Current Major Iteration Count (TCD $n$ \_CITER)****Table 19-28. TCD $n$ \_CITER Field Descriptions**

Field	Description
15 E_LINK	<p>Enable channel-to-channel linking on minor-loop complete. As the channel completes the minor loop, this flag enables linking to another channel, defined by the LINKCH field. The link target channel initiates a channel service request via an internal mechanism that sets the TCD<math>n</math>_CSR[START] bit of the specified channel.</p> <p>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.</p> <p>0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.</p> <p><b>Note:</b> This bit must be equal to the BITER.E_LINK bit. Otherwise, a configuration error is reported.</p>
14–9 LINKCH	<p>Link channel number. If channel-to-channel linking is enabled (E_LINK = 1), then after the minor loop is exhausted, the eDMA engine initiates a channel service request to the channel defined by these six bits by setting that channel's TCD<math>n</math>_CSR[START] bit.</p> <p>0–63 Link to DMA channel 0–63</p>
14–0 or 8–0 CITER	<p>Current major iteration count. This 9-bit (E_LINK = 1) or 15-bit (E_LINK = 0) 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 (e.g., 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.</p> <p><b>Note:</b> When the CITER field is initially loaded by software, it must be set to the same value as that contained in the BITER field.</p> <p><b>Note:</b> If the channel is configured to execute a single service request, the initial values of BITER and CITER should be 0x0001.</p>

Figure 19-32. TCD $n$  Destination Address Signed Offset (TCD $n$ \_DOFF)Table 19-29. TCD $n$ \_DOFF Field Descriptions

Field	Description
15–0 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.

Figure 19-33. TCD $n$  Destination Last Address Adjustment (TCD $n$ \_DLAST\_SGA)Table 19-30. TCD $n$ \_DLAST\_SGA Field Descriptions

Field	Description
31–0 DLAST_SGA	Destination last address adjustment or the memory address for the next transfer control descriptor to be loaded into this channel (scatter/gather). If (TCD $n$ _CSR[E_SG] = 0) then <ul style="list-style-type: none"><li>Adjustment value added to the destination address at the completion of the major iteration count. This value can apply to restore the destination address to the initial value or adjust the address to reference the next data structure.</li></ul> else <ul style="list-style-type: none"><li>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, else a configuration error is reported.</li></ul>

Figure 19-34. TCD $n$  Beginning Major Iteration Count (TCD $n$ \_BITER)

Table 19-31. TCDn\_BITER Field Descriptions

Field	Description
15 E_LINK	<p>Enables channel-to-channel linking on minor loop complete. As the channel completes the minor loop, this flag enables the linking to another channel, defined by BITER.LINKCH. The link target channel initiates a channel service request via an internal mechanism that sets the TCDn_CSR[START] bit of the specified channel. If channel linking disables, the BITER value extends 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.</p> <p>0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.</p> <p><b>Note:</b> When the software loads the TCD, 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.</p>
14–9 LINKCH	<p>Link channel number. If channel-to-channel linking is enabled (E_LINK = 1), then after the minor loop is exhausted, the eDMA engine initiates a channel service request at the channel defined by these six bits by setting that channel's TCDn_CSR[START] bit.</p> <p>0–63 Link to DMA channel 0–63</p> <p><b>Note:</b> When the software loads the TCD, 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.</p>
14–0 or 8–0 BITER	<p>Starting major iteration count. As the transfer control descriptor is first loaded by software, this 9-bit (E_LINK = 1) or 15-bit (E_LINK = 0) 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.</p> <p><b>Note:</b> When the software loads the TCD, 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. If the channel is configured to execute a single service request, the initial values of BITER and CITER should be 0x0001.</p>

Address: 0xFC04\_501E + (0x20 × n) (TCDn\_CSR)

Access: User read/write

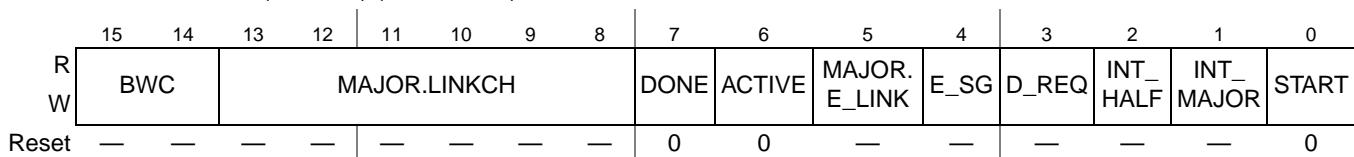


Figure 19-35. TCDn Control and Status (TCDn\_CSR)

**Table 19-32. TCDn\_CSR Field Descriptions**

Field	Description
15–14 BWC	<p>Bandwidth control. Throttles the amount of bus bandwidth consumed by the eDMA. In general, as the eDMA processes the 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 crossbar switch (XBS).</p> <ul style="list-style-type: none"> <li>00 No eDMA engine stalls</li> <li>01 Reserved</li> <li>10 eDMA engine stalls for 4 cycles after each r/w</li> <li>11 eDMA engine stalls for 8 cycles after each r/w</li> </ul> <p><b>Note:</b> If the source and destination sizes are equal, this field is ignored between the first and second transfers and after the last write of each minor loop. This behavior is a side effect of reducing start-up latency.</p>
13–8 MAJOR_LINKCH	<p>Link channel number.</p> <p>If (MAJOR_E_LINK = 0) then</p> <ul style="list-style-type: none"> <li>• No channel-to-channel linking (or chaining) is performed after the major loop counter is exhausted.</li> <li>else</li> <li>• After the major loop counter is exhausted, the eDMA engine initiates a channel service request at the channel defined by these six bits by setting that channel's TCDn_CSR[START] bit.</li> </ul> <p>0–63 Link to DMA channel 0–63</p>
7 DONE	<p>Channel done. This flag indicates the eDMA has completed the major loop. The eDMA engine sets it as the CITER count reaches zero; The software clears it, or the hardware when the channel is activated.</p> <p><b>Note:</b> This bit must be cleared to write the MAJOR_E_LINK or E_SG bits.</p>
6 ACTIVE	<p>Channel active. This flag signals the channel is currently in execution. It is set when channel service begins, and the eDMA clears it as the minor loop completes or if any error condition is detected.</p>
5 MAJOR_E_LINK	<p>Enable channel-to-channel linking on major loop complete. As the channel completes the major loop, this flag enables the linking to another channel, defined by MAJOR_LINKCH. The link target channel initiates a channel service request via an internal mechanism that sets the TCDn_CSR[START] bit of the specified channel.</p> <p><b>Note:</b> To support the dynamic linking coherency model, this field is forced to zero when written to while the TCDn_CSR[DONE] bit is set.</p> <ul style="list-style-type: none"> <li>0 The channel-to-channel linking is disabled.</li> <li>1 The channel-to-channel linking is enabled.</li> </ul>
4 E_SG	<p>Enable scatter/gather processing. As the channel completes the major loop, this flag enables scatter/gather processing in the current channel. If enabled, the eDMA engine uses DLAST_SGA as a memory pointer to a 0-modulo-32 address containing a 32-byte data structure loaded as the transfer control descriptor into the local memory.</p> <p><b>Note:</b> To support the dynamic scatter/gather coherency model, this field is forced to zero when written to while the TCDn_CSR[DONE] bit is set.</p> <ul style="list-style-type: none"> <li>0 The current channel's TCD is normal format.</li> <li>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 major loop completes its execution.</li> </ul>
3 D_REQ	<p>Disable request. If this flag is set, the eDMA hardware automatically clears the corresponding EDMA_ERQ{H,L} bit when the current major iteration count reaches zero.</p> <ul style="list-style-type: none"> <li>0 The channel's EDMA_ERQ{H,L} bit is not affected.</li> <li>1 The channel's EDMA_ERQ{H,L} bit is cleared when the major loop is complete.</li> </ul>

**Table 19-32. TCD $n$ \_CSR Field Descriptions (continued)**

Field	Description
2 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_INT 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. The halfway complete interrupt disables when BITER values are less than two. 0 The half-point interrupt is disabled. 1 The half-point interrupt is enabled.
1 INT_MAJOR	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_INT 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.
0 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.

## 19.5 Functional Description

This section provides an overview of the microarchitecture and functional operation of the eDMA module.

### 19.5.1 eDMA Microarchitecture

The eDMA module is partitioned into two major modules: the eDMA engine and the transfer-control descriptor local memory. Additionally, the eDMA engine is further partitioned into four submodules:

- eDMA Engine
  - Address Path:

This block implements registered versions of two channel transfer control descriptors, channel x and channel y, and manages all master bus-address calculations. All the channels provide the same functionality. This structure allows data transfers associated with one channel to be preempted after the completion of a read/write sequence if a higher priority channel activation 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 provides a mechanism (enabled by DCHPRI $n$ [ECP]) where a large data move operation can be preempted to minimize the time another channel is blocked from execution.

When any channel is selected to execute, the contents of its TCD are read from local memory and loaded into the address path channel x registers for a normal start and into channel y registers for a preemption start. After the minor loop completes execution, the address path hardware writes the new values for the TCD $n$ \_{SADDR, DADDR, CITER} back to local memory. If the major iteration count is exhausted, additional processing are performed, including the final address pointer updates, reloading the TCD $n$ \_CITER field, and a possible fetch of the next TCD $n$  from memory as part of a scatter/gather operation.

— Data Path:

This block implements the bus master read/write datapath. It includes 16 bytes of register storage and the necessary multiplex logic to support any required data alignment. The internal read data bus is the primary input, and the internal write data bus is the primary output.

The address and data path modules directly support the 2-stage pipelined internal bus. The address path module represents the 1st stage of the bus pipeline (address phase), while the data path module implements the 2nd stage of the pipeline (data phase).

— Program Model/Channel Arbitration:

This block implements the first section of the eDMA programming model as well as the channel arbitration logic. The programming model registers are connected to the internal peripheral bus (not shown). The eDMA peripheral request inputs and interrupt request outputs are also connected to this block (via control logic).

— Control:

This block provides all the control functions for the eDMA engine. For data transfers where the source and destination sizes are equal, the eDMA engine performs a series of source read/destination write operations until the number of bytes specified in the minor loop byte count has moved. For descriptors where the sizes are not equal, multiple accesses of the smaller size data are required for each reference of the larger size. As an 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.

- Transfer Control Descriptor Memory

— Memory Controller:

This logic implements the required dual-ported controller, managing accesses from the eDMA engine as well as references from the internal peripheral bus. As noted earlier, in the event of simultaneous accesses, the eDMA engine is given priority and the peripheral transaction is stalled.

— Memory Array: TCD storage is implemented using a single-port, synchronous RAM array.

### 19.5.2 eDMA Basic Data Flow

The basic flow of a data transfer can be partitioned into three segments. As shown in [Figure 19-36](#), the first segment involves the channel activation. In the diagram, this example uses the assertion of the eDMA peripheral request signal to request service for channel  $n$ . Channel activation via software and the  $TCDn\_CSR[START]$  bit follows the same basic flow as peripheral requests. The eDMA request input signal is registered internally and then routed through the eDMA engine: first through the control module, then into the program model and channel arbitration. In the next cycle, the channel arbitration performs, using the fixed-priority or round-robin algorithm. After arbitration is complete, the activated channel number is sent through the address path and converted into the required address to access the local memory for  $TCDn$ . Next, the TCD memory is accessed and the required descriptor read from the local memory and loaded into the eDMA engine address path channel x or y registers. The TCD memory is 64 bits wide to minimize the time needed to fetch the activated channel descriptor and load it into the address path channel x or y registers.

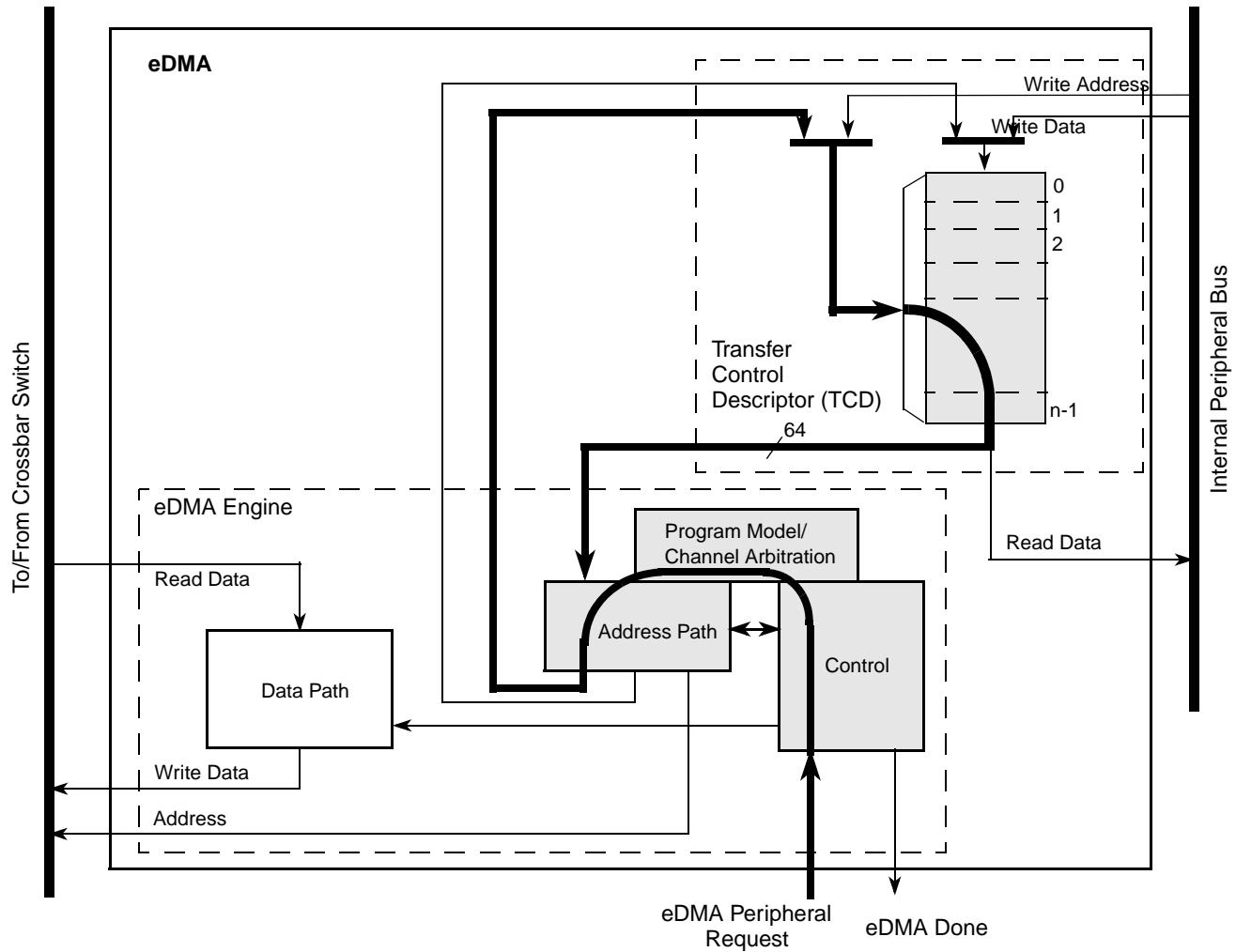


Figure 19-36. eDMA Operation, Part 1

In the second part of the basic data flow (Figure 19-37), 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 block until it is gated onto the internal bus during the destination write. This source read/destination write processing continues until the minor byte count has transferred.

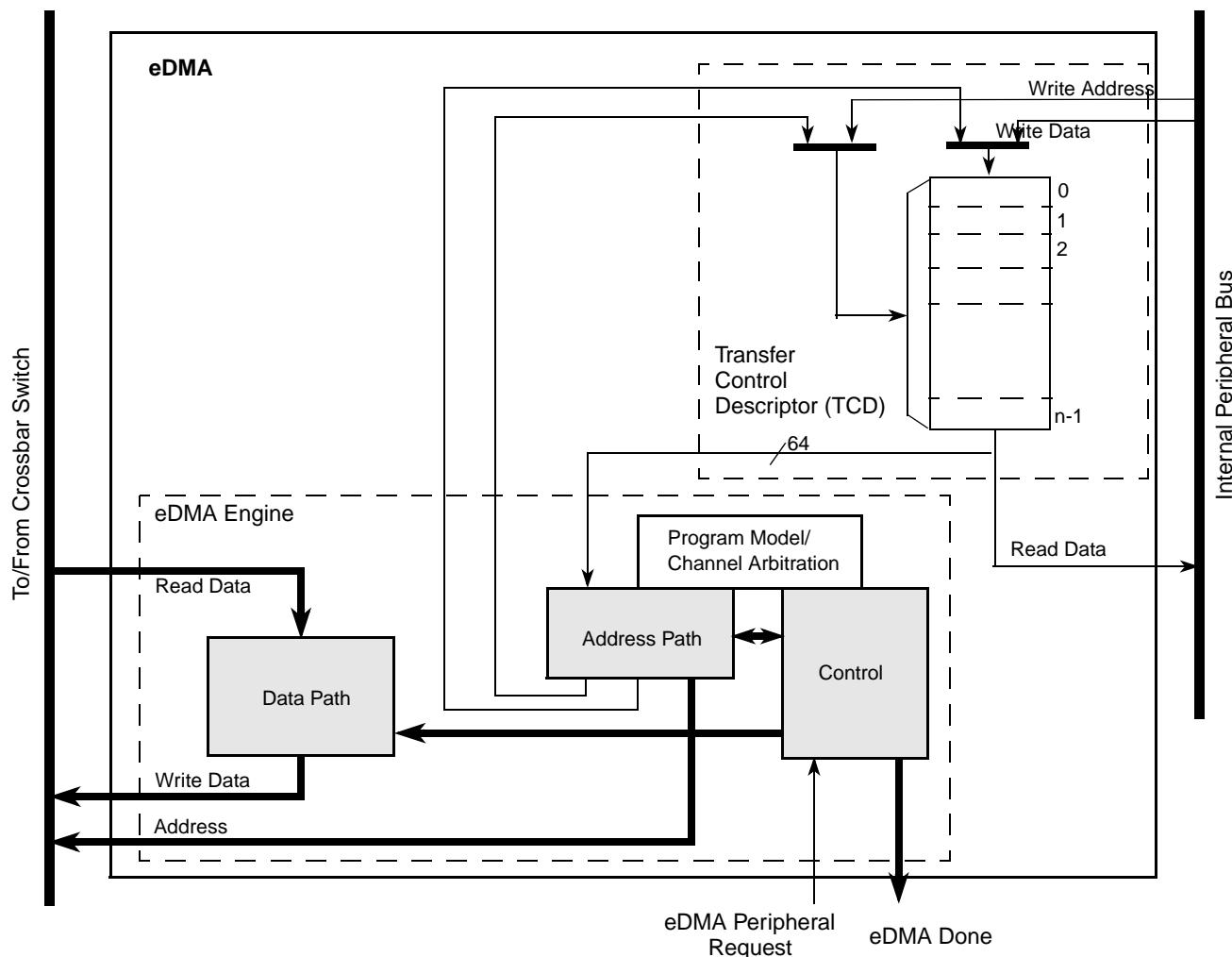


Figure 19-37. eDMA Operation, Part 2

After the minor byte count has moved, the final phase of the basic data flow performs. In this segment, the address path logic performs the required updates to certain fields in the appropriate TCD, e.g., SADDR, DADDR, CITER. If the major iteration count is exhausted, additional operations are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Assertion of an optional interrupt request also 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 19-38](#).

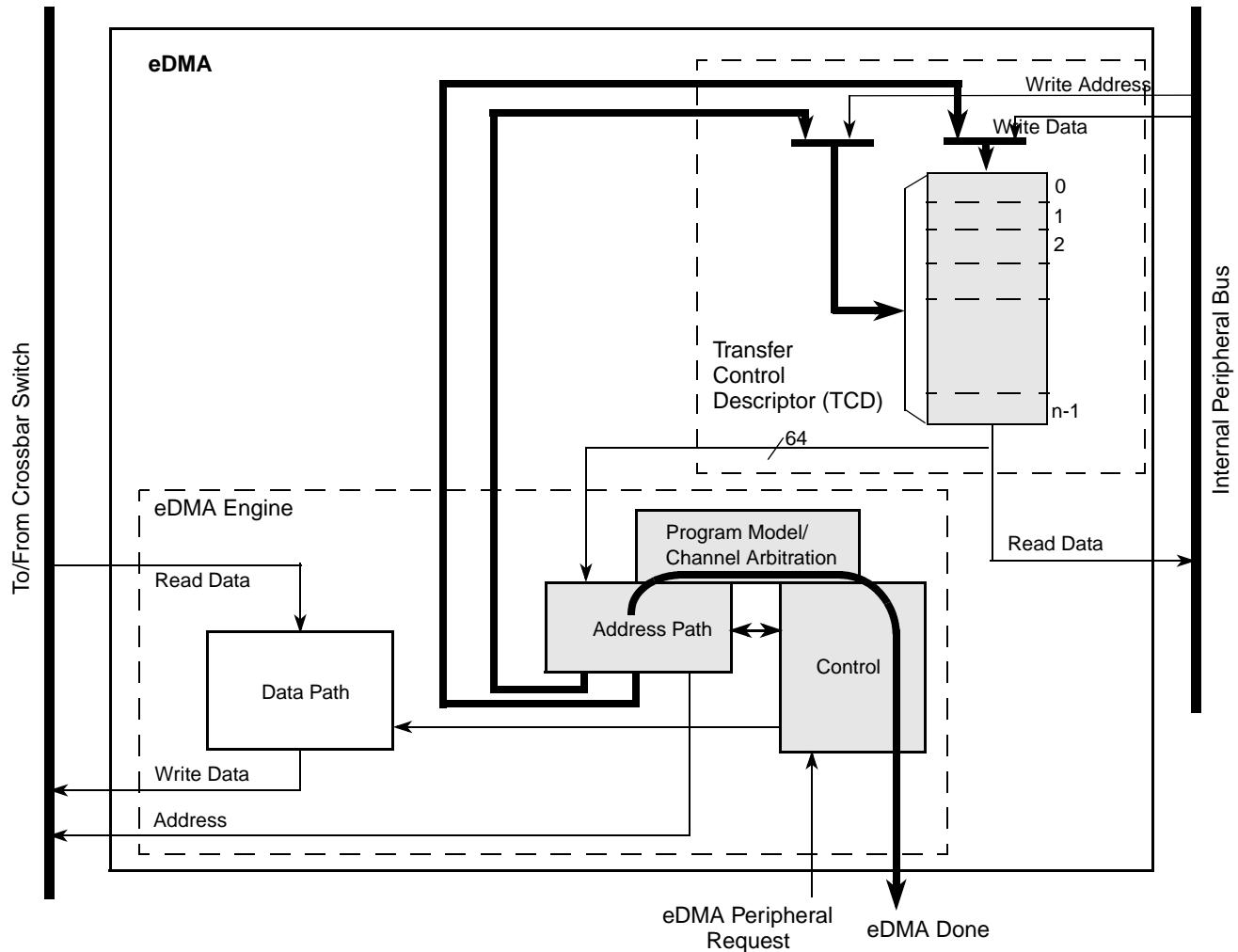


Figure 19-38. eDMA Operation, Part 3

## 19.6 Initialization/Application Information

### 19.6.1 eDMA Initialization

A typical initialization of the eDMA has the following sequence:

1. Write the EDMA\_CR if a configuration other than the default is desired.
2. Write the channel priority levels into the DCHPRI $n$  registers if a configuration other than the default is desired.
3. Enable error interrupts in the EDMA\_EEIs if so desired.
4. Write the 32-byte TCD for each channel that may request service.
5. Enable any hardware service requests via the EDMA\_ERQ.
6. Request channel service by software (setting the TCD $n$ \_CSR[START] bit) or hardware (slave device asserting its eDMA 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 eDMA engine reads the entire TCD, including the TCD control and status fields ([Table 19-33](#)) for the selected channel into its internal address path module. As the TCD is read, the first transfer is initiated on the internal bus unless a configuration error is detected. Transfers from the source (as defined by the source address,  $TCDn\_SADDR$ ) to the destination (as defined by the destination address,  $TCDn\_DADDR$ ) continue until the specified number of bytes ( $TCDn\_NBYTES$ ) are transferred. When transfer is complete, the eDMA engine's local  $TCDn\_SADDR$ ,  $TCDn\_DADDR$ , and  $TCDn\_CITER$  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 executes (interrupts, major loop channel linking, and scatter/gather operations) if enabled.

**Table 19-33. TCD Control and Status Fields**

<b><math>TCDn\_CSR</math> Field Name</b>	<b>Description</b>
START	Control bit to start channel explicitly 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

[Table 19-34](#) 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).

**Table 19-34. Example of Multiple Loop Iterations**

DMA Request	Minor Loop	Major Loop	Current Major Loop Iteration Count (CITER)
			3
DMA Request	Minor Loop	Major Loop	2
DMA Request	Minor Loop		1

Table 19-35 lists the memory array terms and how the TCD settings interrelate.

**Table 19-35. Memory Array Terms**

xADDR: (Starting Address)	xSIZE (size of one data transfer)	Minor Loop (NBYTES in Minor Loop, often the same value as xSIZE)	Offset (xOFF): number of bytes added to current address after each transfer (often the same value as xSIZE)
.	.	Minor Loop	Each DMA source (S) and destination (D) has its own: Address (xADDR) Size (xSIZE) Offset (xOFF) Modulo (xMOD) Last Address Adjustment (xLAST) where $x = S$ or $D$
.	.	Minor Loop	Peripheral queues typically have size and offset equal to NBYTES.
.	.	Last Minor Loop	
xLAST: Number of bytes added to current address after major loop (typically used to loop back)			

## 19.6.2 DMA Programming Errors

The eDMA 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 (EDMA\_ES[GPE]) and channel priority error (EDMA\_ES[CPE]).

For all error types other than Group or channel priority errors, the channel number causing the error is recorded in the EDMA\_ES. 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 once that group has been selected as the active group. For example:

1. The eDMA is configured for fixed group and fixed channel arbitration modes.
2. Group 3 is the highest priority and all channels are unique in that group.
3. Group 2 is the next highest priority and has two channels with the same priority level.
4. If Group 3 has any service requests, those requests will be executed.
5. Once all of Group 3 requests have completed, Group 2 will be the next active group.
6. If Group 2 has a service request, then an undefined channel in Group 2 will be selected and a channel priority error will occur.
7. This will repeat until all of Group 2 requests have been removed or a higher priority Group 3 request comes in.

In this sequence, for item 2, the eDMA Acknowledge lines will assert only if the selected channel is requesting service via the eDMA peripheral request signal. If interrupts are enabled for all channels, the user will get an error interrupt, but the channel number for the EDMA\_ERR and the error interrupt request line may be wrong because they reflect the selected channel. A group priority error is global and any request in any group will cause a group priority error.

If priority levels are not unique, when any channel requests service, a channel priority error is reported. The highest (channel/group) priority with an active request is selected, but the lowest numbered (channel/group) with that priority is selected by arbitration and executed by the eDMA engine. The hardware service request handshake signals, error interrupts, and error reporting is associated with the selected channel.

## 19.6.3 DMA Arbitration Mode Considerations

### 19.6.3.1 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 - i.e. no other groups will 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.

### 19.6.3.2 Round Robin Group Arbitration, Fixed Channel Arbitration

The occurrence of one or more DMA requests from one or more groups, the channel with the highest priority from a specific group will be 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.

Once the channel request is serviced, the group round robin algorithm will select 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 just 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 will always be serviced before lower priority channels in the same group, and thus the lower priority channels will never be serviced. The advantage of this scenario is that no one group will 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 could prevent the servicing of lower priority channels in the same group.

### 19.6.3.3 Round Robin Group Arbitration, Round Robin Channel Arbitration

Groups will be serviced as described in Section 19.6.3.2, but this time channels will be serviced in channel number order. Only one channel 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 will 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 will be serviced.

This scenario ensures that all channels will be guaranteed service at some point, regardless of the request rates. However, the potential latency could be quite high. All channels are treated equally. Priority levels are not used in round robin/round robin mode.

### 19.6.3.4 Fixed Group Arbitration, Round Robin Channel Arbitration

The highest priority group with a request will be serviced. Lower priority groups will be 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 19.6.3.1, but all the channels in the highest priority group will get serviced. Service latency will be short on the highest priority group, but could potentially get very much longer and longer as the group priority decreases.

## 19.6.4 DMA Transfer

### 19.6.4.1 Single Request

To perform a simple transfer of n bytes of data with one activation, set the major loop to one ( $TCDn\_CITER = TCDn\_BITER = 1$ ). The data transfer begins after the channel service request is acknowledged and the channel is selected to execute. After the transfer is complete, the  $TCDn\_CSR[DONE]$  bit is set and an interrupt generates 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 longword-wide port located at 0x2000. The address offsets are programmed in increments to match the transfer size: 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.

#### Example 19-1. Single Request DMA Transfer

```
TCDn_CITER = TCDn_BITER = 1
TCDn_NBYTES = 16
```

#### Enhanced Direct Memory Access (eDMA)

```
TCDn_SADDR = 0x1000
TCDn_SOFF = 1
TCDn_ATTR[SSIZE] = 0
TCDn_SLAST = -16
TCDn_DADDR = 0x2000
TCDn_DOFF = 4
TCDn_ATTR[DSIZE] = 2
TCDn_DLAST_SGA= -16
TCDn_CSR[INT_MAJ] = 1
TCDn_CSR[START] = 1 (Should be written last after all other fields have been initialized)
All other TCDn fields = 0
```

This generates the following event sequence:

1. User write to the TCDn\_CSR[START] bit requests channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes: TCDn\_CSR[DONE] = 0, TCDn\_CSR[START] = 0, TCDn\_CSR[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 from location 0x1000, read byte from location 0x1001, read byte from 0x1002, read byte from 0x1003.
  - b) Write longword to location 0x2000 → first iteration of the minor loop.
  - c) Read byte from location 0x1004, read byte from location 0x1005, read byte from 0x1006, read byte from 0x1007.
  - d) Write longword to location 0x2004 → second iteration of the minor loop.
  - e) Read byte from location 0x1008, read byte from location 0x1009, read byte from 0x100A, read byte from 0x100B.
  - f) Write longword to location 0x2008 → third iteration of the minor loop.
  - g) Read byte from location 0x100C, read byte from location 0x100D, read byte from 0x100E, read byte from 0x100F.
  - h) Write longword to location 0x200C → last iteration of the minor loop → major loop complete.
6. The eDMA engine writes: TCDn\_SADDR = 0x1000, TCDn\_DADDR = 0x2000, TCDn\_CITER = 1 (TCDn\_BITER).
7. The eDMA engine writes: TCDn\_CSR[ACTIVE] = 0, TCDn\_CSR[DONE] = 1, EDMA\_INT[n] = 1.
8. The channel retires and the eDMA goes idle or services the next channel.

#### 19.6.4.2 Multiple Requests

Besides transferring 32 bytes via two hardware requests, the next example is the same as previous. 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 EDMA\_ERQ, the slave device initiates channel service requests.

```

TCDn_CITER = TCDn_BITER = 2
TCDn_SLAST = -32
TCDn_DLAST_SGA = -32

```

This would generate the following sequence of events:

1. First hardware (eDMA peripheral) request for channel service.
2. The channel is selected by arbitration for servicing.
3. eDMA engine writes:  $TCDn\_CSR[DONE] = 0$ ,  $TCDn\_CSR[START] = 0$ ,  $TCDn\_CSR[ACTIVE] = 1$ .
4. eDMA engine reads: channel  $TCDn$  data from local memory to internal register file.
5. The source to destination transfers are executed as follows:
  - a) Read byte from location 0x1000, read byte from location 0x1001, read byte from 0x1002, read byte from 0x1003.
  - b) Write longword to location 0x2000 → first iteration of the minor loop.
  - c) Read byte from location 0x1004, read byte from location 0x1005, read byte from 0x1006, read byte from 0x1007.
  - d) Write longword to location 0x2004 → second iteration of the minor loop.
  - e) Read byte from location 0x1008, read byte from location 0x1009, read byte from 0x100A, read byte from 0x100B.
  - f) Write longword to location 0x2008 → third iteration of the minor loop.
  - g) Read byte from location 0x100C, read byte from location 0x100D, read byte from 0x100E, read byte from 0x100F.
  - h) Write longword to location 0x200C → last iteration of the minor loop.
6. eDMA engine writes:  $TCDn\_SADDR = 0x1010$ ,  $TCDn\_DADDR = 0x2010$ ,  $TCDn\_CITER = 1$ .
7. eDMA engine writes:  $TCDn\_CSR[ACTIVE] = 0$ .
8. The channel retires → one iteration of the major loop. The eDMA goes idle or services the next channel.
9. Second hardware (eDMA peripheral) requests channel service.
10. The channel is selected by arbitration for servicing.
11. eDMA engine writes:  $TCDn\_CSR[DONE] = 0$ ,  $TCDn\_CSR[START] = 0$ ,  $TCDn\_CSR[ACTIVE] = 1$ .
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 from location 0x1010, read byte from location 0x1011, read byte from 0x1012, read byte from 0x1013.
  - b) Write longword to location 0x2010 → first iteration of the minor loop.
  - c) Read byte from location 0x1014, read byte from location 0x1015, read byte from 0x1016, read byte from 0x1017.
  - d) Write longword to location 0x2014 → second iteration of the minor loop.

- e) Read byte from location 0x1018, read byte from location 0x1019, read byte from 0x101A, read byte from 0x101B.
  - f) Write longword to location 0x2018 → third iteration of the minor loop.
  - g) Read byte from location 0x101C, read byte from location 0x101D, read byte from 0x101E, read byte from 0x101F.
  - h) Write longword to location 0x201C → last iteration of the minor loop → major loop complete.
14. eDMA engine writes:  $TCDn\_SADDR = 0x1000$ ,  $TCDn\_DADDR = 0x2000$ ,  $TCDn\_CITER = 2$  ( $TCDn\_BITER$ ).
15. eDMA engine writes:  $TCDn\_CSR[ACTIVE] = 0$ ,  $TCDn\_CSR[DONE] = 1$ ,  $EDMA\_INT[n] = 1$ .
16. The channel retires → major loop complete. The eDMA goes idle or services the next channel.

#### 19.6.4.3 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 2. MOD is a 5-bit field for the source and destination in the TCD, and it specifies which lower address bits 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 19-36** 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 19-36. Modulo Feature Example**

Transfer Number	Address
1	0x12345670
2	0x12345674
3	0x12345678
4	0x1234567C
5	0x12345670
6	0x12345674

#### 19.6.5 eDMA $TCDn$ Status Monitoring

##### 19.6.5.1 Minor Loop Complete

There are two methods to test for minor loop completion when using software initiated service requests. The first is to read the  $TCDn\_CITER$  field and test for a change. (Another method may be extracted from the sequence shown below). The second method is to test the  $TCDn\_CSR[START]$  bit and the  $TCDn\_CSR[ACTIVE]$  bit. The minor-loop-complete condition is indicated by both bits reading zero after

the `TCDn_CSR[START]` was set. Polling the `TCDn_CSR[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:

	TCD $n$ _CSR bits			State
	START	ACTIVE	DONE	
1	1	0	0	Channel service request via software
2	0	1	0	Channel is executing
3a	0	0	0	Channel has completed the minor loop and is idle
3b	0	0	1	Channel has completed the major loop and is idle

The best method to test for minor-loop completion when using hardware (peripheral) initiated service requests is to read the `TCDn_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:

	TCD $n$ _CSR bits			State
	START	ACTIVE	DONE	
1	0	0	0	Channel service request via hardware (peripheral request asserted)
2	0	1	0	Channel is executing
3a	0	0	0	Channel has completed the minor loop and is idle
3b	0	0	1	Channel has completed the major loop and is idle

For both activation types, the major-loop-complete status is explicitly indicated via the `TCDn_CSR[DONE]` bit.

The `TCDn_CSR[START]` bit is cleared automatically when the channel begins execution regardless of how the channel activates.

### 19.6.5.2 Active Channel TCD $n$ Reads

The eDMA reads back the true `TCDn_SADDR`, `TCDn_DADDR`, and `TCDn_NBYTES` values if read while a channel executes. The true values of the `SADDR`, `DADDR`, and `NBYTES` are the values the eDMA engine currently uses 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.

### 19.6.5.3 Preemption Status

Preemption is available only when fixed arbitration is selected for both group and channel arbitration modes. A preemptive situation is one in which a preempt-enabled channel runs 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 actively running relative priority outstanding requests become undefined. Channel and/or group priorities are treated as equal (constantly rotating) when round-robin arbitration mode is selected.

The TCD $n$ \_CSR[ACTIVE] bit for the preempted channel remains asserted throughout the preemption. The preempted channel is temporarily suspended while the preempting channel executes one major loop iteration. If two TCD $n$ \_CSR[ACTIVE] bits are set simultaneously in the global TCD map, a higher priority channel is actively preempting a lower priority channel.

## 19.6.6 Channel Linking

Channel linking (or chaining) is a mechanism where one channel sets the TCD $n$ \_CSR[START] bit of another channel (or itself), therefore initiating a service request for that channel. When properly enabled, the EDMA engine automatically performs this operation at the major or minor loop completion.

The minor loop channel linking occurs at the completion of the minor loop (or one iteration of the major loop). The TCD $n$ \_CITER[E\_LINK] field determines whether a minor loop link is requested. When enabled, the channel link is made after each iteration of the major 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, the initial fields of:

```
TCD $n$ _CITER[E_LINK] = 1
TCD $n$ _CITER[LINKCH] = 0xc
TCD $n$ _CITER[CITER] value = 0x4
TCD $n$ _CSR[MAJOR_E_LINK] = 1
TCD $n$ _CSR[MAJOR_LINKCH] = 0x7
```

executes as:

1. Minor loop done → set TCD12\_CSR[START] bit
2. Minor loop done → set TCD12\_CSR[START] bit
3. Minor loop done → set TCD12\_CSR[START] bit
4. Minor loop done, major loop done → set TCD7\_CSR[START] bit

When minor loop linking is enabled (TCD $n$ \_CITER[E\_LINK] = 1), the TCD $n$ \_CITER[CITER] field uses a nine bit vector to form the current iteration count. When minor loop linking is disabled (TCD $n$ \_CITER[E\_LINK] = 0), the TCD $n$ \_CITER[CITER] field uses a 15-bit vector to form the current iteration count. The bits associated with the TCD $n$ \_CITER[LINKCH] field are concatenated onto the CITER value to increase the range of the CITER.

**NOTE**

The TCD $n$ \_CITER[E\_LINK] bit and the TCD $n$ \_BITER[E\_LINK] bit must equal or a configuration error is reported. The CITER and BITER vector widths must be equal to calculate the major loop, half-way done interrupt point.

**Table 19-37** 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 19-37. 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	CSR[MAJOR_E_LINK]	Enable channel-to-channel linking on major loop completion
	CSR[MAJOR_LINKCH]	Link channel number when linking at end of major loop

## 19.6.7 Dynamic Programming

This section provides recommended methods to change the programming model during channel execution.

### 19.6.7.1 Dynamic Channel Linking and Dynamic Scatter/Gather

Dynamic channel linking and dynamic scatter/gather is the process of changing the TCD $n$ \_CSR[MAJOR\_E\_LINK] or TCD $n$ \_CSR[E\_SG] bits during channel execution. These bits are read from the TCD local memory at the end of channel execution, therefore allowing software to enable either feature during channel execution.

Because software can change the configuration during execution, a coherency sequence must be followed. Consider the scenario the user attempts to execute a dynamic channel link by enabling the TCD $n$ \_CSR[MAJOR\_E\_LINK] bit as the eDMA engine retires the channel. The TCD $n$ \_CSR[MAJOR\_E\_LINK] would be set in the programmer's model, but it would be indeterminate whether the actual link was made before the channel retired.

The following coherency sequence is recommended when executing a dynamic channel link or dynamic scatter/gather request:

1. Set the TCD $n$ \_CSR[MAJOR\_E\_LINK] bit.
2. Read back the TCD $n$ \_CSR[MAJOR\_E\_LINK] bit.
3. Test the TCD $n$ \_CSR[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 TCD $n$ \_CSR[MAJOR\_E\_LINK] and TCD $n$ \_CSR[E\_SG] bits to

zero on any writes to a  $TCDn$  after the  $TCDn\_CSR[DONE]$  bit for that channel is set, indicating the major loop is complete.

**NOTE**

Software must clear the  $TCDn\_CSR[DONE]$  bit before writing the  $TCDn\_CSR[MAJOR\_E\_LINK]$  or  $TCDn\_CSR[E\_SG]$  bits. The  $TCDn\_CSR[DONE]$  bit is cleared automatically by the eDMA engine after a channel begins execution.

# Chapter 20

## FlexBus

### 20.1 Introduction

This chapter describes external bus data transfer operations and error conditions. It describes transfers initiated by the ColdFire processor (or any other bus master) and includes detailed timing diagrams showing the interaction of signals in supported bus operations.

#### NOTE

- In this chapter, unless otherwise noted, clock refers to the FB\_CLK used for the external bus ( $f_{sys}/2$  or  $f_{sys}/4$  depending on MISCCR2[FBHALF] in the chip configuration module).
- This device's pinout does not contain separate address and data signals. Therefore, non-multiplexed mode cannot be used. Ignore any references to non-multiplexed address/data throughout this chapter.

#### 20.1.1 Overview

A multi-function external bus interface called the FlexBus interface controller is provided on the device with basic functionality of interfacing to slave-only devices. It can be directly connected to the following asynchronous or synchronous devices with little or no additional circuitry:

- External boot ROMs
- Flash memories
- Programmable logic devices
- Other simple target (slave) devices

For asynchronous devices, a simple chip-select based interface can be used.

The FlexBus interface has up to six general purpose chip-selects,  $\overline{FB\_CS}[5:0]$ . The actual number of chip selects available depends upon the device and its pin configuration. Chip-select  $\overline{FB\_CS0}$  can be dedicated to boot memory access and programmed to be byte (8 bits), word (16 bits), or longword (32 bits) wide. Control signal timing is compatible with common ROM and flash memories.

#### 20.1.2 Features

Key FlexBus features include:

- Six independent, user-programmable chip-select signals ( $\overline{FB\_CS}[5:0]$ ) that can interface with external SRAM, PROM, EPROM, EEPROM, flash, and other peripherals
- 8-, 16-, and 32-bit port sizes with configuration for multiplexed address and data buses

- Byte-, word-, longword-, and 16-byte line-sized transfers
- Programmable burst- and burst-inhibited transfers selectable for each chip select and transfer direction
- Programmable address-setup time with respect to the assertion of chip select
- Programmable address-hold time with respect to the negation of chip select and transfer direction

### 20.1.3 Modes of Operation

The external interface is a configurable multiplexed bus set to one of the following modes:

- Multiplexed 32-bit address and 32-bit data
- Multiplexed 32-bit address and 16-bit data (non-multiplexed 16-bit address and 16-bit data)
- Multiplexed 32-bit address and 8-bit data (non-multiplexed 24-bit address and 8-bit data)

## 20.2 External Signals

This section describes the external signals involved in data-transfer operations.

**Table 20-1. FlexBus Signal Summary**

Signal Name	I/O <sup>1</sup>	Description
FB_AD[31:0]	I/O	Address/data bus, FB_AD[31:0].
FB_CS[5:0]	O	General purpose chip-selects. The actual number of chip selects available depends upon the device and its pin configuration. See <a href="#">Table 2-2</a> for more details.
FB_BE/BWE[3:0]	O	Byte enable/byte write enable
FB_OE	O	Output enable
FB_R/W	O	Read/write. 1 = Read, 0 = Write
FB_ALE	O	Address latch enable
FB_TSIZ[1:0]	O	Transfer size
FB_TBST	O	Burst transfer indicator
FB_TA	I	Transfer acknowledge

<sup>1</sup> Because this device shares the FlexBus signals with the NAND flash controller, these signal directions are only valid when the FlexBus controls them. The directions may change during NAND flash cycles.

### 20.2.1 Address and Data Buses (FB\_ADn)

The number of byte lanes carrying the data is determined by the port size associated with the matching chip select.

In multiplexed mode, the FB\_ADn bus carries the address and data. The full 32-bit address is driven on the first clock of a bus cycle (address phase). Following the first clock, the data is driven on the bus (data

phase). During the data phase, the address continues driving on the pins not used for data. For example, in 16-bit mode the address continues driving on FB\_AD[15:0] and in 8-bit mode the address continues driving on FB\_AD[23:0].

Because this device shares the FlexBus signals with the NAND flash controller, these signals tristate between bus cycles.

### **20.2.2 Chip Selects (FB\_CS[5:0])**

The chip-select signal indicates which device is selected. A particular chip-select asserts when the transfer address is within the device's address space, as defined in the base- and mask-address registers. The actual number of chip selects available depends upon the pin configuration.

### **20.2.3 Byte Enables/Byte Write Enables (FB\_BE/BWE[3:0])**

When driven low, the byte enable (FB\_BE/BWE[3:0]) outputs indicate data is to be latched or driven onto a byte of the data bus. FB\_BE/BWE<sub>n</sub> signals are asserted only to the memory bytes used during read or write accesses. A configuration option is provided to assert these signals on reads and writes (byte enable) or writes only (byte-write enable).

The FB\_BE/BWE<sub>n</sub> signals are asserted during accesses to on-chip peripherals but not to on-chip SRAM or cache. For external SRAM or flash devices, the FB\_BE/BWE<sub>n</sub> outputs must be connected to individual byte strobe signals.

### **20.2.4 Output Enable (FB\_OE)**

The output enable signal (FB\_OE) is sent to the interfacing memory and/or peripheral to enable a read transfer. FB\_OE is only asserted during read accesses when a chip select matches the current address decode.

Because this device shares the FlexBus signals with the NAND flash controller, this signal tristates between bus cycles.

### **20.2.5 Read/Write (FB\_R/W)**

The processor drives the FB\_R/W signal to indicate the current bus operation direction. It is driven high during read bus cycles and low during write bus cycles.

Because this device shares the FlexBus signals with the NAND flash controller, this signal tristates between bus cycles.

### **20.2.6 Address Latch Enable (FB\_ALE)**

The assertion of FB\_ALE indicates that the device has begun a bus transaction and the address and attributes are valid. FB\_ALE is asserted for one bus clock cycle. FB\_ALE may be used externally to capture the bus transfer address ([Figure 20-8](#)). This device can extend this signal until the first positive clock edge after FB\_CS<sub>n</sub> asserts. See CSCR<sub>n</sub>[EXTS] and [Section 20.4.9, “Extended Transfer Start”](#).

Because this device shares the FlexBus signals with the NAND flash controller, this signal tristates between bus cycles.

### 20.2.7 Transfer Size (FB\_TSIZ[1:0])

For memory accesses, these signals, along with  $\overline{\text{FB\_TBST}}$ , indicate the data transfer size of the current bus operation. The interface supports byte, word, and longword operand transfers and allows accesses to 8-, 16-, and 32-bit data ports.

For misaligned transfers, FB\_TSIZ[1:0] indicates the size of each transfer. For example, if a longword access through a 32-bit port device occurs at a misaligned offset of 0x1, a byte is transferred first (FB\_TSIZ[1:0] = 01), a word is transferred next at offset 0x2 (FB\_TSIZ[1:0] = 10), and the final byte is transferred at offset 0x4 (FB\_TSIZ[1:0] = 01).

For aligned transfers larger than the port size, FB\_TSIZ[1:0] behaves as follows:

- If bursting is used, FB\_TSIZ[1:0] is driven to the transfer size.
- If bursting is inhibited, FB\_TSIZ[1:0] first shows the entire transfer size and then shows the port size.

**Table 20-2. Data Transfer Size**

FB_TSIZ[1:0]	Transfer Size
00	4 bytes (longword)
01	1 byte
10	2 bytes (word)
11	16 bytes (line)

For burst-inhibited transfers, FB\_TSIZ[1:0] changes with each  $\overline{\text{FB\_TS}}$  assertion to reflect the next transfer size. For transfers to port sizes smaller than the transfer size, FB\_TSIZ[1:0] indicates the size of the entire transfer on the first access and the size of the current port transfer on subsequent transfers. For example, for a longword write to an 8-bit port, FB\_TSIZ[1:0] equals 00 for the first transaction and 01 for the next three transactions. If bursting is used for longword write to an 8-bit port, FB\_TSIZ[1:0] is driven to 00 for the entire transfer.

### 20.2.8 Transfer Burst (FB\_TBST)

Transfer burst indicates that a burst transfer is in progress as driven by the device. A burst transfer can be two to 16 beats depending on FB\_TSIZ[1:0] and the port size.

#### NOTE

When burst ( $\overline{\text{FB\_TBST}} = 0$ ), transfer size is 16 bytes (FB\_TSIZ[1:0] = 11) and the address is misaligned within the 16-byte boundary, the external device must be able to wrap around the address.

## 20.2.9 Transfer Acknowledge (FB\_TA)

This signal indicates the external data transfer is complete. When the processor recognizes  $\overline{\text{FB\_TA}}$  during a read cycle, it latches the data and then terminates the bus cycle. When the processor recognizes  $\overline{\text{FB\_TA}}$  during a write cycle, the bus cycle is terminated.

If auto-acknowledge is disabled ( $\text{CSCR}_n[\text{AA}] = 0$ ), the external device drives  $\overline{\text{FB\_TA}}$  to terminate the bus transfer; if auto-acknowledge is enabled ( $\text{CSCR}_n[\text{AA}] = 1$ ),  $\overline{\text{FB\_TA}}$  is generated internally after a specified number of wait states, or the external device may assert external  $\overline{\text{FB\_TA}}$  before the wait-state countdown, terminating the cycle early. The device negates  $\overline{\text{FB\_CS}_n}$  one cycle after the last  $\overline{\text{FB\_TA}}$  asserts. During read cycles, the peripheral must continue to drive data until  $\overline{\text{FB\_TA}}$  is recognized. For write cycles, the processor continues driving data one clock after  $\overline{\text{FB\_CS}_n}$  is negated.

The number of wait states is determined by  $\text{CSCR}_n$  or the external  $\overline{\text{FB\_TA}}$  input. If the external  $\overline{\text{FB\_TA}}$  is used, the peripheral has total control on the number of wait states.

### NOTE

External devices should only assert  $\overline{\text{FB\_TA}}$  while the  $\overline{\text{FB\_CS}_n}$  signal to the external device is asserted.

Because this device shares the FlexBus signals with the NAND flash controller, this signal tristates between bus cycles.

## 20.3 Memory Map/Register Definition

The following tables describe the registers and bit meanings for configuring chip-select operation.

[Table 20-3](#) shows the chip-select register memory map.

The actual number of chip select registers available depends upon the device and its pin configuration. If the device does not support certain chip select signals or the pin is not configured for a chip-select function, then that corresponding set of chip-select registers has no effect on an external pin.

### NOTE

You must set  $\text{CSMR}_0[\text{V}]$  before the chip select registers take effect.

**Table 20-3. FlexBus Chip Select Memory Map**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
$0x\text{FC}00\_8000 + (n \times 0xC)$	Chip-Select Address Register ( $\text{CSAR}_n$ ) $n = 0 - 5$	32	R/W	0x0000_0000	<a href="#">20.3.1/20-6</a>
$0x\text{FC}00\_8004 + (n \times 0xC)$	Chip-Select Mask Register ( $\text{CSMR}_n$ ) $n = 0 - 5$	32	R/W	0x0000_0000	<a href="#">20.3.2/20-6</a>
$0x\text{FC}00\_8008 + (n \times 0xC)$	Chip-Select Control Register ( $\text{CSCR}_n$ ) $n = 0 - 5$	32	R/W	See Section	<a href="#">20.3.3/20-7</a>

### 20.3.1 Chip-Select Address Registers (CSAR0 – CSAR5)

The CSAR $n$  registers specify the chip-select base addresses.

## **NOTE**

Because the FlexBus module is one of the slaves connected to the crossbar switch, it is only accessible within a certain memory range. The only applicable address ranges for which the chip-selects can be active are 0x0000\_0000 – 0x3FFF\_FFFF and 0xC000\_0000 – 0xDFFF\_FFFF. Set the CSAR $n$  registers appropriately.

**Figure 20-1. Chip-Select Address Registers (CSAR $n$ )**

**Table 20-4. CSARn Field Descriptions**

Field	Description
31–16 BA	Base address. Defines the base address for memory dedicated to chip-select $\overline{FB\_CSn}$ . BA is compared to bits 31–16 on the internal address bus to determine if chip-select memory is being accessed.
15–0	Reserved, must be cleared.

### 20.3.2 Chip-Select Mask Registers (CSMR0 – CSMR5)

CSMRn registers specify the address mask and allowable access types for the respective chip-selects.

Address: 0xFC00_8004 (CSMR0)	Access: User read/write
0xFC00_8010 (CSMR1)	
0xFC00_801C (CSMR2)	
0xFC00_8028 (CSMR3)	
0xFC00_8034 (CSMR4)	
0xFC00_8040 (CSMR5)	

**Figure 20-2. Chip-Select Mask Registers (CSMR<sub>n</sub>)**

**Table 20-5. CSMR $n$  Field Descriptions**

Field	Description
31–16 BAM	<p>Base address mask. Defines the chip-select block size by masking address bits. Setting a BAM bit causes the corresponding CSAR bit to be a don't care in the decode.</p> <p>0 Corresponding address bit is used in chip-select decode. 1 Corresponding address bit is a don't care in chip-select decode.</p> <p>The block size for <math>\overline{FB\_CS}_n</math> is <math>2^n</math>; <math>n = (\text{number of bits set in respective CSMR[BAM]}) + 16</math>. For example, if CSAR0 equals 0x0000 and CSMR0[BAM] equals 0x0008, <math>\overline{FB\_CS}_0</math> addresses two discontinuous 64 KB memory blocks: one from 0x40_0000 – 0x40_FFFF and one from 0x48_0000 – 0x48_FFFF. Likewise, for <math>\overline{FB\_CS}_0</math> to access 32 MB of address space starting at location 0x00_0000, <math>\overline{FB\_CS}_1</math> must begin at the next byte after <math>\overline{FB\_CS}_0</math> for a 16 MB address space. Therefore, CSAR0 equals 0x0000, CSMR0[BAM] equals 0x01FF, CSAR1 equals 0x0200, and CSMR1[BAM] equals 0x00FF.</p>
15–9	Reserved, must be cleared.
8 WP	<p>Write protect. Controls write accesses to the address range in the corresponding CSAR. Attempting to write to the range of addresses for which CSAR<math>n</math>[WP] is set results in a bus error termination of the internal cycle and no external cycle.</p> <p>0 Read and write accesses are allowed 1 Only read accesses are allowed</p>
7–1	Reserved, must be cleared.
0 V	<p>Valid bit. Indicates whether the corresponding CSAR, CSMR, and CSCR contents are valid. Programmed chip-selects do not assert until V bit is set (except for <math>\overline{FB\_CS}_0</math>, which acts as the global chip-select). Reset clears each CSMR<math>n</math>[V].</p> <p><b>Note:</b> At reset, no chip-select other than <math>\overline{FB\_CS}_0</math> can be used until the CSMR0[V] is set. Afterward, <math>\overline{FB\_CS}[5:0]</math> functions as programmed.</p> <p>0 Chip-select invalid 1 Chip-select valid</p>

### 20.3.3 Chip-Select Control Registers (CSCR0 – CSCR5)

Each CSCR $n$  controls the auto-acknowledge, address setup and hold times, port size, burst capability, and number of wait states. To support the global chip-select,  $\overline{FB\_CS}_0$ , the CSCR0 reset values differ from the

other CSCRs.  $\overline{FB\_CS0}$  allows address decoding for an external device to serve as the boot memory before system initialization and configuration are completed.

Address: 0xFC00_8008 (CSCR0) 0xFC00_8014 (CSCR1) 0xFC00_8020 (CSCR2) 0xFC00_802C (CSCR3) 0xFC00_8038 (CSCR4) 0xFC00_8044 (CSCR5)										Access: User read/write				
										23	22	21	20	19 18 17 16
										SWSEN	EXTS	ASET		RDAH WRAH
R										0	0	1	1	1 1 1 1
W										0	0	0	0	0 0 0 0
Reset: CSCR0										0	0	0	0	1 1 1 1
Reset: CSCR1–5										0	0	0	0	0 0 0 0
										15	14	13	12	11 10 9 8
										R	WS	BLS	AA	7 6 5 4
R										W	PS	BEM	BSTR	BSTW
Reset: CSCR0										1	1	1	1	0 0 0 0
Reset: CSCR1–5										0	0	0	0	0 0 0 0
										See Note	See Note	1	0	0 0 0 0
										0	0	0	0	0 0 0 0

**Note:** The PS reset value depends upon the chosen chip configuration (RCON[5:4] for parallel configuration or SBF\_RCON[31:30] for serial boot configuration).

Figure 20-3. Chip-Select Control Registers (CSCRn)

Table 20-6. CSCRn Field Descriptions

Field	Description
31–26 SWS	Secondary wait states. The number of wait states inserted before an internal transfer acknowledge is generated for a burst transfer except for the first termination, which is controlled by the wait state count. The secondary wait state is used only if the SWSEN bit is set. Otherwise, the WS value is used for all burst transfers.
25–24	Reserved, must be cleared
23 SWSEN	Secondary wait state enable. 0 The WS value inserts wait states before an internal transfer acknowledge is generated for all transfers. 1 The SWS value inserts wait states before an internal transfer acknowledge is generated for burst transfer secondary terminations.
22 EXTS	Enable extended transfer start. See <a href="#">Section 20.4.9, “Extended Transfer Start”</a> . 0 $\overline{FB\_TS}$ asserts for one bus clock cycle 1 $\overline{FB\_TS}$ remains asserted until the first positive clock edge after $\overline{FB\_CSn}$ asserts
21–20 ASET	Address setup. This field controls the assertion of the chip-select with respect to assertion of a valid address and attributes. The address and attributes are considered valid at the same time $\overline{FB\_ALE}$ asserts. 00 Assert $\overline{FB\_CSn}$ on first rising clock edge after address is asserted. (Default $\overline{FB\_CS0}$ ) 01 Assert $\overline{FB\_CSn}$ on second rising clock edge after address is asserted. 10 Assert $\overline{FB\_CSn}$ on third rising clock edge after address is asserted. 11 Assert $\overline{FB\_CSn}$ on fourth rising clock edge after address is asserted. (Default $\overline{FB\_CS0}$ )

**Table 20-6. CSCR $n$  Field Descriptions (Continued)**

Field	Description																	
19–18 RDAH	Read address hold or deselect. This field controls the address and attribute hold time after the termination during a read cycle that hits in the chip-select address space.																	
	<b>Note:</b> The hold time applies only at the end of a transfer. Therefore, during a burst transfer or a transfer to a port size smaller than the transfer size, the hold time is only added after the last bus cycle.																	
	The number of cycles the address and attributes are held after $\overline{\text{FB\_CS}n}$ negation depends on the value of CSCR $n$ [AA] as shown below.																	
	<table border="1"> <thead> <tr> <th data-bbox="638 494 719 530">RDAH</th><th data-bbox="845 494 943 530">AA = 0</th><th data-bbox="1041 494 1122 530">AA = 1</th></tr> </thead> <tbody> <tr> <td data-bbox="584 551 780 614">00 (<math>\overline{\text{FB\_CS}n}</math> Default)</td><td data-bbox="845 551 943 614">1 cycle</td><td data-bbox="1041 551 1122 614">0 cycles</td></tr> <tr> <td data-bbox="584 629 780 665">01</td><td data-bbox="845 629 943 665">2 cycles</td><td data-bbox="1041 629 1122 665">1 cycles</td></tr> <tr> <td data-bbox="584 680 780 715">10</td><td data-bbox="845 680 943 715">3 cycles</td><td data-bbox="1041 680 1122 715">2 cycles</td></tr> <tr> <td data-bbox="584 730 780 794">11 (<math>\overline{\text{FB\_CS}0}</math> Default)</td><td data-bbox="845 730 943 794">4 cycles</td><td data-bbox="1041 730 1122 794">3 cycles</td></tr> </tbody> </table>			RDAH	AA = 0	AA = 1	00 ( $\overline{\text{FB\_CS}n}$ Default)	1 cycle	0 cycles	01	2 cycles	1 cycles	10	3 cycles	2 cycles	11 ( $\overline{\text{FB\_CS}0}$ Default)	4 cycles	3 cycles
RDAH	AA = 0	AA = 1																
00 ( $\overline{\text{FB\_CS}n}$ Default)	1 cycle	0 cycles																
01	2 cycles	1 cycles																
10	3 cycles	2 cycles																
11 ( $\overline{\text{FB\_CS}0}$ Default)	4 cycles	3 cycles																
17–16 WRAH	Write address hold or deselect. This field controls the address, data, and attribute hold time after the termination of a write cycle that hits in the chip-select address space.																	
	<b>Note:</b> The hold time applies only at the end of a transfer. Therefore, during a burst transfer or a transfer to a port size smaller than the transfer size, the hold time is only added after the last bus cycle.																	
	00 Hold address and attributes one cycle after $\overline{\text{FB\_CS}n}$ negates on writes. (Default $\overline{\text{FB\_CS}n}$ )																	
	01 Hold address and attributes two cycles after $\overline{\text{FB\_CS}n}$ negates on writes.																	
	10 Hold address and attributes three cycles after $\overline{\text{FB\_CS}n}$ negates on writes.																	
	11 Hold address and attributes four cycles after $\overline{\text{FB\_CS}n}$ negates on writes. (Default $\overline{\text{FB\_CS}0}$ )																	
15–10 WS	Wait states. The number of wait states inserted after $\overline{\text{FB\_CS}n}$ asserts and before an internal transfer acknowledge is generated (WS = 0 inserts zero wait states, WS = 0x3F inserts 63 wait states). If AA is reserved, $\overline{\text{FB\_TA}}$ must be asserted by the external system regardless of the number of generated wait states. In that case, the external transfer acknowledge ends the cycle. An external $\overline{\text{FB\_TA}}$ supersedes the generation of an internal $\overline{\text{FB\_TA}}$ .																	
9 BLS	Byte-lane shift. Determines if data on FB_AD appears left-justified or right-justified during the data phase of a FlexBus access.																	
	0 Not shifted. Data is left-justified on FB_AD.																	
	1 Shifted. Data is right justified on FB_AD.																	
8 AA	Auto-acknowledge enable. Determines the assertion of the internal transfer acknowledge for accesses specified by the chip-select address.																	
	0 No internal $\overline{\text{FB\_TA}}$ is asserted. Cycle is terminated externally																	
	1 Internal transfer acknowledge is asserted as specified by WS																	
	<b>Note:</b> If AA is set for a corresponding $\overline{\text{FB\_CS}n}$ and the external system asserts an external $\overline{\text{FB\_TA}}$ before the wait-state countdown asserts the internal $\overline{\text{FB\_TA}}$ , the cycle is terminated. Burst cycles increment the address bus between each internal termination.																	

**Table 20-6. CSCR $n$  Field Descriptions (Continued)**

Field	Description
7–6 PS	Port size. Specifies the data port width associated with each chip-select. It determines where data is driven during write cycles and where data is sampled during read cycles. 00 32-bit port size. Valid data sampled and driven on FB_D[31:0] 01 8-bit port size. Valid data sampled and driven on FB_D[31:24] 1x 16-bit port size. Valid data sampled and driven on FB_D[31:16]
5 BEM	Byte-enable mode. Specifies the byte enable operation. Certain memories have byte enables that must be asserted during reads and writes. BEM can be set in the relevant CSCR to provide the appropriate mode of byte enable support for these SRAMs. 0 $\overline{\text{FB\_BE/BWE}}$ is not asserted for reads. $\overline{\text{FB\_BE/BWE}}$ is asserted for data write only. 1 $\overline{\text{FB\_BE/BWE}}$ is asserted for read and write accesses.
4 BSTR	Burst-read enable. Specifies whether burst reads are used for memory associated with each FB_CS $n$ . 0 Data exceeding the specified port size is broken into individual, port-sized, non-burst reads. For example, a longword read from an 8-bit port is broken into four 8-bit reads. 1 Enables data burst reads larger than the specified port size, including longword reads from 8- and 16-bit ports, word reads from 8-bit ports, and line reads from 8-, 16-, and 32-bit ports.
3 BSTW	Burst-write enable. Specifies whether burst writes are used for memory associated with each FB_CS $n$ . 0 Break data larger than the specified port size into individual, port-sized, non-burst writes. For example, a longword write to an 8-bit port takes four byte writes. 1 Enables burst write of data larger than the specified port size, including longword writes to 8 and 16-bit ports, word writes to 8-bit ports, and line writes to 8-, 16-, and 32-bit ports.
2–0	Reserved, must be cleared.

## 20.4 Functional Description

### 20.4.1 Chip-Select Operation

Each chip-select has a dedicated set of registers for configuration and control:

- Chip-select address registers (CSAR $n$ ) control the base address space of the chip-select. See [Section 20.3.1, “Chip-Select Address Registers \(CSAR0 – CSAR5\).”](#)
- Chip-select mask registers (CSMR $n$ ) provide 16-bit address masking and access control. See [Section 20.3.2, “Chip-Select Mask Registers \(CSMR0 – CSMR5\).”](#)
- Chip-select control registers (CSCR $n$ ) provide port size and burst capability indication, wait-state generation, address setup and hold times, and automatic acknowledge generation features. See [Section 20.3.3, “Chip-Select Control Registers \(CSCR0 – CSCR5\).”](#)

$\overline{\text{FB\_CS0}}$  is a global chip-select after reset and provides external boot memory capability.

### 20.4.1.1 General Chip-Select Operation

When a bus cycle is routed to the FlexBus, the device first compares its address with the base address and mask configurations programmed for chip-selects 0 to 5 (configured in CSCR0 – CSCR5). The results depend on if the address matches or not as shown in [Table 20-7](#).

**Table 20-7. Results of Address Comparison**

Address Matches CSAR $n$ ?	Result
Yes, one CSAR	The appropriate chip-select is asserted, generating an external bus cycle as defined in the chip-select control register. If CSMR[WP] is set and a write access is performed, the internal bus cycle terminates with a bus error, no chip select is asserted, and no external bus cycle is performed.
No	The chip-select signals are not driven. However, the FlexBus runs an external bus cycle with external termination.
Yes, multiple CSARs	The chip-select signals are driven. However, they are driven using an external burst-inhibited bus cycle with external termination on a 32-bit port.

### 20.4.1.2 8-, 16-, and 32-Bit Port Sizing

Static bus sizing is programmable through the port size bits, CSCR[PS]. The processor always drives a 32-bit address on the FB\_AD bus regardless of the external device's address size. The external device must connect its address lines to the appropriate FB\_AD bits from FB\_AD0 upward. Its data bus must be connected to FB\_AD[7:0] from FB\_AD31 downward. No bit ordering is required when connecting address and data lines to the FB\_AD bus. For example, a full 16-bit address/16-bit data device connects its addr[15:0] to FB\_AD[16:1] and data[15:0] to FB\_AD[31:16]. See [Figure 20-4](#) for a graphical connection.

### 20.4.1.3 Global Chip-Select Operation

FB\_CS0, the global (boot) chip-select, supports external boot memory accesses before system initialization. Its operation differs from other external chip-select outputs after system reset.

After system reset, FB\_CS0 is asserted for every external access. No other chip-select can be used until the valid bit, CSMR0[V], is set; at this point FB\_CS0 functions as configured. After this, FB\_CS[5:1] can be used as well. At reset during parallel boot, the logic levels on the FB\_AD[5:4] signals determine global chip-select port size. During serial boot, the value of SBF\_RCON[31:30] determine the port size.

See [Chapter 10, “Chip Configuration Module \(CCM\),”](#) for more information.

## 20.4.2 Data Transfer Operation

Data transfers between the chip and other devices involve these signals:

- Address/data bus (FB\_AD[31:0])

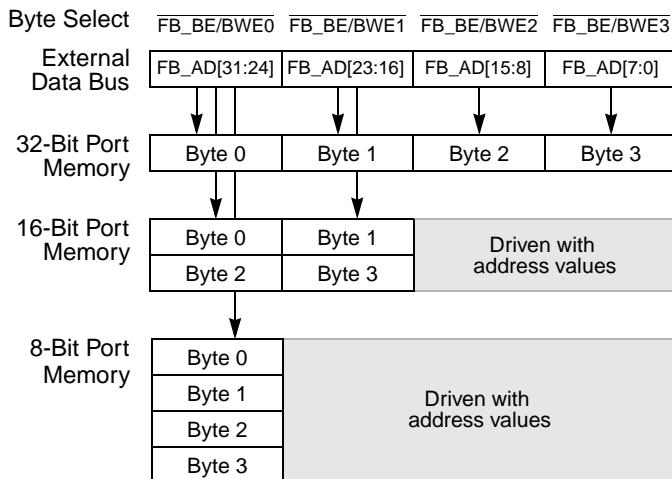
- Control signals (FB\_ALE, FB\_TA, FB\_CS<sub>n</sub>, FB\_OE, FB\_BE/BWE[3:0])
- Attribute signals (FB\_R/W, FB\_TBST, FB\_TSIZ[1:0])

The address, write data, FB\_ALE, FB\_CS<sub>n</sub>, and all attribute signals change on the rising edge of the FlexBus clock (FB\_CLK). Read data is latched into the device on the rising edge of the clock.

The FlexBus supports byte-, word-, longword-, and 16-byte (line) operand transfers and allows accesses to 8-, 16-, and 32-bit data ports. Transfer parameters (address setup and hold, port size, the number of wait states for the external device being accessed, automatic internal transfer termination enable or disable, and burst enable or disable) are programmed in the chip-select control registers (CSCRs). See [Section 20.3.3, “Chip-Select Control Registers \(CSCR0 – CSCR5\).”](#)

### 20.4.3 Data Byte Alignment and Physical Connections

The device aligns data transfers in FlexBus byte lanes with the number of lanes depending on the data port width. [Figure 20-4](#) shows the byte lanes that external memory connects to and the sequential transfers of a longword transfer for the supported port sizes when byte lane shift is disabled. For example, an 8-bit memory connects to the single lane FB\_AD[31:24] (FB\_BE/BWE0). A longword transfer through this 8-bit port takes four transfers, starting with the MSB to the LSB. A longword transfer through a 32-bit port requires one transfer on each four-byte lane of the FlexBus.



**Figure 20-4. Connections for External Memory Port Sizes (CSCRn[BLS] = 0)**

[Figure 20-5](#) shows the byte lanes that external memory connects to and the sequential transfers of a longword transfer for the supported port sizes when byte lane shift is enabled.

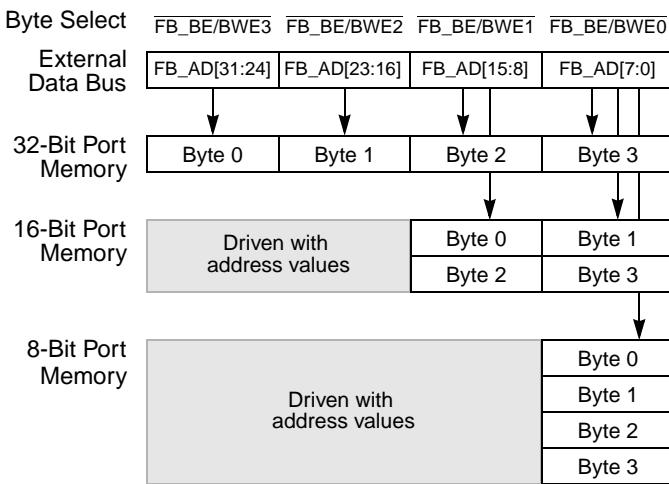


Figure 20-5. Connections for External Memory Port Sizes (CSCRn[BLS] = 1)

#### 20.4.4 Address/Data Bus Multiplexing

The interface supports a single 32-bit wide multiplexed address and data bus (FB\_AD[31:0]). The full 32-bit address is always driven on the first clock of a bus cycle. During the data phase, the FB\_AD[31:0] lines used for data are determined by the programmed port size for the corresponding chip select. The device continues to drive the address on any FB\_AD[31:0] lines not used for data.

The table below lists the supported combinations of address and data bus widths.

Table 20-8. FlexBus Multiplexed Operating Modes

Port Size and Phase		FB_AD					
		[31:24]	[23:16]	[15:8]	[7:0]		
32-bit	Address phase	Address					
	Data phase	Data					
16-bit	Address phase	Address					
	Data phase	Data		Address			
8-bit	Address phase	Address					
	Data phase	Data	Address				

#### 20.4.5 Bus Cycle Execution

As shown in [Figure 20-8](#) and [Figure 20-10](#), basic bus operations occur in four clocks:

1. S0: At the first clock edge, the address, attributes, and FB\_ALE are driven.
2. S1: FB\_CS<sub>n</sub> is asserted at the second rising clock edge to indicate the device selected; by that time, the address and attributes are valid and stable. FB\_ALE is negated at this edge.

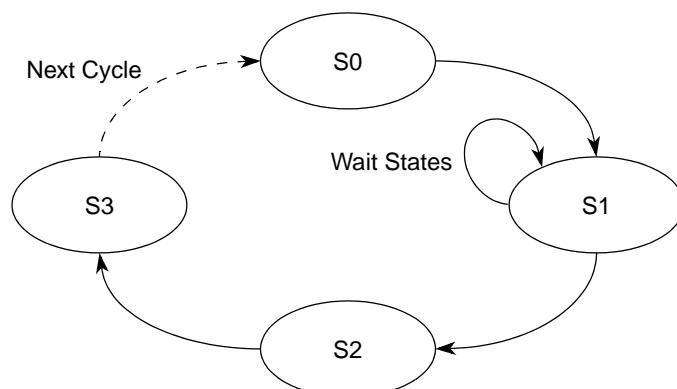
For a write transfer, data is driven on the bus at this clock edge and continues to be driven until one clock cycle after  $\overline{FB\_CSn}$  negates. For a read transfer, data is also driven into the device during this cycle.

External slave asserts  $\overline{FB\_TA}$  at this clock edge.

3. S2: Read data and  $\overline{FB\_TA}$  are sampled on the third clock edge.  $\overline{FB\_TA}$  can be negated after this edge and read data can then be tri-stated.
4. S3:  $\overline{FB\_CSn}$  is negated at the fourth rising clock edge. This last clock of the bus cycle uses what would be an idle clock between cycles to provide hold time for address, attributes, and write data.

#### 20.4.5.1 Data Transfer Cycle States

An on-chip state machine controls the data-transfer operation in the device. Figure 20-6 shows the state-transition diagram for basic read and write cycles.



**Figure 20-6. Data-Transfer-State-Transition Diagram**

Table 20-10 describes the states as they appear in subsequent timing diagrams.

**Table 20-10. Bus Cycle States**

State	Cycle	Description
S0	All	The read or write cycle is initiated. On the rising clock edge, the device places a valid address on FB_AD[31:0], asserts FB_ALE, and drives FB_R/W high for a read and low for a write.
S1	All	FB_ALE is negated on the rising edge of FB_CLK, and $\overline{FB\_CSn}$ is asserted. Data is driven on FB_AD[31:X] for writes, and FB_AD[31:X] is tristated for reads. Address continues to be driven on the FB_AD pins that are unused for data.  If $\overline{FB\_TA}$ is recognized asserted, then the cycle moves on to S2. If $\overline{FB\_TA}$ is not asserted internally or externally, then the S1 state continues to repeat.
	Read	Data is driven by the external device before the next rising edge of FB_CLK (the rising edge that begins S2) with $\overline{FB\_TA}$ asserted.

**Table 20-10. Bus Cycle States (Continued)**

<b>State</b>	<b>Cycle</b>	<b>Description</b>
S2	All	For internal termination, $\overline{FB\_CSn}$ is negated and the internal system bus transfer is completed. For external termination, the external device should negate $\overline{FB\_TA}$ , and the $\overline{FB\_CSn}$ chip select negates after the rising edge of $FB\_CLK$ at the end of S2.
	Read	The processor latches data on the rising clock edge entering S2. The external device can stop driving data after this edge. However, data can be driven until the end of S3 or any additional address hold cycles.
S3	All	Address, data, and $FB\_R/W$ go invalid off the rising edge of $FB\_CLK$ at the beginning of S3, terminating the read or write cycle.

## 20.4.6 FlexBus Timing Examples

### NOTE

Because this device shares the FlexBus signals with the NAND flash controller, all signals, except the chip selects, tristate between bus cycles.

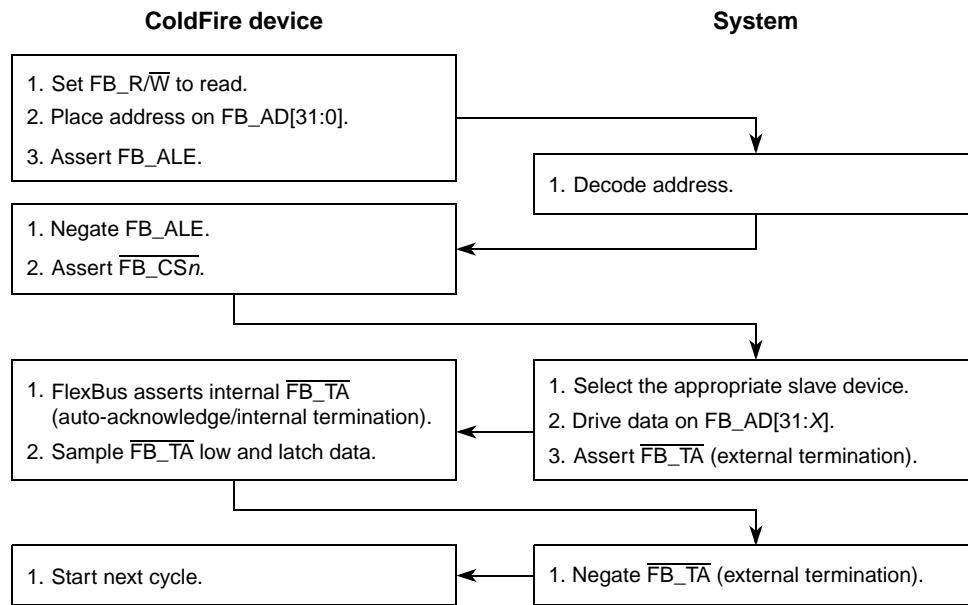
The following sections include timing diagrams that reference signals only available in non-multiplexed mode. This device's pinout does not support this mode. Therefore, ignore any references in the timing diagrams to the  $FB_A$  and  $FB_D$  signals.

### 20.4.6.1 Basic Read Bus Cycle

During a read cycle, the ColdFire device receives data from memory or a peripheral device. [Figure 20-7](#) is a read cycle flowchart.

### NOTE

Throughout this chapter  $FB\_AD[31:X]$  indicates a 32-, 16-, or 8-bit wide data bus.  $FB\_AD[Y:0]$  is an address bus that can be 32-, 24-, or 16 -bits in width.

**Figure 20-7. Read Cycle Flowchart**

The read cycle timing diagram is shown in [Figure 20-8](#).

#### **NOTE**

In the next set of timing diagrams, the dotted lines indicate  $\overline{FB\_TA}$ ,  $\overline{FB\_OE}$ , and  $\overline{FB\_CSn}$  timing when internal termination is used ( $CSCR[AA] = 1$ ). The external and internal  $\overline{FB\_TA}$  assert at the same time; however,  $\overline{FB\_TA}$  is not driven externally for internally-terminated bus cycles.

#### **NOTE**

The processor drives the data lines during the first clock cycle of the transfer with the full 32-bit address. This may be ignored by standard connected devices using non-multiplexed address and data buses. However, some applications may find this feature beneficial.

The address and data busses are muxed between the FlexBus and NAND flash controller. At the end of the read bus cycles the address signals are indeterminate.

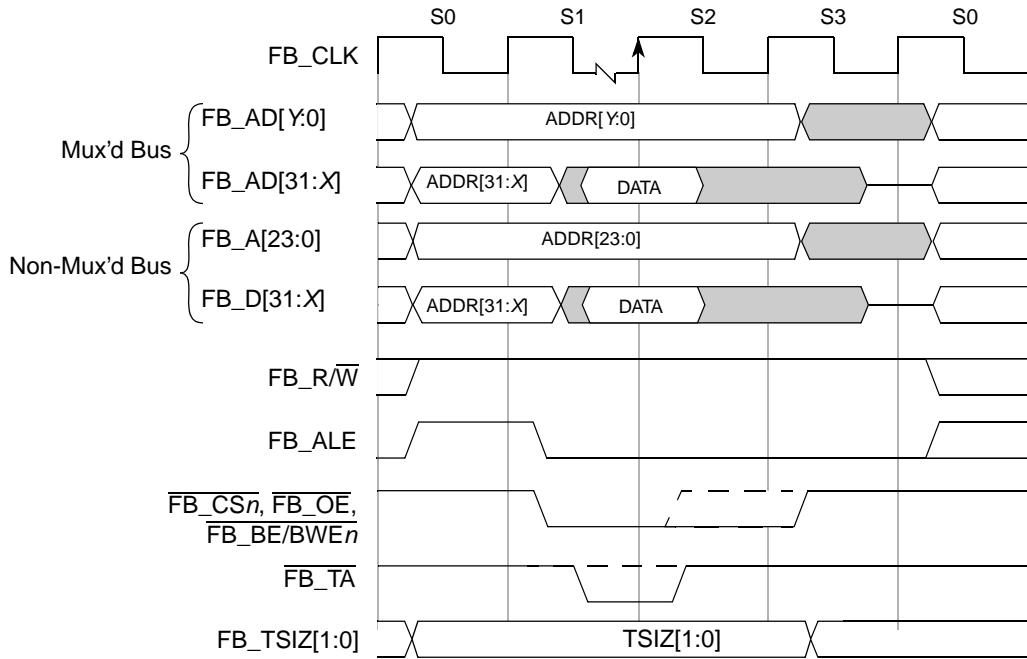


Figure 20-8. Basic Read-Bus Cycle

#### 20.4.6.2 Basic Write Bus Cycle

During a write cycle, the device sends data to memory or to a peripheral device. [Figure 20-9](#) shows the write cycle flowchart.

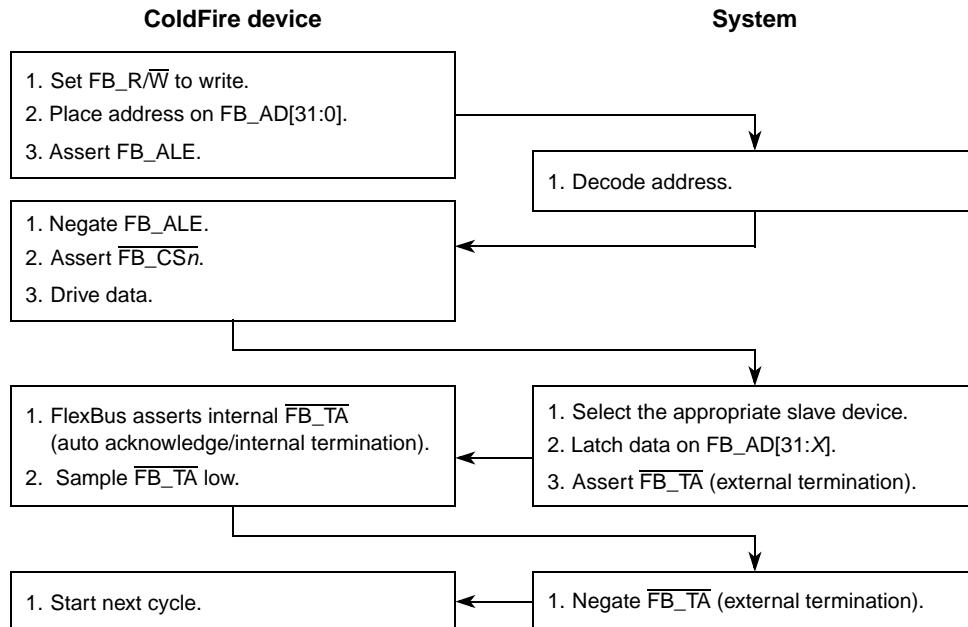


Figure 20-9. Write-Cycle Flowchart

Figure 20-10 shows the write cycle timing diagram.

### NOTE

The address and data busses are muxed between the FlexBus and NAND flash controller. At the end of the write bus cycles, the address signals are indeterminate.

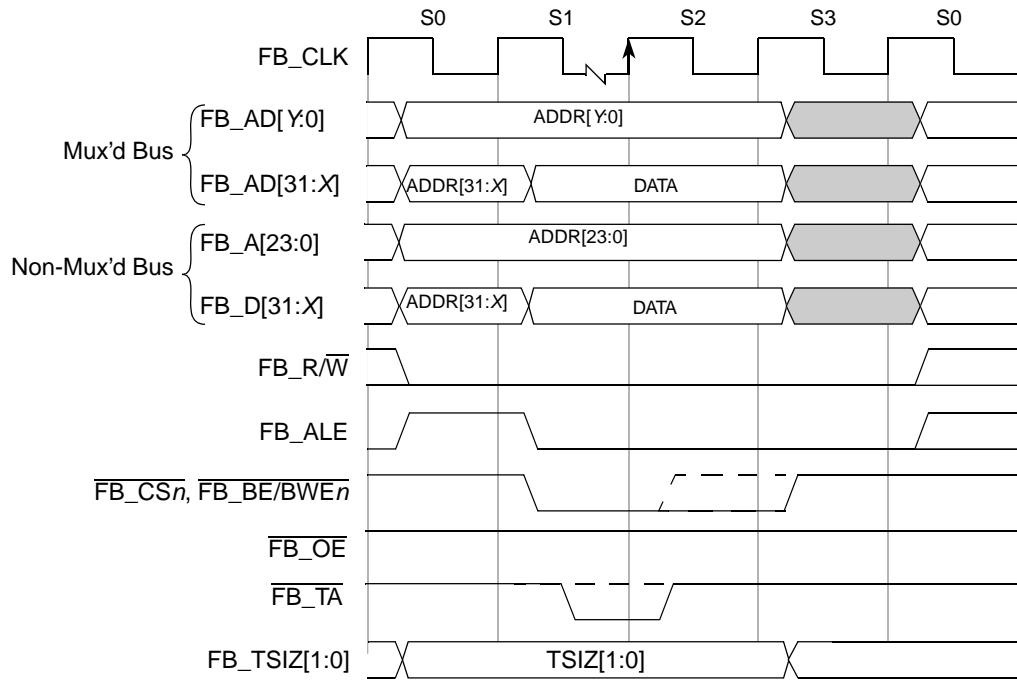
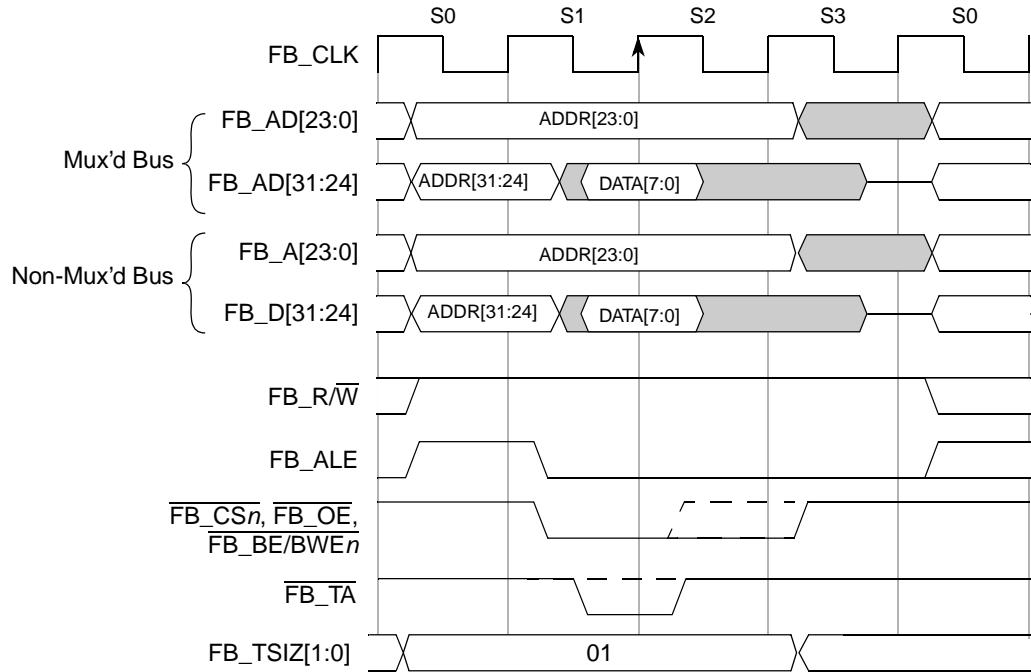


Figure 20-10. Basic Write-Bus Cycle

#### 20.4.6.3 Bus Cycle Sizing

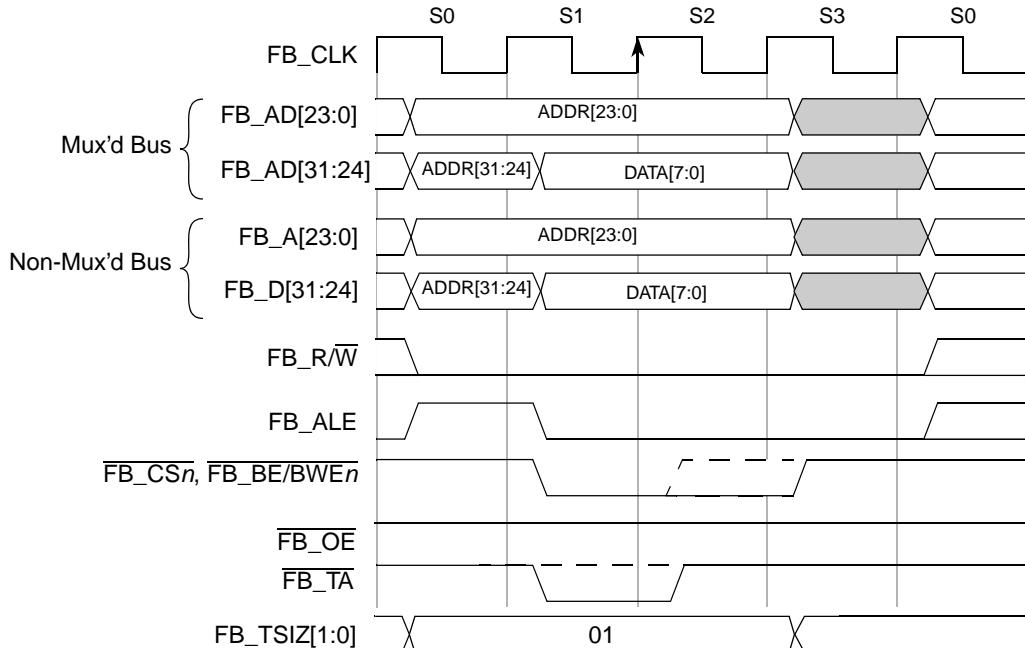
This section shows timing diagrams for various port size scenarios. Figure 20-11 illustrates the basic byte read transfer to an 8-bit device with no wait states. The address is driven on the full FB\_AD[31:8 ] bus in the first clock. The device tristates FB\_AD[31:24] on the second clock and continues to drive address on

FB\_AD[23:0] throughout the bus cycle. The external device returns the read data on FB\_AD[31:24] and may tristate the data line or continue driving the data one clock after FB\_TA is sampled asserted.



**Figure 20-11. Single Byte-Read Transfer**

Figure 20-12 shows the similar configuration for a write transfer. The data is driven from the second clock on FB\_AD[31:24].



**Figure 20-12. Single Byte-Write Transfer**

Figure 20-13 illustrates the basic word read transfer to a 16-bit device with no wait states. The address is driven on the full FB\_AD[31:8 :0] bus in the first clock. The device tristates FB\_AD[31:24] on the second clock and continues to drive the address on FB\_AD[15:0 ] throughout the bus cycle. The external device returns the read data on FB\_AD[31:16], and may tristate the data line or continue driving the data one clock after FB\_TA is sampled asserted.

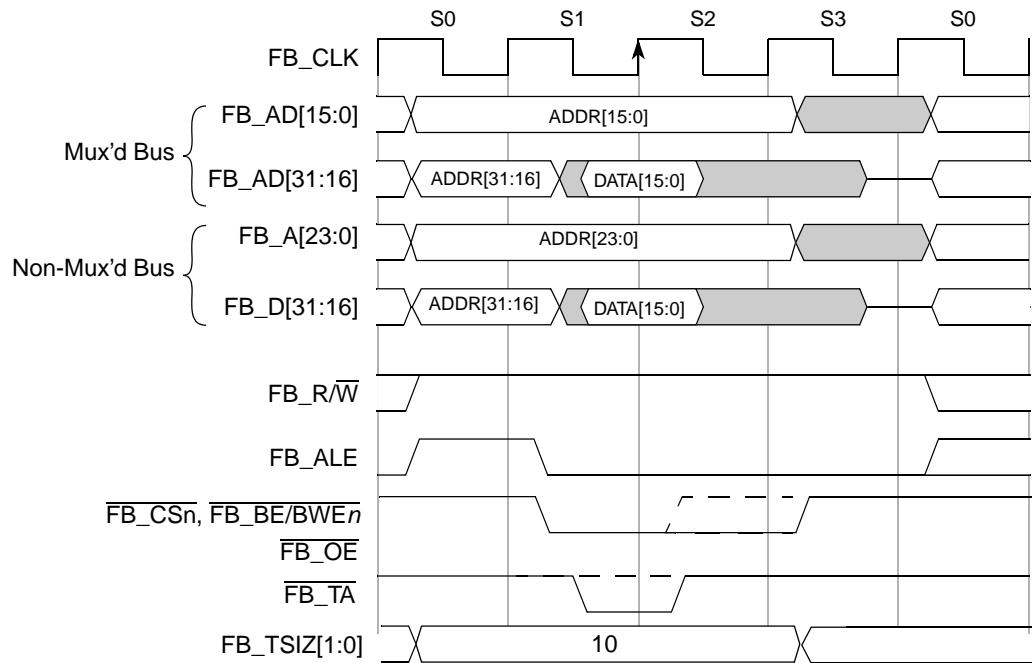
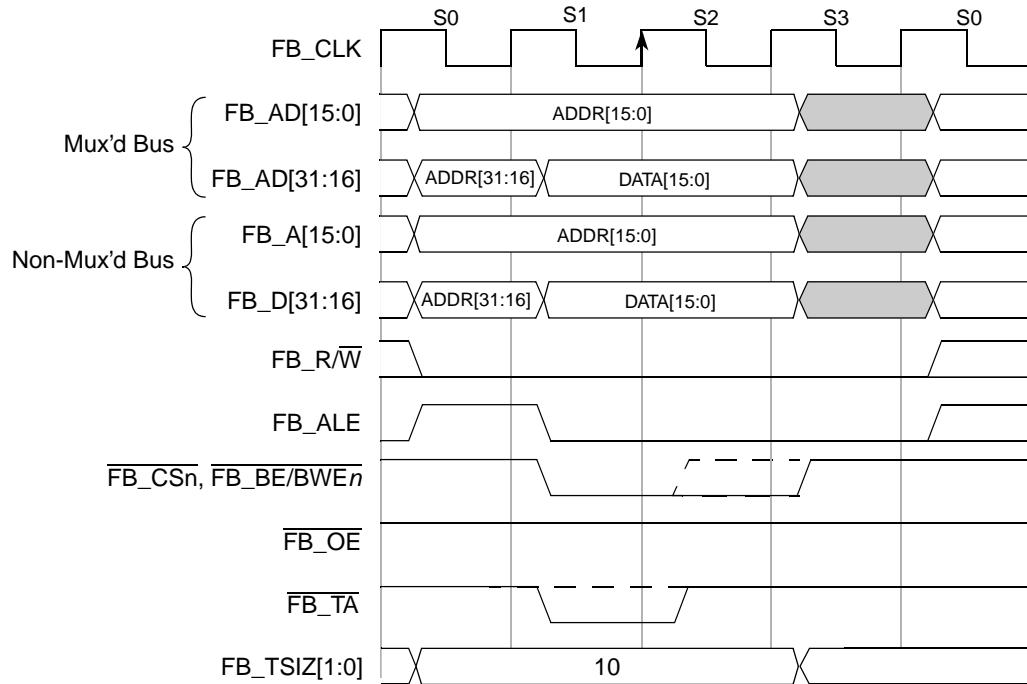


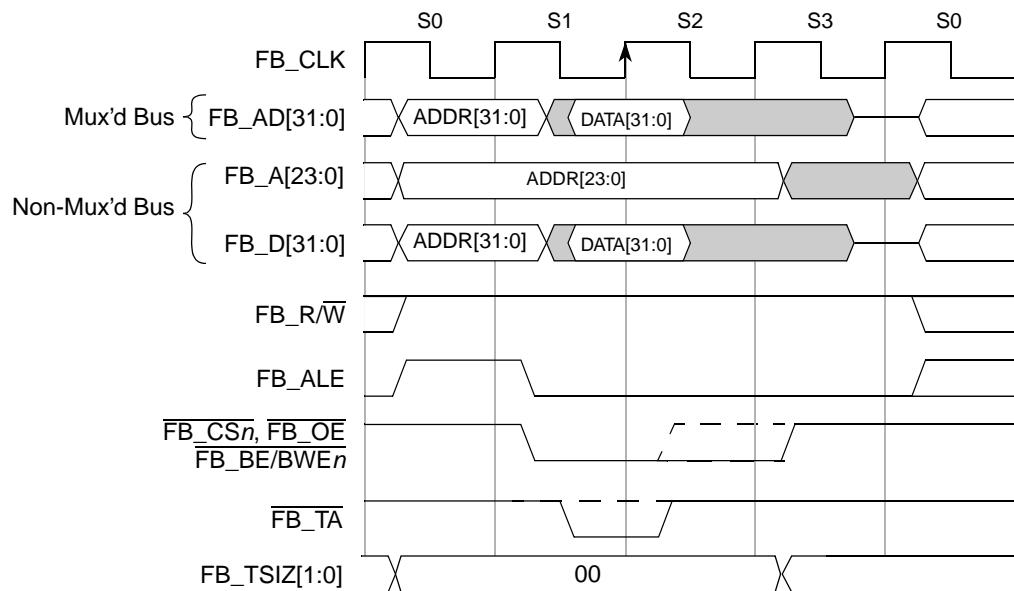
Figure 20-13. Single Word-Read Transfer

Figure 20-14 shows the similar configuration for a write transfer. The data is driven from the second clock on FB\_AD[31:16].



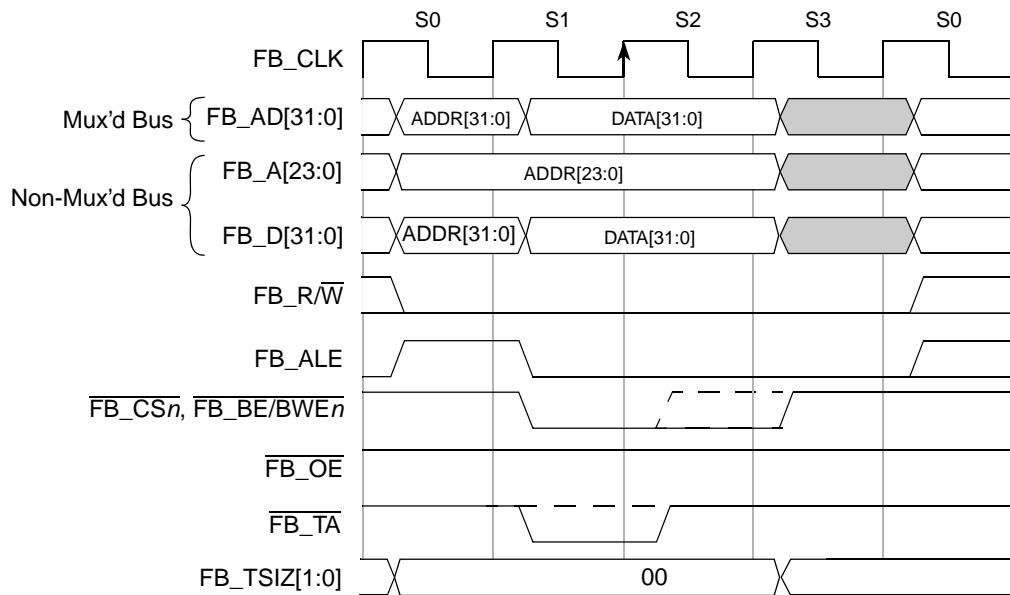
**Figure 20-14. Single Word-Write Transfer**

Figure 20-15 depicts a longword read from a 32-bit device.



**Figure 20-15. Longword-Read Transfer**

Figure 20-16 illustrates the longword write to a 32-bit device.



**Figure 20-16. Longword-Write Transfer**

#### 20.4.6.4 Timing Variations

The FlexBus module has several features that can change the timing characteristics of a basic read- or write-bus cycle to provide additional address setup, address hold, and time for a device to provide or latch data.

##### 20.4.6.4.1 Wait States

Wait states can be inserted before each beat of a transfer by programming the CSCR $n$  registers. Wait states can give the peripheral or memory more time to return read data or sample write data.

Figure 20-17 and Figure 20-18 show the basic read and write bus cycles (also shown in Figure 20-8 and Figure 20-13) with the default of no wait states.

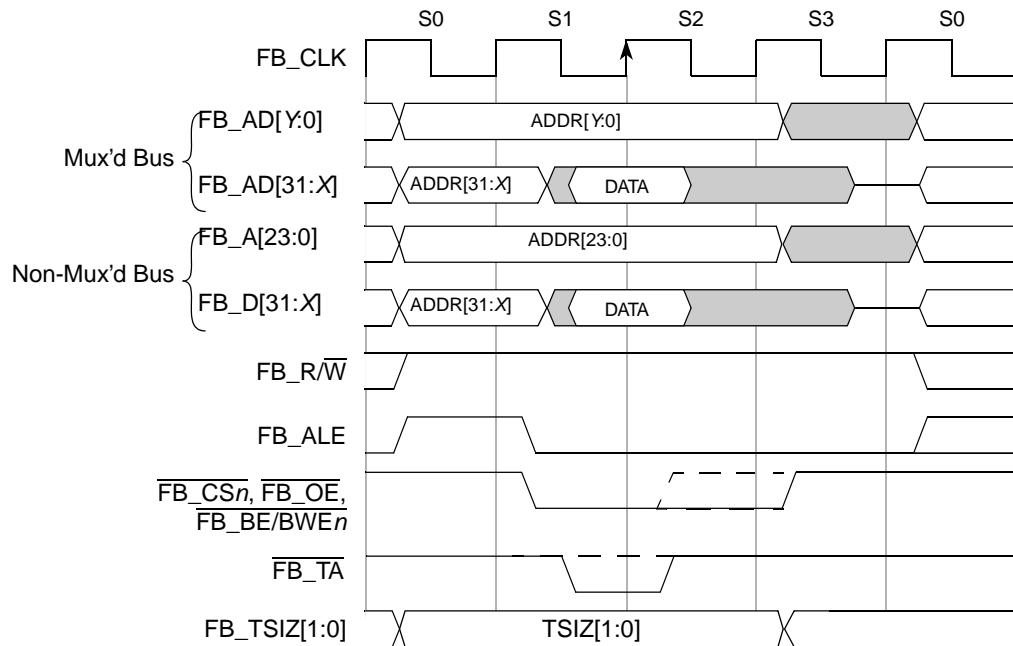


Figure 20-17. Basic Read-Bus Cycle (No Wait States)

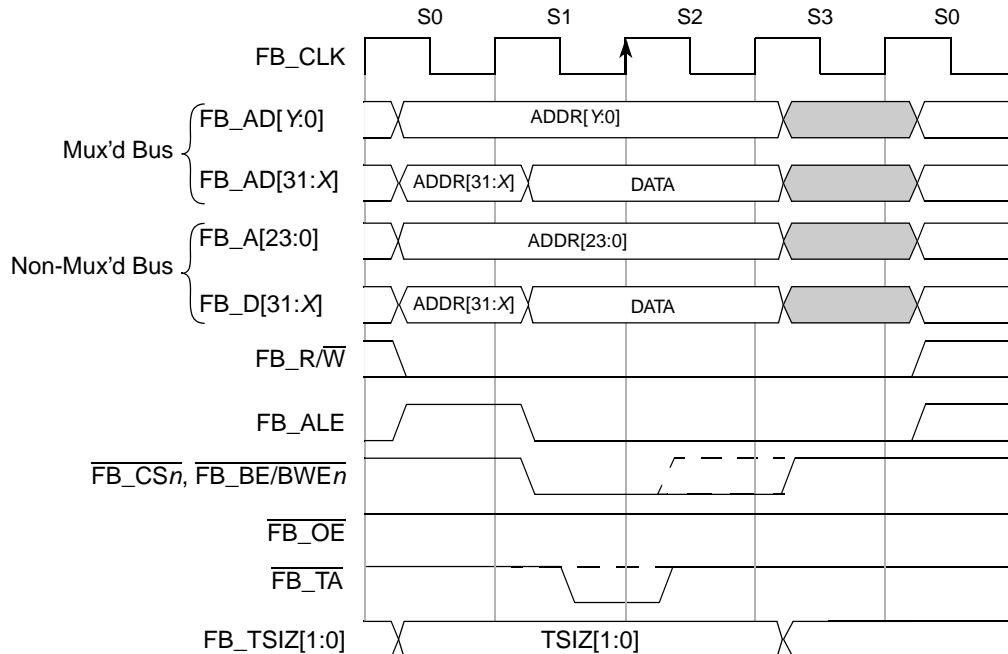
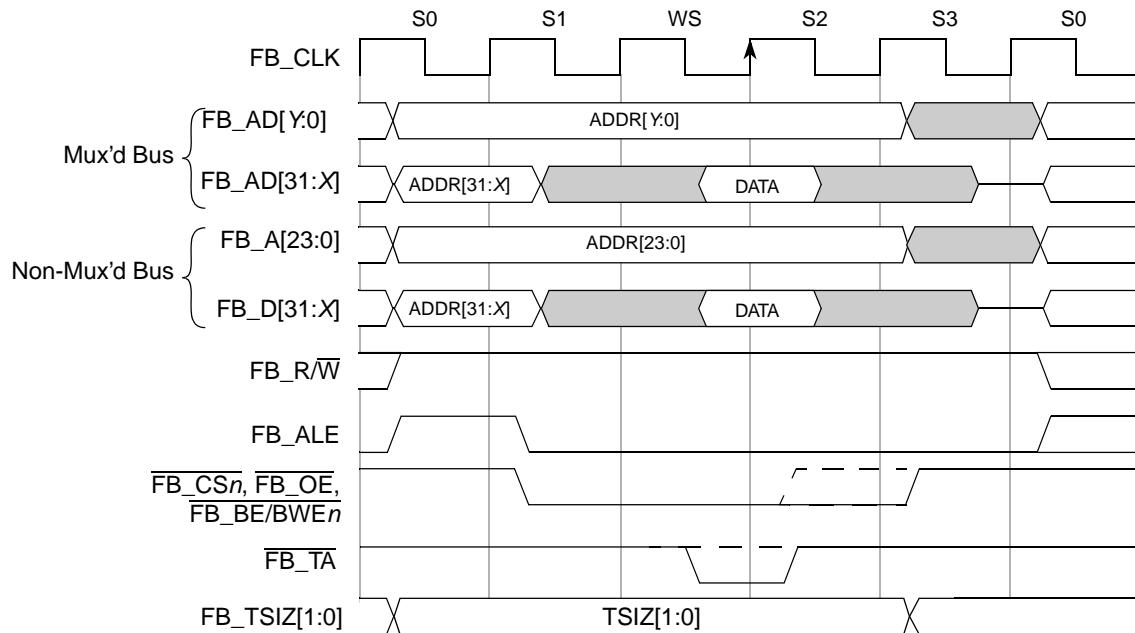
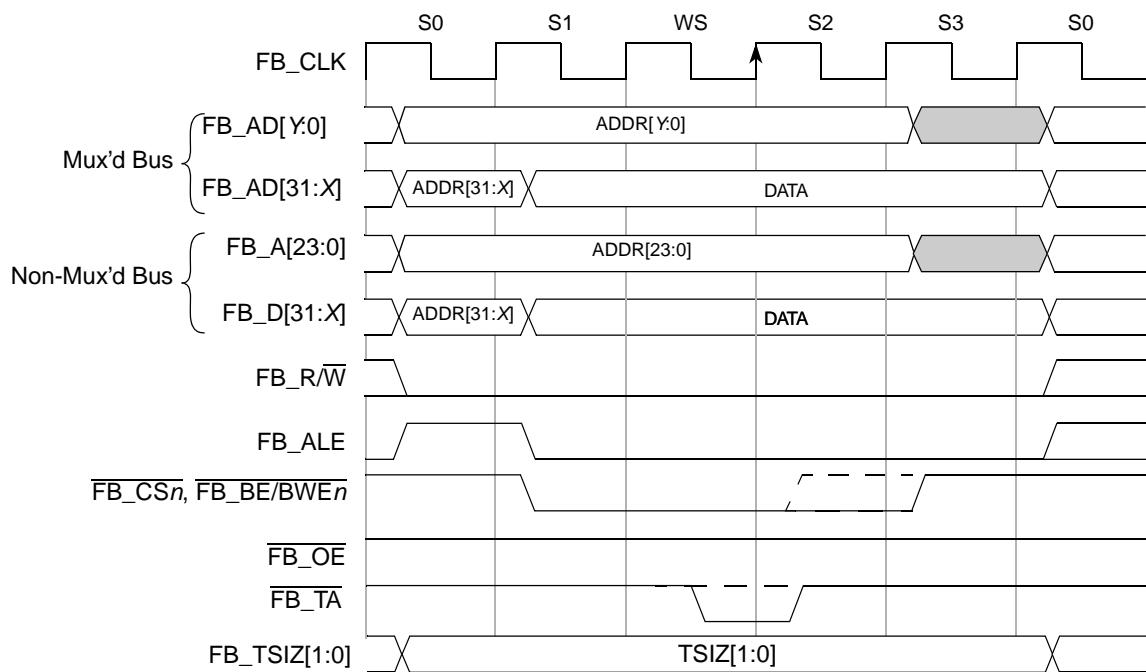


Figure 20-18. Basic Write-Bus Cycle (No Wait States)

If wait states are used, the S1 state repeats continuously until the chip-select auto-acknowledge unit asserts internal transfer acknowledge or the external FB\_TA is recognized as asserted. [Figure 20-19](#) and [Figure 20-20](#) show a read and write cycle with one wait state.



**Figure 20-19. Read-Bus Cycle (One Wait State)**



**Figure 20-20. Write-Bus Cycle (One Wait State)**

#### 20.4.6.4.2 Address Setup and Hold

The timing of the assertion and negation of the chip selects, byte selects, and output enable can be programmed on a chip-select basis. Each chip-select can be programmed to assert one to four clocks after address-latch enable (FB\_ALE) is asserted. Figure 20-21 and Figure 20-22 show read- and write-bus cycles with two clocks of address setup.

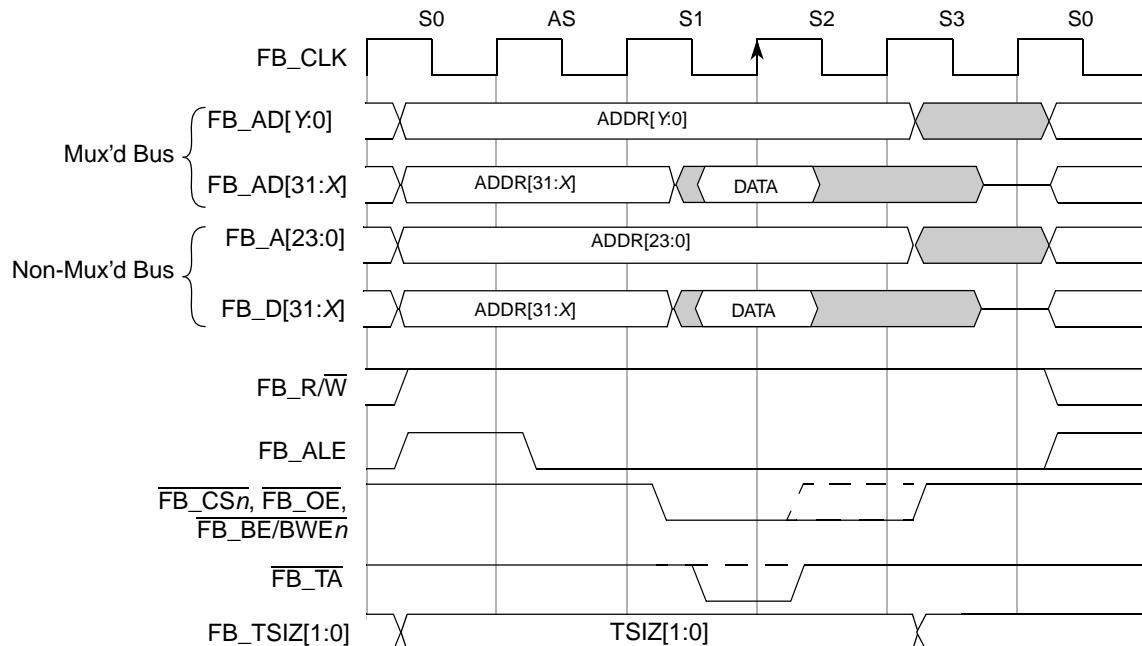


Figure 20-21. Read-Bus Cycle with Two-Clock Address Setup (No Wait States)

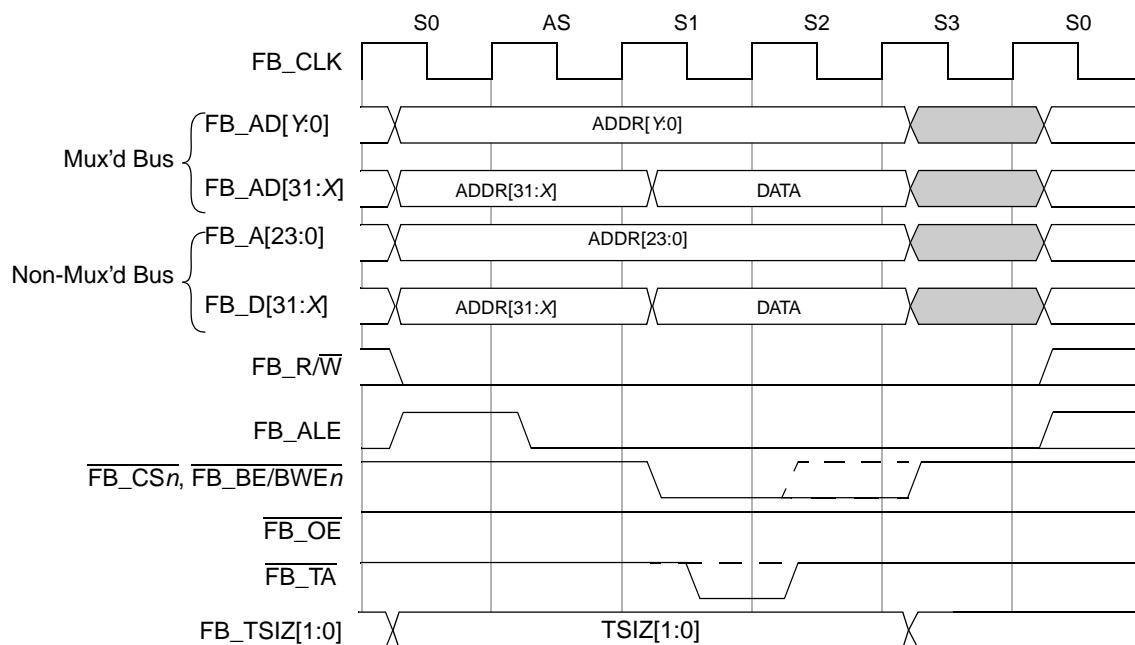


Figure 20-22. Write-Bus Cycle with Two Clock Address Setup (No Wait States)

In addition to address setup, a programmable address hold option for each chip select exists. Address and attributes can be held one to four clocks after chip-select, byte-selects, and output-enable negate. Figure 20-23 and Figure 20-24 show read and write bus cycles with two clocks of address hold.

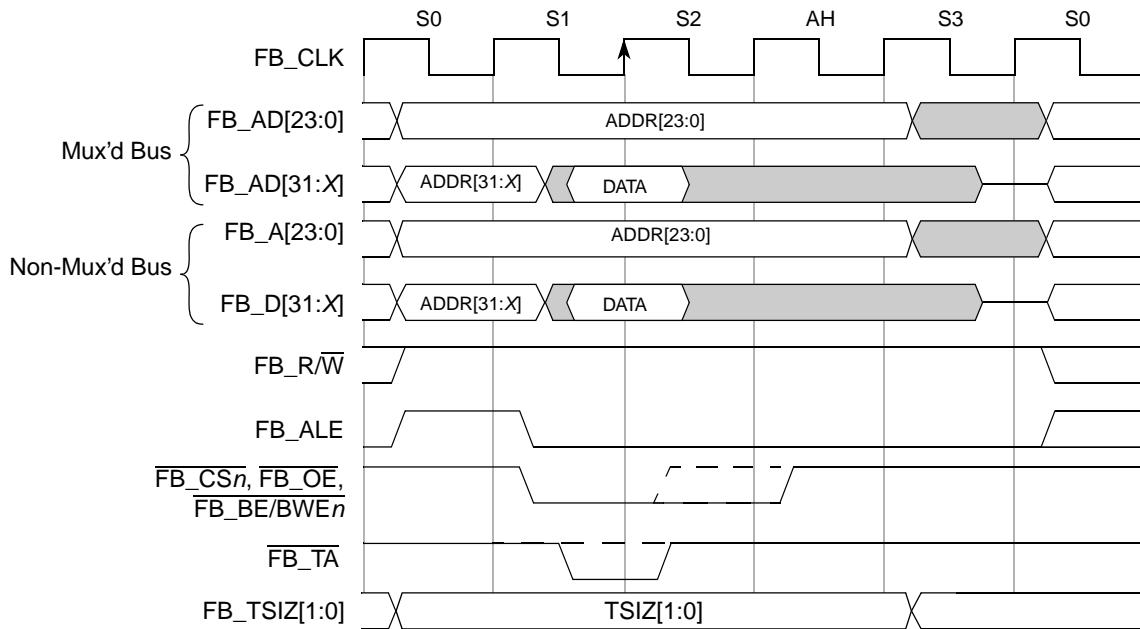


Figure 20-23. Read Cycle with Two-Clock Address Hold (No Wait States)

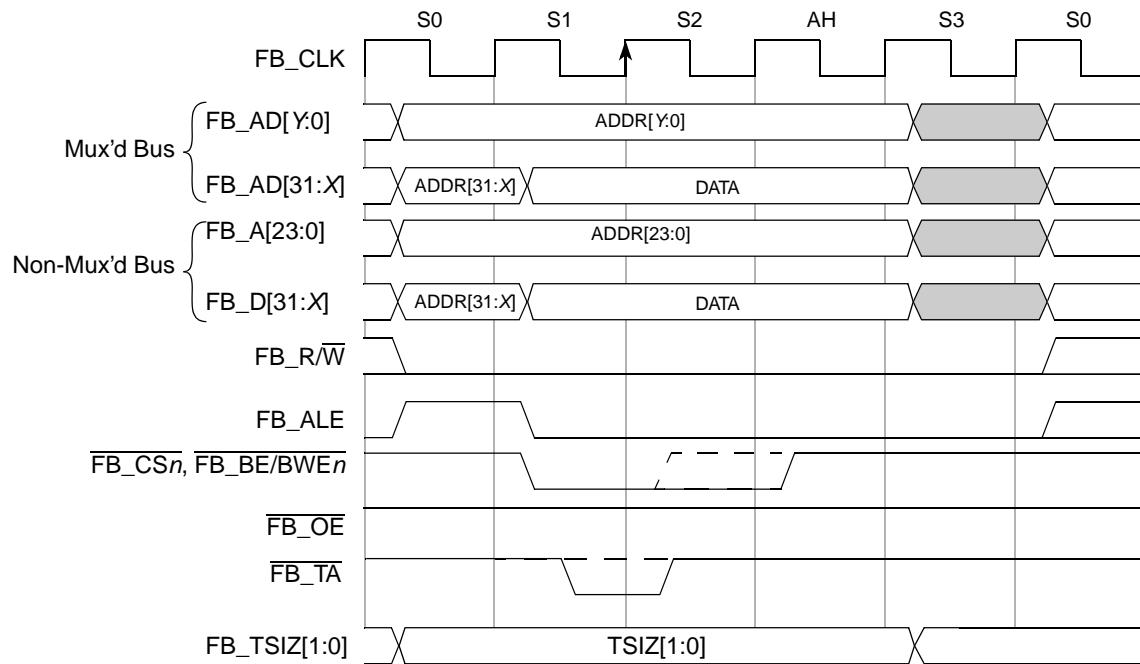
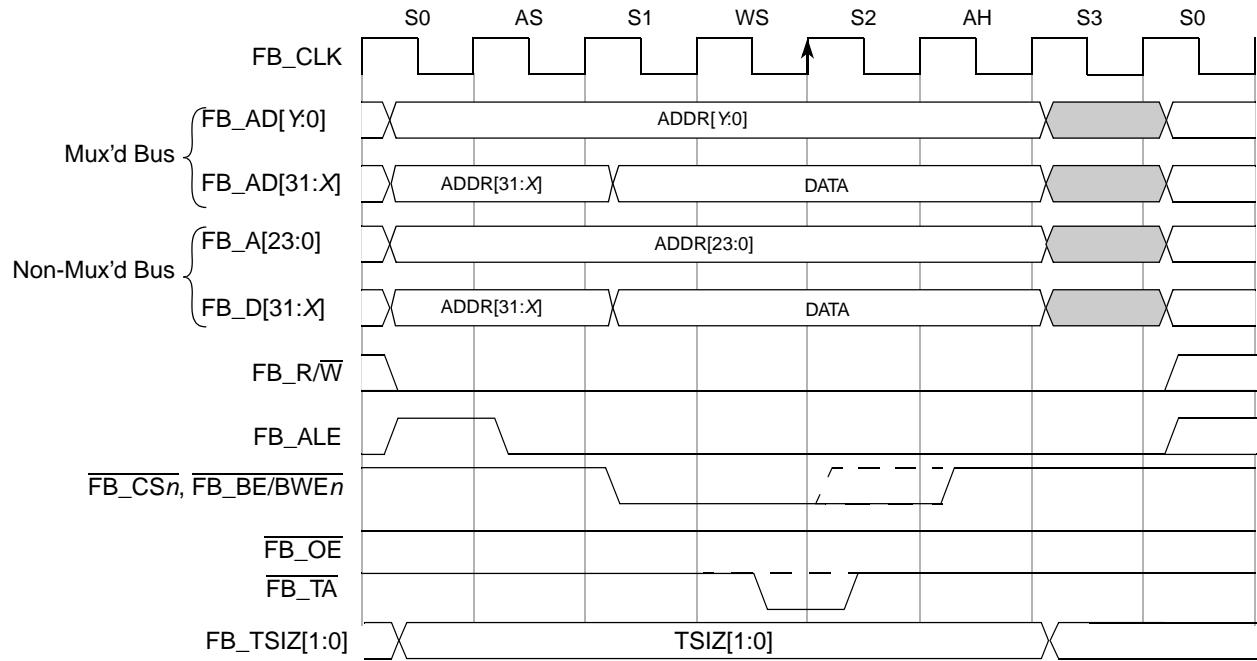


Figure 20-24. Write Cycle with Two-Clock Address Hold (No Wait States)

Figure 20-25 shows a bus cycle using address setup, wait states, and address hold.



**Figure 20-25. Write Cycle with Two-Clock Address Setup and Two-Clock Hold (One Wait State)**

## 20.4.7 Burst Cycles

The device can be programmed to initiate burst cycles if its transfer size exceeds the port size of the selected destination. The initiation of a burst cycle is encoded on the size pins. For burst transfers to smaller port sizes, FB\_TSIZ[1:0] indicates the size of the entire transfer. For example, with bursting enabled, a word transfer to an 8-bit port takes two beats (two byte-sized transfers), for which FB\_TSIZ[1:0] equals 10 throughout. A longword transfer to an 8-bit port would take a 4-byte burst cycle, for which FB\_TSIZ[1:0] equals 00 throughout.

With bursting disabled, any transfer larger than the port size breaks into multiple individual transfers. With bursting enabled, an access larger than port size results in a burst cycle of multiple beats. [Table 20-11](#) shows the result of such transfer translations.

**Table 20-11. Transfer Size and Port Size Translation**

Port Size PS[1:0]	Transfer Size FB_TSIZ[1:0]	Burst-Inhibited: Number of Transfers Burst Enabled: Number of Beats
01 (8-bit)	10 (word)	2
	00 (longword)	4
	11 (line)	16
1x (16-bit)	00 (longword)	2
	11 (line)	8
00 (32-bit)	11 (line)	4

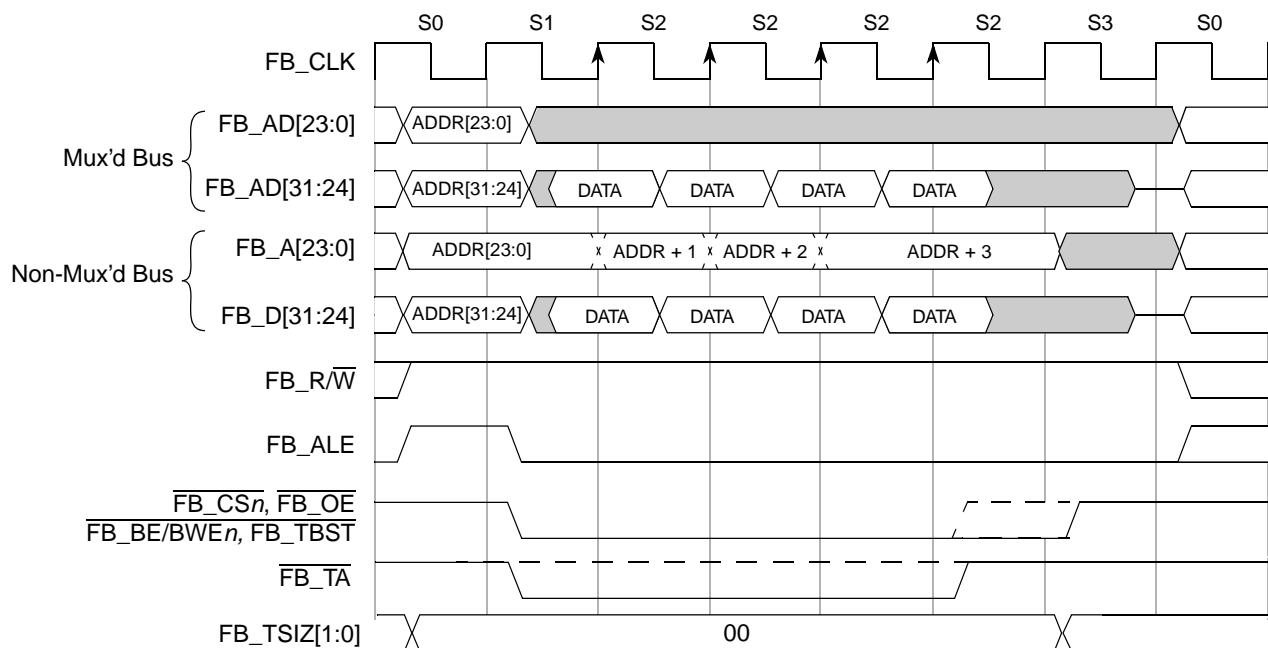
The FlexBus can support 2-1-1-1 burst cycles to maximize system performance. Delaying termination of the cycle can add wait states. If internal termination is used, different wait state counters can be used for the first access and the following beats.

The CSCR<sub>n</sub> registers enable bursting for reads, writes, or both. Memory spaces can be declared burst-inhibited for reads and writes by clearing the appropriate CSCR<sub>n</sub>[BSTR,BSTW] bits.

**Figure 20-26** shows a longword read to an 8-bit device programmed for burst enable. The transfer results in a 4-beat burst and the data is driven on FB\_AD[31:24]. The transfer size is driven at longword (00) throughout the bus cycle.

### NOTE

In multiplexed address/data mode, the address is driven on FB\_AD only during the first cycle for internally- and externally-terminated cycles.

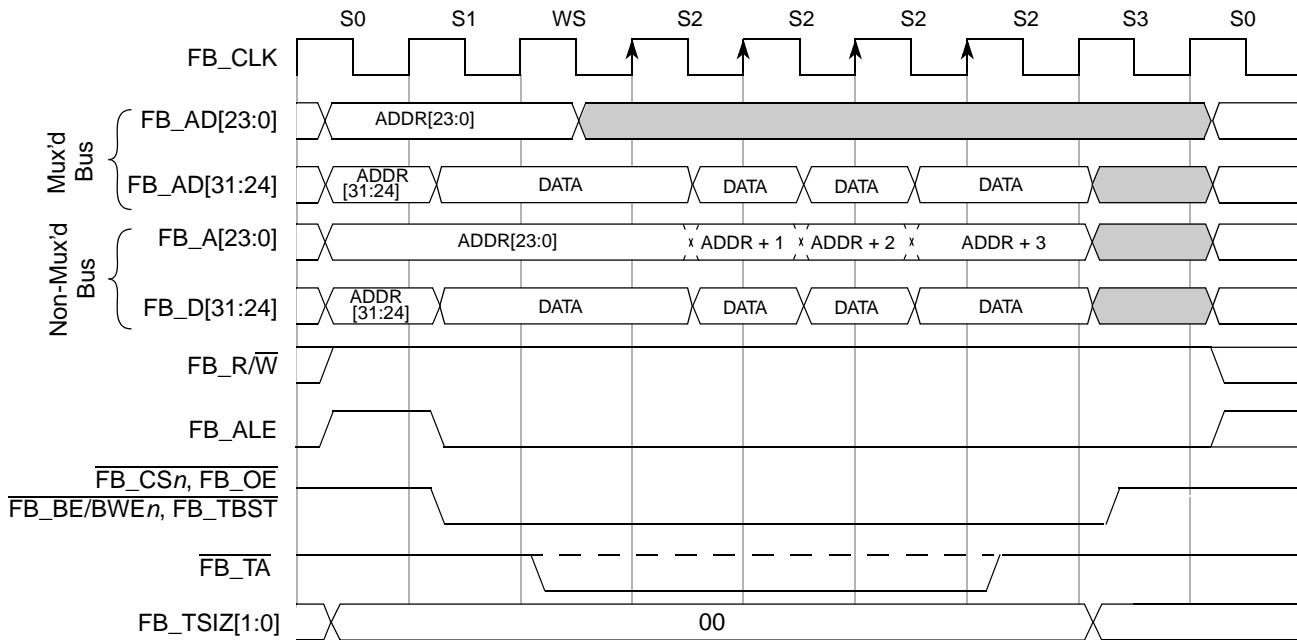


**Figure 20-26. Longword-Read Burst from 8-Bit Port 2-1-1-1 (No Wait States)**

**Figure 20-27** shows a longword write to an 8-bit device with burst enabled. The transfer results in a 4-beat burst and the data is driven on FB\_AD[31:24]. The transfer size is driven at longword (00) throughout the bus cycle.

### NOTE

The first beat of any write burst cycle has at least one wait state. If the bus cycle is programmed for zero wait states (CSCR<sub>n</sub>[WS] = 0), one wait state is added. Otherwise, the programmed number of wait states are used.



**Figure 20-27. Longword-Write Burst to 8-Bit Port 3-1-1-1 (No Wait States)**

Figure 20-28 shows a longword read from an 8-bit device with burst inhibited. The transfer results in four individual transfers. The transfer size is driven at longword (00) during the first transfer and at byte (01) during the next three transfers.

#### NOTE

There is an extra clock of address setup (AS) for each burst-inhibited transfer between states S0 and S1.

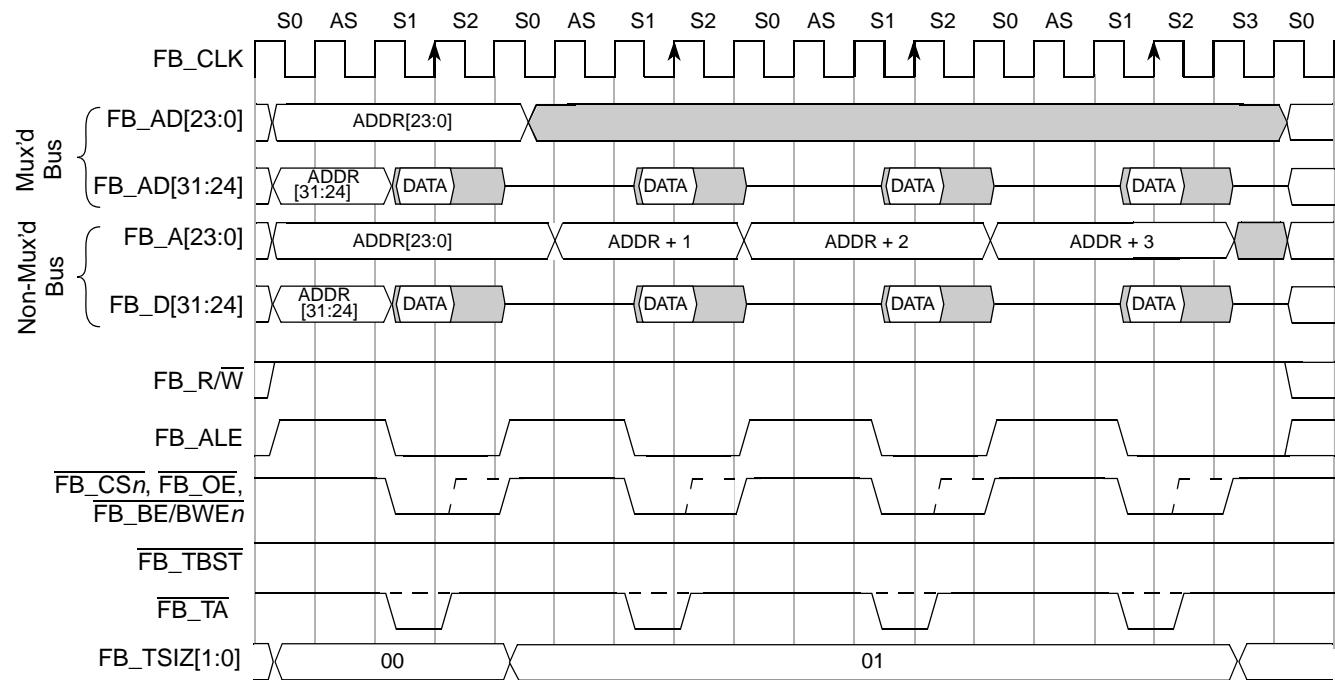
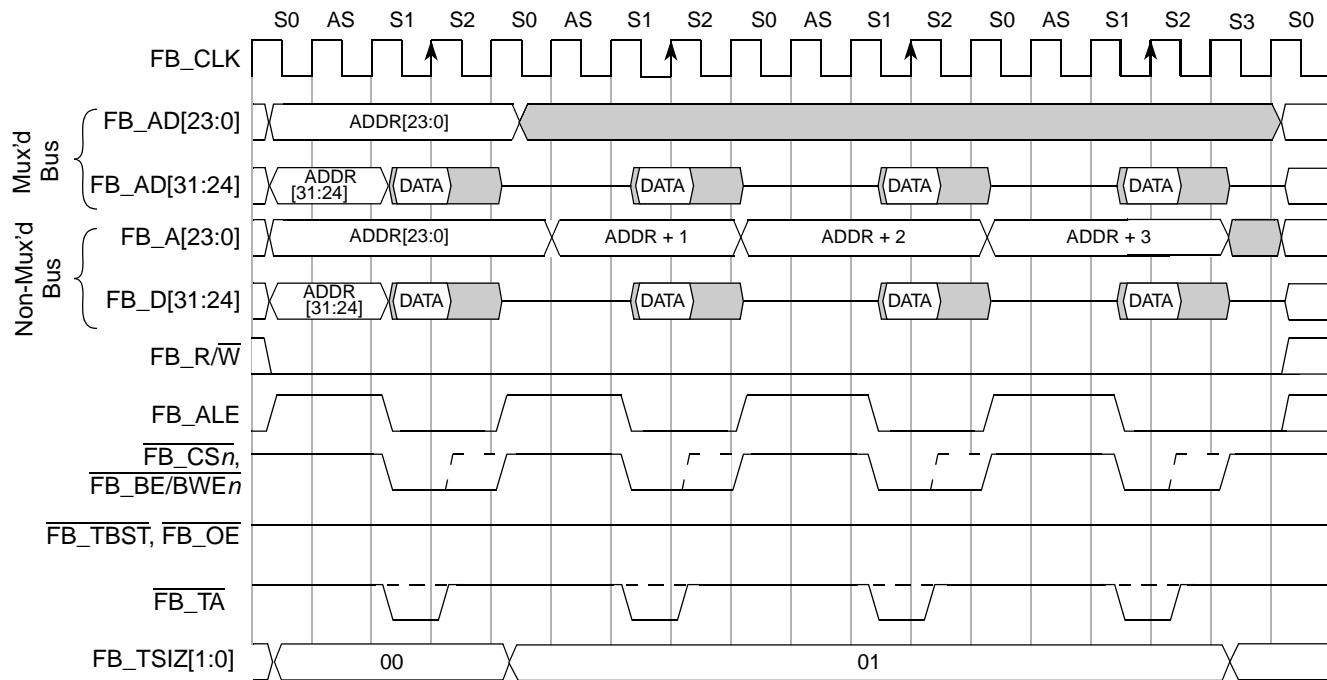


Figure 20-28. Longword-Read Burst-Inhibited from 8-Bit Port (No Wait States)

Figure 20-29 shows a longword write to an 8-bit device with burst inhibited. The transfer results in four individual transfers. The transfer size is driven at longword (00) during the first transfer and at byte (01) during the next three transfers.

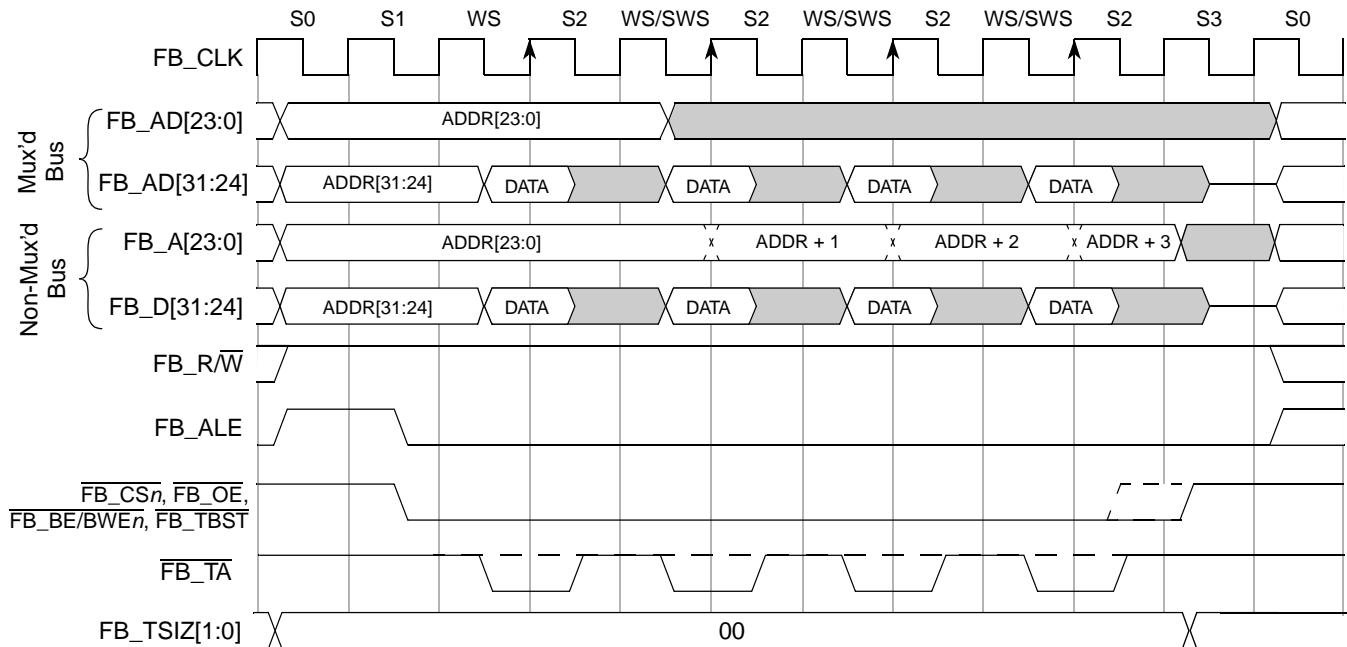


**Figure 20-29. Longword-Write Burst-Inhibited to 8-Bit Port (No Wait States)**

Figure 20-30 illustrates another read burst transfer, but in this case a wait state is added between individual beats.

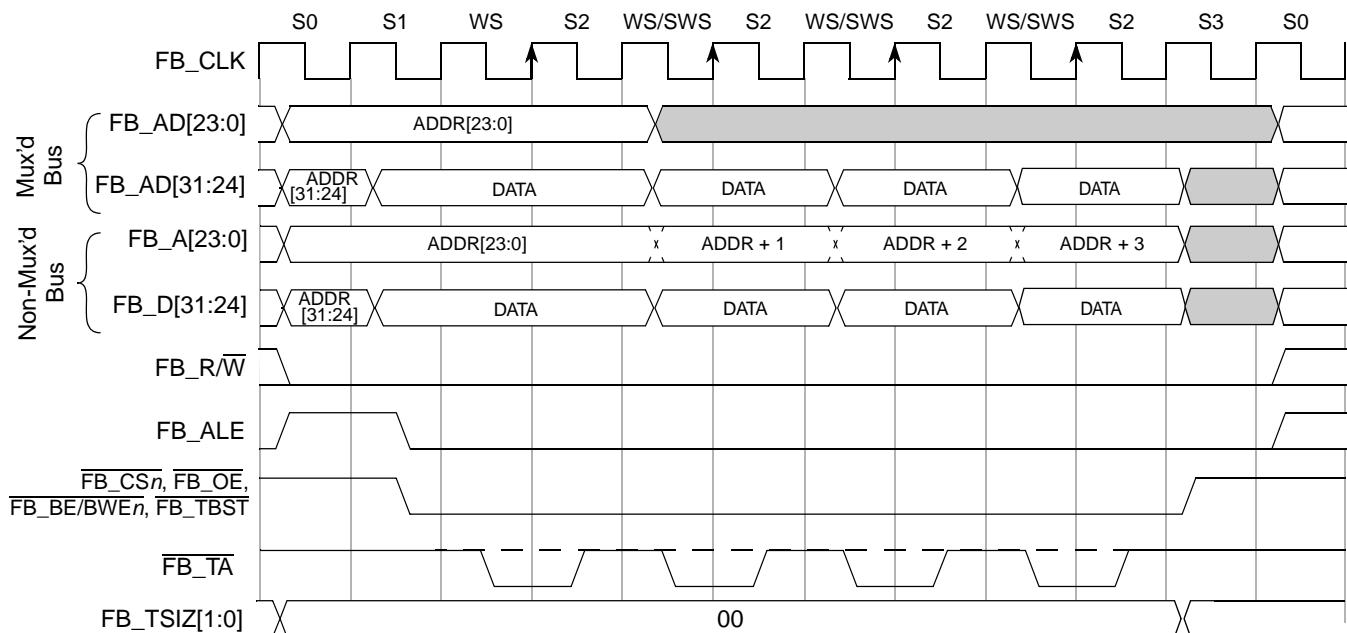
#### NOTE

CSCR $n$ [WS] determines the number of wait states in the first beat. However, for subsequent beats, the CSCR $n$ [WS] (or CSCR $n$ [SWS] if CSCR $n$ [SWSEN] is set) determines the number of wait states.



**Figure 20-30. Longword-Read Burst from 8-Bit Port 3-2-2-2 (One Wait State)**

Figure 20-30 illustrates a write burst transfer with one wait state.

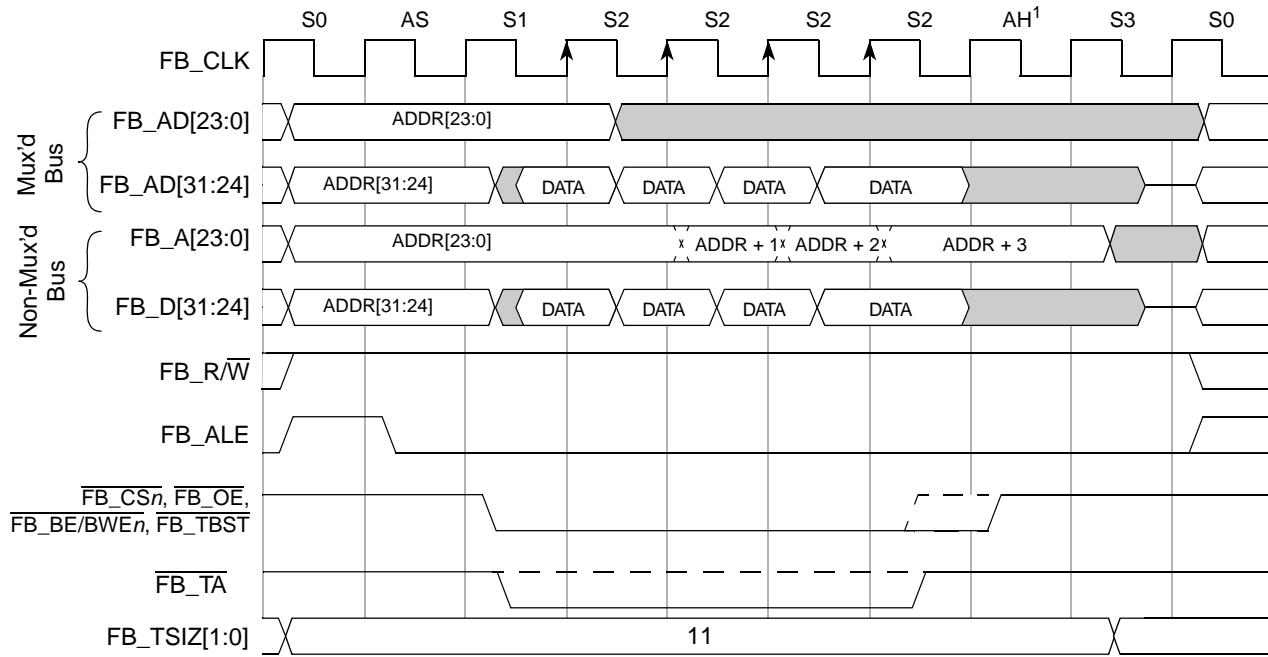


**Figure 20-31. Longword-Write Burst to 8-Bit Port 3-2-2-2 (One Wait State)**

If address setup and hold are used, only the first and last beat of the burst cycle are affected. [Figure 20-32](#) shows a read cycle with one clock of address setup and address hold.

### NOTE

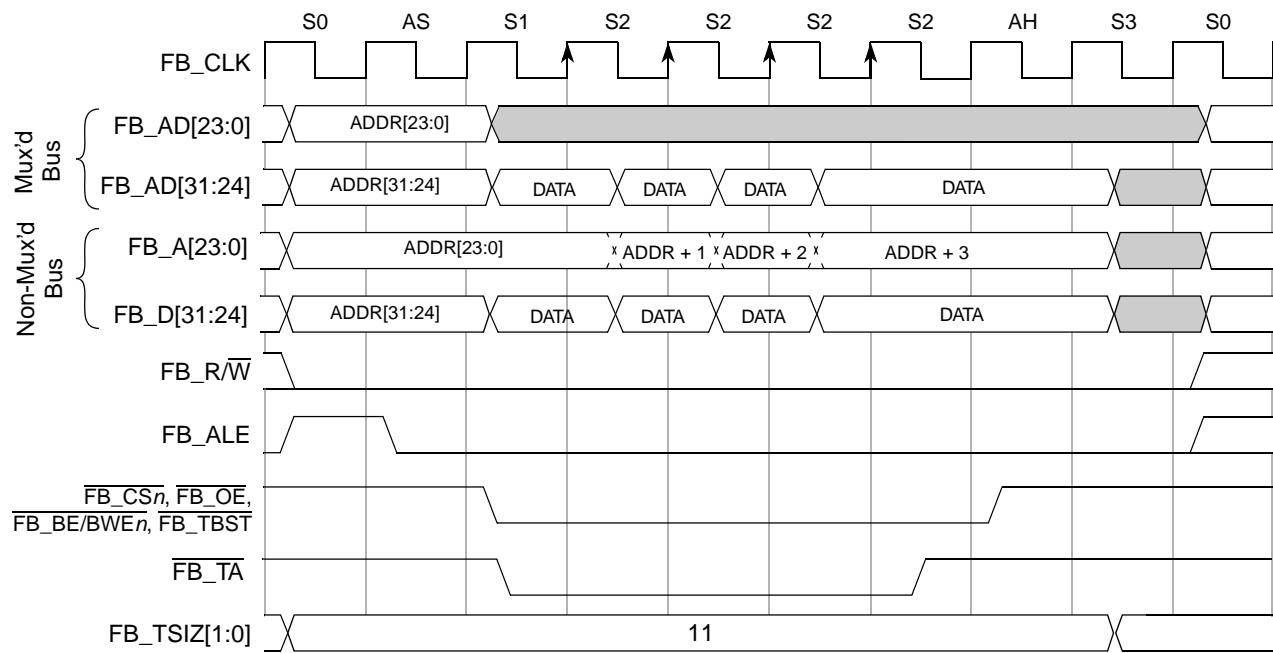
In multiplexed address/data mode, the address is driven on FB\_AD only during the first cycle for internally- and externally-terminated cycles.



<sup>1</sup> The address hold time depends on the setting of CSCR<sub>n</sub>[AA]. See [Section 20.3.3, “Chip-Select Control Registers \(CSCR0 – CSCR5\)](#), for more details.

**Figure 20-32. Longword-Read Burst from 8-Bit Port 3-1-1-1 (Address Setup and Hold)**

Figure 20-33 shows a write cycle with one clock of address setup and address hold.



**Figure 20-33. Longword-Write Burst to 8-Bit Port 3-1-1-1 (Address Setup and Hold)**

## 20.4.8 Misaligned Operands

Because operands, unlike opcodes, can reside at any byte boundary, they are allowed to be misaligned.

- Byte operand is properly aligned at any address
- Word operand is misaligned at an odd address
- Longword is misaligned at any address not a multiple of four

Although the processor enforces no alignment restrictions for data operands (including program counter (PC) relative data addressing), misaligned operands require additional bus cycles.

Instruction words and extension words (opcodes) must reside on word boundaries. Attempting to prefetch a misaligned instruction word causes an address-error exception.

The processor core converts misaligned, cache-inhibited operand accesses to multiple aligned accesses. [Example 20-1](#) shows the transfer of a longword operand from a byte address to a 32-bit port. First, a byte transfers at an offset of 0x1. The slave device supplies the byte and acknowledges the data transfer. When the processor starts the second cycle, a word transfers with a byte offset of 0x2. The next two bytes are transferred in this cycle. In the third cycle, byte 3 transfers. The byte offset is now 0x0, the port supplies the final byte, and the operation completes.

	31	24 23	16 15	8 7	0	FB_A[2:0]
Transfer 1	—	Byte 0	—	—	0	001
Transfer 2	—	—	Byte 1	Byte 2	1	010
Transfer 3	Byte 3	—	—	—	2	100

**Example 20-1. A Misaligned Longword Transfer (32-Bit Port)**

If an operand is cacheable and is misaligned across a cache-line boundary, both lines are loaded into the cache. The example in [Example 20-2](#) differs from the one in [Example 20-1](#) because the operand is word-sized and the transfer takes only two bus cycles.

	31	24 23	16 15	8 7	0	FB_A[2:0]
Transfer 1	—	—	—	Byte 0	0	001
Transfer 2	Byte 0	—	—	—	1	100

**Example 20-2. A Misaligned Word Transfer (32-Bit Port)**

## 20.4.9 Extended Transfer Start

The FB\_TS signal indicates that a bus transaction has begun and the address and attributes are valid. By default, the FB\_TS signal asserts for a single bus clock cycle see. When CSCRn[EXTS] is set, the transfer start signal asserts and remain asserted until the first positive clock edge after FB\_CS<sub>n</sub> asserts. See [Figure 20-34](#).

When EXTS is set, CSCRn[WS] must be programmed to have at least one primary wait state.

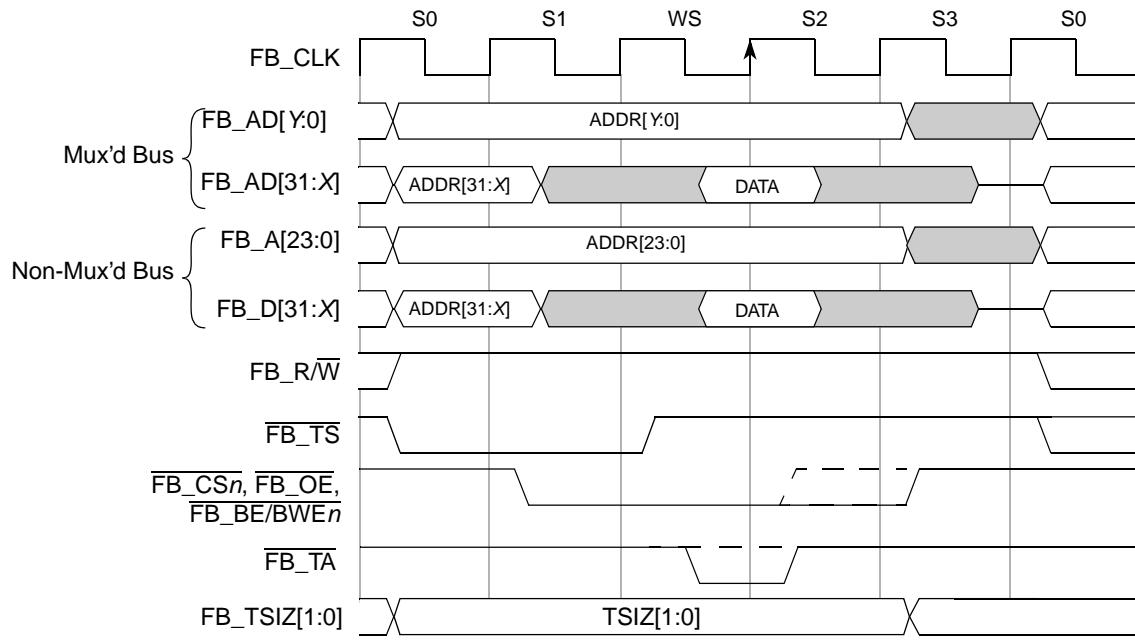


Figure 20-34. Read-Bus Cycle with  $\text{CSCR}_n[\text{EXTS}] = 1$  (One Wait State)

#### 20.4.10 Bus Errors

If the auto-acknowledge feature is not enabled for the address that generates the error, the bus cycle can be terminated by asserting **FB\_TA**. If the processor must manage a bus error differently, asserting an interrupt to the core along with **FB\_TA** when the bus error occurs can invoke an interrupt handler.

The device also includes a bus monitor that generates a bus error for unterminated cycles.

# Chapter 21

## DDR1/2 SDRAM Memory Controller (DDRAMC)

### 21.1 Overview

The DDR1/2 memory controller supports high performance applications for 8-bit DDR1 or DDR2 SDRAM memories. On the MCF5441x, DDR1 has not been completely characterized and is not recommended for use at this time.

#### 21.1.1 Block Diagram

Figure 21-1 shows a high-level block diagram of the DDR SDRAM controller.

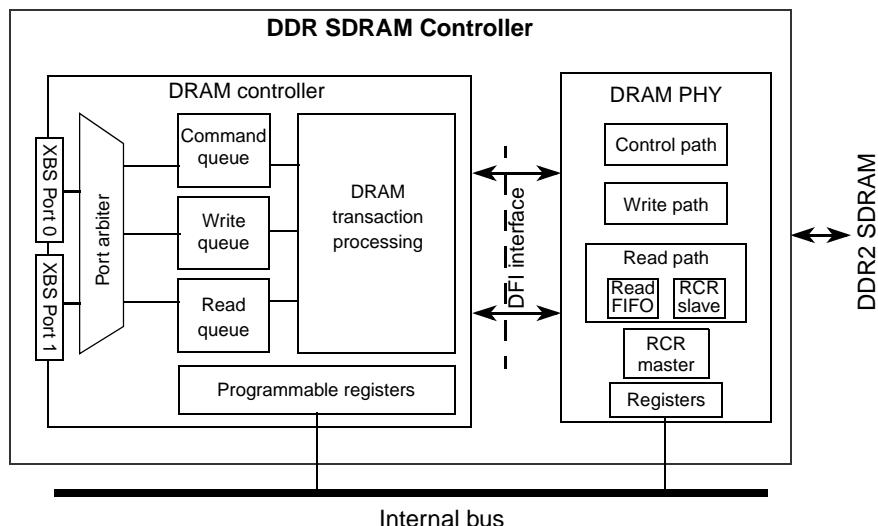


Figure 21-1. SDRAMC Block Diagram

#### 21.1.2 Features

The features of this memory controller include:

- Supports dual data rate (DDR) SDRAM
- Supports DDR2 SDRAM
- Fully pipelined command, read, and write data interfaces to the memory controller
- Advanced bank look-ahead features for high memory throughput
- Programmable register interface to control memory device parameters and protocols including auto pre-charge

- Full initialization of memory on memory controller reset
- Memory datapath size of 8-bit
- Clock frequencies up to 250 MHz supported (500 MBps data rate)
- Integrated with a DFI-compliant PHY, which contains a read clock recovery (RCR) block for reliable data capture timing
- On-die termination (ODT)
- Automatic refresh generation with programmable refresh intervals
- Self-refresh and power-down modes to reduce system power consumption

## 21.2 Modes of Operation

The DDR memory controller operates in the following modes.

### 21.2.1 DDR1 and DDR2

This is the main mode of normal operation. All other modes are parts of this mode.

### 21.2.2 Low Power Modes

There are several low power modes available with the memory controller. The low power modes are listed from least to most power saving.

#### NOTE

It is not possible to exit one low power mode and enter another low power mode simultaneously. Plan for a minimum delay between exit and entry of the two low power modes of 15 cycles in which the memory controller must remain stable.

#### 21.2.2.1 Memory Power-Down

The memory controller sets the memory devices into power-down, which reduces the overall power consumption of the system, but has the least effect of all the low power modes. In this mode, the memory controller and memory clocks are fully operational, but the CKE input bit to the memory devices is negated. The memory controller continues to monitor memory refresh needs and automatically brings the memory out of power-down to perform these refreshes. When a refresh is required, the CKE input bit to the memory devices is re-enabled, which brings the memory devices out of power-down. After the refresh completes, the memory devices are returned to power-down by negating the CKE input bit.

#### 21.2.2.2 Memory Power-Down with Memory Clock Gating

The memory controller sets the memory devices into power-down and gates off the clock to the memory devices. Refreshes are handled as in the memory power-down mode (mode 1), with the exception that gating on the memory clock is removed before asserting the CKE pin. After the refresh completes, the memory devices are returned to power-down with the clock gated. Before the memory devices are removed from power-down, the clock is gated on again.

**NOTE**

Do not use this mode for memory devices that do not support memory clock gating. Clock gating is not supported for standard DDR1 and DDR2 devices.

When set into this mode, the memory controller attempts to place the memory devices in power-down and gate off the memory clock. The memory functions unpredictably and may hang.

### **21.2.2.3 Memory Self-Refresh**

The memory controller sets the memory devices into self-refresh. In this mode, the memory controller and memory clocks are fully operational and the CKE input bit to the memory devices is negated. Since the memory automatically refreshes its contents, the memory controller does not need to send explicit refreshes to the memory.

### **21.2.2.4 Memory Self-Refresh with Memory Clock Gating**

The memory controller sets the memory devices into self-refresh and gates off the clock to the memory devices. Before the memory devices are removed from self-refresh, the clock are gated on again.

### **21.2.2.5 Memory Self-Refresh with Memory and Controller Clock Gating**

This is the deepest low power mode of the memory controller. The memory controller sets the memory devices into self-refresh and gates off the clock to the memory devices. In addition, the clock to the memory controller and the programming parameters are gated off, except to a small portion of the DLL, which must remain active to maintain the lock. Before the memory devices are removed from self-refresh, the memory controller and memory clocks are gated on.

## **21.3 Signal Description**

Table 21-1 describes the DDR1/2 controller's external memory interface signals.

**Table 21-1. Signal Properties**

Name	Function	I/O	Reset	Pull Up
SD_A[14:0]	Selects the column when <u>SD_CAS</u> is asserted Selects the row when <u>SD_RAS</u> is asserted.	O	0000	Active
SD_BA[2:0]	Selects the SDRAM Bank when <u>SD_RAS</u> is asserted (precharge or active) or when <u>SD_CAS</u> is asserted (read or write).	O	0	Passive
<u>SD_CAS</u>	Column address select	O	1	Active
SD_CKE	Clock enable. Asserts when the clock is valid	O	0	Active
<u>SD_CLK</u> <u>SD_CLK</u>	Differential clock	O	0 1	Active
<u>SD_CS</u>	Chip select	O	1	Active
SD_D[7:0]	Write or read data from the SDRAM	I/O	—	Active

**Table 21-1. Signal Properties (continued)**

Name	Function	I/O	Reset	Pull Up
SD_DM	Data mask	O	1	Active
SD_DQS SD_DQS	Differential data strobe	I/O	—	Active
SD_ODT	On-die-termination	O	0	Active
SD_RAS	Row address select	O	1	Active
SD_WE	Write enable	O	1	Active
SD_VREF	Voltage supply reference	—	—	—
SD_VTT	Voltage supply for termination resistors	—	—	—

### 21.3.1 Detailed Signal Descriptions

Table 21-2 describes the external signals in more detail with timing to SD\_CLK and how the signals interact with each other.

**Table 21-2. SDRAM Interface—Detailed Signal Descriptions**

Signal	I/O	Description	
SD_A[14:0]	O	Provides the row address for ACTIVE commands, and the column address and AUTO PRECHARGE bit for READ/WRITE commands, to select one location out of the memory array in the respective bank. A10 is sampled during a precharge command to determine whether the PRECHARGE applies to one bank (A10 LOW) or all banks (A10 HIGH). If only one bank is to be precharged, the bank is selected by BA0, BA1 and BA2. The address outputs also provide the op-code during a MODE REGISTER SET command. BA0, BA1 and BA2 define which mode register is loaded during the MODE REGISTER SET (MRS or EMRS's)	
		<b>Timing</b>	Assertion/Negation— Occurs synchronously with SD_CLK
SD_BA[2:0]	O	BA0, BA1 and BA2 define which bank an ACTIVE, READ, WRITE, or PRECHARGE command is being applied.	
		<b>Timing</b>	Assertion/Negation— Occurs synchronously with SD_CLK
SD_CAS	O	Command input. Along with SD_CS, SD_RAS, and SD_WE defines the current command.	
		<b>State Meaning</b>	Please see for the SDRAM commands.
		<b>Timing</b>	Assertion/Negation— Occurs synchronously with SD_CLK
SD_CKE	O	CKE must be maintained high throughout READ and WRITE accesses. Input buffers, excluding SD_CLK, SD_CLK, and SD_CKE, are disabled during POWER DOWN or SELF REFRESH.	
		<b>State Meaning</b>	Asserted— Activates internal clock signals and device input buffers and output drivers. Negated—Deactivates internal clock signals and device input buffers and output drivers.
		<b>Timing</b>	Assertion — Asynchronous for SELF-REFRESH exit and for output disable Negation— Occurs synchronously with SD_CLK

**Table 21-2. SDRAM Interface—Detailed Signal Descriptions (continued)**

<b>Signal</b>	<b>I/O</b>	<b>Description</b>
<b>SD_CLK</b> <b>SD_CLK̄</b>	O	SD_CLK and SD_CLK̄ are differential clock outputs. All address and control output signals are sent on the crossing of the positive edge of SD_CLK and the negative edge of SD_CLK̄. Output data is referenced to the crossing of SD_CLK and SD_CLK̄ (both directions of crossing).
		<b>Timing</b> Command signals are synchronously with the rising edge of this clock. The data signals can change on both the rising and falling edge of the clock.
<b>SD_CS</b>	O	SD_CS provides for external chip selection on systems with multiple chips. SD_CS̄ is considered part of the command code.
		<b>State Meaning</b> Asserted— Commands for the selected chip will occur Negated—All commands are masked.
		<b>Timing</b> Assertion/Negation— Occurs synchronously with SD_CLK
<b>SD_D[7:0]</b>	I/O	Data bus
		<b>Timing</b> Assertion/Negation— Occurs on both crossing of SD_CLK and SD_CLK̄ on write command. Synchronous with SD_DQS input on read command.
<b>SD_DM</b>	O	Output mask signal for write data. During Reads, SD_DM̄ may be driven high, low, or floating.
		<b>State Meaning</b> Asserted— Data is written to SDRAM Negation— Data is masked
		<b>Timing</b> Assertion/Negation— Occurs on both crossing of SD_CLK and SD_CLK̄.
<b>SD_DQS</b> <b>SD_DQS̄</b>	I/O	Edge-aligned with read data, centered in write data. Used to capture data.
		<b>State Meaning</b> Asserted— Similar to a clock signal, the edges are more important than being asserted or negated.
		<b>Timing</b> Assertion/Negation—Occurs on both crossing of SD_CLK and SD_CLK̄ on write command. Asynchronous with SD_CLK and SD_CLK̄ on read command.
<b>SD_ODT</b>	O	SD_ODT enables termination resistance internal to the DDR2 SDRAM.
		<b>State Meaning</b> Asserted— Enable termination resistance Negation— Disable termination resistance
		<b>Timing</b> Assertion/Negation— Occurs synchronously with SD_CLK
<b>SD_RAS</b>	O	Command input. Along with SD_CS, SD_CAS, and SD_WE defines the current command.
		<b>State Meaning</b> Please see <a href="#">Table 21-3</a> for SDRAM commands.
		<b>Timing</b> Assertion/Negation— Occurs synchronously with SD_CLK.
<b>SD_WE</b>	O	Command input. Along with SD_CS, SD_CAS, and SD_RAS defines the current command.
		<b>State Meaning</b> Please see <a href="#">Table 21-3</a> for SDRAM commands.
		<b>Timing</b> Assertion/Negation— Occurs synchronously with SD_CLK.
<b>SD_VREF</b>	—	SDRAM reference voltage. Reference voltage for differential I/O pad cells. Should be half the voltage of the memory used in the system. For example, 2.5V DDR results in an SD_VREF of 1.25V. See the device's datasheet for the voltages and tolerances for the various memory modes.
<b>SD_VTT</b>	—	Voltage supply for termination resistors

**Table 21-3. SDRAM Commands**

Function	CKE	CS	RAS	CAS	WE	BA[2:0]	A[10]	ADDR
(Extended) mode register set	H	L	L	L	L	V	V	V
Refresh	H	L	L	L	H	X	X	X
Self refresh entry	H→L	L	L	L	H	X	X	X
Self refresh exit	L→H	H	X	X	X	X	X	X
		L	H	H	H			
Single bank precharge	H	L	L	H	L	V	L	X
Precharge all banks	H	L	L	H	L	X	H	X
Bank activate	H	L	L	H	H	V	V	V
Write	H	L	H	L	L	V	L	V
Write with auto-precharge	H	L	H	L	L	V	H	V
Read	H	L	H	L	H	V	L	V
Read with auto-precharge	H	L	H	L	H	V	H	V
No operation	H	L	H	H	H	X	X	X
Device deselect	H	H	X	X	X	X	X	X
Power down entry	H→L	H	X	X	X	X	X	X
		L	H	H	H			
Power down exit	L→H	H	X	X	X	X	X	X
		L	H	H	H			
H = High L = Low V = Valid X = Don't Care								

## 21.4 Memory Map/Register Definition

Table 21-4 is the DDR1/2 controller memory map.

**Table 21-4. DDRMC Memory Map**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
<b>DDR controller registers</b>					
0xFC0B_8000	DDR control register 0 (DDR_CR00)	32	R/W	0x0000_0000	<a href="#">21.4.1/21-9</a>
0xFC0B_8004	DDR control register 1 (DDR_CR01)	32	R/W	0x0000_0000	<a href="#">21.4.2/21-9</a>
0xFC0B_8008	DDR control register 2 (DDR_CR02)	32	R/W	0x0000_0000	<a href="#">21.4.3/21-10</a>
0xFC0B_800C	DDR control register 3 (DDR_CR03)	32	R/W	0x0000_0000	<a href="#">21.4.4/21-11</a>
0xFC0B_8010	DDR control register 4 (DDR_CR04)	32	R/W	0x0000_0000	<a href="#">21.4.5/21-12</a>

**Table 21-4. DDRMC Memory Map (continued)**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0B_8014	DDR control register 5 (DDR_CR05)	32	R/W	0x0000_0000	<a href="#">21.4.6/21-12</a>
0xFC0B_8018	DDR control register 6 (DDR_CR06)	32	R/W	0x0000_0000	<a href="#">21.4.7/21-13</a>
0xFC0B_801C	DDR control register 7 (DDR_CR07)	32	R/W	0x0000_0000	<a href="#">21.4.8/21-14</a>
0xFC0B_8020	DDR control register 8 (DDR_CR08)	32	R/W	0x0000_0000	<a href="#">21.4.9/21-15</a>
0xFC0B_8024	DDR control register 9 (DDR_CR09)	32	R/W	0x0000_0000	<a href="#">21.4.10/21-15</a>
0xFC0B_8028	DDR control register 10 (DDR_CR10)	32	R/W	0x0000_0000	<a href="#">21.4.11/21-16</a>
0xFC0B_802C	DDR control register 11 (DDR_CR11)	32	R/W	0x0000_0000	<a href="#">21.4.12/21-17</a>
0xFC0B_8030	DDR control register 12 (DDR_CR12)	32	R/W	0x0000_0000	<a href="#">21.4.13/21-18</a>
0xFC0B_8034	DDR control register 13 (DDR_CR13)	32	R/W	0x0002_0000	<a href="#">21.4.14/21-19</a>
0xFC0B_8038	DDR control register 14 (DDR_CR14)	32	R/W	0x0000_0000	<a href="#">21.4.15/21-20</a>
0xFC0B_803C	DDR control register 15 (DDR_CR15)	32	R/W	0x0000_0000	<a href="#">21.4.16/21-21</a>
0xFC0B_8040	DDR control register 16 (DDR_CR16)	32	R/W	0x0000_0000	<a href="#">21.4.17/21-21</a>
0xFC0B_8044	DDR control register 17 (DDR_CR17)	32	R/W	0x0000_0000	<a href="#">21.4.18/21-22</a>
0xFC0B_8048	DDR control register 18 (DDR_CR18)	32	R/W	0x0000_0000	<a href="#">21.4.19/21-23</a>
0xFC0B_804C	DDR control register 19 (DDR_CR19)	32	R/W	0x0000_0000	<a href="#">21.4.20/21-24</a>
0xFC0B_8050	DDR control register 20 (DDR_CR20)	32	R/W	0x0C00_0000	<a href="#">21.4.21/21-24</a>
0xFC0B_8054	DDR control register 21 (DDR_CR21)	32	R/W	0x0000_0000	<a href="#">21.4.22/21-25</a>
0xFC0B_8058	DDR control register 22 (DDR_CR22)	32	R/W	0x0000_0000	<a href="#">21.4.23/21-26</a>
0xFC0B_805C	DDR control register 23 (DDR_CR23)	32	R/W	0x1000_0000	<a href="#">21.4.24/21-27</a>
0xFC0B_8060	DDR control register 24 (DDR_CR24)	32	R/W	0x0000_0000	<a href="#">21.4.25/21-28</a>
0xFC0B_8064	DDR control register 25 (DDR_CR25)	32	R/W	0x0000_0000	<a href="#">21.4.26/21-29</a>
0xFC0B_8068	DDR control register 26 (DDR_CR26)	32	R/W	0x0000_0000	<a href="#">21.4.27/21-30</a>
0xFC0B_806C	DDR control register 27 (DDR_CR27)	32	R	0x0000_0000	<a href="#">21.4.28/21-30</a>
0xFC0B_8070	DDR control register 28 (DDR_CR28)	32	R/W	0x0000_0000	<a href="#">21.4.29/21-31</a>
0xFC0B_8074	DDR control register 29 (DDR_CR29)	32	R/W	0x0000_0000	<a href="#">21.4.30/21-32</a>
0xFC0B_8078	DDR control register 30 (DDR_CR30)	32	R	0x0000_0000	<a href="#">21.4.31/21-33</a>
0xFC0B_807C	DDR control register 31 (DDR_CR31)	32	R/W	0x0000_0000	<a href="#">21.4.32/21-33</a>
0xFC0B_8080	DDR control register 32 (DDR_CR32)	32	R/W	0x0000_0000	<a href="#">21.4.33/21-34</a>
0xFC0B_8084	DDR control register 33 (DDR_CR33)	32	R/W	0x0000_0000	<a href="#">21.4.34/21-34</a>
0xFC0B_8088	DDR control register 33 (DDR_CR34)	32	R/W	0x0000_0000	<a href="#">21.4.35/21-35</a>
0xFC0B_808C	DDR control register 35 (DDR_CR35)	32	R/W	0x0000_0000	<a href="#">21.4.36/21-35</a>
0xFC0B_8090	DDR control register 36 (DDR_CR36)	32	R/W	0x0000_0000	<a href="#">21.4.37/21-36</a>

Table 21-4. DDRMC Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0B_8094	DDR control register 37 (DDR_CR37)	32	R/W	0x0000_0000	<a href="#">21.4.38/21-36</a>
0xFC0B_8098	DDR control register 38 (DDR_CR38)	32	R/W	0x0000_0000	<a href="#">21.4.39/21-36</a>
0xFC0B_809C	DDR control register 39 (DDR_CR39)	32	R/W	0x0000_0000	<a href="#">21.4.40/21-37</a>
0xFC0B_80A0	DDR control register 40 (DDR_CR40)	32	R/W	0x0000_0000	<a href="#">21.4.41/21-37</a>
0xFC0B_80A4	DDR control register 41 (DDR_CR41)	32	R/W	0x0000_0000	<a href="#">21.4.42/21-38</a>
0xFC0B_80A8	DDR control register 42 (DDR_CR42)	32	R/W	0x0000_0000	<a href="#">21.4.43/21-38</a>
0xFC0B_80AC	DDR control register 43 (DDR_CR43)	32	R/W	0x0000_0000	<a href="#">21.4.44/21-38</a>
0xFC0B_80B0	DDR control register 44 (DDR_CR44)	32	R	0x0000_2040	<a href="#">21.4.45/21-39</a>
0xFC0B_80B4	DDR control register 45 (DDR_CR45)	32	R/W	0x0000_0000	<a href="#">21.4.46/21-39</a>
0xFC0B_80B8	Reserved	32	—	—	<a href="#">21.4.47/21-39</a>
0xFC0B_80BC	Reserved	32	—	—	<a href="#">21.4.48/21-40</a>
0xFC0B_80C0	Reserved	32	—	—	<a href="#">21.4.49/21-40</a>
0xFC0B_80C4	Reserved	32	—	—	<a href="#">21.4.50/21-40</a>
0xFC0B_80C8	Reserved	32	—	—	<a href="#">21.4.51/21-40</a>
0xFC0B_80CC	Reserved	32	—	—	<a href="#">21.4.52/21-40</a>
0xFC0B_80D0	Reserved	32	—	—	<a href="#">21.4.53/21-40</a>
0xFC0B_80D4	DDR control register 53 (DDR_CR53)	32	R	0x0000_0000	<a href="#">21.4.54/21-40</a>
0xFC0B_80D8	Reserved	32	—	—	<a href="#">21.4.55/21-40</a>
0xFC0B_80DC	DDR control register 55 (DDR_CR55)	32	R	0x0000_0000	<a href="#">21.4.56/21-41</a>
0xFC0B_80E0	DDR control register 56 (DDR_CR56)	32	R/W	0x0000_0000	<a href="#">21.4.57/21-41</a>
0xFC0B_80E4	DDR control register 57 (DDR_CR57)	32	R/W	0x0000_0000	<a href="#">21.4.58/21-42</a>
0xFC0B_80E8	DDR control register 58 (DDR_CR58)	32	R/W	0x0000_0000	<a href="#">21.4.59/21-42</a>
0xFC0B_80EC	DDR control register 59 (DDR_CR59)	32	R/W	0x0000_0000	<a href="#">21.4.60/21-43</a>
0xFC0B_80F0	DDR control register 60 (DDR_CR60)	32	R/W	0x0000_0000	<a href="#">21.4.61/21-43</a>
0xFC0B_80F4	Reserved	32	R/W	0x0000_0000	<a href="#">21.4.62/21-43</a>
0xFC0B_80F8	Reserved	32	R/W	0x0000_0000	<a href="#">21.4.63/21-44</a>
0xFC0B_80FC	Reserved	32	R/W	0x0000_0000	<a href="#">21.4.64/21-44</a>
0xFC0B_8180	RCR control register (DDR_RCR)	32	R/W	0x0000_0000	<a href="#">21.4.65/21-44</a>
0xFC0B_81AC	DDR I/O pad control register (DDR_PADCR)	32	R/W	0x0000_0000	<a href="#">21.4.66/21-44</a>

## 21.4.1 DDR Control Register 0 (DDR\_CR00)

Address: 0xFC0B\_8000 (DDR\_CR00)

Access: User 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	0	0	0	0	0	0	0	WP	0	0	0	0	0	0	0	RP	0	0	0	0	0	0	ADD	0	0	0	0	0	0	AGE		
W								0								0							COL									
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		

Figure 21-2. DDR Control Register 0 (DDR\_CR00)

Table 21-5. DDR\_CR00 Field Descriptions

Field	Description
31–25	Reserved, must be cleared.
24 WP0	Sets the port 0 write command priority. 0 Highest 1 Lowest
23–17	Reserved, must be cleared.
16 RP0	Sets the port 0 read command priority. 0 Highest 1 Lowest
15–9	Reserved, must be cleared.
8 ADDCOL	Enables address collision/data coherency detection as a condition when using the placement logic to fill the command queue. 0 Disable 1 Enable
7–1	Reserved, must be cleared.
0 AGE	Enables aging of commands in the command queue when using the placement logic to fill the command queue. The total number of cycles required to decrement the priority value on a command by one is the product of the values in DDR_CR16[AGECNT] and DDR_CR17[CMDAGE] parameters. 0 Disable 1 Enable

## 21.4.2 DDR Control Register 1 (DDR\_CR01)

Address: 0xFC0B\_8004 (DDR\_CR01)

Access: User 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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RP	0	0	0	0	0	0	WP	0	0	0	0	0	0	AGE		
W																AREF							1									
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		

Figure 21-3. DDR Control Register 1 (DDR\_CR01)

**Table 21-6. DDR\_CR01 Field Descriptions**

Field	Description
31–25	Reserved, must be cleared.
24 AREF	Trigger auto-refresh at boundary specified by AUTO_REFRESH_MODE. This bit is write-only and always reads zero. If there are any open banks when this parameter is set, the memory controller automatically closes these banks before issuing the auto-refresh command. 0 No action 1 Issue auto-refresh to the DRAM devices.
23–17	Reserved, must be cleared.
16 AP	Auto-precharge enable for DRAM devices. 0 Disable. Memory banks stay open until another request requires this bank, the maximum open time (TRAS_MAX) elapses, or a refresh command closes all banks. 1 Enable. All read and write transactions must be terminated by an auto precharge command. If a transaction consists of multiple read or write bursts, only the last command is issued with an auto precharge. <b>Note:</b> This bit cannot be changed after DDR_CR09[START] is set.
15–9	Reserved, must be cleared.
8 WP1	Sets the port 1 write command priority. 0 Highest 1 Lowest
7–1	Reserved, must be cleared.
0 RP1	Sets the port 1 read command priority. 0 Highest 1 Lowest

### 21.4.3 DDR Control Register 2 (DDR\_CR02)

Address: 0xFC0B\_8008 (DDR\_CR02)

Access: User 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	0	0	0	0	0	0	0	0	CC	0	0	0	0	0	0	BIG	0	0	0	0	0	0	BNK	0	0	0	0	0	0	AREF	MODE	
W									APEN							END								SPT								
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		

**Figure 21-4. DDR Control Register 2 (DDR\_CR02)****Table 21-7. DDR\_CR02 Field Descriptions**

Field	Description
31–25	Reserved, must be cleared.
24 CCAPEN	Concurrent auto-precharge enable. Allow controller to issue commands to other banks while a bank is in auto precharge. Some DRAM devices do not allow one bank to auto pre-charge while another bank is reading or writing. The JEDEC standard allows concurrent auto pre-charge. Set this parameter for the DRAM device being used. 0 Disabled 1 Enabled
23–17	Reserved, must be cleared.

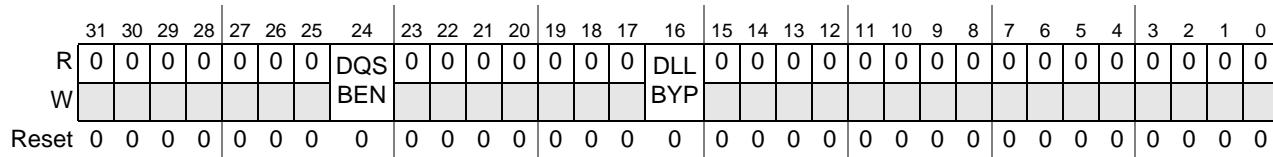
**Table 21-7. DDR\_CR02 Field Descriptions (continued)**

Field	Description
16 BIGEND	Big endian enable. Set byte ordering as little endian or big endian. 0 Little endian 1 Big endian
15–9	Reserved, must be cleared.
8 BNKSPT	Bank split enable for command queue placement logic. 0 Disabled 1 Enabled
7–1	Reserved, must be cleared.
0 AREFMODE	Auto refresh mode. Define auto refresh to occur at the next burst or command boundary. 0 Issue refresh on the next DRAM burst boundary, even if the current command is not complete 1 Issue refresh on the next command boundary. If a refresh is required to memory, the controller delays this refresh until the end of the current transaction (if the transaction is fully contained inside a single page), or until the current transaction hits the end of the current page.

#### 21.4.4 DDR Control Register 3 (DDR\_CR03)

Address: 0xFC0B\_800C (DDR\_CR03)

Access: User read/write

**Figure 21-5. DDR Control Register 3 (DDR\_CR03)****Table 21-8. DDR\_CR03 Field Descriptions**

Field	Description
31–25	Reserved, must be cleared.
24 DQSBEN	Configures the DQS pin as single-ended or differential. 0 Single-ended 1 Differential
23–17	Reserved, must be cleared.
16 DLLBYP	DLL bypass enable. <b>Note:</b> This bit must be set.
15–0	Reserved, must be cleared.

## 21.4.5 DDR Control Register 4 (DDR\_CR04)

Address: 0xFC0B\_8010 (DDR\_CR04)

Access: User 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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	QK	0	0	0	0	0	0	0	8	0	0	0	0	0	0	DRV	
W																REF								BNK							DQS	
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	

Figure 21-6. DDR Control Register 4 (DDR\_CR04)

Table 21-9. DDR\_CR04 Field Descriptions

Field	Description
31–17	Reserved, must be cleared.
16 QKREF	Enable quick self-refresh. Allows user to interrupt memory initialization to enter self-refresh mode when a power loss is detected during the initialization process. 0 Continue memory initialization 1 Interrupt memory initialization and enter self-refresh mode
15–9	Reserved, must be cleared.
8 8BNK	Eight bank mode. Number of banks on the DRAMs. 0 4 banks 1 8 banks
7–1	Reserved, must be cleared.
0 DRV DQS	Selects if the DQ/DQS output enables are driven active when controller is idle. 0 Deassert the output enables when idle 1 Drive the output enables active when idle

## 21.4.6 DDR Control Register 5 (DDR\_CR05)

Address: 0xFC0B\_8014 (DDR\_CR05)

Access: User 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	0	0	0	0	0	0	0	INT	0	0	0	0	0	0	0	INT	0	0	0	0	0	0	0	INT	0	0	0	0	0	0	FST	
W								WR								RD								PRE							WR	
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	

Figure 21-7. DDR Control Register 5 (DDR\_CR05)

Table 21-10. DDR\_CR05 Field Descriptions

Field	Description
31–25	Reserved, must be cleared.
24 INTWR	Allows the controller to interrupt a combined write command with auto precharge with another write command to the same bank. 0 Disable 1 Enable
23–17	Reserved, must be cleared.

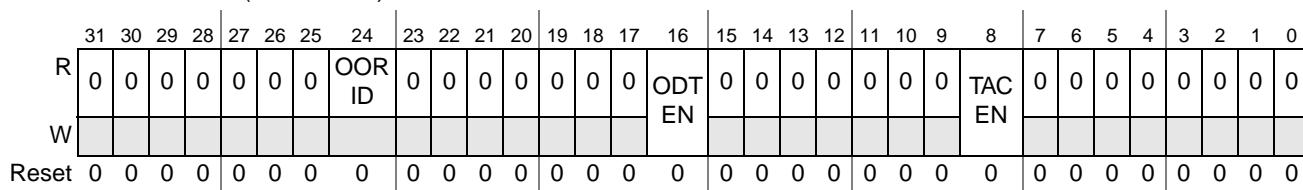
**Table 21-10. DDR\_CR05 Field Descriptions (continued)**

Field	Description
16 INTRD	Allows the controller to interrupt a combined read command with auto-precharge with another read command to the same bank. 0 Disable 1 Enable
15–9	Reserved, must be cleared.
8 INTPRE	Allows the controller to interrupt an auto-precharge command with another command. 0 Disable 1 Enable. The current operation is interrupted. However, the bank is precharged as if the current operation were allowed to continue.
7–1	Reserved, must be cleared.
0 FSTWR	Defines when write commands are issued to DRAM devices. 0 The memory controller issues a write command to the DRAM devices when it has received enough data for one DRAM burst. Write data can be sent in any cycle relative to the write command. This mode also allows for multi-word write command data to arrive in non-sequential cycles. 1 The memory controller issues a write command to the DRAM devices after the first word of the write data is received by the memory controller. The first word can be sent at any time relative to the write command. In this mode, multi-word write command data must be available to the memory controller in sequential cycles.

### 21.4.7 DDR Control Register 6 (DDR\_CR06)

Address: 0xFC0B\_8018 (DDR\_CR06)

Access: User read/write

**Figure 21-8. DDR Control Register 6 (DDR\_CR06)****Table 21-11. DDR\_CR06 Field Descriptions**

Field	Description
31–25	Reserved, must be cleared.
24 OORID	Source ID of the command that caused an out-of-range interrupt request. This bit is read only.
23–17	Reserved, must be cleared.
16 ODTEN	Indicates that ODT with CAS latency 3 is supported. This bit must always be set. 0 Reserved 1 ODT with CAS latency 3 is supported but is not DFI compliant. This disables the interrupt bit for ODT-with-CAS3 and disables the OVL error.
15–9	Reserved, must be cleared.

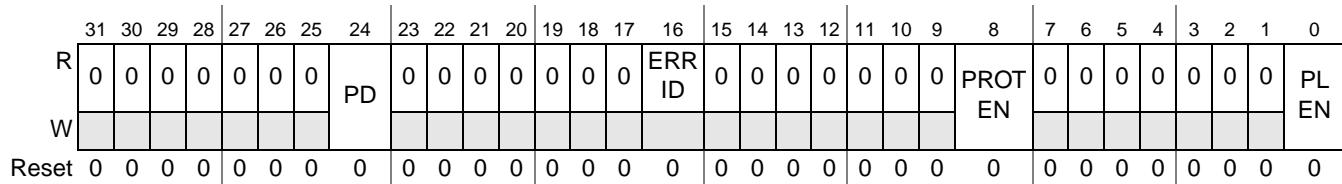
**Table 21-11. DDR\_CR06 Field Descriptions (continued)**

Field	Description
8 TACEN	Enable extra turn-around clock between back-to-back reads/writes to different chip selects. The additional clock may be needed at higher clock frequencies. The turn-off and turn-on time of termination resistors are not scalable. At higher clock frequencies, it is possible that these times may overlap, resulting in two active resistors while the DQS line is still active. This could compromise the signal integrity of the DQS signal. The additional clock prevents this overlap. 0 No additional clocking required 1 Additional clock added for back-to-back reads or writes that occur to different banks
7–0	Reserved, must be cleared.

## 21.4.8 DDR Control Register 7 (DDR\_CR07)

Address: 0xFC0B\_801C (DDR\_CR07)

Access: User read/write

**Figure 21-9. DDR Control Register 7 (DDR\_CR07)****Table 21-12. DDR\_CR07 Field Descriptions**

Field	Description
31–25	Reserved, must be cleared.
24 PD	Power down. Disable clock enable and set DRAMs in the power-down state. 0 Enable full power state 1 The memory controller completes processing of the current burst for the current transaction (if any), issues a precharge all command, and disables the clock enable signal to the DRAM devices. Any subsequent commands in the command queue are suspended until this bit is cleared.
23–17	Reserved, must be cleared.
16 ERRID	Port command error ID. Source ID of the command that caused the port command error. This bit is read-only.
15–9	Reserved, must be cleared.
8 PROTEN	Enables port address range protection and interrupt generation logic. 0 Disabled 1 Enabled. All incoming addresses are checked against valid address ranges and an out-of-range interrupt occurs if the check fails.
7–1	Reserved, must be cleared.
0 PLEN	Placement enable. Enable placement logic to fill the command queue. 0 Disabled. The command queue is a straight FIFO. 1 Enabled. The command queue is filled according to the placement logic factors.

## 21.4.9 DDR Control Register 8 (DDR\_CR08)

Address: 0xFC0B\_8020 (DDR\_CR08)

Access: User 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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	R2R	0	0	0	0	0	0	0	PU	0	0	0	0	0	0	PRI EN	
W																TAC								REF								
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		

Figure 21-10. DDR Control Register 8 (DDR\_CR08)

Table 21-13. DDR\_CR08 Field Descriptions

Field	Description
31–25	Reserved, must be cleared.
24 REDUC	Controls the width of the memory datapath. 0 Reserved 1 8-bit <b>Note:</b> The entire user datapath is used regardless of this setting. Only burst length value of 4 is supported.
23–17	Reserved, must be cleared.
16 R2RTAC	Adds an extra turn-around clock for back-to-back reads to different chip selects. 0 Disabled 1 Enabled
15–9	Reserved, must be cleared.
8 PUREF	Allow power-up via self-refresh instead of full memory initialization. This field allows you to skip full initialization when the DRAM devices are in a known self-refresh state. 0 Disabled 1 Enabled <b>Note:</b> For this silicon revision, clear this bit. This may result in a $t_{XSNR}$ violation. Consult your memory vendor to understand the exact implications of this.
7–1	Reserved, must be cleared.
0 PRIEN	Enable priority for command queue placement logic. 0 Disabled 1 Enabled

## 21.4.10 DDR Control Register 9 (DDR\_CR09)

Address: 0xFC0B\_8024 (DDR\_CR09)

Access: User 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	0	0	0	0	0	0	0	0	STA	0	0	0	0	0	0	SR	0	0	0	0	0	0	0	RW	0	0	0	0	0	0	REG DIMM	
W									RT							EF								EN								
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		

Figure 21-11. DDR Control Register 9 (DDR\_CR09)

Table 21-14. DDR\_CR09 Field Descriptions

Field	Description
31–25	Reserved, must be cleared.
24 START	Initiate command processing in the controller. 0 The memory controller does not issue any commands to the DRAM devices or respond to any signal activity except for reading and writing parameters. 1 After setting this bit, the memory controller responds to inputs from the ASIC. The memory controller begins its initialization routine. <b>Note:</b> You must wait for the initialization complete bit to set in DDR_CR27[INT_STATUS] and for the <i>dfl_init_complete</i> signal to be asserted from the PHY before submitting any transactions.
23–17	Reserved, must be cleared.
16 SREF	Place DRAMs into self-refresh mode. 0 Disable self-refresh mode. 1 Initiate self-refresh mode of the DRAM devices. The burst of the current transaction (if any) completes, all banks are closed, the self-refresh command is issued to the DRAM, and the clock enable signal is negated. The system remains in self-refresh mode until this bit is cleared. The DRAM devices return to normal operating mode after the self-refresh exit time (TXSR) of the device. The memory controller resumes processing of the commands from the interruption point.
15–9	Reserved, must be cleared.
8 RWEN	Enable read/write grouping for command queue placement logic. 0 Disabled 1 Enabled
7–1	Reserved, must be cleared.
0 REGDIMM	Enables registered DIMM operations to control the address and command pipeline of the memory controller. 0 Normal operation 1 Enable registered DIMM operation

### 21.4.11 DDR Control Register 10 (DDR\_CR10)

Address: 0xFC0B\_8028 (DDR\_CR10)

Access: User 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	0	0	0	0	0	0	0	INT	0	0	0	0	0	0	0	TREF	0	0	0	0	0	0	0	TRAS	0	0	0	0	0	0	SWP	EN
W								WBR								EN								TRAS								
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		

Figure 21-12. DDR Control Register 10 (DDR\_CR10)

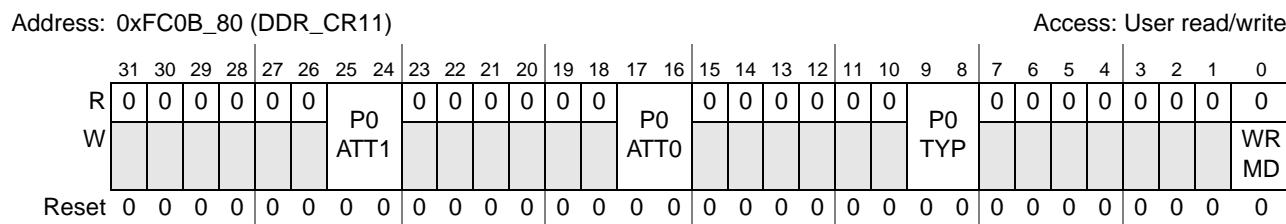
Table 21-15. DDR\_CR10 Field Descriptions

Field	Description
31–25	Reserved, must be cleared.
24 INTWBR	Allows the controller to interrupt a write burst to the DRAMs with a read command. Some memory devices do not support this functionality. 0 Read commands cannot interrupt write commands 1 Read commands can interrupt write commands

**Table 21-15. DDR\_CR10 Field Descriptions (continued)**

Field	Description
23–17	Reserved, must be cleared.
16 TREFEN	Enables refresh commands. If command refresh mode is configured, then refresh commands are automatically issued based on the internal DDR_CR31[TREF] counter and any refresh commands sent through the command interface or the register interface. 0 Refresh commands disabled 1 Refresh commands enabled
15–9	Reserved, must be cleared.
8 TRAS	Defines the t <sub>TRAS</sub> lockout setting for the DRAM device. t <sub>TRAS</sub> lockout allows the memory controller to execute auto pre-charge commands before the TRAS_MIN parameter expires. 0 t <sub>TRAS</sub> lockout not supported by memory device 1 t <sub>TRAS</sub> lockout supported by memory device
7–1	Reserved, must be cleared.
0 SWPEN	Enables swapping of the active command for a new higher-priority command when using the placement logic. 0 Disabled 1 Enabled

## 21.4.12 DDR Control Register 11 (DDR\_CR11)

**Figure 21-13. DDR Control Register 11 (DDR\_CR11)****Table 21-16. DDR\_CR11 Field Descriptions**

Field	Description
31–26	Reserved, must be cleared.
25–24 P0ATT1	Allowed transaction types for port 0 address range 1. This field is only used when DDR_CR07[PROTEN] is set to verify incoming addresses are of a valid type and range. 00 No access 01 Read only 10 Write only 11 Read and write <b>Note:</b> This field resets to no access. Therefore, you must initialize this field to access the memory.
23–18	Reserved, must be cleared.

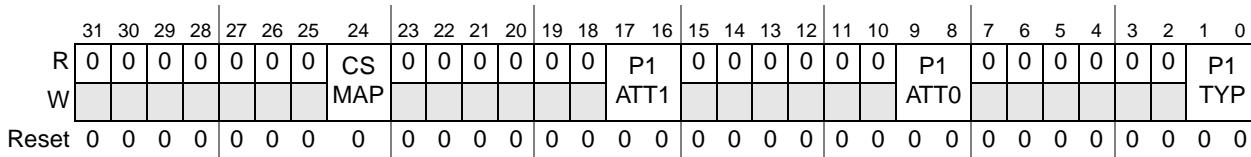
**Table 21-16. DDR\_CR11 Field Descriptions (continued)**

Field	Description
17–16 P0ATT0	Allowed transaction types for port 0 address range 0. This field is only used when DDR_CR07[PROTEN] is set to verify incoming addresses are of a valid type and range. 00 No access 01 Read only 10 Write only 11 Read and write <b>Note:</b> This field resets to no access. Therefore, you must initialize this field to access the memory.
15–10	Reserved, must be cleared.
9–8 P0TYP	Clock domain relativity between port 0 and the controller core. 00 Asynchronous 01 2:1 port:core pseudo-synchronous 01 1:2 port:core pseudo-synchronous 11 Synchronous
7–1	Reserved, must be cleared.
0 WRMD	Write mode register data to the DRAMs. This bit is write-only. 0 No write occurs 1 Write the mode parameters (EMRS register) in the DRAM devices. Each subsequent setting of this bit writes the EMRS register of the next chip select. This parameter always reads zero. The mode registers are automatically written at initialization of the memory controller. There is no need to initiate a mode register write after setting DDR_CR09[START], unless the values in these registers needs to be changed after initialization. <b>Note:</b> This parameter may not be changed when the memory is in power-down mode (CKE is negated).

### 21.4.13 DDR Control Register 12 (DDR\_CR12)

Address: 0xFC0B\_8030 (DDR\_CR12)

Access: User read/write

**Figure 21-14. DDR Control Register 12 (DDR\_CR12)****Table 21-17. DDR\_CR12 Field Descriptions**

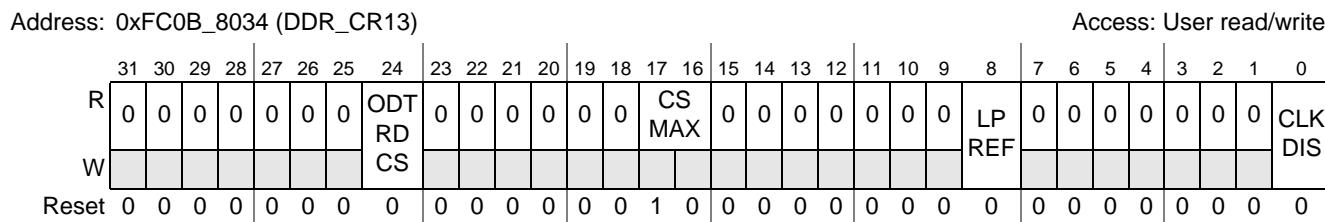
Field	Description
31–25	Reserved, must be cleared.
24 CSMAP	Defines if the chip select is enabled. 0 Chip select disabled 1 Chip select enabled
23–18	Reserved, must be cleared.

**Table 21-17. DDR\_CR12 Field Descriptions (continued)**

Field	Description
17–16 P1ATT1	Allowed transaction types for port 1 address range 1. This field is only used when DDR_CR07[PROTEN] is set to verify incoming addresses are of a valid type and range. 00 No access 01 Read only 10 Write only 11 Read and write <b>Note:</b> This field resets to no access. Therefore, you must initialize this field to access the memory.
15–10	Reserved, must be cleared.
9–8 P1ATT0	Allowed transaction types for port 1 address range 0. This field is only used when DDR_CR07[PROTEN] is set to verify incoming addresses are of a valid type and range. 00 No access 01 Read only 10 Write only 11 Read and write <b>Note:</b> This field resets to no access. Therefore, you must initialize this field to access the memory.
7–2	Reserved, must be cleared.
1–0 P1TYP	Clock domain relativity between port 1 and the controller core. 00 Asynchronous 01 2:1 port:core pseudo-synchronous 01 1:2 port:core pseudo-synchronous 11 Synchronous

#### 21.4.14 DDR Control Register 13 (DDR\_CR13)

DDR\_CR13[ODTRDCS] must be set.

**Figure 21-15. DDR Control Register 13 (DDR\_CR13)****Table 21-18. DDR\_CR13 Field Descriptions**

Field	Description
31–25	Reserved, must be cleared.
24 ODTRDCS	Determines if the chip select has termination when a read occurs on the chip select. 0 Reserved 1 CS has active ODT termination when CS performs a read <b>Note:</b> This bit must be set.
23–18	Reserved, must be cleared.

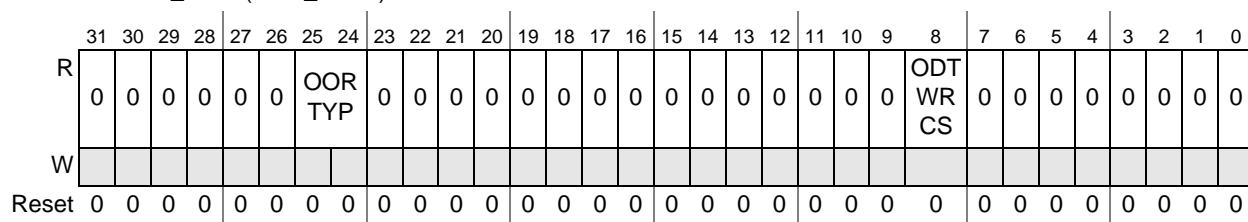
**Table 21-18. DDR\_CR13 Field Descriptions (continued)**

Field	Description
17–16 CSMAX	Maximum number of chip selects supported by the memory controller. This bit is read-only. 00 Zero 01 One 10 Two 11 Reserved <b>Note:</b> This field may not necessarily indicate how many chip selects are available externally on this device.
15–9	Reserved, must be cleared.
8 LPREF	Disables refreshes while in low power mode. 0 Refreshes occur 1 Refreshes do not occur
7–1	Reserved, must be cleared.
0 CLKDIS	Enables/disables the memory clocks. 0 Enable memory clocks 1 Disable memory clocks

### 21.4.15 DDR Control Register 14 (DDR\_CR14)

Address: 0xFC0B\_8038 (DDR\_CR14)

Access: User read/write

**Figure 21-16. DDR Control Register 14 (DDR\_CR14)****Table 21-19. DDR\_CR14 Field Descriptions**

Field	Description
31–26	Reserved, must be cleared.
25–24 OORTYP	Type of command that caused an out-of-range interrupt request. This bit is read-only.
23–9	Reserved, must be cleared.
8 ODTWRCS	Determines if the chip select has termination when a write occurs on the chip select. 0 Reserved 1 CS has active ODT termination when the CS performs a write <b>Note:</b> This bit must be set.
7–0	Reserved, must be cleared.

## 21.4.16 DDR Control Register 15 (DDR\_CR15)

Address: 0xFC0B_803C (DDR_CR15)																Access: User read/write									
R																W									
ADD PINS																ODT RES									
Reset																QFU LL									
0 0																0 0 0 0 0 0 0 0 0 0									

Figure 21-17. DDR Control Register 15 (DDR\_CR15)

Table 21-20. DDR\_CR15 Field Descriptions

Field	Description
31–27	Reserved, must be cleared.
26–24 ADDPINS	Defines the difference between the maximum number of address pins configured (16) and the actual number of pins used. The user address is automatically shifted so that the user address space is mapped contiguously into the memory map based on the value of this parameter. For details, refer to <a href="#">Section 21.5.2.1, “DDR SDRAM Address Mapping Options”</a> .
23–10	Reserved, must be cleared.
9–8 ODTRES	On-die termination resistance setting for all DRAM devices. 00 Termination disabled 01 75 Ohm 10 150 Ohm 11 Reserved
7–2	Reserved, must be cleared.
1–0 QFULL	Quantity that determines the command queue is full.

## 21.4.17 DDR Control Register 16 (DDR\_CR16)

Address: 0xFC0B_8040 (DDR_CR16)																Access: User read/write									
R																W									
COLSIZ																CKEDLY									
0 0																CASLAT									

Figure 21-18. DDR Control Register 16 (DDR\_CR16)

Table 21-21. DDR\_CR16 Field Descriptions

Field	Description
31–27	Reserved, must be cleared.
26–24 COLSIZ	Difference between the 12 column pins available and the number being used. The user address is automatically shifted so that the user address space is mapped contiguously into the memory map based on the value of this parameter. For details, refer to <a href="#">Section 21.5.2.1, “DDR SDRAM Address Mapping Options”</a> .
23–19	Reserved, must be cleared.

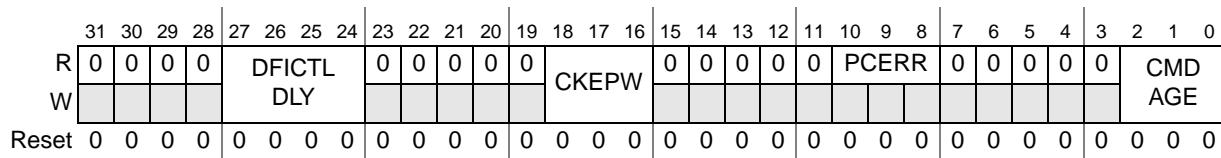
**Table 21-21. DDR\_CR16 Field Descriptions (continued)**

Field	Description
18–16 CKEDLY	Additional cycles to delay CKE for status reporting.
15–11	Reserved, must be cleared.
10–8 CASLAT	Encoded column address strobe (CAS) latency sent to DRAMs during initialization. The value programmed into this parameter is dependent on the memory device, since the same CASLAT value may have different meanings to different memories. This is programmed into the DRAM devices at initialization. The CAS encoding is specified in the DRAM data sheet, and should correspond to the DDR_CR20[LATLIN] parameter.
7–3	Reserved, must be cleared.
2–0 AGECNT	Initial value of master aging-rate counter for command aging. When using the placement logic to fill the command queue, the command aging counters are decremented one each time the master aging-rate counter counts down AGECNT cycles.

### 21.4.18 DDR Control Register 17 (DDR\_CR17)

Address: 0xFC0B\_8044 (DDR\_CR17)

Access: User read/write

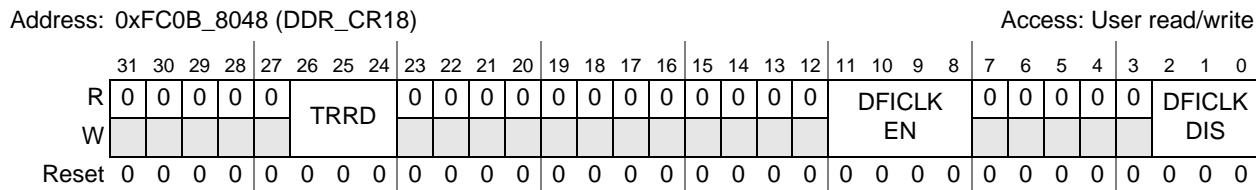
**Figure 21-19. DDR Control Register 17 (DDR\_CR17)****Table 21-22. DDR\_CR17 Field Descriptions**

Field	Description
31–28	Reserved, must be cleared.
27–24 DFICTLDLY	Contains the DFI $t_{CTRL\_DELAY}$ timing parameter. This field should be programmed with the number of cycles that the PHY requires to send a power-down or self-refresh command to the DRAM devices.
23–19	Reserved, must be cleared.
18–16 CKEPW	Minimum CKE pulse width in cycles.
15–11	Reserved, must be cleared.

**Table 21-22. DDR\_CR17 Field Descriptions (continued)**

Field	Description
10–8 PCERR	Type of error and access type that caused the port command error. This bit is read-only. <b>Bit 2:</b> 0 No address range error 1 The input address was outside all valid address ranges. Valid addresses are within PnSTAD $m$ and PnENDAD $m$ . <b>Bit 1:</b> 0 No transaction type error 1 The input address is valid, but the transaction type requested (read, write) was not valid for the address range. Valid types are defined in the PnATT $m$ fields. <b>Bit 0:</b> Indicates the transaction type that generated the error 0 Read 1 Write
7–3	Reserved, must be cleared.
2–0 CMDAGE	Initial value of individual command aging counters associated with each command in the command queue. When using the placement logic to fill the command queue, the command aging counters decrement one each time the master aging-rate counter counts down DDR_CR16[AGECNT] cycles.

### 21.4.19 DDR Control Register 18 (DDR\_CR18)

**Figure 21-20. DDR Control Register 18 (DDR\_CR18)****Table 21-23. DDR\_CR18 Field Descriptions**

Field	Description
31–27	Reserved, must be cleared.
26–24 TRRD	Defines the DRAM activate to activate delay for different banks (TRRD) in cycles.
23–12	Reserved, must be cleared.
11–8 DFICLKEN	Contains the DFI t <sub>DRAM_CLK_ENABLE</sub> timing parameter. <b>Note:</b> This parameter is currently unused in the memory controller.
7–3	Reserved, must be cleared.
2–0 DFICLKDIS	Contains the DFI t <sub>DRAM_CLK_DISABLE</sub> timing parameter. Program this field with the number of cycles that the PHY requires to disable the clock after the <i>dfi_dram_clk_disable</i> signal is asserted. The recommended value for this field is 0x2.

## 21.4.20 DDR Control Register 19 (DDR\_CR19)

Address: 0xFC0B\_804C (DDR\_CR19) Access: User 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	0	0	0	0	0	0	0	0	0	0	0	0	WRLAT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		
W					APREBIT												TWTR												TRTP				
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 21-21. DDR Control Register 19 (DDR\_CR19)

Table 21-24. DDR\_CR19 Field Descriptions

Field	Description
31–28	Reserved, must be cleared.
27–24 APREBIT	Location of the auto precharge bit in the DRAM address in decimal encoding.
23–20	Reserved, must be cleared.
19–16 WRLAT	Defines the DRAM write latency (WRLAT) when the write command is issued to the time the write data is presented to the DRAM devices in cycles. <b>Note:</b> This parameter must be set to 0x1 when the memory controller is in DDR1 mode.
15–12	Reserved, must be cleared.
11–8 TWTR	Sets the number of cycles needed to switch from a write to a read operation, as dictated by the DDR SDRAM specification.
7–3	Reserved, must be cleared.
2–0 TRTP	Defines the DRAM read to precharge time (TRTP) in cycles.

## 21.4.21 DDR Control Register 20 (DDR\_CR20)

Address: 0xFC0B\_8050 (DDR\_CR20) Access: User 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	0	0	0	0	MAXCOL	0	0	0	0				AREFINIT	0	0	0	0				LATGATE	0	0	0	0	0	0	0	0	0	0	
W																																
Reset	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	0	

Figure 21-22. DDR Control Register 20 (DDR\_CR20)

**Table 21-25. DDR\_CR20 Field Descriptions**

Field	Description
31–28	Reserved, must be cleared.
27–24 MAXCOL	Maximum width of the column address in the DRAM devices. This field is read-only. This value can be used to set the DDR_CR16[COLSIZ] field. COLSIZ = MAXCOL – number of column bits in the memory device 0x0 Zero ... 0xC 12 Else Reserved
23–20	Reserved, must be cleared.
19–16 AREFINIT	Number of auto-refresh commands to execute during DRAM initialization.
15–12	Reserved, must be cleared.
11–8 LATGATE	Adjusts data capture gate open time by half cycles. This parameter is programmed differently than LATLIN field when there are fixed offsets in the flight path between the memories and the memory controller for clock gating. When this field is larger than LATLIN, the data capture window becomes shorter. A value smaller than LATLIN may have no effect on the data capture window, depending on the fixed offsets in the ASIC and the board.
7–3	Reserved, must be cleared.
2–0 LATLIN	Sets the CAS latency linear value in half cycle increments. This sets an internal adjustment for the delay from when the read command is sent from the memory controller to when data is received back. The window of time in which the data is captured is a fixed length. This field adjusts the start of this data capture window. Not all linear values are supported for the memory devices being used. Refer to the device's data sheet for valid values. 0000 Reserved 0001 Reserved 0010 1 cycle 0011 1.5 cycles ... 1111 7.5 cycles

### 21.4.22 DDR Control Register 21 (DDR\_CR21)

Address: 0xFC0B_8054 (DDR_CR21)																Access: User read/write																				
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					
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	0	0	0	0	0

**Figure 21-23. DDR Control Register 21 (DDR\_CR21)**

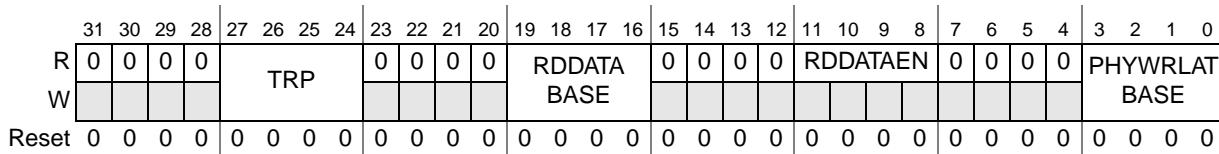
**Table 21-26. DDR\_CR21 Field Descriptions**

Field	Description
31–28	Reserved, must be cleared.
27–24 PHYWRLAT	Indicates the calculated DFI $t_{PHY\_WRLAT}$ timing parameter. This bit is read only. $PHYWRLAT = DDR\_CR22[WRLATBASE] + DDR\_CR57[WRLATADJ] + DDR\_CR09[REGDIMM] - WRLAT\_WIDTH'h3$
23–20	Reserved, must be cleared.
19–16 PHYRDLAT	Contains the $t_{PHY\_RDLAT}$ timing parameter.
15–12	Reserved, must be cleared.
11–8 CTRLUPDMIN	Contains the DFI $t_{CTRLUPD\_MIN}$ timing parameter. This bit is read only.
7–5	Reserved, must be cleared.
4–0 TDAL	Defines the auto-precharge write recovery time when auto-precharge is enabled (DDR_CR01[AP] is set), in cycles. This is defined internally as $t_{RP}$ (pre-charge time) + auto-precharge write recovery time. Not all memories use this parameter. If $t_{DAL}$ is defined in the memory specification, then program this parameter to the specified value. If the memory does not specify a $t_{DAL}$ time, then program this parameter to $t_{WR} + t_{RP}$ <b>Note:</b> Do not program this parameter with a value of 0x0. Else, the memory controller does not function properly when auto-precharge is enabled.

### 21.4.23 DDR Control Register 22 (DDR\_CR22)

Address: 0xFC0B\_8058 (DDR\_CR22)

Access: User read/write

**Figure 21-24. DDR Control Register 22 (DDR\_CR22)****Table 21-27. DDR\_CR22 Field Descriptions**

Field	Description
31–28	Reserved, must be cleared.
27–24 TRP	Defines the DRAM precharge command time (TRP) in cycles.
23–20	Reserved, must be cleared.
19–16 RDDATABSE	Sets the DFI base value for the $t_{RDDATA\_EN}$ timing parameter. Set this field to 2.
15–12	Reserved, must be cleared.

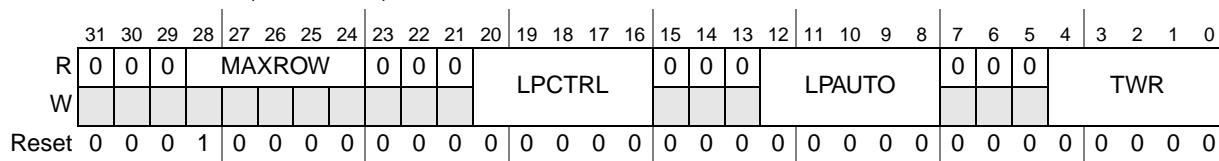
**Table 21-27. DDR\_CR22 Field Descriptions (continued)**

Field	Description
11–8 RDDATAEN	Contains the calculated DFI $t_{RDDATA\_EN}$ timing parameter. These bits are read-only. $RDDATAEN = RDDATABBASE + DDR\_CR57[RDLATADJ] + DDR\_CR09[REGDIMM] - RDLAT\_WIDTH'h3$
7–4	Reserved, must be cleared.
3–0 WRLATBASE	Sets the DFI base value for the $t_{PHY\_WRLAT}$ timing parameter. Set this field to 2.

### 21.4.24 DDR Control Register 23 (DDR\_CR23)

Address: 0xFC0B\_805C (DDR\_CR23)

Access: User read/write

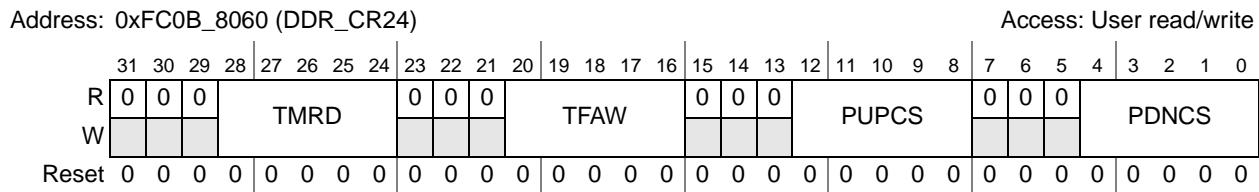
**Figure 21-25. DDR Control Register 23 (DDR\_CR23)****Table 21-28. DDR\_CR23 Field Descriptions**

Field	Description													
31–29	Reserved, must be cleared.													
28–24 MAXROW	Maximum width of the memory address bus. This byte is read-only and always reads 0x10. This value can be used to set the DDR_CR15[ADDPINS] field: $ADDPINS = MAXROW - \text{number of row bits in the memory device}$ . <b>Note:</b> The memory controller supports a 16-bit address bus. However, only 15 address lines are available externally on this device.													
23–21	Reserved, must be cleared.													
20–16 LPCTRL	Controls entry into the low power modes. <table border="1"> <tr> <th>LPCTRL bit</th> <th>Description</th> </tr> <tr> <td>4</td> <td>Memory power-down mode (mode 1)</td> </tr> <tr> <td>3</td> <td>Memory power-down with memory clock gating mode (mode 2)</td> </tr> <tr> <td>2</td> <td>Memory self-refresh mode (mode 3)</td> </tr> <tr> <td>1</td> <td>Memory self-refresh with memory clock gating mode (mode 4)</td> </tr> <tr> <td>0</td> <td>Memory self-refresh with memory and controller clock gating mode (mode 5)</td> </tr> </table> 0 Disable 1 Enable		LPCTRL bit	Description	4	Memory power-down mode (mode 1)	3	Memory power-down with memory clock gating mode (mode 2)	2	Memory self-refresh mode (mode 3)	1	Memory self-refresh with memory clock gating mode (mode 4)	0	Memory self-refresh with memory and controller clock gating mode (mode 5)
LPCTRL bit	Description													
4	Memory power-down mode (mode 1)													
3	Memory power-down with memory clock gating mode (mode 2)													
2	Memory self-refresh mode (mode 3)													
1	Memory self-refresh with memory clock gating mode (mode 4)													
0	Memory self-refresh with memory and controller clock gating mode (mode 5)													
15–13	Reserved, must be cleared.													

**Table 21-28. DDR\_CR23 Field Descriptions (continued)**

Field	Description													
12–8 LPAUTO	Enables automatic entry into the low power mode on idle.													
	<table border="1"> <thead> <tr> <th>LPAUTO bit</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>4</td> <td>Memory power-down mode (mode 1)</td> </tr> <tr> <td>3</td> <td>Memory power-down with memory clock gating mode (mode 2)</td> </tr> <tr> <td>2</td> <td>Memory self-refresh mode (mode 3)</td> </tr> <tr> <td>1</td> <td>Memory self-refresh with memory clock gating mode (mode 4)</td> </tr> <tr> <td>0</td> <td>Memory self-refresh with memory and controller clock gating mode (mode 5)</td> </tr> </tbody> </table>		LPAUTO bit	Description	4	Memory power-down mode (mode 1)	3	Memory power-down with memory clock gating mode (mode 2)	2	Memory self-refresh mode (mode 3)	1	Memory self-refresh with memory clock gating mode (mode 4)	0	Memory self-refresh with memory and controller clock gating mode (mode 5)
LPAUTO bit	Description													
4	Memory power-down mode (mode 1)													
3	Memory power-down with memory clock gating mode (mode 2)													
2	Memory self-refresh mode (mode 3)													
1	Memory self-refresh with memory clock gating mode (mode 4)													
0	Memory self-refresh with memory and controller clock gating mode (mode 5)													
	0 Automatic entry into this mode is disabled. You may enter the modes manually by setting the associated LPCTRL bit. 1 The controller/memory automatically enters this mode when the proper counters expire, and only if the associated LPCTRL bit is set.													
7–5	Reserved, must be cleared.													
4–0 TWR	Defines the DRAM write recovery time (TWR) parameter in cycles.													

### 21.4.25 DDR Control Register 24 (DDR\_CR24)

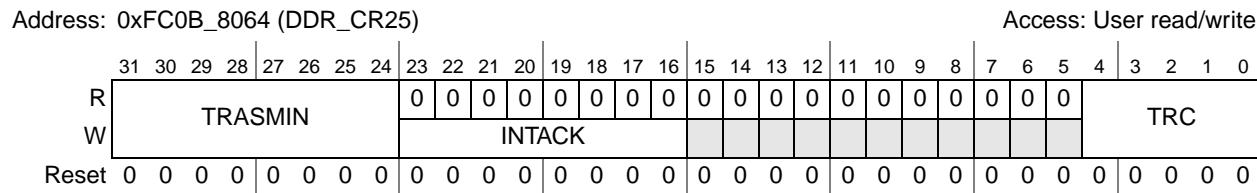
**Figure 21-26. DDR Control Register 24 (DDR\_CR24)****Table 21-29. DDR\_CR24 Field Descriptions**

Field	Description
31–29	Reserved, must be cleared.
28–24 TMRD	DRAM TMRD parameter in cycles.
23–21	Reserved, must be cleared.
20–16 TFAW	Defines the DRAM t <sub>F<sub>A</sub>W</sub> parameter in cycles.
15–13	Reserved, must be cleared.

**Table 21-29. DDR\_CR24 Field Descriptions (continued)**

Field	Description
12–8 PUPCS	Off-chip driver (OCD) pull-up adjustment setting for DRAMs for the chip select. The memory controller issues OCD adjustment commands during power-up. <b>Bit 4:</b> 0 Decrement OCD settings 1 Increment OCD settings <b>Bits 3–0:</b> Number of OCD adjustment commands to issue
7–5	Reserved, must be cleared.
4–0 PDNCS	Off-chip driver (OCD) pull-down adjustment setting for DRAMs for the chip select. The memory controller issues OCD adjustment commands during power-up. <b>Bit 4:</b> 0 Decrement OCD settings 1 Increment OCD settings <b>Bits 3–0:</b> Number of OCD adjustment commands to issue

## 21.4.26 DDR Control Register 25 (DDR\_CR25)

**Figure 21-27. DDR Control Register 25 (DDR\_CR25)****Table 21-30. DDR\_CR25 Field Descriptions**

Field	Description
31–24 TRASMIN	Defines the DRAM minimum row active time (TRAS_MIN) in cycles.
23–16 INTACK	Clear mask of the INTSTATUS parameter. This field is write-only. 0 No effect 1 Clear the corresponding bit in INTSTATUS
15–5	Reserved, must be cleared.
4–0 TRC	Defines the DRAM period between active commands for the same bank (TRC) in cycles.

### 21.4.27 DDR Control Register 26 (DDR\_CR26)

Address: 0xFC0B_8068 (DDR_CR26)																Access: User read/write											
R																INTMASK											
W																TRFC											
Reset																0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0											

Figure 21-28. DDR Control Register 26 (DDR\_CR26)

Table 21-31. DDR\_CR26 Field Descriptions

Field	Description
31–25	Reserved, must be cleared.
24–16 INTMASK	Mask for controller interrupt signals from the INTSTATUS parameter.
15–8 TRFC	Defines the DRAM refresh command time (TRFC) in cycles.
7–0 TRCD_INT	Defines the DRAM RAS to CAS delay in cycles.

### 21.4.28 DDR Control Register 27 (DDR\_CR27)

Address: 0xFC0B_806C (DDR_CR27)																Access: User read/write											
R																OORLEN											
W																0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0											
Reset																INTSTATUS											

Figure 21-29. DDR Control Register 27 (DDR\_CR27)

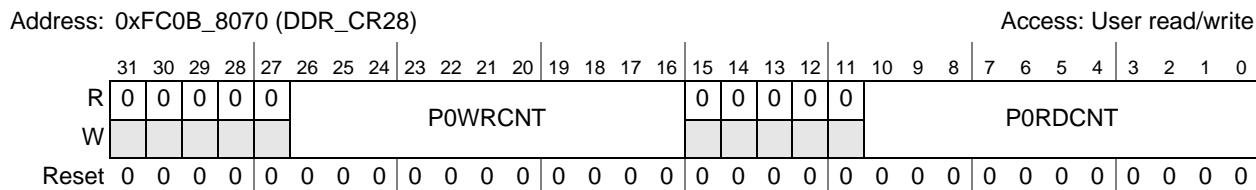
Table 21-32. DDR\_CR27 Field Descriptions

Field	Description
31–26	Reserved, must be cleared.
25–16 OORLEN	Length of command that caused an out-of-range interrupt request. This field is read-only.

**Table 21-32. DDR\_CR27 Field Descriptions (continued)**

Field	Description																												
15–9	Reserved, must be cleared.																												
8–0 INTSTATUS	Status of interrupt features in the controller. This field is read-only.																												
	<table border="1"> <thead> <tr> <th>INTSTATUS bit</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>8</td> <td>Logical OR of INTSTATUS[7:0]</td> </tr> <tr> <td>7</td> <td>Reserved</td> </tr> <tr> <td>6</td> <td>Reserved</td> </tr> <tr> <td>5</td> <td>Reserved</td> </tr> <tr> <td>4</td> <td>Address cross page boundary detected</td> </tr> <tr> <td>3</td> <td>DRAM initialization complete</td> </tr> <tr> <td>2</td> <td>Error was found with command channel in a port</td> </tr> <tr> <td>1</td> <td>Multiple accesses outside the defined physical memory space detected</td> </tr> <tr> <td>0</td> <td>A single access outside the defined physical memory space detected</td> </tr> </tbody> </table>									INTSTATUS bit	Description	8	Logical OR of INTSTATUS[7:0]	7	Reserved	6	Reserved	5	Reserved	4	Address cross page boundary detected	3	DRAM initialization complete	2	Error was found with command channel in a port	1	Multiple accesses outside the defined physical memory space detected	0	A single access outside the defined physical memory space detected
INTSTATUS bit	Description																												
8	Logical OR of INTSTATUS[7:0]																												
7	Reserved																												
6	Reserved																												
5	Reserved																												
4	Address cross page boundary detected																												
3	DRAM initialization complete																												
2	Error was found with command channel in a port																												
1	Multiple accesses outside the defined physical memory space detected																												
0	A single access outside the defined physical memory space detected																												

### 21.4.29 DDR Control Register 28 (DDR\_CR28)

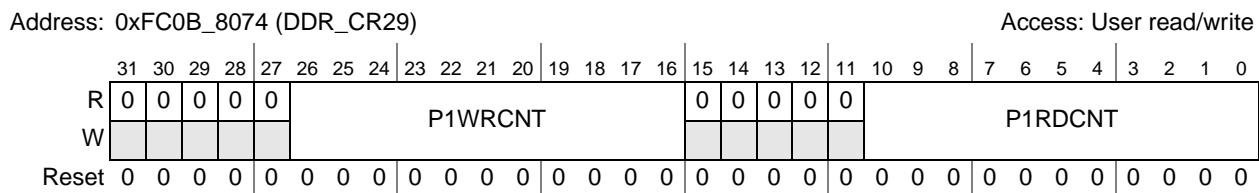
**Figure 21-30. DDR Control Register 28 (DDR\_CR28)****Table 21-33. DDR\_CR28 Field Descriptions**

Field	Description
31–27	Reserved, must be cleared.
26–16 P0WRCNT	Number of bytes for INCR write command on port 0. The logic subdivides an INCR request into memory controller core commands of the size of this parameter. The logic continues sending bursts of this size as the previous request is transmitted by the crossbar port. If the INCR command is terminated on an unnatural boundary, the logic discards the unnecessary words. The value defined in this parameter must be a multiple of four. Clearing this parameter causes the port to issue commands of zero length to the memory controller core, which the core interprets as the pre-configured value of 128 bytes.

**Table 21-33. DDR\_CR28 Field Descriptions (continued)**

Field	Description
15–11	Reserved, must be cleared.
10–0 P0RDCNT	Number of bytes for INCR read command on port 0. The logic subdivides an INCR request into memory controller core commands of the size of this parameter. The logic continues requesting bursts of this size as soon as the previous request is received by the crossbar port. If the INCR command is terminated on an unnatural boundary, the logic discards the unnecessary words. The value defined in this parameter must be a multiple of four. Clearing this parameter causes the port to issue commands of zero length to the memory controller core, which the core interprets as the pre-configured value of 128 bytes.

### 21.4.30 DDR Control Register 29 (DDR\_CR29)

**Figure 21-31. DDR Control Register 29 (DDR\_CR29)****Table 21-34. DDR\_CR29 Field Descriptions**

Field	Description
31–27	Reserved, must be cleared.
26–16 P1WRCNT	Number of bytes for INCR write command on port 1. The logic subdivides an INCR request into memory controller core commands of the size of this parameter. The logic continues sending bursts of this size as the previous request is transmitted by the crossbar port. If the INCR command is terminated on an unnatural boundary, the logic discards the unnecessary words. The value defined in this parameter must be a multiple of four. Clearing this parameter causes the port to issue commands of zero length to the memory controller core, which the core interprets as the pre-configured value of 128 bytes.
15–11	Reserved, must be cleared.
10–0 P1RDCNT	Number of bytes for INCR read command on port 1. The logic subdivides an INCR request into memory controller core commands of the size of this parameter. The logic continues requesting bursts of this size as soon as the previous request is received by the crossbar port. If the INCR command is terminated on an unnatural boundary, the logic discards the unnecessary words. The value defined in this parameter must be a multiple of four. Clearing this parameter causes the port to issue commands of zero length to the Databahn MC core, which the core interprets as the pre-configured value of 128 bytes.

### 21.4.31 DDR Control Register 30 (DDR\_CR30)

Address: 0xFC0B_8078 (DDR_CR30)																Access: User read-only															
R		PHYUPDRESP														W		CTRLUPDMAX													
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	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 21-32. DDR Control Register 30 (DDR\_CR30)

Table 21-35. DDR\_CR30 Field Descriptions

Field	Description
31–30	Reserved, must be cleared.
29–16 PHYUPDRESP	Contains the DFI $t_{PHYUPD\_RESP}$ timing parameter. This field is read-only.
15–14	Reserved, must be cleared.
13–0 CTRLUPDMAX	Contains the DFI $t_{CTRLUPD\_MAX}$ timing parameter. This field is read-only.

### 21.4.32 DDR Control Register 31 (DDR\_CR31)

Address: 0xFC0B_807C (DDR_CR31)																Access: User read/write															
R		TREF														W		PHYUPD_TYPE0													
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	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 21-33. DDR Control Register 31 (DDR\_CR31)

Table 21-36. DDR\_CR31 Field Descriptions

Field	Description
31–30	Reserved, must be cleared.
29–16 TREF	Defines the DRAM cycles between refresh commands (TREF) in cycles.
15–14	Reserved, must be cleared.
13–0 PHYUPD_TYPE0	Contains the DFI $t_{PHYUPD\_TYPE0}$ timing parameter. This field is read-only.

### 21.4.33 DDR Control Register 32 (DDR\_CR32)

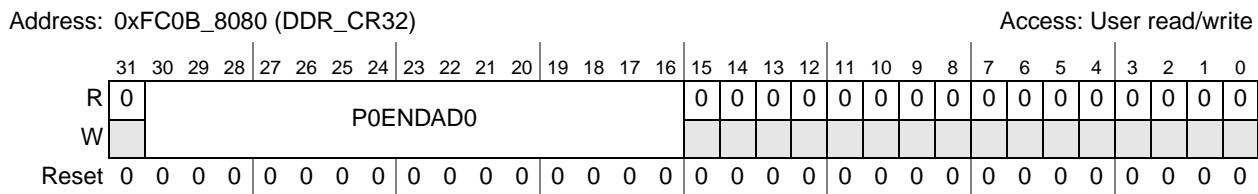


Figure 21-34. DDR Control Register 32 (DDR\_CR32)

Table 21-37. DDR\_CR32 Field Descriptions

Field	Description
31	Reserved, must be cleared.
30–16 P0ENDAD0	End address of port 0 address range 0. This is only used when the DDR_CR07[PROTEN] is set to verify that incoming addresses are of a valid type and range. The granularity of this parameter is in kilobytes. For example, if P0STAD1 = 0x00_0000, P0ENDAD1 = 0x00_0002, and P0ATT1 = 0x1, then port 0 has read-only access to the first 2KB of the memory.
15–0	Reserved, must be cleared.

### 21.4.34 DDR Control Register 33 (DDR\_CR33)

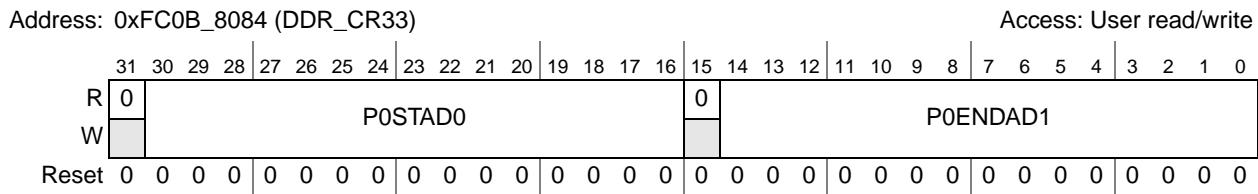


Figure 21-35. DDR Control Register 33 (DDR\_CR33)

Table 21-38. DDR\_CR33 Field Descriptions

Field	Description
31	Reserved, must be cleared.
30–16 P0STAD0	Start address of port 0 address range 0. This is only used when the DDR_CR07[PROTEN] is set to verify that incoming addresses are of a valid type and range. The granularity of this parameter is in kilobytes. For example, if P0STAD1 = 0x00_0000, P0ENDAD1 = 0x00_0002, and P0ATT1 = 0x1, then port 0 has read-only access to the first 2KB of the memory.
15	Reserved, must be cleared.
14–0 P0ENDAD1	End address of port 0 address range 1. This is only used when the DDR_CR07[PROTEN] is set to verify that incoming addresses are of a valid type and range. The granularity of this parameter is in kilobytes. For example, if P0STAD1 = 0x00_0000, P0ENDAD1 = 0x00_0002, and P0ATT1 = 0x1, then port 0 has read-only access to the first 2KB of the memory.

### 21.4.35 DDR Control Register 34 (DDR\_CR34)

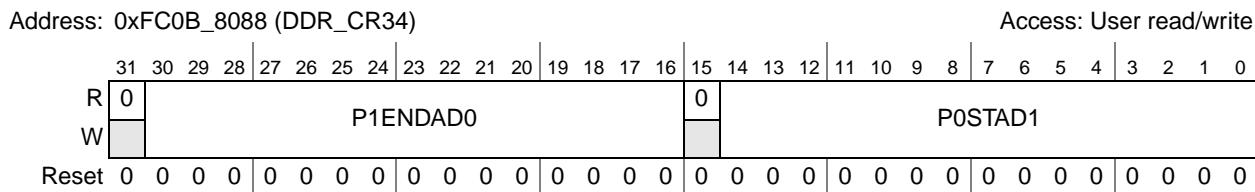


Figure 21-36. DDR Control Register 34 (DDR\_CR34)

Table 21-39. DDR\_CR34 Field Descriptions

Field	Description
31	Reserved, must be cleared.
30–16 P1ENDAD0	End address of port 1 address range 0. This is only used when the DDR_CR07[PROTEN] is set to verify that incoming addresses are of a valid type and range. The granularity of this parameter is in kilobytes. For example, if P0STAD1 = 0x00_0000, P0ENDAD1 = 0x00_0002, and P0ATT1 = 0x1, then port 0 has read-only access to the first 2KB of the memory.
15	Reserved, must be cleared.
14–0 P0STAD1	Start address of port 0 address range 1. This is only used when the DDR_CR07[PROTEN] is set to verify that incoming addresses are of a valid type and range. The granularity of this parameter is in kilobytes. For example, if P0STAD1 = 0x00_0000, P0ENDAD1 = 0x00_0002, and P0ATT1 = 0x1, then port 0 has read-only access to the first 2KB of the memory.

### 21.4.36 DDR Control Register 35 (DDR\_CR35)

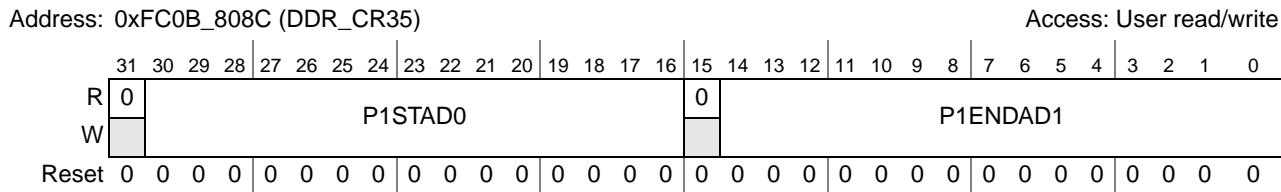


Figure 21-37. DDR Control Register 35 (DDR\_CR35)

Table 21-40. DDR\_CR35 Field Descriptions

Field	Description
31	Reserved, must be cleared.h
30–16 P1STAD0	Start address of port 1 address range 0. This is only used when the DDR_CR07[PROTEN] is set to verify that incoming addresses are of a valid type and range. The granularity of this parameter is in kilobytes. For example, if P0STAD1 = 0x00_0000, P0ENDAD1 = 0x00_0002, and P0ATT1 = 0x1, then port 0 has read-only access to the first 2KB of the memory.
15	Reserved, must be cleared.
14–0 P1ENDAD1	End address of port 1 address range 1. This is only used when the DDR_CR07[PROTEN] is set to verify that incoming addresses are of a valid type and range. The granularity of this parameter is in kilobytes. For example, if P0STAD1 = 0x00_0000, P0ENDAD1 = 0x00_0002, and P0ATT1 = 0x1, then port 0 has read-only access to the first 2KB of the memory.

#### 21.4.37 DDR Control Register 36 (DDR\_CR36)

Address: 0xFC0B\_8090 (DDR\_CR36)

Access: User read/write

**Figure 21-38. DDR Control Register 36 (DDR\_CR36)**

**Table 21-41. DDR\_CR36 Field Descriptions**

Field	Description
31–15	Reserved, must be cleared.
14–0 P1STAD1	Start address of port 1 address range 1. This is only used when the DDR_CR07[PROTEN] is set to verify that incoming addresses are of a valid type and range. The granularity of this parameter is in kilobytes. For example, if P0STAD1 = 0x00_0000, P0ENDAD1 = 0x00_0002, and P0ATT1 = 0x1, then port 0 has read-only access to the first 2KB of the memory.

#### 21.4.38 DDR Control Register 37 (DDR\_CR37)

Address: 0xFC0B\_8094 (DDR\_CR37)

Access: User read/write

**Figure 21-39. DDR Control Register 37 (DDR\_CR37)**

**Table 21-42. DDR CR37 Field Descriptions**

Field	Description
31–16	Reserved, must be cleared.
15–0 EMRS2D	Data to program into memory mode register 2 for the chip select. The contents of this parameter are programmed into the DRAM at initialization or when the DDR_CR11[WRMD] is set. Consult the DRAM specification for the correct settings for this field.

#### 21.4.39 DDR Control Register 38 (DDR\_CR38)

Address: 0xFC0B\_8098 (DDR\_CR38)

Access: User read/write

**Figure 21-40. DDR Control Register 38 (DDR\_CR38)**

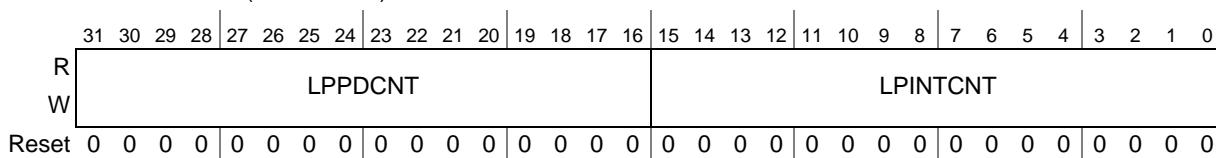
**Table 21-43. DDR\_CR38 Field Descriptions**

Field	Description
31–16 LPEXTCNT	Counts the number of idle cycles before memory self-refresh in memory clock gating low power mode.
15–0	Reserved, must be cleared.

#### 21.4.40 DDR Control Register 39 (DDR\_CR39)

Address: 0xFC0B\_809C (DDR\_CR39)

## Access: User read/write



**Figure 21-41. DDR Control Register 39 (DDR\_CR39)**

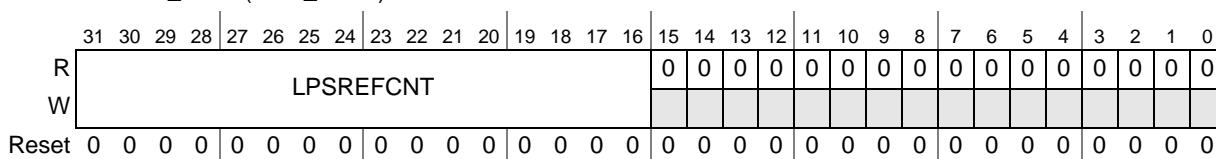
**Table 21-44. DDR\_CR39 Field Descriptions**

Field	Description
31–16 LPPDCNT	Counts the number of idle cycles before memory power-down or power-down in memory clock gating low power mode.
15–0 LPINTCNT	Counts the number of idle cycles before memory self-refresh in memory and controller clock gating low power mode.

#### 21.4.41 DDR Control Register 40 (DDR CR40)

Address: 0xFC0B\_80A0 (DDR CR40)

Access: User read/write



**Figure 21-42. DDR Control Register 40 (DDR CR40)**

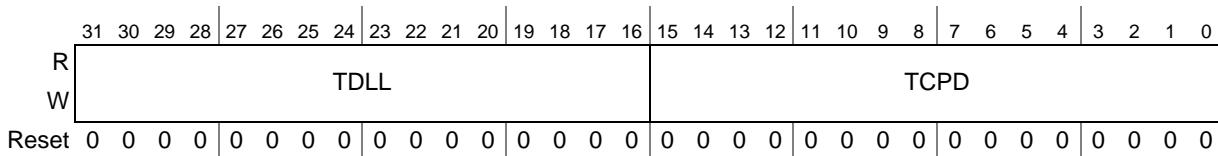
**Table 21-45. DDR CR40 Field Descriptions**

Field	Description
31–16 LPSREFCNT	Counts the number of cycles to the next memory self-refresh low power mode.
15–0	Reserved, must be cleared.

#### 21.4.42 DDR Control Register 41 (DDR\_CR41)

Address: 0xFC0B\_80A4 (DDR\_CR41)

Access: User read/write



**Figure 21-43. DDR Control Register 41 (DDR\_CR41)**

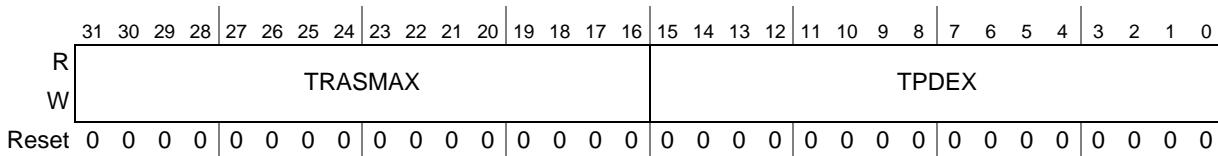
**Table 21-46. DDR\_CR41 Field Descriptions**

Field	Description
31–16 TDLL	DLL lock time in cycles. <b>Note:</b> This field must be cleared.
15–0 TCPD	Defines the DRAM TCPD (clock enable to precharge delay time) parameter in cycles.

#### 21.4.43 DDR Control Register 42 (DDR\_CR42)

Address: 0xFC0B\_80A8 (DDR\_CR42)

Access: User read/write



**Figure 21-44. DDR Control Register 42 (DDR\_CR42)**

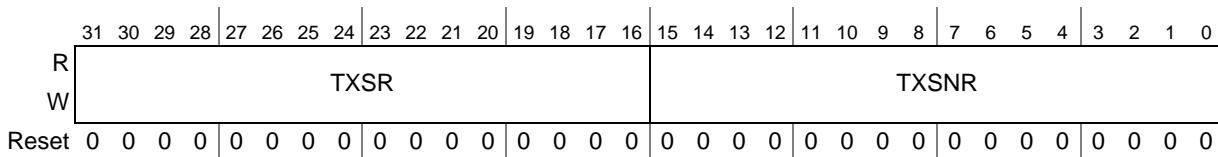
**Table 21-47. DDR\_CR42 Field Descriptions**

Field	Description
31–16 TRASMAX	Defines the DRAM maximum row active time (TRAS_MAX) in cycles.
15–0 TPDEX	Defines the DRAM power-down exit command period in cycles.

#### 21.4.44 DDR Control Register 43 (DDR\_CR43)

Address: 0xFC0B\_80AC (DDR\_CR43)

## Access: User read/write



**Figure 21-45. DDR Control Register 43 (DDR\_CR43)**

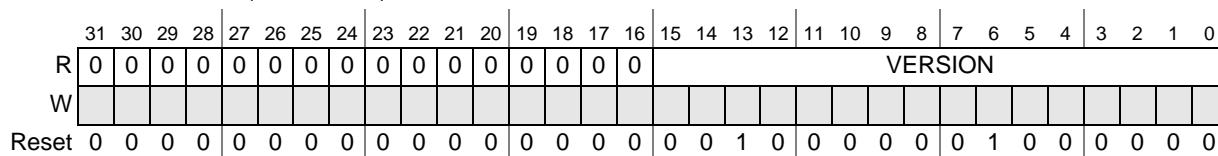
**Table 21-48. DDR\_CR43 Field Descriptions**

Field	Description
31–16 TXSR	Defines the DRAM self-refresh exit time (TXSR) in cycles.
15–0 Txsnr	Defines the DRAM TXSNR parameter in cycles.

#### 21.4.45 DDR Control Register 44 (DDR\_CR44)

Address: 0xFC0B\_80B0 (DDR CR44)

Access: User read/write



**Figure 21-46. DDR Control Register 44 (DDR\_CR44)**

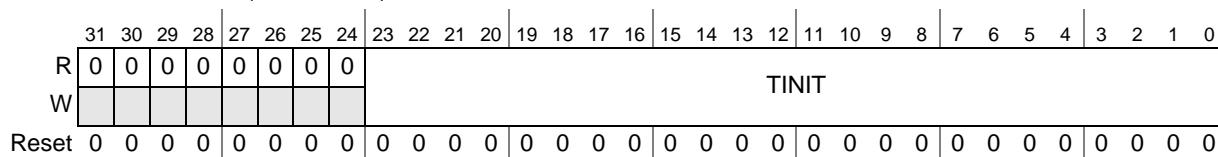
**Table 21-49. DDR\_CR44 Field Descriptions**

Field	Description
31–16	Reserved, must be cleared.
15–0 VERSION	Controller version number. This field is read only and always reads 0x2040.

#### **21.4.46 DDR Control Register 45 (DDR\_CR45)**

Address: 0xFC0B\_80B4 (DDR\_CR45)

Access: User read/write



**Figure 21-47. DDR Control Register 45 (DDR CR45)**

**Table 21-50. DDR CR45 Field Descriptions**

Field	Description
31–24	Reserved, must be cleared.
23–0 TINIT	Defines the DRAM initialization time in cycles.

#### 21.4.47 DDR Control Register 46 (DDR\_CR46)

This register location is reserved.

#### 21.4.48 DDR Control Register 47 (DDR\_CR47)

This register location is reserved.

#### 21.4.49 DDR Control Register 48 (DDR\_CR48)

This register location is reserved.

#### 21.4.50 DDR Control Register 49 (DDR\_CR49)

This register location is reserved.

#### 21.4.51 DDR Control Register 50 (DDR\_CR50)

This register location is reserved.

#### 21.4.52 DDR Control Register 51 (DDR\_CR51)

This register location is reserved.

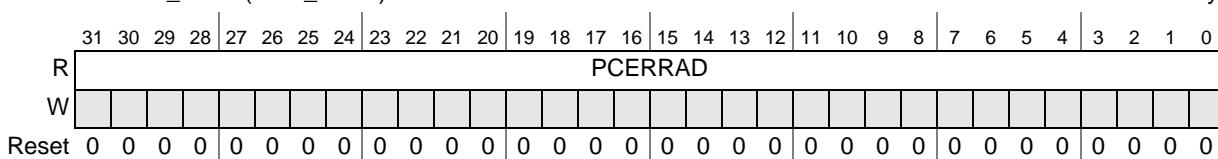
#### 21.4.53 DDR Control Register 52 (DDR\_CR52)

This register location is reserved.

#### 21.4.54 DDR Control Register 53 (DDR\_CR53)

Address: 0xFC0B\_80D4 (DDR\_CR53)

Access: User read-only



**Figure 21-48. DDR Control Register 53 (DDR CR53)**

**Table 21-51. DDR CR53 Field Descriptions**

Field	Description
31–0 PCERRAD	Address of the command that caused a port command error condition. This field is read-only.

#### 21.4.55 DDR Control Register 54 (DDR\_CR54)

This register location is reserved.

### 21.4.56 DDR Control Register 55 (DDR\_CR55)

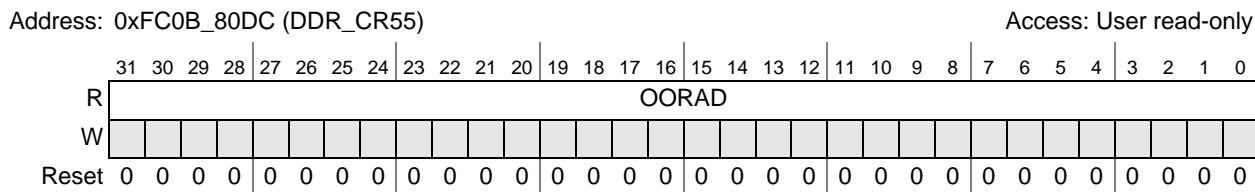


Figure 21-49. DDR Control Register 55 (DDR\_CR55)

Table 21-52. DDR\_CR55 Field Descriptions

Field	Description
31–0 OORAD	Address of the command that caused an out-of-range interrupt. This field is read-only. <b>Note:</b> See DDR_CR56[OOR32] for bit 32 of this address.

### 21.4.57 DDR Control Register 56 (DDR\_CR56)

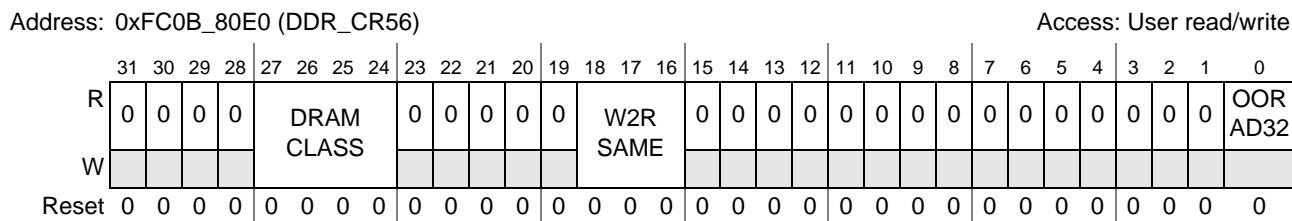


Figure 21-50. DDR Control Register 56 (DDR\_CR56)

Table 21-53. DDR\_CR56 Field Descriptions

Field	Description
31–28	Reserved, must be cleared.
27–24 DRAMCLASS	Defines the mode of operation of the controller. 0000 DDR1 0001 Reserved 0100 DDR2 Else Reserved
23–19	Reserved, must be cleared.
18–16 W2RSAME	Additional clocks of delay to insert between writes and reads to the same chip select.
15–1	Reserved, must be cleared.
0 OORAD32	Bit 32 of the address of command that caused an out-of-range interrupt. This bit is read-only. <b>Note:</b> See DDR_CR56[OOR32] for bits 31–0 of this address.

## 21.4.58 DDR Control Register 57 (DDR\_CR57)

Address: 0xFC0B_80E4 (DDR_CR57)																Access: User read/write																	
R	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	
W	TMOD								0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	WRLAT	0	0	0	0	RDLAT			
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	0

Figure 21-51. DDR Control Register 57 (DDR\_CR57)

Table 21-54. DDR\_CR57 Field Descriptions

Field	Description
31–24 TMOD	Defines the number of cycles of wait time between mode commands. For write leveling, this is defined as the number of cycles of wait time after a MRS command to the ODT enable.
23–12	Reserved, must be cleared.
11–8 WRLATADJ	Adjusts the relative timing between DFI write commands and the <i>dfi_wrdata_en</i> signal to conform to PHY timing requirements. When this field is cleared, <i>dfi_wrdata_en</i> asserts on the same cycle as the <i>dfi_address</i> . Set WRLATADJ = DDR_CR19[WRLAT].
7–4	Reserved, must be cleared.
3–0 RDLATADJ	Adjusts the relative timing between DFI read commands and the <i>dfi_rddata_en</i> signal to conform to PHY timing requirements. When this field is cleared, <i>dfi_rddata_en</i> asserts one cycle after the <i>dfi_address</i> . Set RDLATADJ = DDR_CR16[CASLAT].

## 21.4.59 DDR Control Register 58 (DDR\_CR58)

Address: 0xFC0B_80E8 (DDR_CR58)																Access: User read/write																		
R	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		
W	EMRS1D								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 21-52. DDR Control Register 58 (DDR\_CR58)

Table 21-55. DDR\_CR58 Field Descriptions

Field	Description
31–16 EMRS1D	Data to program into memory mode register 1 for the chip select. The contents of this parameter are programmed into the DRAM at initialization or if DDR_CR11[WRMD] is set. Consult the DRAM specification for the correct settings for this parameter. <b>Note:</b> The memory controller does not support additive latency. Therefore, the additive latency bits must be cleared. For DDR2 memories, these are bits 5:3 of this field or A[5:3]. <b>Note:</b> The same values must be programmed into EMRS1D[6,2] (A6, A2) and ODTRES.
15–0	Reserved, must be cleared.

## 21.4.60 DDR Control Register 59 (DDR\_CR59)

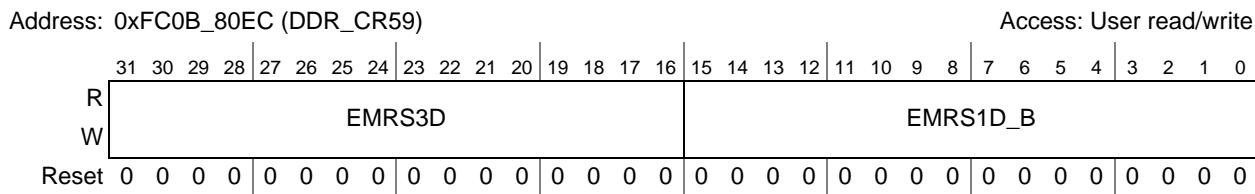


Figure 21-53. DDR Control Register 59 (DDR\_CR59)

Table 21-56. DDR\_CR59 Field Descriptions

Field	Description
31–16 EMRS3D	Data to program into memory mode register 3 for the chip select. The contents of this parameter are programmed into the DRAM at initialization or when the DDR_CR11[WRMD] is set. Consult the DRAM specification for the correct settings for this field.
15–0 EMRS1D_B	<b>Note:</b> This value must match DDR_CR58[EMRS1D].

## 21.4.61 DDR Control Register 60 (DDR\_CR60)

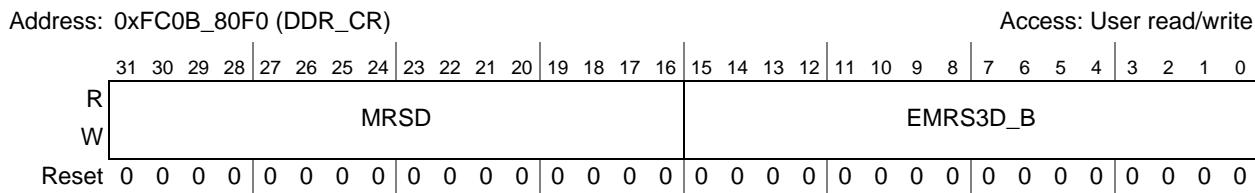


Figure 21-54. DDR Control Register 60 (DDR\_CR60)

Table 21-57. DDR\_CR60 Field Descriptions

Field	Description
31–16 MRSD	MRS data to program into memory mode register 0 for the chip select. The memory controller ignores the programmed value of the DLL reset bit in this field. An internal state machine controls this bit and only sets it during initialization. <b>Note:</b> The memory controller does not support interleaved bursts or burst lengths other than 4. Therefore, you must set MRSD[3:0] to 0010 (A3 = 0 and A[2:0] = 010). The contents of this field, except the DLL reset bit, is programmed into the DRAM at initialization or if DDR_CR11[WRMD] is set. Consult the DRAM specification for the correct settings for this field.
15–0 EMRS3D_B	<b>Note:</b> This value must match DDR_CR59[EMRS3D].

## 21.4.62 DDR Control Register 61 (DDR\_CR61)

This register location is reserved.

### 21.4.63 DDR Control Register 62 (DDR\_CR62)

This register location is reserved.

### 21.4.64 DDR Control Register 63 (DDR\_CR63)

This register location is reserved.

### 21.4.65 RCR Control Register (DDR\_RCR)

The DDR\_RCR register controls the operation of the read clock recovery module.

Address: 0xFC0B_8180 (DDR_RCR)																																
R	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
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		

Figure 21-55. RCR Control Register (DDR\_RCR)

Table 21-58. DDR\_RCR Field Descriptions

Field	Description
31	Reserved, should be cleared.
30 RST	Read clock recovery module reset. Used to force RCR software reset in addition to system reset. 0 No software reset. 1 Force software reset  <b>Note:</b> This bit is write-only; always return 0 on a read.
29–0	Reserved, should be cleared.

### 21.4.66 DDR I/O Pad Control Register (DDR\_PADCR)

This register stores the various control bits for the DQS pad and the pad control logic. The OFFSET\_VALUE is set differently for each type of supported SDRAM.

Offset 0xFC0B_81AC (DDR_PADCR)																Access: User read/write								
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	OFFSET_VALUE							
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	
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	SPARE_DLY0							
W																								
Reset	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-56. I/O Pad Control Register (DDR\_PADCR)

**Table 21-59. DDR\_PADCR Field Descriptions**

Field	Description
31–26	Reserved, must be cleared.
25–24 PAD_ODT_CS	On-die termination of the pads when the read command is issued to the chip-select. 00 ODT disabled 01 75 Ω 10 150 Ω 11 50 Ω
23–18	Reserved, must be cleared.
17–16 OFFSET_VALUE	Adjusts the voltage value of the hysteresis differential amplifier in the DQS pad. 00 400 mV (DDR2) / 200 mV (DDR1) 01 500 mV (DDR2) / 250 mV (DDR1) 10 600 mV (DDR2) / 300 mV (DDR1) 11 700 mV (DDR2) / 350 mV (DDR1)
15–10	Reserved, must be cleared.
9	Reserved, must be set.
8–2	Reserved, must be cleared.
1–0 SPARE_DLY0	<b>Note:</b> This bitfield must be set to 11.

## 21.5 Functional Description

### 21.5.1 High-Level Memory Controller Blocks

The DDR1/2 memory controller can be broken down into two main functional sub-blocks as described below.

#### 21.5.1.1 DDR1/2 Memory Controller Core

The controller core provides two crossbar switch ports to improve system performance by enabling multiple access requests to be presented to the controller at the same time. This allows the controller to pipeline many of the operations (for example, bank activate and precharge), and to reduce the average system access latency and improve utilization of the external memory. The transaction order can be rearranged to maximize the number of in-page accesses.

The controller also provides a programmable register interface to control memory device parameters and protocols including auto pre-charge.

#### 21.5.1.2 DFI-Compliant PHY

The DDR1/2 memory controller interfaces to the external SDRAM bus via a PHY. This PHY is compatible with the DDR PHY interface (DFI) specification. This block transfers control information and read and write data to and from the DRAM devices. The PHY contains three major sub-blocks:

- Logic to convey the control and address signals

- Logic to buffer and manipulate write data
- Logic to delay input DQS to capture read data and to forward read data to the controller

The read data path logic uses the read clock recovery (RCR) hard macro built within the PHY.

## 21.5.2 Address Mapping

The memory controller automatically maps user addresses to the DRAM memory in a contiguous block. Addressing starts at user address 0 and ends at the highest available address according to the size and number of DRAM devices present. This mapping is dependent on how the memory controller is configured.

The mapping of the address space to the internal data storage structure of the DRAM devices is based on the actual size of the DRAM devices available. The size is stored in user-programmable parameters that must be initialized at power up. Certain DRAM devices allow for different mapping options to be chosen, while other DRAM devices depend on the chosen burst length.

### 21.5.2.1 DDR SDRAM Address Mapping Options

The address structure of DDR SDRAM devices contains the following five fields:

- Chip select
- Row
- Bank
- Column
- Datapath

Each of these fields can be individually addressed when accessing the DRAM.

The maximum widths of the fields are based on the configuration settings. The actual widths of the fields may be smaller if the device address width parameters (DDR\_CR15[ADDPINS], DDR\_CR04[8BNK], and DDR\_CR16[COLSIZ]) are programmed differently.

### 21.5.2.2 Maximum Address Space

The maximum user address range is determined by the width of the memory datapath, the number of chip select pins, and the address space of the DRAM device. The maximum amount of memory can be calculated by the following formula:

$$\text{Maximum Memory Size} = \text{Chip Selects} \times 2^{\text{Address}} \times \text{Banks} \times \text{DatapathWidth}$$

**Eqn. 21-1**

For this memory controller, the maximum values for these fields are as follows:

- Chip selects = 1
- Device address = 15 rows + 12 columns = 27
- Number of banks = 8
- Memory datapath width = 1 byte

As a result, the maximum accessible memory area is 1 GB.

### 21.5.2.3 Memory Mapping to Address Space

The maximum allowable address space and mapping into the DRAM devices for the memory controller is shown in [Figure 21-57](#). This map corresponds to a memory device with 15 row bits and 12 column bits.

32	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
Don't care	Chip select	Row										Bank	Column										Data path									

**Figure 21-57. Memory Controller Memory Map: Maximum**

The only valid Datapath setting is zero. The DDR\_CR15[ADDPINS] and DDR\_CR16[COLSIZ] fields can each range from the maximum configured for the memory controller to seven bits smaller than the maximum configured. This allows the memory controller to function with a wide variety of memory device sizes.

The settings for the ADDPINS and COLSIZ parameters control how the address map decodes the user address to the DRAM chip selects and row and column addresses. The DDR\_CR04[8BNK] parameter controls the address in DDR2 mode. It is assumed that the values in these parameters never exceed the maximum values configured.

Using the example shown in [Figure 21-57](#), if the memory controller is connected to devices with 13 row and 10 column bits, the maximum accessible memory space is reduced. The accessible memory space for this configuration is 256 MB.

The address map for this configuration is shown in [Figure 21-58](#). Address bits 28–32 are not used. These bits are ignored when generating the address to the DRAM devices.

32	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
Don't care	Chip select	Row										Bank	Column										Data path									

**Figure 21-58. Alternate Memory Map**

#### NOTE

The chip select, row, bank, and column fields address an entire memory word (a byte in this case).

### 21.5.3 Write Data Queue

The write data queue is a write data storage array for transactions. The queue consists of multiple buffers holding write data for the write requests of a particular port. Write data is stored in these buffers for commands in the command queue until the command is processed in the placement logic and needed by the DRAM command arbitration logic.

The buffers can accept data whenever any space is available. The buffers are defined to hold 8 entries. The size of the write data buffers and the burst length programmed into the memory devices affect the overall performance of a single port during the write operation. Each buffer must have a depth of at least twice the number of data words for a memory burst to ensure that the memory controller can continuously burst write data for a port. The buffer should also be large enough to hold enough words to consume data for a single write transaction related to the bus. The data may actually be written into the buffers from the bus

at a later time, depending on the priority of the request and the number of transactions in the command queue.

## 21.5.4 DRAM Command Processing

The DRAM command processing logic processes the commands in the command queue. The logic organizes the commands to the memory devices so that data throughput is maximized. Bank opening and closing cycles are used for data transfers.

The logic uses a variety of factors to determine when to issue bank open and close commands. The logic reviews the entire command queue for a look-ahead of which banks are to be accessed in the future. The timing is then set to meet the TRC and TRAS\_MIN timing parameters of the memory devices, values which were programmed into the memory controller on initialization. This flexibility allows the memory controller to be tuned to extract the maximum performance out of memory devices. The parameters that relate to DRAM device protocol are listed in the [Section 21.4, “Memory Map/Register Definition”](#).

## 21.5.5 Latency

By using the placement logic of the command queue in the memory controller, a new request through any port can be immediately placed at the top of the command queue or can interrupt an ongoing request. This scheme allows a high priority request to be serviced in the shortest possible time. However, since there are many factors that determine the placement into the command queue, there are also many factors that affect the actual latency of the command.

These factors include information about transactions already in the command queue:

- Coherency status — If there is a data coherency conflict with a transaction already in the command queue, the new transaction is placed after the transaction that produced the conflict. The position of the conflicting transaction determines the latency of the high priority read or write command.
- Priority status — If the new command has a higher priority than those already in the command queue, the new request is serviced ahead of the lower priority command. The latency of the new command is lower than the latency of the older command.
- Read, write, and bank information — In general, reads are placed ahead of writes when both are of the same priority level. Read commands are grouped with other read commands of similar priorities, and write commands are grouped with other write commands of similar priorities. Among these groupings, transactions with similar bank and different row destinations are separated as much as possible.

If all of the placement conditions are met, then a new command is placed at the top of the command queue. However, if the new command is of a higher priority than the transaction executing, the current command is interrupted and the new command executes first. The interruption occurs at a natural burst boundary of the DRAMs. The interrupted transaction is placed at the top of the placement queue and resumes after the new request is completed. The page status of the new transaction determines when the current transaction is interrupted:

- If the page for the new transaction is already open, then the current transaction is interrupted at the next natural burst boundary of the DRAM device.

- If the page is not currently open, then the new request is placed at the top of the command queue while its page is prepared.

There are a fixed number of latency cycles in the memory controller, based on the pipeline through the memory controller logic. These steps are:

1. Command passing through the port interface. (fixed)
2. Arbitration through the arbiter. (fixed)
3. Placement into the command queue. (fixed)
4. Memory command generation. (variable)
5. Sending of control signals from the core logic. (fixed)
6. Flight time to the DRAM device. (variable)
7. Flight time from the DRAM device. (variable)
8. For reads, synchronization of read data from the data strobe domain. (fixed)
9. For reads, data pass through the port interface. (fixed)

## 21.5.6 Core Command Queue with Placement Logic

The memory controller core contains a command queue that accepts commands from the arbiter. This command queue uses a placement algorithm to determine the order that commands execute in the core. The placement logic follows many rules to determine where new commands are inserted into the queue, relative to the contents of the command queue at the time. Placement is determined by considering address collisions, source collisions, data collisions, command types and priorities. In addition, the placement logic attempts to maximize efficiency of the memory controller core through command grouping and bank splitting. After being placed into the command queue, the relative order of commands is constant.

Many of the rules used in placement may be individually enabled/disabled. In addition, the queue may be programmed through DDR\_CR07[PLEN] to disable the placement logic entirely, resulting in an in-line queue that services requests in the order they are received. If DDR\_CR07[PLEN] is cleared, the placement algorithm is ignored.

### 21.5.6.1 Rules of the Placement Algorithm

The factors affecting command placement work together to identify where a new command fits into the execution order. They are listed in order of importance.

#### 21.5.6.1.1 Address Collision/Data Coherency Violation

The order that read and write commands are processed in the memory controller is critical to proper system behavior. While reads and writes to different addresses are independent and may be re-ordered without affecting system performance, reads and writes that access the same address are significantly related. If the port requests a read after a write to the same address, then repositioning the read before the write returns the original data, not the changed data. Similarly, if the read was requested ahead of the write but accidentally positioned after the write, then the read returns the new data, not the original data prior to being overwritten. These are significant data coherency mistakes.

To avoid address collisions, reads or writes that access the same chip select, bank, and row as a command already in the command queue are inserted into the command queue after the original command, even if the new command is a higher priority.

This factor may be enabled/disabled through DDR\_CR00[ADDCOL] and should only be disabled if the system can guarantee coherency of reads and writes.

#### **21.5.6.1.2 Source ID Collision**

Each port is assigned a specific source ID that identifies the source uniquely. This allows the memory controller to map data from/to the correct source/destination.

##### **NOTE**

A source ID does contain port identification information, which means that the rules for placement are dependent on the requesting port. There are not source ID collisions between ports.

In general, commands of the same type from the same source ID are placed in the command queue in order. Therefore, a read/write command with the same source ID as a read/write command already in the command queue is processed after the original read/write command.

The behavior of commands of different types from the same source ID is dependent on the user configuration. For this memory controller, the placement of new read/write commands that collide in terms of source ID with existing entries in the command queue only depends on other commands of the same type, not on different types. This means that, if there are no address conflicts, a read command can be executed ahead of a write command with the same source ID. Likewise a write command can be executed ahead of a read command with the same source ID.

This feature is always enabled.

#### **21.5.6.1.3 Write Buffer Collision**

Incoming write requests in the command queue are allocated to one of the two write buffers in the memory controller core automatically based on availability. New write commands are designated to any available buffer. However, back-to-back write requests from a particular source ID are allocated to the same write buffer as the previous command.

Since the controller core must pull data out of the buffers in the order it was stored, if a write command is linked to a buffer that is associated with another command in the queue, then the new command is placed in the command queue after that command, regardless of priority.

This feature is always enabled.

#### **21.5.6.1.4 Priority**

Priorities distinguish important commands from less important commands. Each command is given a priority based on the command type through the programmable fields RP $n$  and WP $n$  (where  $n$  represents the port number). A value of 0 is the highest priority, and value of 1 is the lowest.

The placement algorithm attempts to place higher priority commands ahead of lower priority commands, as long as they have no source ID, write buffer, or address collisions. Higher priority commands are placed lower in the command queue if they access the same address, are from the same requestor, or use the same buffer as lower priority commands already in the command queue.

This feature is enabled through DDR\_CR08[PRIEN].

#### **21.5.6.1.5 Bank Splitting**

Before accesses can be made to two different rows within the same bank, the first active row must be closed (pre-charged) and the new row must be opened (activated). Both activities require some timing overhead. Therefore, for optimization, the placement queue attempts to insert the new command into the command queue such that commands to other banks may execute during this timing overhead. The placement of the new commands still follows priority, source ID, write buffer, and address collision rules.

The placement logic also attempts to optimize the memory controller core by inserting a command to the same bank as an existing command in the command queue immediately after the original. This reduces the overall timing overhead by potentially eliminating one pre-charging/activating cycle. This placement is only possible if there are no priority, source ID, write buffer, or address collisions or conflicts with other commands in the command queue.

All bank splitting features are enabled through DDR\_CR02[BNKSPT].

#### **21.5.6.1.6 Read/Write Grouping**

The memory suffers a small timing overhead when switching from read to write mode. For efficiency, the placement queue attempts to place a new read command sequentially with other read commands in the command queue, or a new write command sequentially with other write commands in the command queue. Grouping is only possible if no priority, source ID, write buffer, or address collision rules are violated.

This feature is enabled through DDR\_CR09[RWEN].

### **21.5.7 Command Execution Order After Placement**

When a command is placed in the command queue, its order relative to the other commands in the queue at that time is fixed. While this provides simplicity in the algorithm, there are drawbacks. For this reason, the memory controller offers two options that affect commands after they are placed in the command queue.

#### **21.5.7.1 High-Priority Command Swapping**

Commands are assigned priority values to ensure that critical commands are executed more quickly in the memory controller than less important commands. Therefore, it is desirable that high-priority commands pass into the memory controller core as soon as possible. The placement algorithm takes priority into account when determining the order of commands, but still allows a scenario in which a high-priority command sits waiting at the top of the command queue while another command, perhaps of a lower priority, is in process.

The high-priority command swapping feature allows this new high-priority command to be executed more quickly. If the swapping function is enabled by DDR\_CR10[SWPEN], then the entry at the top of the command queue is compared with the current command in progress. If the command queue's top entry is a higher priority (not the same priority) and it does not have an address, source ID, or write buffer conflict with the current command being executed, then the original command is interrupted.

If the command is to be interrupted, it is halted after completing the current burst, stored and placed at the top of the queue, and the new command is executed. As long as the command queue is not full, new commands may continue to be inserted into the command queue based on the placement rules, even at the head of the queue ahead of the interrupted command. The top entry in the command queue is executed next. Whenever the interrupted command is resumed, it starts from the point at which it was interrupted.

#### **NOTE**

Priority 0 commands are never interrupted, so set any commands that should not be interrupted to priority 0.

#### **21.5.7.2 Command Aging**

Since commands can be inserted ahead of existing commands in the command queue, the situation could occur where a low priority command remains at the bottom of the queue indefinitely. To avoid such a lockout condition, aging counters are included in the placement logic that measure the number of cycles that each command has been waiting. If command aging is enabled through DDR\_CR00[AGE], then if an aging counter hits its maximum, the priority of the associated command is decremented by one (lower priority commands are executed first). This increases the likelihood that this command moves to the top of the command queue and executed. This feature does not move relative positions in the command queue when it ages; the new priority is considered when placing new commands into the command queue.

Aging is controlled through a master aging-rate counter and command aging counters associated with each command in the command queue. DDR\_CR16[AGECNT] and DDR\_CR17[CMDAGE] hold the initial values for each of these counters, respectively. When the master counter counts down the AGEcnt value, a signal is sent to the command aging counters to decrement. When the command aging counters have completely decremented, then the priority of the associated command is decremented by one number and the counter is reset. Therefore, a command does not age by a priority level until the total elapsed cycles reaches the product of the AGEcnt and CMDAGE values. The maximum number of cycles that any command can wait in the command queue until reaching the top priority level is the product of AGEcnt, CMDAGE, and the number of priority levels in the system.

#### **21.5.8 Low Power Modes**

The various low power modes available for the memory controller are described in [Section 21.2.2, “Low Power Modes”](#). The memory controller may enter and exit these modes in the following ways:

- Automatic entry
- When the memory controller is idle, four timing counters begin counting the cycles of inactivity. If any counter expires, the memory controller enters the low-power mode associated with that counter.
- Manual entry

You may initiate any low power mode by setting the bit of DDR\_CR23[LPCTRL] associated with the desired mode. The memory controller enters the selected low power mode when it has completed its current burst.

Automatic and manual entry methods are both controlled by two parameters: DDR\_CR23[LPCTRL, LPAUTO]. LPCTRL contains individual enable/disable bits for each low-power mode, and LPAUTO enables automatic low-power mode entry for each low-power mode.

### 21.5.8.1 Automatic Entry

Automatic entry occur if the following conditions are true:

- The mode is programmed for automatic entry by setting the relevant bit in DDR\_CR23[LPAUTO]
- The particular mode is enabled in DDR\_CR23[LPCTRL]
- The memory controller is idle.
- The counter associated with this mode expires

There are four counters to cover the five low power modes:

- Separate counter for each of the three memory self-refresh low power modes (modes 3, 4 and 5)
- Memory power-down mode (mode 1) and memory power-down with memory clock gating mode (mode 2) share a counter

The counters determine the number of idle cycles before entering the associated low power mode. All counters are re-initialized each time there is a new read or write transaction entering or executing in the memory controller. This ensures that the memory controller does not enter any of the low power modes when active.

All five low power modes can be entered through automatic entry, and are exited automatically when any of the following conditions occur:

- A new read or write transaction appears at the memory controller interface
- The memory controller must refresh the memory when in either of the power-down modes (mode 1 or 2). After completing the memory refresh, the memory controller re-enters power-down.
- The counter for a deeper low-power mode expires. The memory controller must exit the current low power mode to enter the deeper low power mode. A minimum of 15 cycles occur between exit from one low power mode before entering the next low power mode, even if the counters expire within 15 cycles of each other. The memory controller cannot enter a less deep low power mode, regardless of which counters expire.

### 21.5.8.2 Manual On-Demand Entry

Manual entry occurs if the following conditions are true:

- The mode is programmed for manual entry by clearing the relevant bit in DDR\_CR23[LPAUTO]
- The particular mode is set in DDR\_CR23[LPCTRL]

For manual entry, DDR\_CR23[LPCTRL] triggers entry into the low power modes. The memory controller does not need to be idle when a low power mode bit is enabled. When a particular mode that is

programmed for manual entry is enabled, the memory controller completes the current memory burst access, and then, regardless of the activity inside the memory controller or at the memory interface, it enters the selected low power mode.

If new transactions enter the memory controller while it is in any of the low power modes, they accumulate inside the memory controller's command queue until the queue is full.

Exiting from a manually-entered low power mode is also manual. Clearing DDR\_CR23[LPCTRL] triggers the memory controller to pull the memory devices out of power-down or self-refresh, and command processing resumes.

#### NOTE

In the deepest low power mode (mode 5), the clock to the programming registers module is gated off. However, manual low power mode exit requires you to clear DDR\_CR23[LPCTRL], which is not possible when the clock is off. As a result, never manually activate the deepest low power mode.

If a different DDR\_CR23[LPCTRL] bit is set while in one of the low power modes, or on clearing of the original bit, the memory controller exits the current low power mode. There is at least a 15 cycle delay before the memory controller is fully operational or enters the new low power mode.

#### 21.5.8.3 Register Programming

The low power modes of the memory controller are controlled through DDR\_CR23[LPCTRL] and LPAUTO]. These five-bit parameters each contain one bit for controlling each low power mode.

DDR\_CR23[LPCTRL] enables the associated low power mode, and DDR\_CR23[LPAUTO] sets the entry method into that mode as manual or automatic. [Table 21-60](#) shows the relationship between the five bits of DDR\_CR23[LPCTRL and LPAUTO] and the various low power modes.

**Table 21-60. Low Power Mode Parameters**

Low power mode	Enable	Entry	Counter
Memory power down (mode 1)	LPCTRL[4] = 1	LPAUTO[4] 0 Manual 1 Automatic	DDR_CR39[LPPDCNT]
Memory power down with memory clock gating (mode 2)	LPCTRL[3] = 1	LPAUTO[3] 0 Manual 1 Automatic	DDR_CR39[LPPDCNT]
Memory self-refresh (mode 3)	LPCTRL[2] = 1	LPAUTO[2] 0 Manual 1 Automatic	DDR_CR40[LPSREFCNT]
Memory self-refresh with memory clock gating (mode 4)	LPCTRL[1] = 1	LPAUTO[1] 0 Manual 1 Automatic	DDR_CR38[LPEXTCNT]
Memory self-refresh with memory and controller clock gating (mode 5)	LPCTRL[0] = 1	LPAUTO[0] 0 Manual 1 Automatic	DDR_CR39[LPINTCNT]

When a LPCTRL bit is set, the memory controller checks the corresponding LPAUTO bit.

- If the associated bit is set, the memory controller watches the associated counter for expiration and then enters that low power mode. [Table 21-60](#) shows the correlation between the low power modes and the counters that control each mode's automatic entry.
- If the associated bit is cleared, the memory controller completes its current memory burst access and then enter the specified low power mode.

The values in LPAUTO are only relevant when the associated LPCTRL bit is set.

Multiple bits in LPCTRL and LPAUTO can be set at the same time. When this happens, the memory controller always enters the deepest low power mode of all the modes that are enabled. If the memory controller is already in one low power mode when a deeper low power mode is requested automatically or manually, it must first exit the current low power mode, and then enter the deeper low power mode. A minimum 15 cycle delay occurs before the second entry.

The timing for automatic entry into any of the low power modes is based on the number of idle cycles that have elapsed in the memory controller. There are four counters related to the five low power modes to determine when any particular low power mode is entered if the automatic entry option is chosen. The counters are also shown in [Table 10-1, “Low Power Mode Parameters”](#). Since the two power-down modes share one counter, if you wish to enter memory power-down mode (mode 1) automatically, then you must not enable memory power-down with memory clock gating mode (mode 2).

### 21.5.9 Out-of-Range Address Checking

It is possible that the master attempts to write to an invalid address. For this reason, all incoming addresses are always checked against the addressable physical memory space. If a transaction is addressed to an out-of-range memory location, then bit 0 of DDR\_CR27[INTSTATUS] is set to alert you of this condition. The memory controller records the following information that caused the out-of-range interrupt:

- Address: {DDR\_CR56[OORAD32], DDR\_CR55[OORAD]}
- Source ID: DDR\_CR06[OORID]
- Length: DDR\_CR27[OORLEN]
- Type of transaction: DDR\_CR14[OORTYPE]

Reading the out-of-range parameters initiates the memory controller to empty these parameters and allow them to store out-of-range access information for future errors. Acknowledge the interrupt by setting bit 0 of DDR\_CR25[INTACK], which causes bit 0 of INTSTATUS to clear.

If a second out-of-range access occurs before the first out-of-range interrupt is acknowledged, then bit 1 of DDR\_CR27[INTSTATUS] is set to indicate that multiple out-of-range accesses have occurred. If the out-of-range parameters have been read when the second out-of-range error occurs, then the details for this transaction are stored in the out-of-range parameters. If they have not been read, then the details of the second error are lost.

Even though the address is identified as erroneous, the memory controller still processes the read or write transaction. A read transaction returns random data which you must receive to avoid stalling the memory controller. A write transaction writes the associated data to an unknown location in the memory array,

potentially over-writing other stored data. The command cannot be aborted once accepted into the memory controller.

## **21.6 Initialization/Application Information**

The memory controller requires a sequence for correct operation after power to the microcontroller and memory devices is stable. When initialized, the memory controller automatically initializes the memory devices.

The procedure to initialize the memory controller is as follows:

1. Issue write register commands to configure the DRAM protocols and the settings for the DCC. Keep DDR\_CR09[START] cleared during this initialization step.
2. Set DDR\_CR09[START]. This triggers the memory controller to execute the initialization sequence using the parameters written into the registers. The memory controller waits for the PHY to indicate that the PHY and memory devices are ready to accept commands.

# **Chapter 22**

## **NAND Flash Controller (NFC)**

### **22.1 Introduction**

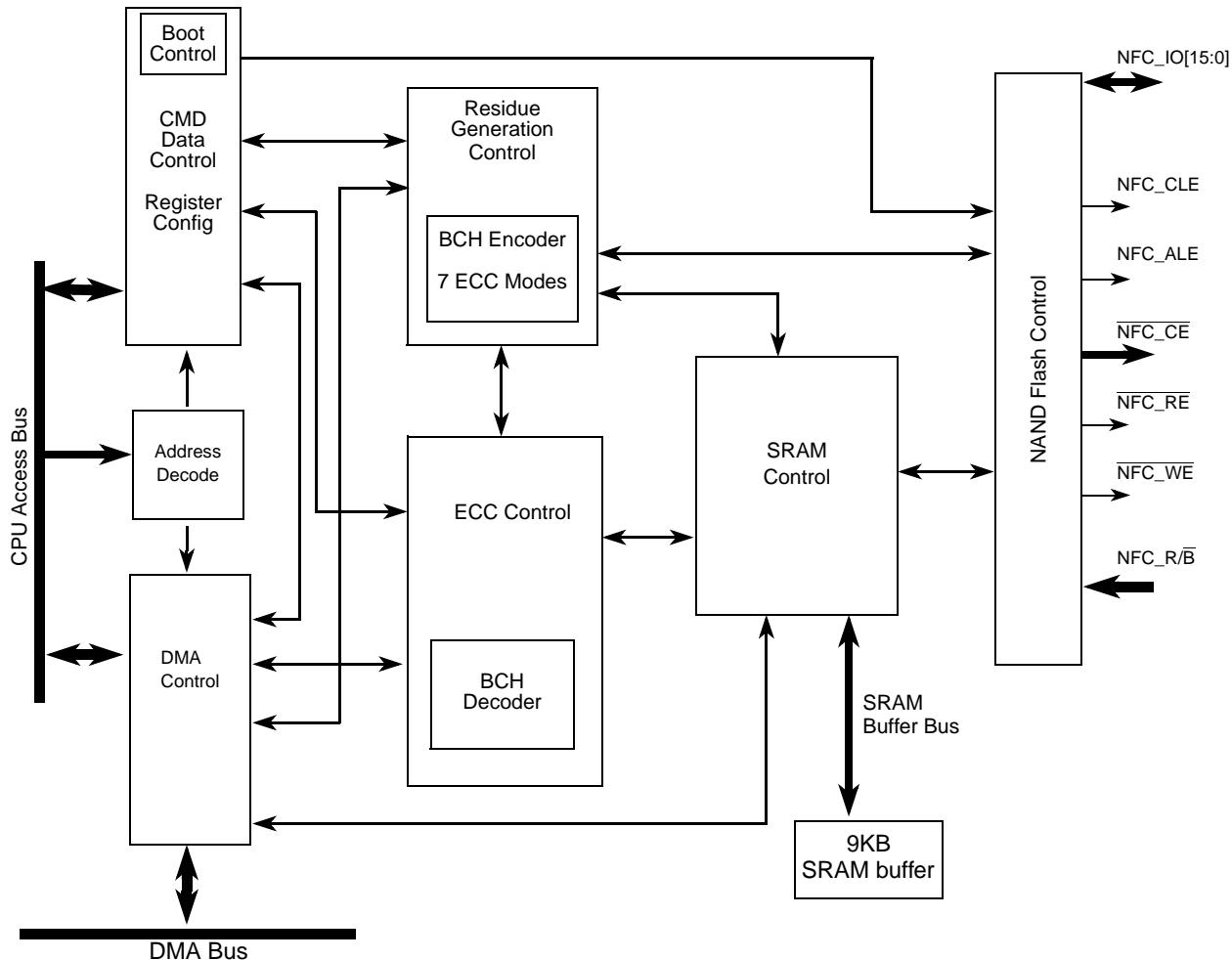
The NAND flash controller (NFC) interfaces to standard NAND flash memory devices. It is composed of various control logic units and a 9 KB SRAM buffer. The NFC provides a glueless interface to 8- and 16-bit NAND flash devices with page sizes of 512 bytes, 2 KB, 4 KB, and 8 KB.

Throughout this chapter the following terms are used:

- Block — (specified by device) smallest erasable unit in a NAND device, consisting of multiple pages
- Page — (specified by device) unit of flash data containing main and spare areas
- Main area of a page— stores data
- Spare area of a page — stores ECC and other software information
- Sector — an elementary transfer unit
  - For devices with pages of 2KB and smaller, this is the same size of the page
  - For devices with pages larger than 2KB, the pages are split into multiple virtual pages. In this case, the sector size is the size of the virtual page
- Virtual page — is the physical page size divided by the splitting factor, NFC\_CFG[PAGECNT]
- ECC — error-correcting code
- BCH (Bose Chaudhuri Hocquenghem) — cyclic error-correcting code that corrects multi-bit errors

## 22.1.1 Block Diagram

See [Figure 22-1](#) for a block diagram of the NAND flash controller.



**Figure 22-1. NAND Flash Controller Block Diagram**

## 22.1.2 Features

The NAND flash controller includes the following features:

- 8- and 16-bit NAND flash interface
- 9 KB RAM buffer
  - Memory-mapped registers and SRAM buffer
- Supports all NAND flash products regardless of density/organization
- Supports flash device commands, such as page read, page program, reset, block erase, read status, read ID, copy-back, multiplane read/program, interleaved read/program, random input/output, read in EDO mode.
- Integrated DMA engine

- Two configurable DMA channels
  - Use DMA channel 1 only to read/write a page for main and spare area of a page
  - Use DMA channel 1 to read/write the main area of a page, and DMA channel 2 for the spare area
- ECC mode
  - In ECC mode, NFC supports 4/6/8/12/16/24/32-bit error correction.
  - ECC mode can be bypassed.
- Boot from page size  $\geq$  2KB flash (x8) without extra control

## 22.2 External Signal Description

The following signals shown in [Table 22-1](#) are used to control NAND flash device.

**Table 22-1. NFC Signal Properties**

Name	Function	I/O	Reset
NFC_ALE	Flash address latch enable	O	1
<u>NFC_CE</u>	Flash chip enable	O	1
NFC_CLE	Flash command latch enable	O	1
NFC_R/ <u>B</u>	Flash ready/busy	I	Pull up
NFC_RE	Flash read enable	O	1
<u>NFC_WE</u>	Flash write enable	O	1
NFC_IO[15:0]	Flash data bus	I/O	—

## 22.3 Memory Map/Register Definition

**Table 22-3. NFC Memory Map**

Offset or Address	Register	Width (bits)	Access	Reset Value	Section/Page
<b>SRAM buffer</b>					
0xFC0F_C000 — 0xFC0F_F8FF	SRAM buffer	64	R/W	—	<a href="#">22.4.1/22-15</a>
<b>General Registers</b>					
0xFC0F_FF00	Flash command 1 (NFC_CMD1)	32	R/W	0x30FF_0000	<a href="#">22.3.1/22-4</a>
0xFC0F_FF04	Flash command 2 (NFC_CMD2)	32	R/W	0x007E_E000	<a href="#">22.3.2/22-5</a>
0xFC0F_FF08	Column address (NFC_CAR)	32	R/W	0x0000_0000	<a href="#">22.3.3/22-6</a>
0xFC0F_FF0C	Row address (NFC_RAR)	32	R/W	0x1100_0000	<a href="#">22.3.4/22-6</a>

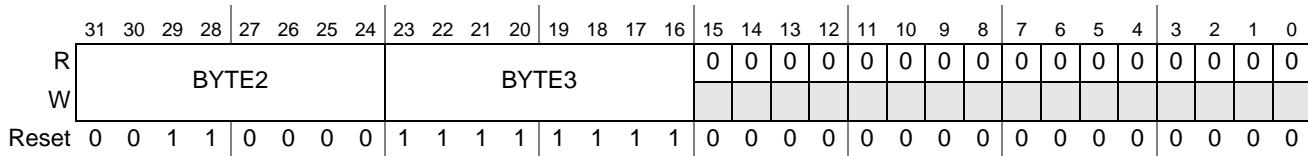
**Table 22-3. NFC Memory Map (continued)**

Offset or Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0F_FF10	Flash command repeat (NFC_RPT)	32	R/W	0x0000_0000	<a href="#">22.3.5/22-7</a>
0xFC0F_FF14	Row address increment (NFC_RAI)	32	R/W	0x0000_0001	<a href="#">22.3.6/22-7</a>
0xFC0F_FF18	Flash status 1 (NFC_SR1)	32	R	0x0000_0000	<a href="#">22.3.7/22-8</a>
0xFC0F_FF1C	Flash status 2 (NFC_SR2)	32	R	0x0000_0000	<a href="#">22.3.8/22-8</a>
0xFC0F_FF20	DMA 1 address register (NFC_DMA1)	32	R/W	0x0000_0000	<a href="#">22.3.9/22-9</a>
0xFC0F_FF34	DMA 2 address register (NFC_DMA2)	32	R/W	0x0000_0000	<a href="#">22.3.10/22-9</a>
0xFC0F_FF24	DMA configuration register (NFC_DMACFG)	32	R/W	0x0000_0000	<a href="#">22.3.11/22-9</a>
0xFC0F_FF28	Cache swap register (NFC_SWAP)	32	R/W	0xFFE_0FFE	<a href="#">22.3.12/22-10</a>
0xFC0F_FF2C	Sector size register (NFC_SECSZ)	32	R/W	0x0000_0420	<a href="#">22.3.13/22-11</a>
0xFC0F_FF30	Flash configuration register (NFC_CFG)	32	R/W	0x000E_A631	<a href="#">22.3.14/22-11</a>
0xFC0F_FF38	Interrupt status register (NFC_ISR)	32	R/W	0x6000_0000	<a href="#">22.3.15/22-13</a>

### 22.3.1 Flash Command 1 Register (NFC\_CMD1)

Address: 0xFC0F\_FF00 (NFC\_CMD1)

Access: User read/write



**Figure 22-2. Flash Command 1 Register (NFC\_CMD1)**

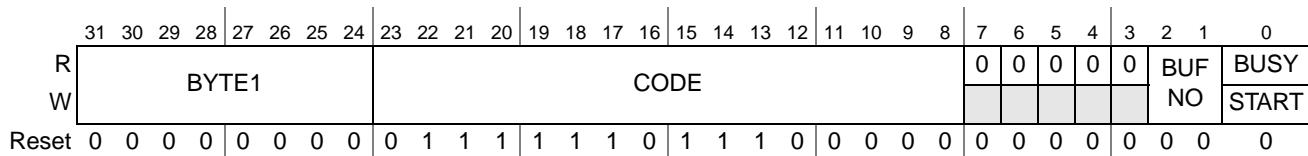
**Table 22-4. NFC\_CMD1 Field Descriptions**

Field	Description
31–24 BYTE2	Second command byte that may be sent to the flash device
23–16 BYTE3	Third command byte that may be sent to the flash device
15–0	Reserved, must be cleared.

### 22.3.2 Flash Command 2 Register (NFC\_CMD2)

Address: 0xFC0F\_FF04 (NFC\_CMD2)

Access: User read/write



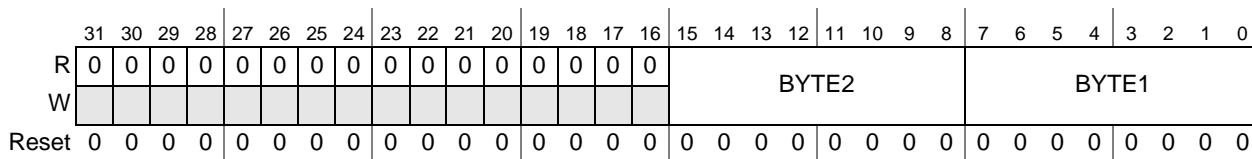
**Figure 22-3. Flash Command 2 Register (NFC\_CMD2)**

**Table 22-5. NFC\_CMD2 Field Descriptions**

Field	Description
31–24 BYTE1	First command byte that may be sent to flash device
23–8 CODE	User-defined flash operation sequencer. Each bit indicates a certain action. (See <a href="#">Table 22-22</a> ) If the bit is set, the corresponding action is executed after writing 1 to START.  The following are some configuration examples (other sequences are possible): 0111_1110_1110_0000 Read data (BYTE1, 5x Address, BYTE2, R/B, read data) 1111_1111_1101_1000 Write page (DMA,BYTE1, 5x Address, write data, BYTE2, R/B, BYTE3, read status) 0100_1110_1101_1000 Block erase (BYTE1, 3x Address, BYTE2, R/B, BYTE3, read status) 0100_1000_0000_0100 Read ID (BYTE1, 1x Address, read ID) 0100_0000_0100_0000 Reset (BYTE1, R/B) 0100_0000_0110_0000 Read burst (BYTE1, R/B, read data) 0111_1110_0000_0000 CMD+address (BYTE1, 5x address) 1111_1111_1100_0000 Write page burst (DMA,BYTE1,5xAddress, write data, BYTE2,R/B)
7–3	Reserved, must be cleared.
2–1 BUFNO	Internal buffer number used for this command
0 BUSY/START	Busy indicator and start command.  During reads: 0 Flash controller is idle - OK to send next command. 1 Command execution is busy <b>Note:</b> This bit is repeated in NFC_ISR.  During writes: 0 No action 1 Command execution starts

### 22.3.3 Column Address Register (NFC\_CAR)

Address: 0xFC0F\_FF08 (NFC\_CAR) Access: User read/write

**Figure 22-4. Column Address Register (NFC\_CAR)****Table 22-6. NFC\_CAR Field Descriptions**

Field	Description
31–16	Reserved, must be cleared.
15–8 BYTE2	Second byte of column address
7–0 BYTE1	First byte of column address

### 22.3.4 Row Address Register (NFC\_RAR)

Address: 0xFC0F\_FF0C (NFC\_RAR) Access: User 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	0	0	0	0	CS	0	0	0	RB	BYTE3	BYTE2	BYTE1																				
W																																
Reset	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

Figure 22-5. Row Address Register (NFC\_RAR)

Table 22-7. NFC\_RAR Field Descriptions

Field	Description
31–29	Reserved, must be cleared.
28 CS	Chip select. Bit mask to enable/disable the chip select. 0 NFC_CE is disabled 1 NFC_CE is enabled
27–25	Reserved, must be cleared.
24 RB	Ready/busy enable. Determines if NFC_R/B is waited on a wait for R/B command. 0 NFC_R/B is disabled 1 NFC_R/B is enabled
23–16 BYTE3	Third byte of row address
15–8 BYTE2	Second byte of row address
7–0 BYTE1	First byte of row address

### 22.3.5 Flash Command Repeat (NFC\_RPT)

Address: 0xFC0F\_FF10 (NFC\_RPT) Access: User 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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	COUNT												
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		

Figure 22-6. Flash Command Repeat Register (NFC\_RPT)

Table 22-8. NFC\_RPT Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 COUNT	16-bit repeat count. Determines how many times NFC_CMD2[CODE] is executed. If 0 or 1, the flash command is executed once.

### 22.3.6 Row Address Increment (NFC\_RAI)

When auto-increment of row address is enabled (NFC\_CFG[AIAD] = 1), the row address is incremented as follows:

```
new{NFC_RAR[BYTE3, BYTE2, BYTE1]} =
    {NFC_RAR[BYTE3], NFC_RAR[BYTE2], NFC_RAR[BYTE1]} + {NFC_RAI[INC3, INC2, INC1]}
```

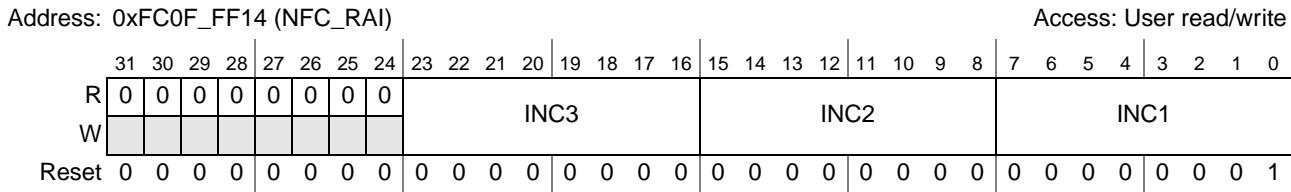


Figure 22-7. Row Address Increment Register (NFC\_RAI)

Table 22-9. NFC\_RAI Fields

Field	Description
31–24	Reserved, must be cleared.
23–16 INC3	Increment for the third byte of row address
15–8 INC2	Increment for the second byte of row address
7–0 INC1	Increment for the first byte of row address

### 22.3.7 Flash Status 1 Register (NFC\_SR1)

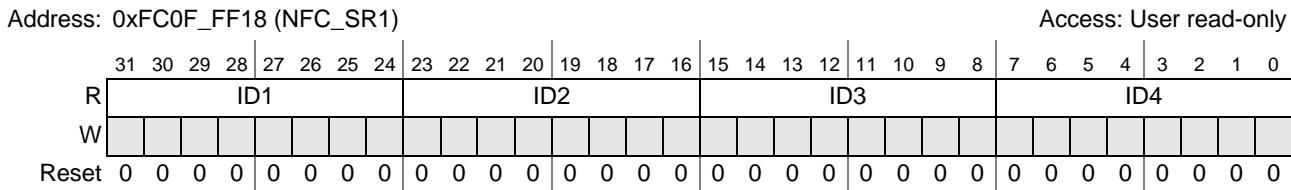


Figure 22-8. Flash Status 1 Register (NFC\_SR1)

Table 22-10. NFC\_SR1 Field Descriptions

Field	Description
31–24 ID1	First byte returned by <i>read ID</i> command
23–16 ID2	Second byte returned by <i>read ID</i> command
15–8 ID3	Third byte returned by <i>read ID</i> command
7–0 ID4	Fourth byte returned by <i>read ID</i> command

### 22.3.8 Flash Status 2 Register (NFC\_SR2)

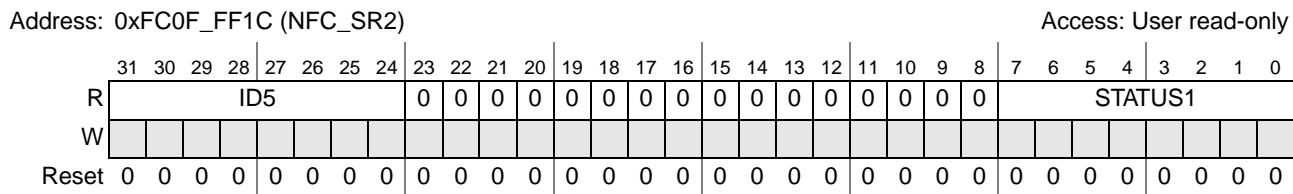


Figure 22-9. Flash Status 2 Register (NFC\_SR2)

Table 22-11. NFC\_SR2 Field Descriptions

Field	Description
31–24 ID5	Fifth byte returned by <i>read ID</i> command
23–8	Reserved, must be cleared.
7–0 STATUS1	Byte returned by <i>read status</i> command

### 22.3.9 DMA1 Address Register (NFC\_DMA1)

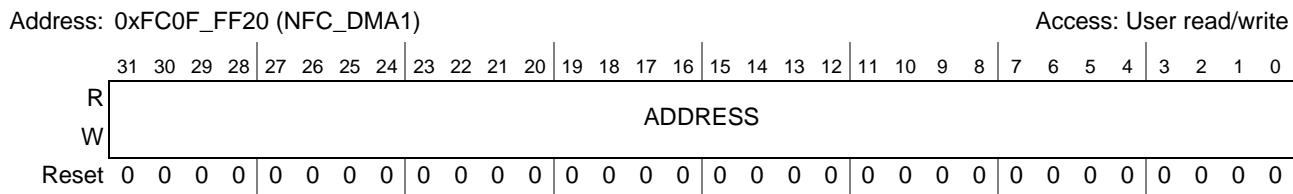


Figure 22-10. DMA1 Address Register (NFC\_DMA1)

Table 22-12. NFC\_DMA1 Field Descriptions

Field	Description
31–0 ADDRESS	DMA channel 1 address

### 22.3.10 DMA2 Address Register (NFC\_DMA2)

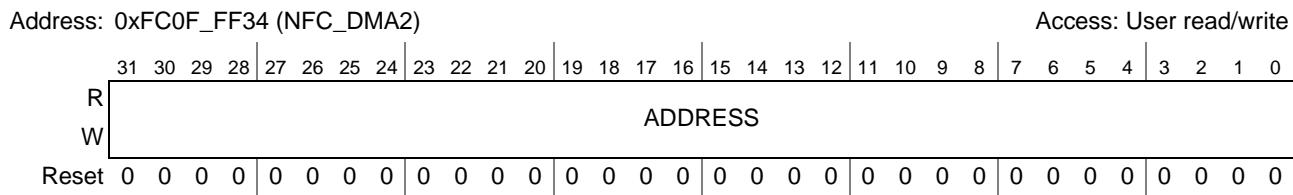


Figure 22-11. DMA2 Address Register (NFC\_DMA2)

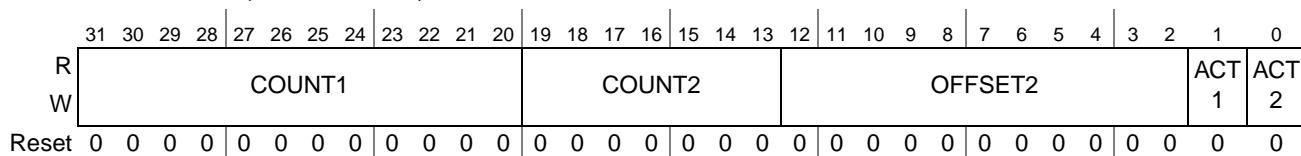
**Table 22-13. NFC\_DMA2 Field Descriptions**

Field	Description
31–0 ADDRESS	DMA channel 2 address

### 22.3.11 DMA Configuration Register (NFC\_DMACFG)

Address: 0xFC0F\_FF24 (NFC\_DMACFG)

Access: User read/write



**Figure 22-12. DMA2 Configuration Register (NFC DMACFG)**

**Table 22-14. NFC DMACFG Field Descriptions**

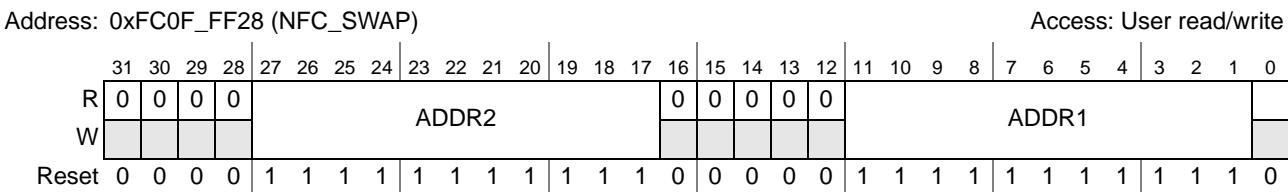
Field	Description
31–20 COUNT1	Byte count to be transferred by DMA for DMA channel 1
19–13 COUNT2	Byte count to be transferred by DMA for DMA channel 2
12–2 OFFSET2	Byte offset for DMA channel 2. DMA channel 2 transfer starts at this offset count.
1 ACT1	0 DMA channel 1 is inactive 1 DMA channel 1 is active, and transfers to memory when triggered
0 ACT2	0 DMA channel 2 is inactive 1 DMA channel 2 is active, and will be used for transfers to memory when triggered

### 22.3.12 Cache Swap Register (NFC\_SWAP)

When DMA transfers data to/from the NFC cache (NFC SRAM buffer), or when the CPU reads or writes data to/from the NFC cache via the internal bus, all accesses that go to NFC\_SWAP[ADDR1], are directed to NFC\_SWAP[ADDR2]. Likewise, all accesses that go to NFC\_SWAP[ADDR2] are directed to NFC\_SWAP[ADDR1].

The feature allows the bad block marker in the first position of the spare area of a page. Because of the way the flash controller interleaves data and ECC bytes on flash devices with page sizes larger than 2 KB, the position of the bad block marker is shifted, and does not appear in the first position of the spare area of the page. The cache swap feature allows consistent swapping of the actual bad block line with the expected bad block line, and causes the operating system to get the bad block marker in the position where it is expected. [Table 22-21](#) gives some examples of usage.

## NAND Flash Controller (NFC)

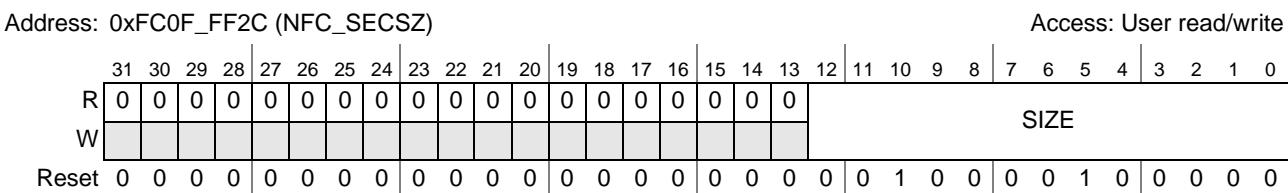


**Figure 22-13. Cache Swap Register (NFC\_SWAP)**

**Table 22-15. NFC\_SWAP Fields**

Field	Description
31–28	Reserved, must be cleared.
27–17 ADDR1	Lower swap address
16–12	Reserved, must be cleared.
11–1 ADDR2	Upper swap address
0	Reserved, must be cleared.

### 22.3.13 Sector Size Register (NFC\_SECSZ)



**Figure 22-14. Sector Size Register (NFC\_SECSZ)**

**Table 22-16. NFC\_SECSZ Field Descriptions**

Field	Description
31–13	Reserved, must be cleared.
12–0 SIZE	<p>Size in bytes of one elementary transfer unit. For devices with pages of 2KB and smaller, this is the physical size of the page in bytes (data bytes + header bytes + ECC bytes) transferred in one page. When pages are larger than 2KB, they must be split in multiple virtual pages. In this case, the sector size is the size of the virtual page. The virtual page size is the physical size divided by the splitting factor, NFC_CFG[PAGECNT]. <a href="#">Table 22-20</a> gives examples on programming this field.</p> <p><b>Note:</b> If only a part of a page to be programmed or read, SIZE can be set to the number of affected bytes, not the page size. Then, ECC and DMA (data bytes) are all performed on the number of bytes, indicated by SIZE.</p> <p><b>Note:</b> For 16-bit data width flash devices, only odd SIZE is supported. If SIZE is even number, the real implemented size is SIZE – 1. So, write size + 1 to this field. For example, if SIZE = 1, no data is written or read.</p> <p><b>Note:</b> When programming NAND memory for boot and using the ECC feature, ensure that SIZE is equal to the default value (data (996) + ECC (60) bytes).</p>

## 22.3.14 Flash Configuration Register (NFC\_CFG)

Address: 0xFC0F_FF30 (NFC_CFG)												Access: User read/write					
R	STOP WERR	ECCAD												ECC SRAM	DMA REQ	ECCMODE	FAST
Reset	0 0 0 0	0 0 0 0				0 0 0 0				0 0 0 0				1 1 1 0			
R	IDCNT	TIMEOUT						16BIT	BT MD	AIAD	AIBN	PAGECNT					
Reset	1 0 1 0	0 1 1 0				0 See Note				1 1	0 0 0 1	0 0 0 1					

Note: Resets to 0 if no boot is performed by the NFC; 1 if booting from NFC.

Figure 22-15. Flash Configuration Register (NFC\_CFG)

Table 22-17. NFC\_CFG Field Descriptions

Field	Description
31 STOPWERR	0 No stop on write error 1 Auto sequencer stops on a write error
30–22 ECCAD	Byte address in SRAM where ECC status is written.
21 ECCSRAM	0 Do not write ECC status to SRAM 1 Write ECC status to SRAM
20 DMAREQ	0 Do not transfer sector after ECC done 1 After ECC done, transfer sector using DMA
19–17 ECCMODE	000 No correction, ECC bypass 001 4-error correction (8 ECC bytes) 010 6-error correction (12 ECC bytes) 011 8-error correction (15 ECC bytes) 100 12-error correction (23 ECC bytes) 101 16-error correction (30 ECC bytes) 110 24-error correction (45 ECC bytes) 111 32-error correction (60 ECC bytes)
16 FAST	0 Slow flash timing. Clock in read data on rising edge of read strobe 1 Fast flash timing. Clock in read data a half clock later than rising edge of read strobe See Section 22.4.5, "Fast Flash Configuration for EDO" for more details.
15–13 IDCNT	Number of bytes that are read for the <i>read id</i> command.

Table 22-17. NFC\_CFG Field Descriptions (continued)

Field	Description
12–8 TIMEOUT	The number of flash_clk cycles from NFC_WE high to either: <ul style="list-style-type: none"> <li>• NAND flash busy (<math>t_{WB}</math>), or</li> <li>• NFC_RE low (<math>t_{WHR}</math>)</li> </ul> After the last command is issued to flash, before sampling NFC_R/B, the NFC must wait $t_{WB}$ clocks. After $t_{WB}$ clocks: <ul style="list-style-type: none"> <li>• if NFC_R/B is sampled as high, the NFC considers the command to be a timeout, and the flash memory is idle. The NFC can issue new commands to the flash memory.</li> <li>• if NFC_R/B is sampled as low, the NAND flash memory is busy.</li> </ul> When reading the status or ID from the NAND flash memory, after the last command is issued to flash, the NFC must wait for $t_{WHR}$ cycles. The NFC then negates NFC_RE to low to read the valid status or ID. <b>Note:</b> $t_{WB}$ exists in page program/read, block erase, etc. Refer to the NAND flash datasheet for details of $t_{WB}$ and $t_{WHR}$ .
7 16BIT	0 8-bit wide flash mode 1 16-bit wide flash mode
6 BTMD	0 Normal mode 1 Boot mode <b>Note:</b> Resets to 0 if no boot is performed from the NFC, 1 if NFC boot is performed
5 AIAD	0 Do not auto-increment flash row address 1 Auto-increment flash row address
4 AIBN	0 Do not auto-increment buffer number 1 Auto increment buffer number
3–0 PAGECNT	Number of virtual pages (in one physical flash page) to be programmed or read, etc.

### 22.3.15 Interrupt Status Register (NFC\_ISR)

Address: 0xFC0F\_FF38 (NFC\_ISR)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	WERR	DONE	IDLE	0	WERR NS	CMD BUSY	RES BUSY	ECC BUSY	DMA BUSY	WERR EN	DONE EN	IDLE EN	0	0	0	0
W													WERR CLR	DON ECLR	IDLE CLR	
Reset	0	1	1	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	0	0	0	0	0	0	0	0	0	0	0	ECCBN		DMABN	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 22-16. Interrupt Status Register (NFC\_ISR)

**Table 22-18. NFC\_ISR Field Descriptions**

<b>Field</b>	<b>Description</b>
31 WERR	Write error interrupt. Set if an error condition is detected during a flash read status command. Sticky bit
30 DONE	Done interrupt. Set if command processing is done.
29 IDLE	Command idle interrupt. Set if command done, and residue engine, ECC engine and DMA engine are idle.
28	Reserved, must be cleared.
27 WERRNS	Write error status. Set if an error condition was detected during the last flash read status command. Non-sticky bit.
26 CMDBUSY	Set if command execution busy, cleared otherwise.
25 RESBUSY	Set if residue engine busy, cleared otherwise.
24 ECCBUSY	Set if ECC engine busy, cleared otherwise.
23 DMABUSY	Set if DMA engine busy, cleared otherwise.
22 WERREN	Enable bit for NFC_ISR[WERR].
21 DONEEN	Enable bit for NFC_ISR[DONE].
20 IDLEEN	Enable bit for command NFC_ISR[IDLE].
19 WERRCLR	Clear bit for NFC_ISR[WERR]. Writing 1 to this bit clears NFC_ISR[WERR].
18 DONECLR	Clear bit for NFC_ISR[DONE]. Writing 1 to this bit clears NFC_ISR[DONE].
17 IDLECLR	Clear bit for NFC_ISR[IDLE]. Writing 1 to this bit clears NFC_ISR[IDLE].
16–6	Reserved, must be cleared.
5–4 RESBN	Residue buffer number. Buffer number corresponding with the current residue block task.
3–2 ECCBN	ECC buffer number. Buffer number corresponding with the current ECC task.
1–0 DMABN	DMA buffer number - Buffer number corresponding with the current DMA task.

## 22.4 Functional Description

The NFC executes commands on a single or bank of external NAND flash chips. The NFC supports commands such as read, program, reset, erase, status read, read ID.

The NFC block contains a DMA engine and built-in ECC logic. For each read or write, the NFC performs ECC calculations on-the-fly. Two DMA channels are organized for each read or write: one for the main area, and one for the spare area. It is possible to disable the second DMA channel, and transfer main and spare data with only the first DMA channel.

Page size supported is 512, 2K, 4K and 8K bytes. There are 8 different ECC settings provided: 0, 4, 6, 8, 12, 16, 24 and 32 bits errors. They use 0, 8, 12, 15, 23, 30, 45 and 60 ECC bytes. The ECC works on page sizes of 512+spares bytes, 1K+spares bytes, 2K+spares bytes. The ECC algorithm used is a BCH code.

The error corrector can write ECC status to the spare area, since the read is pipelined. This means, while the current page is transferred from flash to buffer, the previous page is ECC corrected, and the page before that is transferred using DMA. Because of the pipelining, it is difficult to inform the CPU in the foreground of ECC errors. To solve this, ECC status is written to the auxiliary area of the sector, and transferred to memory. See [Section 22.4.2, “Error Corrector Status”](#) for more information. It’s up to the CPU to inspect the ECC result in memory, and act appropriately.

As described, reads are pipelined. However, writes are flow-through; no advance operations are done during a write. If there is a problem found during a write, the command sequence may be interrupted, and the CPU is informed.

Each page read, page write, page erase, read ID, or read status command sequence needs CPU attention only once. The CPU needs to prepare the DMA to point to the data, write correct values to all registers, and start the command. After command completion, the NFC block may interrupt the CPU.

The block allows command repeat, which is useful for write, read and erase, and allows processing multiple pages with just one command given by the CPU. No bank interleaving is supported during command repeat.

Booting from NAND flash is optional. The feature is activated when the reset configuration indicates a boot from NAND flash. See [Chapter 10, “Chip Configuration Module \(CCM\)”](#), for details on selecting NFC boot on this device. If boot feature is activated, first the NFC issues a reset command (0xFF) to the flash, then NFC reads four pages from block 0. Each page is 1056 bytes. The boot pages are protected by 32-bit error correction, which means that of the 1056 bytes, 996 bytes are user bytes and 60 bytes are ECC bytes. When the data from the boot pages is read, successfully error corrected, and stored in the NFC SRAM, the NFC indicates to the CPU that its boot code is visible in the NFC SRAM, and visible on addresses 0x000 to 0xF8F (3984 bytes total).

If the boot image from block 0 cannot be corrected, because there are more than 32 errors in one or several pages, boot is retried on the blocks at row addresses 256, 512, and 768. If it still fails after these retries, boot from the NFC is aborted and the processor begins execution using the default memory mapped to the FlexBus.

Right after boot, a special address hashing function is active on all reads and writes done to NFC SRAM. This hashing function interleaves the page data from the four boot pages in such a way that all user data is visible in address range 0x000 to 0xF8F instead of four different ranges, one for each page. This hashing

is controlled by NFC\_CFG[BTMD], and the hashing should be turned off by the CPU after finishing reading/executing the boot image, and before normal operations of the NFC. See [Figure 22-25](#) and [Section 22.4.1, “NFC Buffer Memory Space”](#).

Page size at boot is set to 1056 bytes to be compatible with a large number of NFC devices, without needing additional power-on reset flags to indicate the boot device.

- Compatible with 8-wide SLC and MLC devices with page size of 2 KB + 64 bytes spare
- Compatible with 8-wide SLC and MLC devices with page size of 4 KB and larger
- Compatible with 16-wide SLC and MLC devices with page size of 2 KB + 64 bytes spare
- Compatible with 16-wide SLC and MLC devices with page size of 4 KB and larger
- Not compatible with devices with 512 bytes page size.

## 22.4.1 NFC Buffer Memory Space

[Figure 22-17](#) shows the organization of the buffer memory space in the NFC. The memory's size is  $1152 \times 64$  bit, and is separated into four buffers, each with incontinuous physical address. For example, buffer 0's physical address is  $(0xFC0F\_C000 + 0x20 \times i) - (0xFC0F\_C007 + 0x20 \times i)$ .

However, when the CPU writes or reads a buffer in non-boot mode, the CPU address is continuous, since there's an address transition inside NFC: `sram_physical_addr[13:3] = {cpu_addr[11:3],cpu_addr[13:12]}`

So, in non-boot mode, the address ranges are:

- Buffer 0: 0xFC0F\_C000 – 0xFC0F\_C8FF
- Buffer 1: 0xFC0F\_D000 – 0xFC0F\_D8FF
- Buffer 2: 0xFC0F\_E000 – 0xFC0F\_E8FF
- Buffer 3: 0xFC0F\_F000 – 0xFC0F\_F8FF

See [Figure 22-25](#) for the different operations between the NFC\_CFG[BTMD] settings.

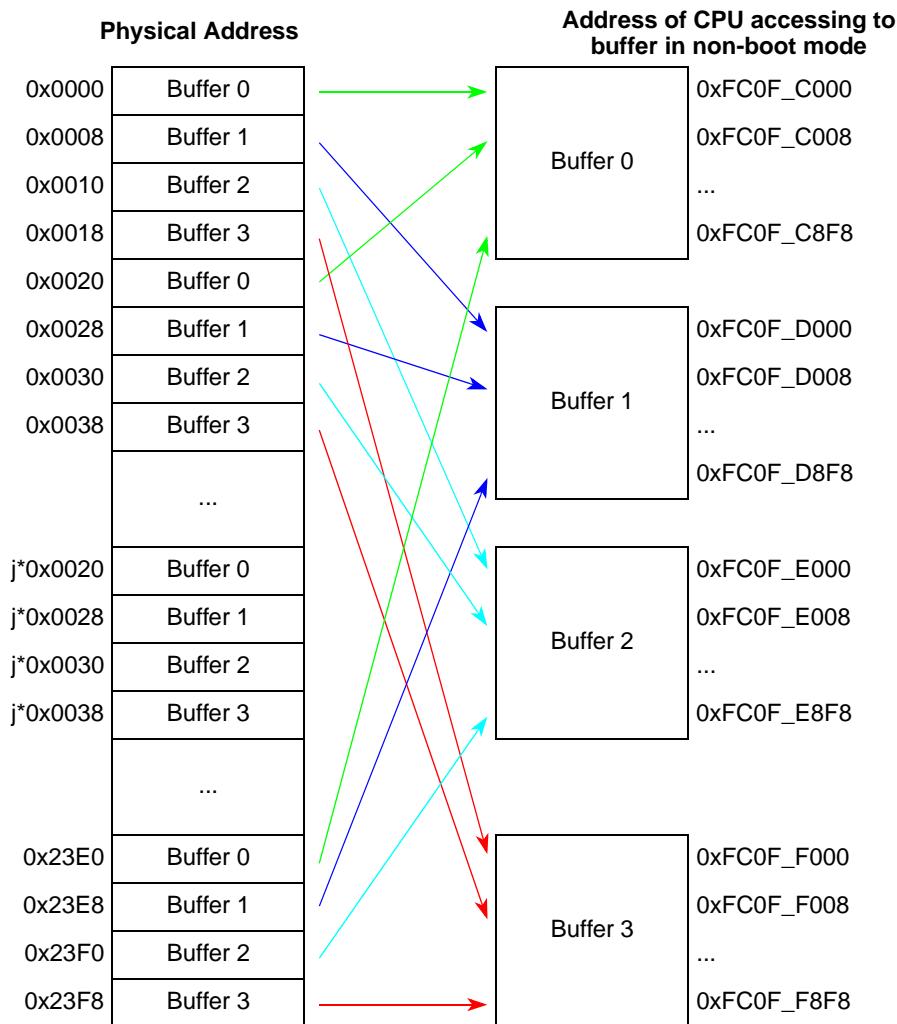


Figure 22-17. NFC Buffer Memory Space

## 22.4.2 Error Corrector Status

The ECC engine determines if a page is correctable. If correctable, it corrects error bits, and indicates error number. Otherwise, CORFAIL bit is asserted as shown in [Table 22-19](#). For a bad block management strategy to work, it may be necessary for the processor to obtain this information.

The error corrector writes the status word to a byte location to the SRAM buffer, defined by NFC\_CFG[ECCAD]. It is selectable if the status is written or not with NFC\_CFG[ECCSRAM]. If the status is written to the SRAM buffer, it becomes effectively part of the flash data, and is processed like the flash data. Most likely, the status byte is written to memory as part of the page header. Once in memory, the ECC status is visible to the CPU, while CPU parses the rest of the flash header. No interrupt on error or status is available because this increases the interrupt load on the CPU. (The interrupt would be independent of the command done interrupt.) It is not possible to stop reading when ECC failed.

The organization of the status byte is given in [Table 22-19](#).

Table 22-19. ECC Status Word

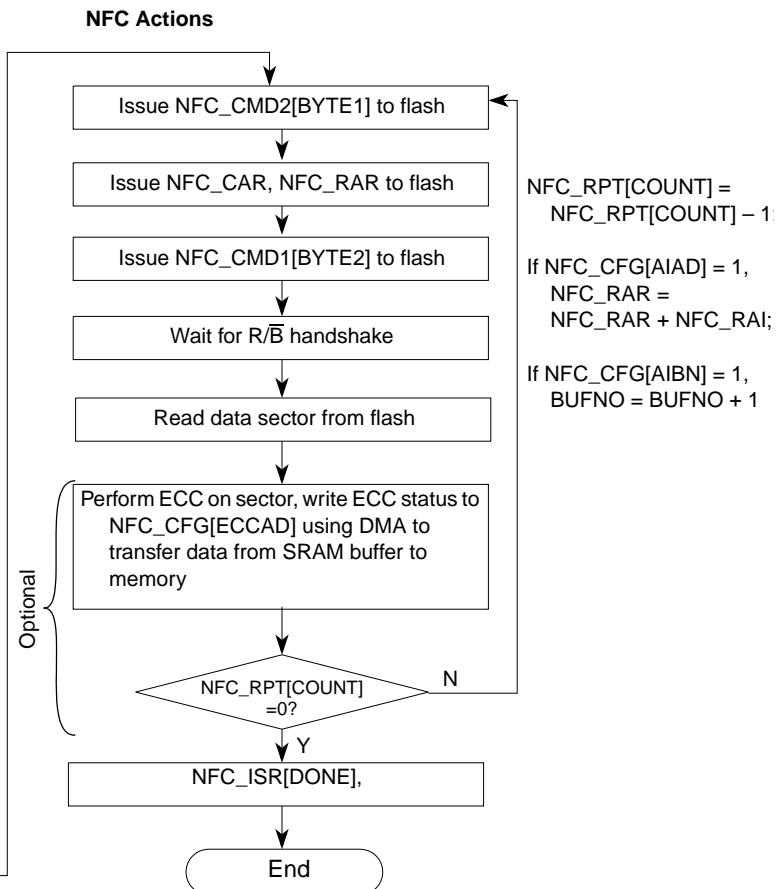
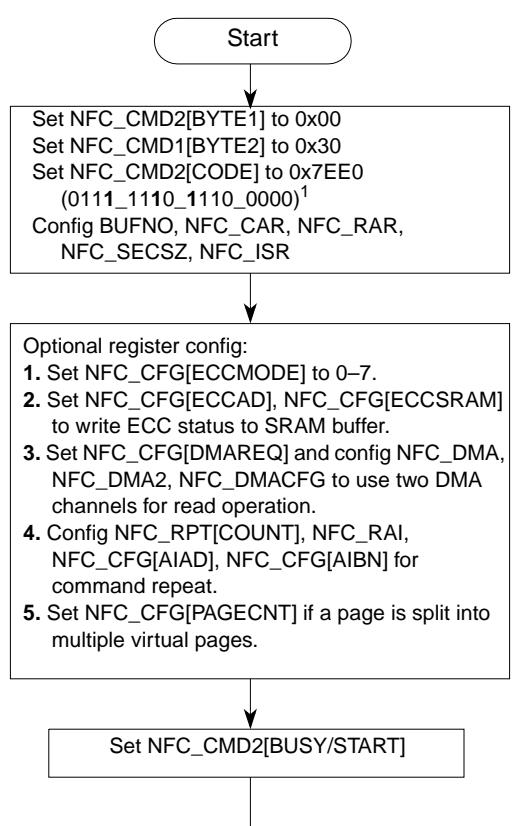
Field	Definition
7 CORFAIL	0 Page has been successfully corrected 1 Page is uncorrectable
5–0 ERROR_COUNT	Number of errors that have been corrected in this page

## 22.4.3 NFC Basic Commands

### 22.4.3.1 Page Read

This command reads pages from the NAND flash. Figure 22-18 is the general flow chart of read operation.

#### Register Config



#### Note:

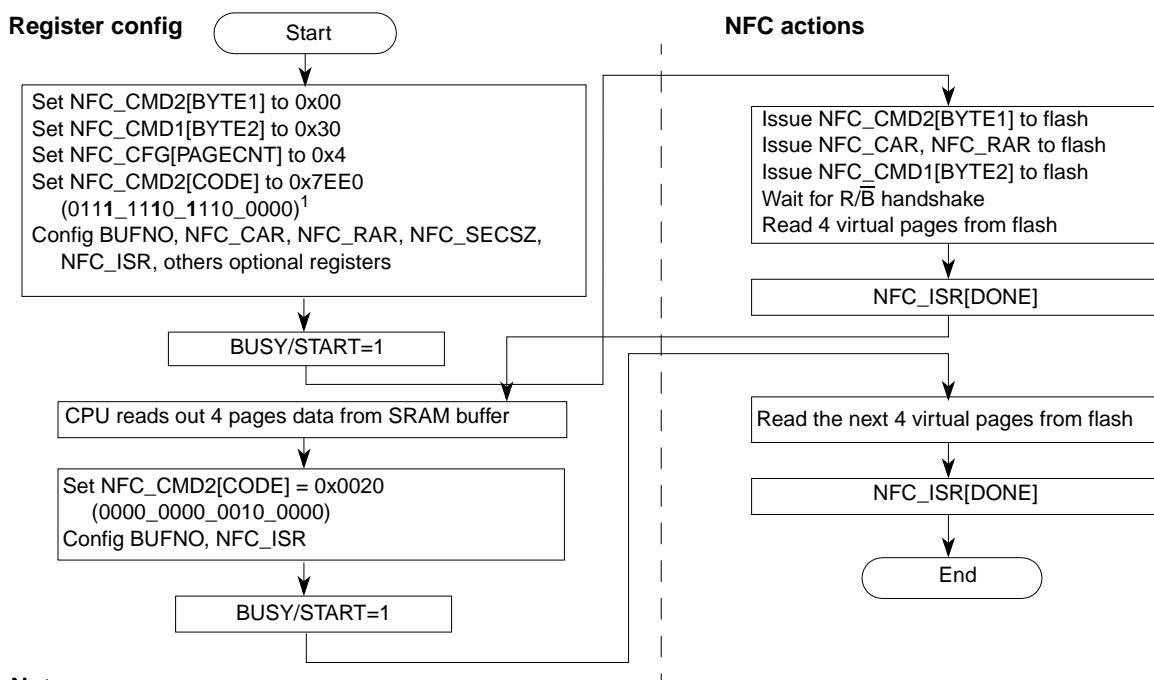
- <sup>1</sup> COL\_ADDR2, NFC\_RAR[BYTE3], and NFC\_CMD1[BYTE2] (bold) are not necessary for some flash devices. See their data sheets for detail. For example:  
 If the flash only has one column address, then NFC\_CMD2[CODE] = 0110\_1110\_1110\_0000;  
 If the flash only has two row addresses, then NFC\_CMD2[CODE] = 0111\_1100\_1110\_0000;  
 If flash does not need the second command 0x30 for read, then NFC\_CMD2[CODE] = 0110\_1110\_0110\_0000.

Figure 22-18. Flow Chart of Read Operation

Figure 22-19 shows a particular case: one page is split into 8 virtual pages (see [Section 22.4.6, “Organization of the Data in the NAND Flash”](#)), and DMA is not used. The SRAM buffer can hold data for four (virtual) pages at most. The CPU must transfer data out of the SRAM buffer after the first four virtual pages are read from flash. Otherwise, the next four virtual pages data overwrite the buffer. So, the read operation has following steps:

- Configure registers as shown in [Figure 22-18](#). NFC\_CFG[PAGECNT] = 4, start commands, wait for NFC\_ISR[DONE]
- CPU reads data from buffer, set NFC\_CMD2[CODE] = 0x20 (only enable read data)
- Start commands to read out the next 4 virtual pages, wait for NFC\_ISR[DONE]

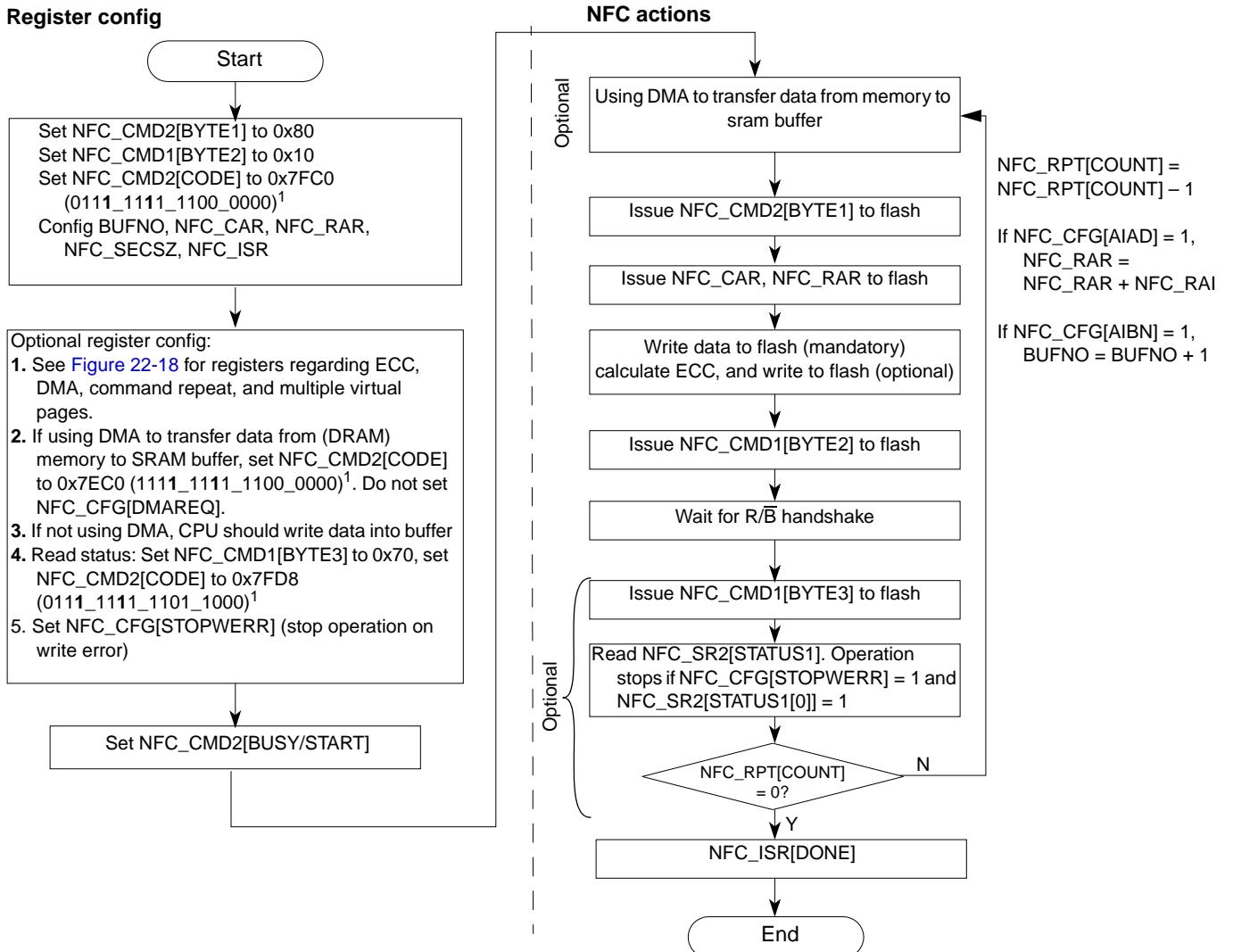
If DMA is used to transfer data from SRAM buffer to memory instead of CPU, the flow in [Figure 22-18](#) is used: set NFC\_CFG[PAGECNT] = 8, set NFC\_CFG[DMAREQ] = 1, configure DMA registers, start commands. A pipeline ([Section 22.4, “Functional Description”](#)) controls the read operation.



**Figure 22-19. Flow Chart of Read Operation, NFC\_CFG[PAGECNT] = 8, No DMA**

### 22.4.3.2 Page Program

This command programs pages to the NAND flash. [Figure 22-20](#) is the general flow of page program operation.



**Note:**

<sup>1</sup> COL\_ADDR2 and NFC\_RAR[BYTE3] (bold) are not necessary for some flash devices. See their data sheets for detail. See the footnote of [Figure 22-18](#).

**Figure 22-20. Flow Chart of Page Program Operation**

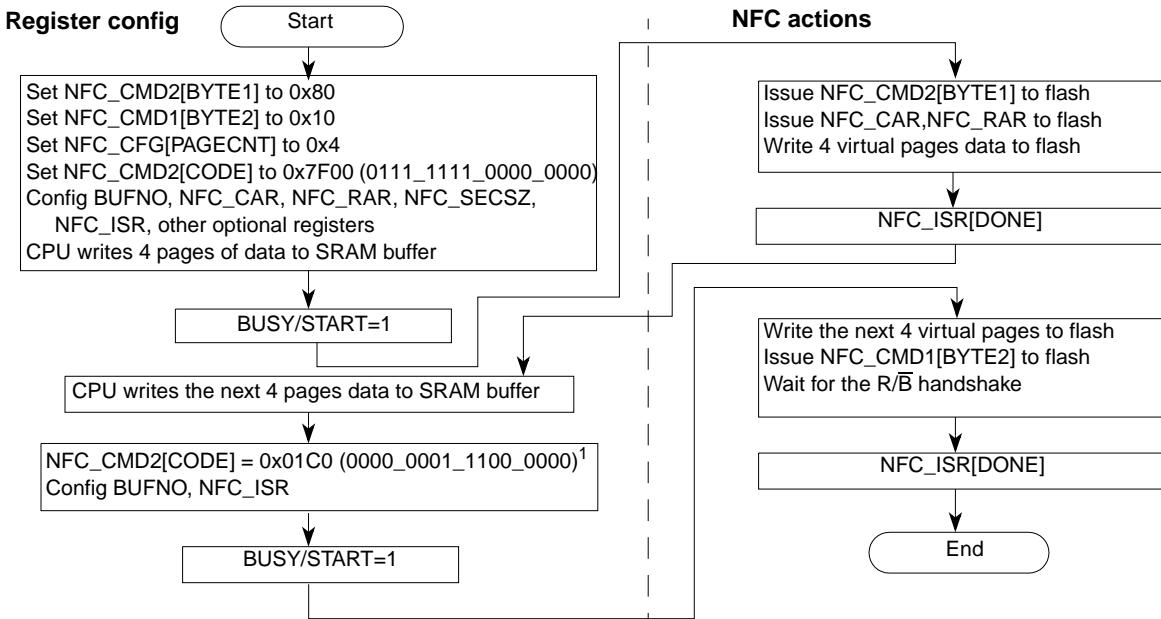
[Figure 22-21](#) is the particular case which is similar as [Figure 22-19](#). The CPU writes at most four virtual pages of data into the buffer before the first start command. Set NFC\_CFG[PAGECNT] to 4 and set NFC\_CMD2[CODE] twice:

- First, set it to 0x7F00 (0111\_1111\_0000\_0000). The NFC issues NFC\_CMD2[BYTE1], address cycles, four virtual pages data to flash. After NFC\_ISR[DONE] is set, the CPU can write the next four virtual pages data into the SRAM buffer.

## NAND Flash Controller (NFC)

- Second, set CODE to 0x01C0 (0000\_0001\_1100\_0000). The NFC sends the next four virtual pages of data to flash, issues NFC\_CMD1[BYTE2], waits for R/B handshake, and waits for NFC\_ISR[DONE] to set.

Like the read operation, if DMA transfers data from memory to NFC SRAM buffer (instead of the CPU), the flow in [Figure 22-20](#) is used and set NFC\_CFG[PAGECNT] to 0x8.



### Note:

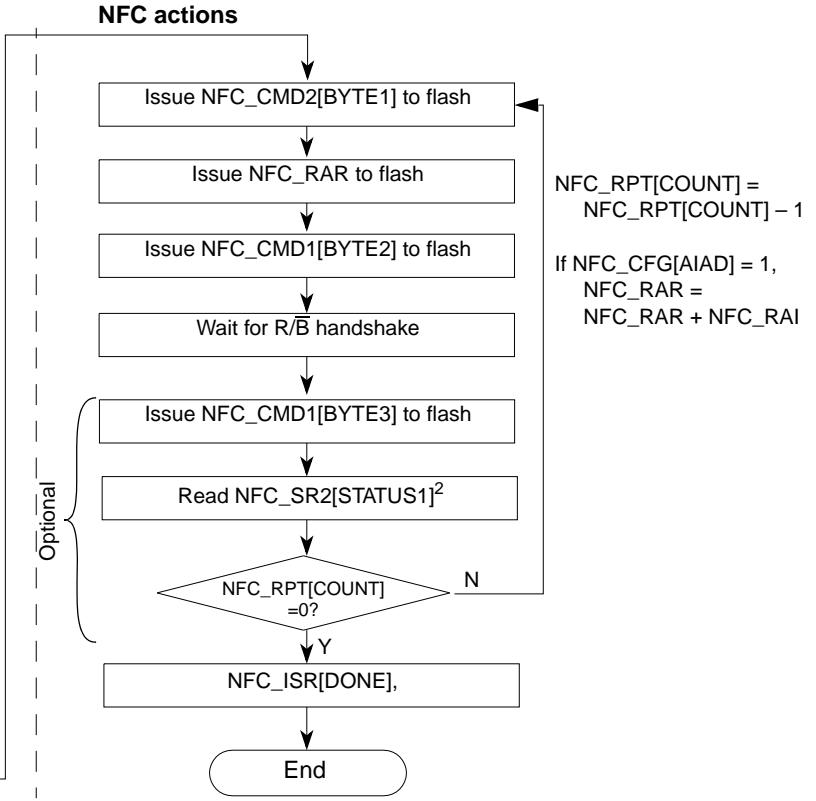
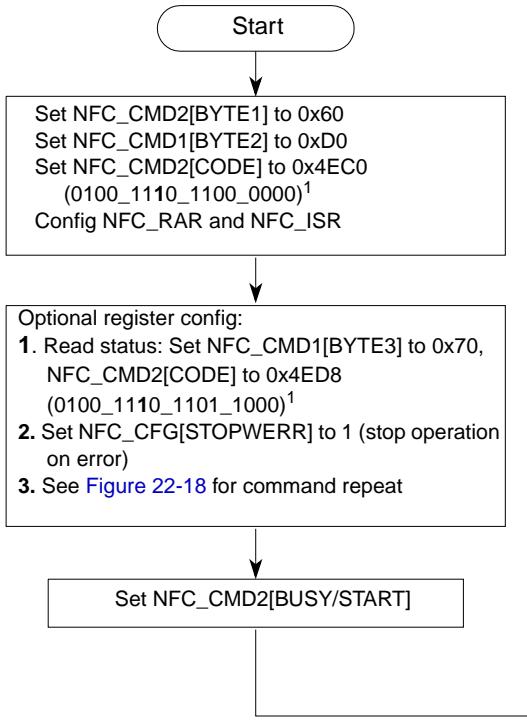
- If you want to read the status after the second 0x10 command, set NFC\_CMD1[BYTE3] to 0x70 and NFC\_CMD2[CODE] to 0x01D8 (0000\_0001\_1101\_1000). Then, after "Wait for the R/B handshake", the NFC issues NFC\_CMD1[BYTE3] to flash, and reads the status. If NFC\_CFG[STOPWERR] is set and NFC\_SR2[STATUS1[0]]=1, operation stops. Otherwise, NFC\_ISR[DONE] comes out. The COL\_ADDR2 and NFC\_RAR[BYTE3] of the first NFC\_CMD2[CODE] may not be necessary. See note 1 of [Figure 22-18](#).

**Figure 22-21. Flow Chart of Page Program Operation, NFC\_CFG[PAGECNT] = 8, No DMA**

### 22.4.3.3 Block Erase

This command is used to erase blocks.

#### Register config



#### Note:

<sup>1</sup> NFC\_RAR[BYTE3] (bold) is not necessary for some flash devices. See their data sheets for detail.

<sup>2</sup> If NFC\_CFG[STOPWERR] is set and NFC\_SR2[STATUS1[0]]=1, operation stops.

Figure 22-22. Flow Chart of Block Erase Operation

#### 22.4.3.4 Read ID

This command reads the flash ID.

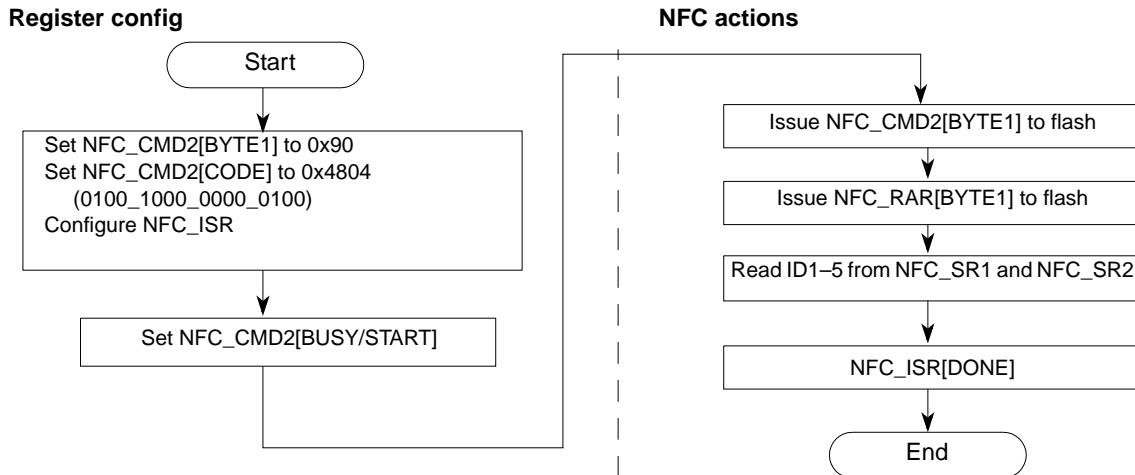


Figure 22-23. Flow Chart of Read ID Operation

#### 22.4.3.5 Reset

This command sends a single reset command to the flash.

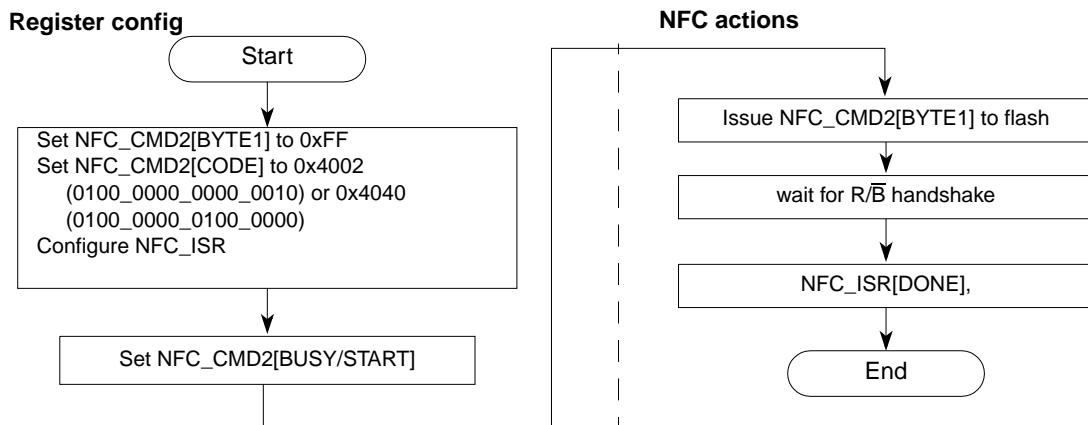


Figure 22-24. Flow Chart of Reset Operation

#### 22.4.4 NAND Flash Boot

For booting, the power-on reset values of some registers are:

- Sector size is 1056 bytes: NFC\_SECSZ = 1056
- Flash is defined as 8-bit: NFC\_CFG[16BIT] = 0
- ECC correction depth is 32 bits errors: NFC\_CFG[ECCMODE] = 0x7
- Boot sector address is 0: NFC\_CMD2[BUFNO] = 00

Boot-up occurs after power-on reset. After boot, the following happens:

1. The NFC issues reset command 0xFF to the flash.
2. Four boot blocks are identified in the flash at row addresses 0, 256, 512, and 768.
3. The flash controller burst reads four pages from the first boot block to memory. A total of 4 KB are read in this way.
4. If there is no ECC failure during the burst read, the boot is successful.  
If there is an ECC failure during the burst read, the process is started again from the next boot block. If the fourth block still has ECC failure, boot is unsuccessful, which means it is not possible to read a reliable boot image from the flash. The processor begins instruction execution using the default boot memory mapped to the FlexBus.
5. CPU access is held until boot completion.
6. After boot, the boot image is visible in the memory map at address 0x00 – 0xF8F. A special hash function is active on the NFC SRAM read to have the boot image in one continuous address range, and not in four address ranges (one for each page). The NFC\_CFG[BTMD] bit controls this hash function, and this bit should be cleared after the CPU has read/executed the boot image, and before it operates the NFC in standard mode. See [Figure 22-25](#).

#### NOTE

There is no difference in how data is transferred between SRAM buffer and the flash: one page uses one buffer. But, how the CPU writes/reads the buffer is different. In non-boot mode, the write/read address is the SRAM physical address. In boot mode, the write/read address is based on the buffers. See [Section 22.4.1, “NFC Buffer Memory Space”](#).

Suppose the boot code is U0, U1, U2, U3, ... each an 8-byte data. The flash device data width is 8-bit.

Data organization in the NAND flash is:

Page0: U0, U4, U8, U12, ...; Page1: U1, U5, U9, U13, ...; Page2: U2, U6, U10, U14, ...; Page3: U3, U7, U11, U15, ...

Page	0	1	2	3
U0	U1	U2	U3	
U4	U5	U6	U7	
U8	U9	U10	U11	
U12	U13	U14	U15	
...	...	...	...	...

Physical address:  
0x00 – 1F    0x20 – 3F    0x40 – 5F    0x60 – 7F    ...

Buffer	0	1	2	3
	U0	U1	U2	U3
	U4	U5	U6	U7
	U8	U9	U10	U11
	U12	U13	U14	U15
	...	...	...	...

When boot starts, NFC\_CFG[BTMD] is set. After boot completes, if the CPU reads data from the 0x0000 of the SRAM buffer, the data is U0, U1, U2, U3, U4, U5, U6, U7, ...

If NFC\_CFG[BTMD] is cleared (non-boot mode), the CPU reads

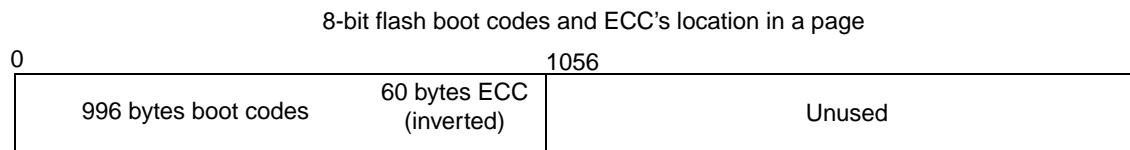
- U0, U4, U8, U12, ... from address 0x0000 (buffer0),
- U1, U5, U9, U13, ... from address 0x1000 (buffer1),
- U2, U6, U10, U14, ... from address 0x2000 (buffer2),
- U3, U7, U11, U15, ... from address 0x3000 (buffer3).

**Figure 22-25. Boot and NFC\_CFG[BTMD]**

The CPU boot code should be split into four pages as shown in [Figure 22-25](#). Boot sector encoding cannot be done by the NFC block itself. It must be done in software. Each page contains 996-byte boot codes and 60-byte ECC codes.

The ECC codes should be inverted because when NFC is doing a page program, it sends the inverted ECC codes to flash. When reading, it inverts them again to get the correct ECC codes for error correction. And only page reads are performed in boot operation. So, software should do the ECC code inversion after boot sector encoding.

For an 8-bit flash, the upper 1 KB of data is not read.



**Figure 22-26. Boot Data's Location in a Page**

## 22.4.5 Fast Flash Configuration for EDO

Normally, read out data goes valid after the high-to-low transition of  $\overline{\text{RE}}$ , and invalid on the low-to-high transition ([Figure 22-27](#))  $t_{RHOH} < t_{REH}$ . NFC sampled the read data at the negedge of `flash_clk`, and because the data is invalid at that time, a latch is used here to maintain the valid data during the high period of `flash_clk`, so that NFC can sample correct data.

Some flash devices contain a EDO (enhanced data out) feature, where the data can be held until the next high-to-low  $\overline{\text{RE}}$  transition ([Figure 22-28](#)),  $t_{RHOH} > t_{REH}$ . The read data is valid at the negedge of `flash_clk`, NFC can sample data directly without latching it. To support the EDO feature, the NFC must work in fast mode (NFC\_CFG[FAST] set). The NFC clock from the clock module must be configured fast enough (usually  $> 33\text{MHz}$ ) according to the data sheet of flash devices. The NFC clock frequency is determined by the PLL\_DR[OUTDIV5] bitfield. See [Section 8.2.2, “PLL Divider Register \(PLL\\_DR\)”](#), for details.

\* **Serial access Cycle after Read**(CLE=L,  $\overline{WE}=H$ , ALE=L)

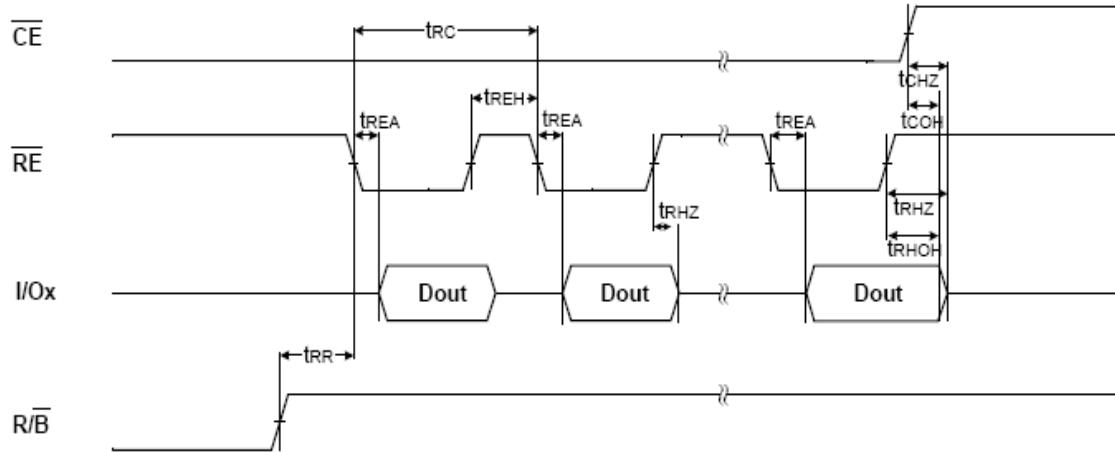


Figure 22-27. Read Operation

**Serial Access Cycle after Read**(EDO Type, CLE=L,  $\overline{WE}=H$ , ALE=L)

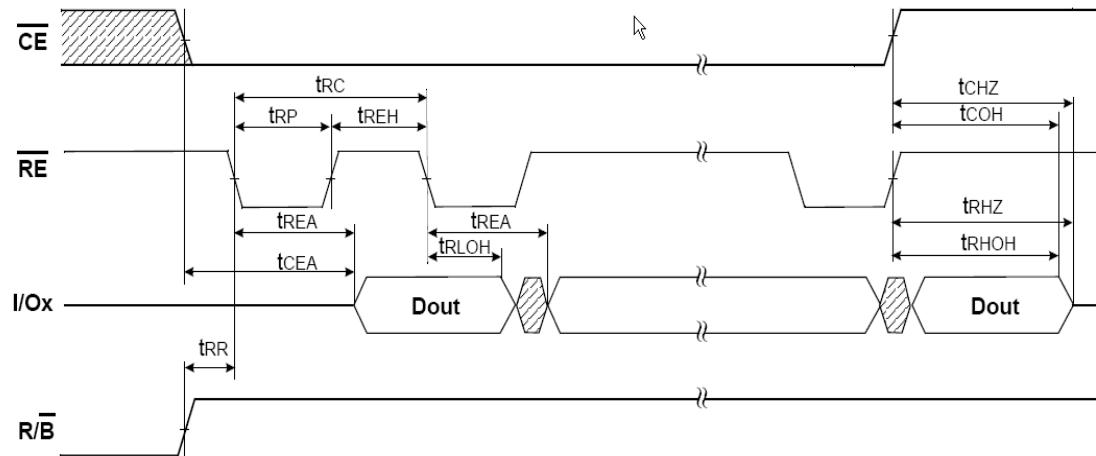


Figure 22-28. Read Operation, EDO type

## 22.4.6 Organization of the Data in the NAND Flash

Pages on the flash can be split into multiple virtual ECC/DMA pages. The parameter that controls this is NFC\_CFG[PAGECNT]. This parameter gives the number of virtual ECC/DMA pages in one flash page. See [Section 22.3.14, “Flash Configuration Register \(NFC\\_CFG\)”](#) for more details.

The virtual page is split into a user (main) area and ECC (spare) area. Data in the user area can be set or used by the application, while data in the ECC area is set and used by the ECC.

## NAND Flash Controller (NFC)

The following tables give virtual-to-physical mappings for various flash devices and their recommended settings.

**Table 22-20. Virtual-to-Physical Mappings of Different Flash<sup>1,2</sup>**

Flash page size (main+spare) bytes	NFC_CFG [ECC MODE]	ECC bits	NFC_CFG [PAGE CNT]	Sector size (bytes)	Virtual page user size (bytes)	Mapping
512 + 16	000	0	1	528	528	VirtualPage_0[527:0] = Physical[527:0]
512 + 16	001	4	1	528	520	VirtualPage_0[519:0] = Physical[519:0]
2048 + 64	000	0	1	2112	2112	VirtualPage_0[2111:0] = Physical[2111:0]
2048 + 64	101	16	1	2112	2082	VirtualPage_0[2081:0] = Physical[2081:0]
2048 + 64	110	24	1	2112	2067	VirtualPage_0[2066:0] = Physical[2066:0]
2048 + 64	111	32	1	2112	2052	VirtualPage_0[2051:0] = Physical[2051:0]
2048 + 64	000	0	4	528	528	VirtualPage_0[527:0] = Physical[527:0] VirtualPage_1[527:0] = Physical[1055:528] VirtualPage_2[527:0] = Physical[1583:1056] VirtualPage_3[527:0] = Physical[2111:1584]
2048 + 64	001	4	4	528	520	VirtualPage_0[519:0] = Physical[519:0] VirtualPage_1[519:0] = Physical[1047:528] VirtualPage_2[519:0] = Physical[1575:1056] VirtualPage_3[519:0] = Physical[2103:1584]
4096 + 128	000	0	2	2112	2112	VirtualPage_0[2111:0] = Physical[2111:0] VirtualPage_1[2111:0] = Physical[4223:2112]
4096 + 128	101	16	2	2112	2082	VirtualPage_0[2081:0] = Physical[2081:0] VirtualPage_1[2081:0] = Physical[4193:2112]
4096 + 128	110	24	2	2112	2067	VirtualPage_0[2066:0] = Physical[2066:0] VirtualPage_1[2066:0] = Physical[4178:2112]
4096 + 128	111	32	2	2112	2052	VirtualPage_0[2051:0] = Physical[2051:0] <sup>3</sup> VirtualPage_1[2051:0] = Physical[4163:2112]
4096 + 128	000	0	8 <sup>4</sup>	528	528	VirtualPage_0[527:0] = Physical[527:0] VirtualPage_1[527:0] = Physical[1055:528] VirtualPage_2[527:0] = Physical[1583:1056] VirtualPage_3[527:0] = Physical[2111:1584] VirtualPage_4[527:0] = Physical[2639:2112] VirtualPage_5[527:0] = Physical[3167:2640] VirtualPage_6[527:0] = Physical[3695:3168] VirtualPage_7[527:0] = Physical[4223:3696]
4096 + 128	001	4	8	528	520	VirtualPage_0[519:0] = Physical[519:0] VirtualPage_1[519:0] = Physical[1047:528] VirtualPage_2[519:0] = Physical[1575:1056] VirtualPage_3[519:0] = Physical[2103:1584] VirtualPage_4[519:0] = Physical[2631:2112] VirtualPage_5[519:0] = Physical[3159:2640] VirtualPage_6[519:0] = Physical[3687:3168] VirtualPage_7[519:0] = Physical[4215:3696]

**Table 22-20. Virtual-to-Physical Mappings of Different Flash<sup>1,2</sup> (continued)**

<b>Flash page size (main+spare) bytes</b>	<b>NFC_CFG [ECC MODE]</b>	<b>ECC bits</b>	<b>NFC_CFG [PAGE CNT]</b>	<b>Sector size (bytes)</b>	<b>Virtual page user size (bytes)</b>	<b>Mapping</b>
4096 + 208	000	0	2	2152	2152	VirtualPage_0[2151:0] = Physical[2151:0] VirtualPage_1[2151:0] = Physical[4303:2152]
4096 + 208	101	16	2	2152	2122	VirtualPage_0[2121:0] = Physical[2121:0] VirtualPage_1[2121:0] = Physical[4273:2152]
4096 + 208	110	24	2	2152	2104	VirtualPage_0[2103:0] = Physical[2103:0] VirtualPage_1[2103:0] = Physical[4255:2152]
4096 + 208	111	32	2	2152	2092	VirtualPage_0[2091:0] = Physical[2091:0] VirtualPage_1[2091:0] = Physical[4243:2152]
4096 + 208	000	0	8	538	538	VirtualPage_0[537:0] = Physical[537:0] VirtualPage_1[537:0] = Physical[1075:538] VirtualPage_2[537:0] = Physical[1613:1076] VirtualPage_3[537:0] = Physical[2151:1614] VirtualPage_4[537:0] = Physical[2689:2152] VirtualPage_5[537:0] = Physical[3227:2690] VirtualPage_6[537:0] = Physical[3765:3228] VirtualPage_7[537:0] = Physical[4304:3766]
4096 + 208	001	4	8	538	530	VirtualPage_0[529:0] = Physical[529:0] VirtualPage_1[529:0] = Physical[1067:538] VirtualPage_2[529:0] = Physical[1605:1076] VirtualPage_3[529:0] = Physical[2143:1614] VirtualPage_4[529:0] = Physical[2681:2152] VirtualPage_5[529:0] = Physical[3219:2690] VirtualPage_6[529:0] = Physical[3757:3228] VirtualPage_7[529:0] = Physical[4295:3766]
4096 + 208	010	6	8	538	526	VirtualPage_0[525:0] = Physical[525:0] VirtualPage_1[525:0] = Physical[1063:538] VirtualPage_2[525:0] = Physical[1601:1076] VirtualPage_3[525:0] = Physical[2139:1614] VirtualPage_4[525:0] = Physical[2677:2152] VirtualPage_5[525:0] = Physical[3215:2690] VirtualPage_6[525:0] = Physical[3753:3228] VirtualPage_7[525:0] = Physical[4291:3766]
4096 + 208	011	8	8	538	523	VirtualPage_0[522:0] = Physical[523:0] VirtualPage_1[522:0] = Physical[1060:538] VirtualPage_2[522:0] = Physical[1598:1076] VirtualPage_3[522:0] = Physical[2136:1614] VirtualPage_4[522:0] = Physical[2674:2152] VirtualPage_5[522:0] = Physical[3212:2690] VirtualPage_6[522:0] = Physical[3750:3228] VirtualPage_7[522:0] = Physical[4288:3766]

<sup>1</sup> Not all settings are shown. It's also possible to split 2K and 4K sectors into 1KB units, which is not shown. For flash devices with more spare bytes, it is possible to correct more errors, at a higher ECC byte count cost.

## NAND Flash Controller (NFC)

- <sup>2</sup> It is possible to split 2 KB and 4 KB pages into virtual pages of 512, 1 KB or 2 KB. The best error protection is achieved with the larger page sizes. For example, splitting a 4 KB page in two 2 KB pages, with 32-bit error correction yields lower uncorrectable error probability than splitting the same page in eight 512-byte pages with 8 bit error correction. For OS compatibility, it may be necessary to use 512 byte pages, however.
- <sup>3</sup> In most applications, this mode is of no use because user size is too small.
- <sup>4</sup> When 4KB page is split into eight virtual pages, if page program/read using DMA, set NFC\_CFG[PAGECNT] to 8. if not using DMA, set NFC\_CFG[PAGECNT] to 4. See [Section 22.4.3.1, “Page Read”](#) and [Section 22.4.3.2, “Page Program”](#) for details.

If flash devices with a physical page size of 4K or more are used, the bad block marker appears as the first byte of the spare area. But, because of the physical-to-virtual mapping, it does not appear in byte 2048 of the virtual page, where its logical place would be. The DMA engine contains the option to swap some bytes, and to make the bad block marker appear in the requested place.

**Table 22-21. Using the Swap Field to Move the Bad Block Marker**

Flash sector size (main + spare) bytes	Bad block marker (physical)	Bad block marker (virtual) Before swap	Bad block marker (expected) After swap	Swap
4096 + 128	4096	Page 1/byte 1984	Page 1/byte 2048	NFC_SWAP[ADDR1] = (1984/8) <sup>1</sup> NFC_SWAP[ADDR2] = (2048/8)
4096 + 208	4096	Page 1/byte 1944	Page 1/byte 2048	NFC_SWAP[ADDR1] = (1944/8) NFC_SWAP[ADDR2] = (2048/8)

<sup>1</sup> Only works with a user page size of at least 2055 bytes. Does not work with ECC mode 111.

## 22.4.7 Flash Command Code Description

The 16-bit command code in NFC\_CMD2[CODE] is defined in [Table 22-22](#). If a bit is set, the action is executed. The command is repeated for the number of the NFC\_RPT[COUNT] value. If NFC\_RPT[COUNT] is zero or one, the command is executed once.

**Table 22-22. NFC\_CMD2[CODE] Detail**

NFC_CMD2 [CODE] bit	Action when Bit is Set
15	Start DMA transfer to read data from memory , and write to SRAM.
14	Send command byte 1 (NFC_CMD2[BYTE1]) to flash
13	Send column address 1 (COL_ADDR1) to flash
12	Send column address 2 (COL_ADDR2) to flash
11	Send row address 1 (NFC_RAR[BYTE1]) to flash
10	Send row address 2 (NFC_RAR[BYTE2]) to flash
9	Send row address 3 (NFC_RAR[BYTE3]) to flash
8	Write data to flash. Total of NFC_CFG[PAGECNT] pages is written to the flash, and equal number of starts is sent to the residue engine. Also, additional starts to the DMA engine are sent, until DMA has transferred the NFC_CFG[PAGECNT] data from memory to NFC.
7	Send command byte 2 (NFC_CMD1[BYTE2]) to flash

**Table 22-22. NFC\_CMD2[CODE] Detail (continued)**

NFC_CMD2 [CODE] bit	Action when Bit is Set
6	Wait for flash R/B handshake
5	Read data from flash. Read is only started if the new NFC_CMD2[BUFNO] is idle. One or more starts are sent to the residue engine, total NFC_CFG[PAGECNT] starts. <b>Note:</b> For reads, DMA is not started. Instead, to start DMA for reads, NFC_CFG[DMAREQ] must be set.
4	Send command byte 3 (NFC_CMD1[BYTE3]) to flash
3	Read flash status
2	Read ID
1	Always set. End-of-command marker used to signal done.
0	Reserved, must be cleared.

## 22.4.8 Interrupts

There are two interrupts to flag the end of a command execution:

1. The done interrupt, NFC\_ISR[DONE]. Use this interrupt if commands are sent back-to-back to the flash. It indicates when a new command can be dispatched. The done interrupt is given before the flash data is corrected and resident in memory, because operation of the ECC engine and DMA engine is pipelined.  
When the done interrupt is tracks command completion, the software may also monitor the NFC\_ISR[ECCBUSY, DMABUSY, ECCBN, DMABN] fields.
  - a) NFC\_ISR[ECCBUSY] indicates that the ECC block is still busy, and reports the buffer number the ECC block is working on in NFC\_ISR[ECCBN].
  - b) NFC\_ISR[DMABUSY] indicates that the DMA block is still busy, and reports the buffer number the DMA block is working on in NFC\_ISR[DMABN].
2. The command idle interrupt, NFC\_ISR[IDLE]. Use this interrupt if you want to use the data produced in the next process. The idle interrupt indicates all command processing has terminated, and the relevant data is now available in memory or the NFC SRAM buffer. When using back-to-back reads to the flash, use of the idle interrupt means the NFC does not operate at its maximum transfer speed, as ECC and DMA are now done in foreground.

When using the done interrupt, transfer completion for write pages can be assumed when the done interrupt is received. When done is received for read pages, the data may still be in flight in the DMA or the ECC. To check this, the CPU should remember the buffer number (NFC\_CMD2[BUFNO]) associated with the command, and wait until the DMA and ECC are either idle, or are both busy on a different buffer number. (The ECC buffer number and DMA buffer number fields do not match the BUFNO specified with command.) You can check on any done interrupt or by polling the register.



# Chapter 23

## Universal Serial Bus Interface – Host Module

This chapter describes the universal serial bus (USB) host module, which implements many industry standards. However, it is beyond the scope of this document to document the intricacies of these standards. Instead, it is left to the reader to refer to the governing specifications. Readers of this chapter are assumed to be fluent in the operation and requirements of a USB network.

The following documents are available from the USB Implementers Forum web page at <http://www.usb.org/developers/docs>:

- *Universal Serial Bus Specification, Revision 2.0*

The following documents are available from the Intel USB Specifications web page at <http://www.intel.com/technology/usb/spec.htm>:

- *Enhanced Host Controller Interface (EHCI) Specification for Universal Serial Bus, Revision 1.0*

### 23.1 Introduction

The processor implements two USB modules: a host module and an On-The-Go (OTG) module. The host and OTG modules can be used with a full-speed/low-speed on-chip transceiver or with an external ULPI transceiver. For more details on the USB OTG module, refer to [Chapter 24, “Universal Serial Bus Interface – On-The-Go Module.”](#)

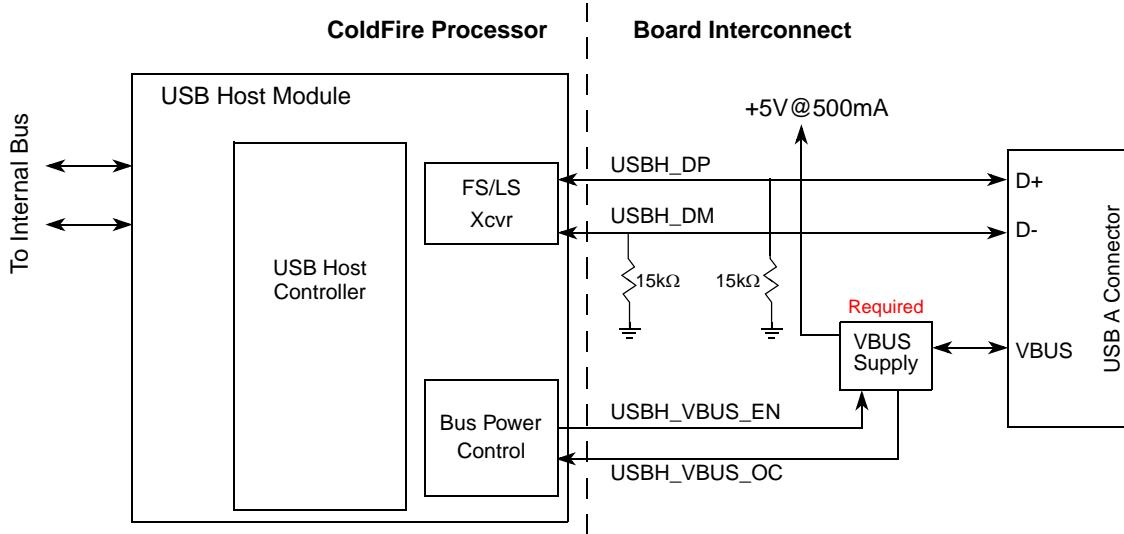
USB host modules must supply 500 mA with a 5 V supply on its downstream port (referred to as VBUS). If the connected device attempts to draw more than the allocated amount of current, the USB host must disable the port and remove power. This processor provides pins for control and status to an external IC capable of managing the VBUS downstream supply.

Register and data structure interfaces conform to the EHCI specification from Intel Corporation, with enhancements to support the embedded environment. The USB controller contains its own DMA (direct memory access) engines that reduce interrupt load on the application processor, and thereby reduce total system bus bandwidth dedicated to servicing the USB interface requirements. The USB controller includes logic to support USB’s low-power suspend features, and for suspended devices to request remote wakeup from the host.

This USB host controller hides all direct interaction with the protocol, but some knowledge of the USB is required to properly configure the device for operation on the local bus and on the USB. This document covers programming requirements, and additional information may be found in the USB specification.

### 23.1.1 Block Diagram

The USB host module is shown below in Figure 23-1.



**Figure 23-1. USB Host Interface Block Diagram**

### 23.1.2 Overview

The USB host module is a USB 2.0-compliant serial interface engine for implementing a USB interface. The registers and data structures for both modules are based on the *Enhanced Host Controller Interface Specification for Universal Serial Bus* from Intel Corporation. The USB controller supports directly connected full- and low-speed peripherals without the need for UHCI or OHCI companion controllers.

The USB host module interfaces to the processor's ColdFire core. The USB controller is programmable to support full-speed (12 Mbps) and low-speed (1.5 Mbps) applications using the on-chip transceiver. The processor's on-chip PLL provides all necessary clocks to the USB controller. For special applications, pin access (via USBCLKIN) is provided for an external USB reference clock (60MHz).

The USB host controller provides control and status signals to interface with external USB host power devices. Use these control and status signals on the chip interface to communicate with external USB host power solutions. USB VBUS is not provided on-chip.

The on-chip FS/LS transceiver does not include USB DP and DM bias resistors. The primary function of the transceiver is the physical signal conditioning of the external USB DP and DM cable signals for a USB 2.0 network. Several USB system elements are not supported on this device as they are available via a standard product from various manufacturers.

### 23.1.3 Features

The USB host module includes the following features:

- Complies with USB specification revision 2.0

- Supports operation as a standalone USB host controller
  - Supports enhanced host controller interface (EHCI)
  - Allows direct connection of full-speed (FS) or low-speed (LS) devices without an OHCI/UHCI companion controller
  - Supported by Linux and other commercially available operating systems
- Includes on-chip full-speed (12 Mbps), and low-speed (1.5 Mbps) transceiver
- Optional UTMI+ Low Pin Interface (ULPI) to support high speed (HS = 480 Mbps) transfers with external PHY, shared with USB OTG module (see MISCCR2[ULPI] in [Chapter 10, “Chip Configuration Module \(CCM\)”](#))
- Allows vendor to define any USB device or class for targeted peripheral list, including USB hubs
- Supports VBUS power enable and VBUS over-current detect to control bus power
- Registers provided for indicating VBUS state to controller
- Suspend mode/low power
  - As host, firmware can suspend individual devices or the entire USB and disable chip clocks for low-power operation
  - Device supports low-power suspend
  - Remote wake-up supported
  - Integrated with the processor’s doze and stop modes for ultra-low power operation

### 23.1.4 Modes of Operation

The USB host controller provides the functionality of one USB 2.0 host. The module is hardwired to an on-chip full-/low-speed transceiver. Speed selection is auto-detected at connect time via sensing of the DP or DM pullup resistor on the connected device using procedures of enumeration in the USB network. The USB host module provides the following modes of operation for the user:

- USB disabled. In this mode, the USB host datapath does not accept transactions received on the USB interface.
- USB enabled. In this mode, the USB host’s datapath is enabled to accept transactions received on the USB interface.
- USB enabled, low power modes. See [Section 23.1.4.1, “Low-Power Modes,”](#) for details.

#### 23.1.4.1 Low-Power Modes

The USB host module is integrated with the Version 3 ColdFire core’s low-power modes (stop, doze and wait). The modes are implemented as follows:

- Stop — The processor stops the clock to the USB host module. In this state, the module ignores traffic on the USB network and does not generate any interrupts or wake-up events. The on-chip transceiver is disabled to save power.
- Wait — The clocks to the USB host module are running.

- Doze — The processor stops the system clocks to the USB host module. However, the 60 MHz transceiver clock remains active. Detection of resume signaling initiates a restart of the module clocks.

## 23.2 External Signal Description

[Table 23-1](#) describes the external signals of the USB host module.

**Table 23-1. USB Host External Signals**

Signal	I/O	Description
USB_CLKIN	I	Optional 60 MHz clock source.
USBH_DM	I/O	D- output of the dual-speed transceiver for the USB Host module.
		<b>State Meaning</b> Asserted—Data 1 Negated—Data 0
		<b>Timing</b> Asynchronous
USBH_DP	I/O	D+ output of the dual-speed transceiver for the USB Host module.
		<b>State Meaning</b> Asserted—Data 1 Negated—Data 0
		<b>Timing</b> Asynchronous
USBH_VBUS_EN	O	Enables off-chip charge pump controller of VBUS for the USB host module.
		<b>State Meaning</b> Asserted—Off-chip charge pump controller enabled Negated—Off-chip charge pump controller disabled
		<b>Timing</b> Asynchronous
USBH_VBUS_OC	I	Communicates short on USB lines occurred for USB host module.
		<b>State Meaning</b> Asserted—Short detected Negated—No short detected
		<b>Timing</b> Synchronous to USB_CLKIN.

### 23.2.1 USB Host Control and Status Signals

To minimize the pin-count on the device, a few USB host control and status signals are implemented on-chip as bit fields in a register within the chip configuration module (CCM).

The host controller status register (UHCSR) is implemented as follows:

- Writes to the UHCSR register from the firmware set the corresponding bits on the USB interface.
- When the USB host module outputs change, the corresponding bits on the UHCSR register are updated, and a maskable interrupt is generated.

The UHCSR register is documented in the CCM chapter, see [Chapter 10, “Chip Configuration Module \(CCM\).”](#)

**Table 23-2. Internal Control and Status Bits for USB Host Module**

<b>Signal</b>	<b>Mnemonic</b>	<b>Direction</b>	<b>Access</b>	<b>Interrupt Trigger?</b>
Port Indicator LED Control	PORTIND[1:0]	Selects LED color for product package.	R	N
Wake-up Event	WKUP	Reflects when a wake-up event has occurred on the USB bus.	R/W	Y
Drive VBUS	DRVVBUS	Enables drive of the 5 V power on VBUS	R/W	N
VBUS Power Fault	PWRFLT	Indicates a power fault occurred on VBUS (overcurrent)	R/W	N
Interrupt Mask	UHMIE	Interrupt enable. When set, changes on WKUP cause an interrupt to be asserted. When cleared, the interrupt is masked.	R/W	N/A
On-chip Transceiver Pull-down Enable	XPDE	Enables 50 kΩ pull-downs on the host controller's DM and DP pins	R/W	N

### 23.3 Memory Map/Register Definitions

This section provides the memory map of the USB host module. Descriptions of these registers can be found in [Chapter 24, “Universal Serial Bus Interface – On-The-Go Module.”](#) See the Section/Page heading in the below table for direct links to the corresponding register description. Addresses and reset values shown here are correct for the USB host module.

**Table 23-3. USB Host Controller Memory Map**

<b>Address</b>	<b>Register</b>	<b>EHCI<sup>1</sup></b>	<b>Width (bits)</b>	<b>Access</b>	<b>Reset</b>	<b>Section/Page</b>
<b>Module Identification Registers</b>						
0xFC0B_4000	Identification Register (ID)	N	32	R	0xE242_FA05	<a href="#">24.3.1.1/24-8</a>
0xFC0B_4004	General Hardware Parameters (HWGENERAL)	N	32	R	0x0000_07C5	<a href="#">24.3.1.2/24-9</a>
0xFC0B_4008	Host Hardware Parameters (HWHOST)	N	32	R	0x1002_0001	<a href="#">24.3.1.3/24-10</a>
0xFC0B_4010	TX Buffer Hardware Parameters (HWTXBUF)	N	32	R	0x8004_0404	<a href="#">24.3.1.5/24-11</a>
0xFC0B_4014	RX Buffer Hardware Parameters (HWRXBUF)	N	32	R	0x0000_0404	<a href="#">24.3.1.6/24-11</a>
<b>Timer Registers</b>						
0xFC0B_4080	General Purpose Timer 0 Load (GPTIMER0LD)	N	32	R/W	0x0000_0000	<a href="#">24.3.2.1/24-12</a>
0xFC0B_4084	General Purpose Timer 0 Control (GPTIMER0CTL)	N	32	R/W	0x0000_0000	<a href="#">24.3.2.2/24-12</a>
0xFC0B_4088	General Purpose Timer 1 Load (GPTIMER1LD)	N	32	R/W	0x0000_0000	<a href="#">24.3.2.1/24-12</a>
0xFC0B_408C	General Purpose Timer 1 Control (GPTIMER1CTL)	N	32	R/W	0x0000_0000	<a href="#">24.3.2.2/24-12</a>
<b>Capability Registers</b>						

**Table 23-3. USB Host Controller Memory Map (continued)**

Address	Register	EHCI <sup>1</sup>	Width (bits)	Access	Reset	Section/Page
0xFC0B_4100	Host Interface Version Number (HCIVERSION)	Y	16	R	0x0100	<a href="#">24.3.3.1/24-13</a>
0xFC0B_4103	Capability Register Length (CAPLENGTH)	Y	8	R	0x40	<a href="#">24.3.3.4/24-15</a>
0xFC0B_4104	Host Structural Parameters (HCSPARAMS)	Y	32	R	0x0001_0011	<a href="#">24.3.3.3/24-14</a>
0xFC0B_4108	Host Capability Parameters (HCCPARAMS)	Y	32	R	0x0000_0006	<a href="#">24.3.3.4/24-15</a>
<b>Operational Registers</b>						
0xFC0B_4140	USB Command (USBCMD)	Y	32	R/W	0x0008_0B00	<a href="#">24.3.4.1/24-17</a>
0xFC0B_4144	USB Status (USBSTS)	Y	32	R/W	0x0000_1080	<a href="#">24.3.4.2/24-19</a>
0xFC0B_4148	USB Interrupt Enable (USBINTR)	Y	32	R/W	0x0000_0000	<a href="#">24.3.4.3/24-22</a>
0xFC0B_414C	USB Frame Index (FRINDEX)	Y	32	R/W	0x0000_0000	<a href="#">24.3.4.4/24-24</a>
0xFC0B_4154	Periodic Frame List Base Address (PERIODICLISTBASE)	Y	32	R/W	0x0000_0000	<a href="#">24.3.4.5/24-25</a>
0xFC0B_4158	Current Asynchronous List Address (ASYNCLISTADDR)	Y	32	R/W	0x0000_0000	<a href="#">24.3.4.7/24-26</a>
0xFC0B_415C	Host TT Asynchronous Buffer Control (TTCTRL)	N	32	R/W	0x0000_0000	<a href="#">24.3.4.9/24-28</a>
0xFC0B_4160	Master Interface Data Burst Size (BURSTSIZE)	N	32	R/W	0x0000_1010	<a href="#">24.3.4.10/24-28</a>
0xFC0B_4164	Host Transmit FIFO Tuning Control (TXFILLTUNING)	N	32	R/W	0x0002_0000	<a href="#">24.3.4.11/24-28</a>
0xFC0B_4180	Configure Flag Register (CONFIGFLAG)	Y	32	R	0x0000_0001	<a href="#">24.3.4.13/24-32</a>
0xFC0B_4184	Port Status/Control (PORTSC1)	Y	32	R/W	0xEC00_0000	<a href="#">24.3.4.14/24-32</a>
0xFC0B_41A8	USB Mode Register (MODE)	N	32	R/W	0x0000_0003	<a href="#">24.3.4.16/24-39</a>

<sup>1</sup> Indicates if the register is present in the EHCI specification.

## 23.4 Functional Description

The USB host module’s functional description is very similar to the USB OTG module in host mode. See [Chapter 24, “Universal Serial Bus Interface – On-The-Go Module,”](#) and the *Enhanced Host Controller Interface (EHCI) Specification for Universal Serial Bus, Revision 1.0* for more information.

# Chapter 24

## Universal Serial Bus Interface – On-The-Go Module

### 24.1 Introduction

This chapter describes the universal serial bus (USB) interface, which implements many industry standards. However, it is beyond the scope of this document to document the intricacies of these standards. Instead, you should refer to the governing specifications. Readers of this chapter are assumed to be fluent in the operation and requirements of a USB network.

Visit the USB Implementers Forum web page at <http://www.usb.org/developers/docs> for:

- *Universal Serial Bus Specification, Revision 2.0*
- *On-The-Go Supplement to the USB 2.0 Specification, Revision 1.0a*

Visit the Intel USB specifications web page at <http://www.intel.com/technology/usb/spec.htm> for:

- *Enhanced Host Controller Interface Specification for Universal Serial Bus, Revision 1.0*

Visit the ULPI web page at <http://www.ulpi.org> for:

- *UTMI+ Specification, Revision 1.0*
- *UTMI Low Pin Interface (ULPI) Specification, Revision 1.0*

#### 24.1.1 Overview

The USB On-The-Go (OTG) module is a USB 2.0-compliant serial interface engine for implementing a USB interface. The registers and data structures are based on the *Enhanced Host Controller Interface Specification for Universal Serial Bus* (EHCI) from Intel Corporation. The USB OTG module can act as a host, a device, or an On-The-Go negotiable host/device on the USB bus.

The USB 2.0 OTG module interfaces to the processor's ColdFire core. The USB controller is programmable to support host, or device operations under firmware control. Full-speed (FS) and low-speed (LS) applications are supported by the integrated on-chip transceiver. The ULPI interface option supports high-speed (HS) applications. The processor's on-chip PLL provides all necessary clocks to the USB controller, including a system interface clock and a 60 MHz clock. For special applications, pin access (via USBCLKIN) is provided for an external 60 MHz reference clock. See [Chapter 10, “Chip Configuration Module \(CCM\)](#),” for more information.

The USB controller provides control and status signals to interface with external USB OTG and USB host power devices. Use these control and status signals on the chip interface and the I<sup>2</sup>C bus to communicate with external USB On-The-Go and USB host power devices.

USB-host modules must supply 500 mA with a 5 V supply on its downstream port (referred to as VBUS); however, the USB OTG standard provides a minimum 8 mA VBUS supply requirement. Optionally, the

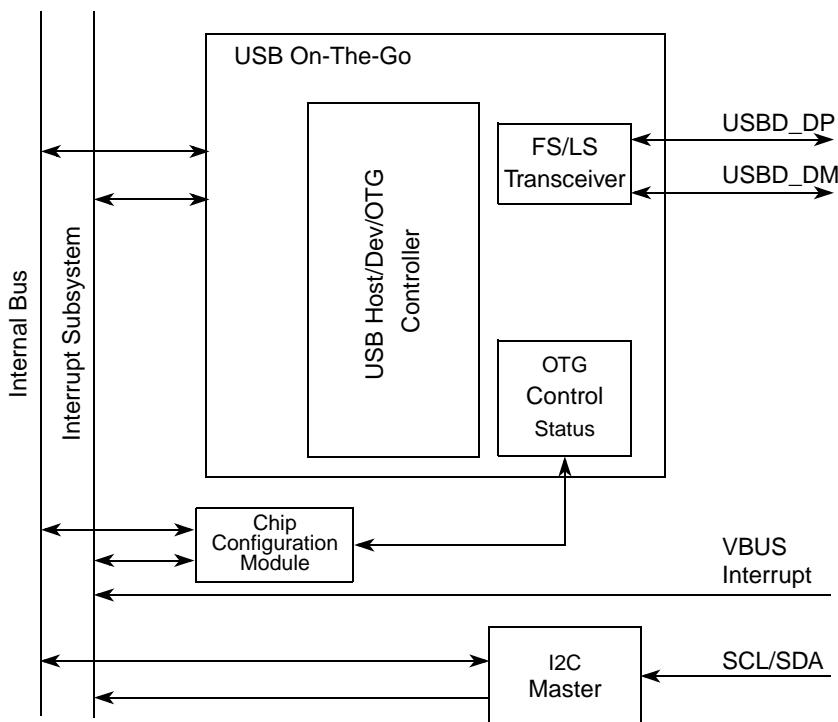
OTG module may supply up to 500 mA to the USB-connected devices. If the connected device attempts to draw more than the allocated amount of current, the USB host must disable the port and remove power. USB VBUS is not provided on-chip. This processor provides pins for control and status to an external IC capable of managing the VBUS downstream supply.

For OTG operations, external circuitry is required to manage the host negotiation protocol (HNP) and session request protocol (SRP). External ICs that are capable of providing the OTG VBUS with support for HNP and SRP, as well as support for programmable pullup and pulldown resistors on the USB DP and DM lines are available from various manufacturers.

The on-chip FS/LS transceiver also includes a programmable pullup resistor on USB DP. This pullup is configurable via the CCM. Also, configurable 50 k $\Omega$  pulldown resistors are available on the USB\_DP and USB\_DM signals of the on-chip transceiver. See [Chapter 10, “Chip Configuration Module \(CCM\),”](#) for more information. The primary function of the transceiver is the physical signal conditioning of the external USB DP and DM cable signals for a USB 2.0 network. Several USB system elements are not supported on the device as they are available via standard products from various manufacturers.

## 24.1.2 Block Diagram

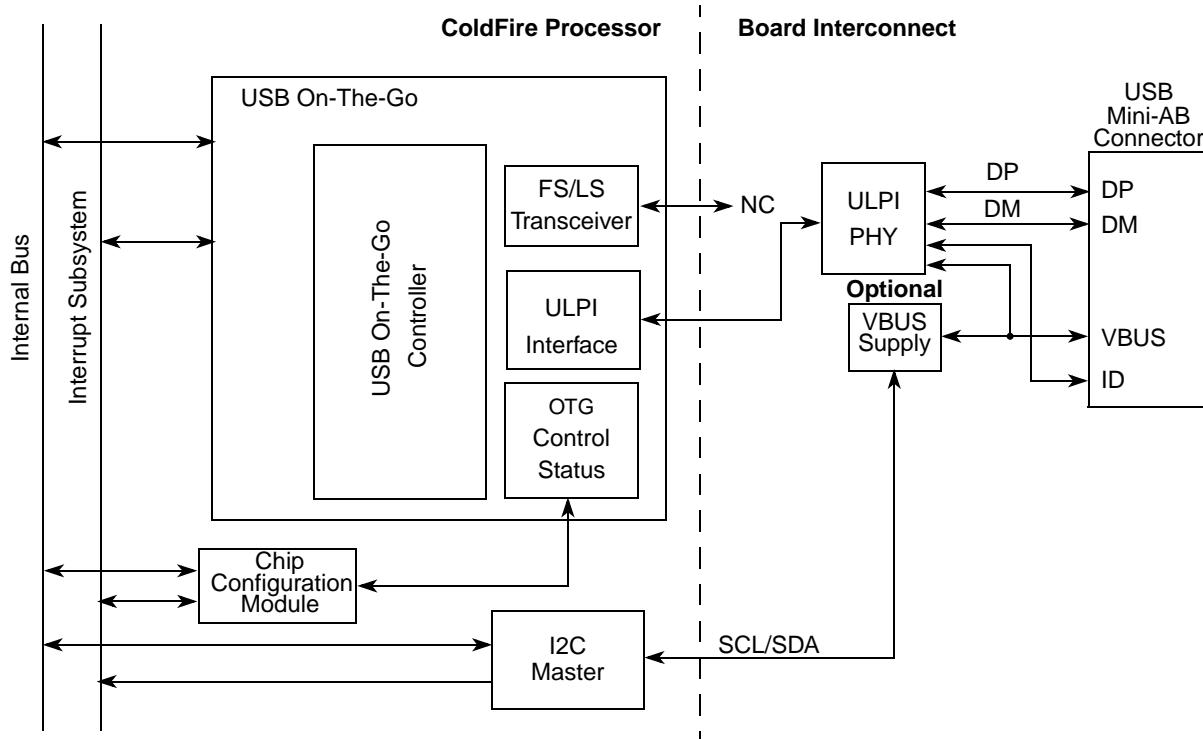
[Figure 24-1](#) shows the USB On-The-Go interface using the on-chip full-speed/low-speed transceiver.



**Figure 24-1. USB On-The-Go with on-chip FS/LS Transceiver Interface Block Diagram**

[Figure 24-2](#) illustrates the On-The-Go (OTG) configuration with an off-chip ULPI transceiver. The ULPI transceiver is an implementation of the HS/FS/LS physical layer which encapsulates the 60+ pin UTMI+ interface using a 12-pin digital interface.

The board-level implementation of a ULPI based product is dependent on the PHY vendor. One possible implementation is shown in [Figure 24-1](#). The ULPI PHY manages USB clocking, DP/DM bias resistors, and the OTG VBUS charge pump. For OTG applications requiring full host power (100 – 500 mA downstream current), an additional USB power-switch chip may be used. This OTG configuration may be used as a USB device, host, or dual-role device under firmware control.



**Figure 24-2. USB On-The-Go module and ULPI transceiver/PHY**

### 24.1.3 Features

The USB On-The-Go module includes these features:

- Complies with USB specification rev 2.0
- USB host mode
  - Supports enhanced-host-controller interface (EHCI).
  - Allows direct connection of FS/LS devices without an OHCI/UHCI companion controller.
  - Supported by Linux and other commercially available operating systems.
- USB device mode
  - Supports full-speed operation via the on-chip transceiver.
  - Supports full-speed/high-speed operation via an external ULPI transceiver.
  - Supports one upstream facing port.
  - Supports four programmable, bidirectional USB endpoints, including endpoint 0. See endpoint configurations:

**Table 24-1. Endpoint Configurations**

Endpoint	Type	FIFO Size	Data Transfer	Comments
0	Bidirectional	Variable	Control	Mandatory
1-3	IN or OUT	Variable	Ctrl, Int, Bulk, or Iso	Optional

- Suspend mode/low power
  - As host, firmware can suspend individual devices or the entire USB and disable chip clocks for low-power operation
  - Device supports low-power suspend
  - Remote wake-up supported for host and device
  - Integrated with the processor’s doze and stop modes for low power operation
- Includes an on-chip full-speed (12 Mbps) and low-speed (1.5 Mbps) transceiver
- Support for off-chip HS/FS/LS transceiver
  - External ULPI transceiver supports high speed (480 Mbps), full speed, and low speed operation in host mode, and high-speed and full-speed operation in device mode (shared with USB host module, see MISCCR2[ULPI] in [Chapter 10, “Chip Configuration Module \(CCM\)”](#))
  - Interface uses 8-bit single-data-rate ULPI data bus
  - ULPI PHY supplies a 60 MHz USB reference clock input to the processor

#### 24.1.4 Modes of Operation

The USB OTG module has two basic operating modes: host and device. Selection of operating mode is accomplished via the USBMODE[CM] bit field.

Speed selection is auto-detected at connect time via sensing of the DP or DM pull-up resistor on the connected device using enumeration procedures in the USB network. The USB OTG module provides these operation modes:

- USB disabled. In this mode, the USB OTG’s datapath does not accept transactions received on the USB interface.
- USB enabled. In this mode, the USB host’s datapath is enabled to accept transactions received on the USB interface.
- USB enabled, low-power modes. See [Section 24.1.4.1, “Low-Power Modes,”](#) for details.

##### 24.1.4.1 Low-Power Modes

The USB OTG module is integrated with the processor’s low-power modes (stop, doze and wait). The modes are implemented as follows:

- Stop — The processor stops the clock to the USB OTG module. In this state, the USB OTG module ignores traffic on the USB and does not generate any interrupts or wake-up events. The on-chip transceiver is disabled to save power.
- Wait — The clocks to the USB OTG module are running.

- Doze — The processor stops the system clocks to the USB OTG module, but the 60 MHz transceiver clock remains active. In doze mode, detection of resume signaling initiates a restart of the module clocks.

## 24.2 External Signal Description

Table 24-2 describes the external signal functionality of the USB OTG module.

### NOTE

The ULPI signals are multiplexed with the Ethernet assembly. This section describes the signal functions when in ULPI mode; refer to [Chapter 15, “Pin-Multiplexing and Control,”](#) for more details.

**Table 24-2. USB OTG Signal Descriptions**

Signal	I/O	Description
<b>On-chip FS/LS transceiver</b>		
USB_CLKIN	I	Optional 60 MHz clock source. This signal is also used for the input clock from a ULPI PHY.
USBOTG_DM	I/O	Data minus. Output of dual-speed transceiver for the USB OTG module.
		<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td><b>State Meaning</b></td><td>Asserted—Data 1 Negated—Data 0</td></tr> </table>
<b>State Meaning</b>	Asserted—Data 1 Negated—Data 0	
<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td><b>Timing</b></td><td>Asynchronous</td></tr> </table>	<b>Timing</b>	Asynchronous
<b>Timing</b>	Asynchronous	
USBOTG_DP	I/O	Data plus. Output of dual-speed transceiver for the USB OTG module.
		<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td><b>State Meaning</b></td><td>Asserted—Data 1 Negated—Data 0</td></tr> </table>
<b>State Meaning</b>	Asserted—Data 1 Negated—Data 0	
<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td><b>Timing</b></td><td>Asynchronous</td></tr> </table>	<b>Timing</b>	Asynchronous
<b>Timing</b>	Asynchronous	
USB_PULLUP	O	Enables an external pull-up on the USBOTG_DP line. This signal is controlled by the UOCSR[BVLD] bit.
		<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td><b>State Meaning</b></td><td>Asserted—Pull-up enabled. UOCSR[BVLD] set. Negated—Pull-up disabled. UOCSR[BVLD] cleared.</td></tr> </table>
<b>State Meaning</b>	Asserted—Pull-up enabled. UOCSR[BVLD] set. Negated—Pull-up disabled. UOCSR[BVLD] cleared.	
<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td><b>Timing</b></td><td>Asynchronous</td></tr> </table>	<b>Timing</b>	Asynchronous
<b>Timing</b>	Asynchronous	

### 24.2.1 USB OTG Control and Status Signals

The USB OTG module uses a number of control and status signals to implement the OTG protocols. The USB OTG module must be able to individually enable and disable the pull-up and pull-down resistors on DP and DM, and it must be able to control and sense the levels on the USB VBUS line.

These control and status signals are implemented on chip as registers within the chip-configuration module (CCM) to minimize the pin-count on the device. With firmware, the system designer uses an external device to manage the OTG functions to implement communications across the I<sup>2</sup>C bus or GPIO pins.

The OTG controller status register (UOCSR) implements as follows:

- Writes to the UOCSR register from the firmware set the corresponding bits on the USB interface.

- When the USB OTG module outputs change, the corresponding bits on the UOCSR register are updated, and a maskable interrupt is generated.

The UOCSR register is documented in the CCM chapter, see [Section 10.3.7, “USB On-the-Go Controller Status Register \(UOCSR\).”](#)

**Table 24-3. Internal Control and Status Bits for USB OTG Module**

Signal	Mnemonic	Direction	Comments	Interrupt Trigger?
DP Pull-down Enable	DPPD	Enables 15 kΩ resistor pull-down on DP	R	Y
DM Pull-down Enable	DMPD	Enables 15 kΩ resistor pull-down on DM	R	Y
VBUS Drive	DRV_VBUS	Enables bus power on VBUS to connected device.	R	Y
VBUS Charge	CRG_VBUS	Enables 8 mA pull-up to charge VBUS.	R	Y
VBUS Discharge	DCR_VBUS	Enables 8 mA pull-down to discharge VBUS.	R	Y
DP Pull-up Enable	DPPU	Enables the 1.5KΩ resistor pull-up on DP	R	Y
A Session Valid	AVLD	Indicates a valid session level for A device detected on VBUS.	R/W	N
B Session Valid	BVLD	Indicates a valid session level for B device detected on VBUS.	R/W	N
Session Valid	VVLD	Indicates valid operating level on VBUS from USB device's perspective.	R/W	N
Session End	SEND	Indicates VBUS fell below the session valid threshold.	R/W	N
VBUS Fault	PWRFLT	Indicates a fault (overcurrent, thermal issue) on VBUS.	R/W	N
Wake-up Event	WKUP	Reflects when a wake-up event occurred on the USB bus.	R/W	Y
Interrupt Mask	UOMIE	Interrupt enable. When this bit is 1, changes on DPPD, DMPD, DPPU, CHRG_VBUS, DCRG_VBUS, or VBUS_PWR cause an interrupt to be asserted. When this bit is 0, the interrupt is masked.	R/W	N/A
On-chip Transceiver Pull-down Enable	XPDE	Enables the on-chip 50 kΩ pull-downs on the OTG controller's DM and DP pins when the on-chip transceiver is used.	R/W	N

## 24.3 Memory Map/Register Definition

This section provides the memory map and detailed descriptions of all USB-interface registers. See [Table 24-4](#) for the memory map of the USB OTG interface. Registers in USB host module are similar, starting with 0xFC0B\_4xxx. See [Chapter 23, “Universal Serial Bus Interface – Host Module,”](#) for the USB host memory map.

**Table 24-4. USB On-The-Go Memory Map**

Address	Register	EHCI <sup>1</sup>	H/D <sup>2</sup>	Width (bits)	Access	Reset	Section/Page
<b>Module Identification Registers</b>							
0xFC0B_0000	Identification Register (ID)	N	H/D	32	R	0xE241_FA05	<a href="#">24.3.1.1/24-8</a>
0xFC0B_0004	General Hardware Parameters (HWGENERAL)	N	H/D	32	R	0x0000_07C5	<a href="#">24.3.1.2/24-9</a>
0xFC0B_0008	Host Hardware Parameters (HWHOST)	N	H/D	32	R	0x1002_0001	<a href="#">24.3.1.3/24-10</a>
0xFC0B_000C	Device Hardware Parameters (HWDEVICE)	N	D	32	R	0x0000_0009	<a href="#">24.3.1.4/24-10</a>
0xFC0B_0010	TX Buffer Hardware Parameters (HWTXBUF)	N	H/D	32	R	0x8004_0604	<a href="#">24.3.1.5/24-11</a>
0xFC0B_0014	RX Buffer Hardware Parameters (HWRXBUF)	N	H/D	32	R	0x0000_0404	<a href="#">24.3.1.6/24-11</a>
<b>Device/Host Timer Registers</b>							
0xFC0B_0080	General Purpose Timer 0 Load (GPTIMER0LD)	N	H/D	32	R/W	0x0000_0000	<a href="#">24.3.2.1/24-12</a>
0xFC0B_0084	General Purpose Timer 0 Control (GPTIMER0CTL)	N	H/D	32	R/W	0x0000_0000	<a href="#">24.3.2.2/24-12</a>
0xFC0B_0088	General Purpose Timer 1 Load (GPTIMER1LD)	N	H/D	32	R/W	0x0000_0000	<a href="#">24.3.2.1/24-12</a>
0xFC0B_008C	General Purpose Timer 1 Control (GPTIMER1CTL)	N	H/D	32	R/W	0x0000_0000	<a href="#">24.3.2.2/24-12</a>
<b>Capability Registers</b>							
0xFC0B_0100	Host Interface Version Number (HCIVERSION)	Y	H	16	R	0x0100	<a href="#">24.3.3.1/24-13</a>
0xFC0B_0103	Capability Register Length (CAPLENGTH)	Y	H/D	8	R	0x40	<a href="#">24.3.3.2/24-14</a>
0xFC0B_0104	Host Structural Parameters (HCSPARAMS)	Y	H	32	R	0x0001_0011	<a href="#">24.3.3.3/24-14</a>
0xFC0B_0108	Host Capability Parameters (HCCPARAMS)	Y	H	32	R	0x0000_0006	<a href="#">24.3.3.4/24-15</a>
0xFC0B_0122	Device Interface Version Number (DCIVERSION)	N	D	16	R	0x0001	<a href="#">24.3.3.5/24-16</a>
0xFC0B_0124	Device Capability Parameters (DCCPARAMS)	N	D	32	R	0x0000_0184	<a href="#">24.3.3.6/24-16</a>
<b>Operational Registers</b>							
0xFC0B_0140	USB Command (USBCMD)	Y	H/D	32	R/W	0x0008_0000	<a href="#">24.3.4.1/24-17</a>
0xFC0B_0144	USB Status (USBSTS)	Y	H/D	32	R/W	0x0000_0080	<a href="#">24.3.4.2/24-19</a>
0xFC0B_0148	USB Interrupt Enable (USBINTR)	Y	H/D	32	R/W	0x0000_0000	<a href="#">24.3.4.3/24-22</a>
0xFC0B_014C	USB Frame Index (FRINDEX)	Y	H/D	32	R/W	0x0000_0000	<a href="#">24.3.4.4/24-24</a>
0xFC0B_0154	Periodic Frame List Base Address (PERIODICLISTBASE)	Y	H	32	R/W	0x0000_0000	<a href="#">24.3.4.5/24-25</a>
0xFC0B_0154	Device Address (DEVICEADDR)	N	D	32	R/W	0x0000_0000	<a href="#">24.3.4.6/24-26</a>

**Table 24-4. USB On-The-Go Memory Map (continued)**

Address	Register	EHCI <sup>1</sup>	H/D <sup>2</sup>	Width (bits)	Access	Reset	Section/Page
0xFC0B_0158	Current Asynchronous List Address (ASYNCLISTADDR)	Y	H	32	R/W	0x0000_0000	24.3.4.7/24-26
0xFC0B_0158	Address at Endpoint List (EPLISTADDR)	N	D	32	R/W	0x0000_0000	24.3.4.8/24-27
0xFC0B_015C	Host TT Asynchronous Buffer Control (TTCTRL)	N	H	32	R/W	0x0000_0000	24.3.4.9/24-28
0xFC0B_0160	Master Interface Data Burst Size (BURSTSIZE)	N	H/D	32	R/W	0x0000_0101	24.3.4.10/24-28
0xFC0B_0164	Host Transmit FIFO Tuning Control (TXFILLTUNING)	N	H	32	R/W	0x0000_0000	24.3.4.11/24-28
0xFC0B_0170	ULPI Register Access (ULPI_VIEWPORT)	N	H/D	32	R/W	0x0000_0000	24.3.4.12/24-30
0xFC0B_0180	Configure Flag Register (CONFIGFLAG)	Y	H/D	32	R	0x0000_0001	24.3.4.13/24-32
0xFC0B_0184	Port Status/Control (PORTSC1)	Y	H/D	32	R/W	0xEC00_0004	24.3.4.14/24-32
0xFC0B_01A4	On-The-Go Status and Control (OTGSC)	N	H/D	32	R/W	0x0000_1020	24.3.4.15/24-36
0xFC0B_01A8	USB Mode Register (MODE)	N	H/D	32	R/W	0x0000_0000	24.3.4.16/24-39
0xFC0B_01AC	Endpoint Setup Status Register (EPSETUPSR)	N	D	32	R/W	0x0000_0000	24.3.4.17/24-40
0xFC0B_01B0	Endpoint Initialization (EPPRIME)	N	D	32	R/W	0x0000_0000	24.3.4.18/24-41
0xFC0B_01B4	Endpoint De-initialize (EPFLUSH)	N	D	32	R/W	0x0000_0000	24.3.4.19/24-41
0xFC0B_01B8	Endpoint Status Register (EPSR)	N	D	32	R	0x0000_0000	24.3.4.20/24-42
0xFC0B_01BC	Endpoint Complete (EPCOMPLETE)	N	D	32	R/W	0x0000_0000	24.3.4.21/24-43
0xFC0B_01C0	Endpoint Control Register 0 (EPCR0)	N	D	32	R/W	0x0080_0080	24.3.4.22/24-44
0xFC0B_01C4	Endpoint Control Register 1 (EPCR1)	N	D	32	R/W	0x0000_0000	24.3.4.23/24-45
0xFC0B_01C8	Endpoint Control Register 2 (EPCR2)	N	D	32	R/W	0x0000_0000	24.3.4.23/24-45
0xFC0B_01CC	Endpoint Control Register 3 (EPCR3)	N	D	32	R/W	0x0000_0000	24.3.4.23/24-45

<sup>1</sup> Indicates if the register is present in the EHCI specification.

<sup>2</sup> Indicates if the register is available in host and/or device modes.

## 24.3.1 Module Identification Registers

Declare the slave interface presence and include a table of the hardware configuration parameters. These registers are not defined by the EHCI specification.

### 24.3.1.1 Identification (ID) Register

Provides a simple way to determine if the module is provided in the system. The ID register identifies the module and its revision.

Address: 0xFC0B\_0000 (ID)

Access: User read-only

	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	0	0	REVISION					1	1	NID				0	0	ID												
W																																		
Reset	1	1	1	0	0	0	1	0	0	1	0	0	0	0	1	1	1	1	1	1	0	1	0	0	0	0	0	1	0	1	0	1		

Figure 24-3. Identification Register (ID)

Table 24-5. ID Field Descriptions

Field	Description
31–24	Reserved, always set to 0xE2.
23–16 REVISION	Revision number of the module.
15–14	Reserved, always set.
13–8 NID	Ones-complement version of the ID bit field.
7–6	Reserved, always cleared.
5–0 ID	Configuration number. This number is set to 0x05.

#### 24.3.1.2 General Hardware Parameters Register (HWGENERAL)

The HWGENERAL register contains parameters defining the particular implementation of the module.

Address: 0xFC0B\_0004 (HWGENERAL)

Access: User read-only

	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SM	PHYM	PHYW	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	1	1	1	1	0	0	0	0	1	0	1

Figure 24-4. General Hardware Parameters Register (HWGENERAL)

Table 24-6. HWGENERAL Field Descriptions

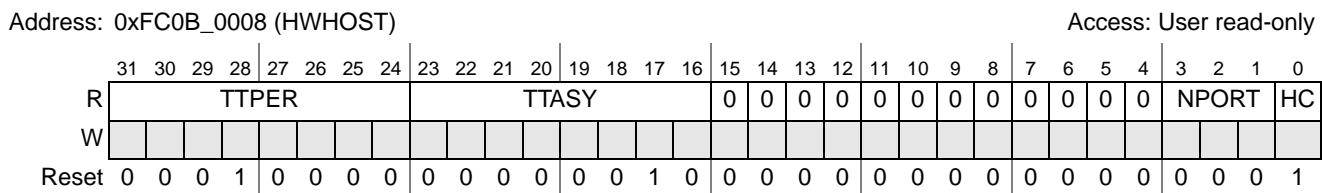
Field	Description
31–11	Reserved, always cleared.
10–9 SM	Serial mode. Indicates presence of serial interface. Always 11. 11 Serial engine is present and defaulted for all FS/LS operations
8–6 PHYM	PHY Mode. Indicates USB transceiver interface used. Always reads 111. 111 Software controlled reset to serial FS
5–4 PHYW	PHY width. Indicates data interface to UTMI transceiver. This field is relevant only for UTMI mode; therefore, it is relevant only to the USB OTG module in UTMI mode. Always reads 00. 00 8-bit data bus (60 MHz)
3	Reserved, always cleared.

**Table 24-6. HWGENERAL Field Descriptions (continued)**

Field	Description
2–1	Reserved. For the USB OTG module, always 10; for the USB host module, always 01.
0	Reserved, always set.

### 24.3.1.3 Host Hardware Parameters Register (HWHOST)

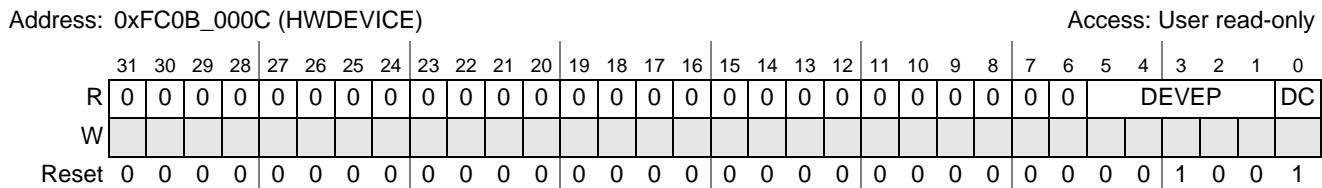
Provides host hardware parameters for this implementation of the module.

**Figure 24-5. Host Hardware Parameters Register (HWHOST)****Table 24-7. HWHOST Field Descriptions**

Field	Description
31–24 TTPER	Transaction translator periodic contexts. Number of supported transaction translator periodic contexts. Always 0x10. 0x10 16
23–16 TTASY	Transaction translator contexts. Number of transaction translator contexts. Always 0x02. 0x02 2
15–4	Reserved, always cleared.
3–1 NPORT	Indicates number of ports in host mode minus 1. Always 0 for the USB OTG module; Always 1 for the USB host module.
0 HC	Indicates module is host capable. Always set.

### 24.3.1.4 Device Hardware Parameters Register (HWDEVICE)

Provides device hardware parameters for this implementation of the USB OTG module.

**Figure 24-6. Device Hardware Parameters Register (HWDEVICE)**

**Table 24-8. HWDEVICE Field Descriptions**

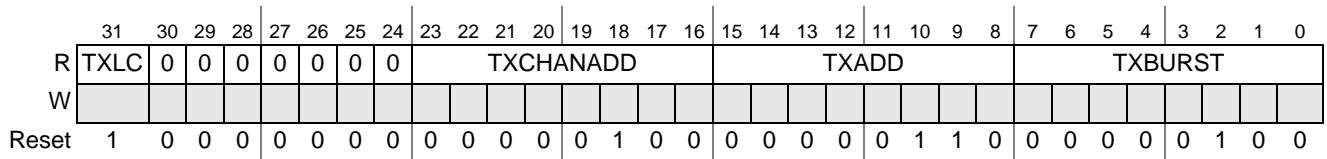
Field	Description
31–6	Reserved, always cleared.
5–1 DEVEP	Device endpoints. The number of supported endpoints. Always 0x04.
0 DC	Indicates the OTG module is device capable. Always set.

#### 24.3.1.5 Transmit Buffer Hardware Parameters Register (HWTXBUF)

Provides the transmit-buffer parameters for this implementation of the module.

Address: 0xFC0B\_0010 (HWTXBUF)

Access: User read-only



**Figure 24-7. Transmit Buffer Hardware Parameters Register (HWTXBUF)**

**Table 24-9. HWTXBUF Field Descriptions**

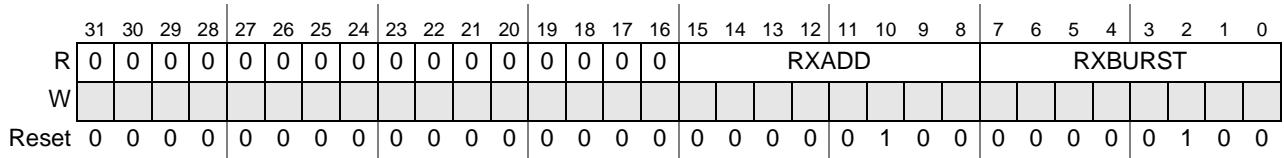
Field	Description
31 TXLC	Transmit local context registers. Indicates how the device transmit context registers implement. Always set on USB OTG module; Always clear on USB host. 0 Store device transmit contexts in the TX FIFO 1 Store device transmit contexts in a register file
30–24	Reserved, always cleared.
23–16 TXCHANADD	Transmit channel address. Number of address bits required to address one channel's worth of TX data. Always 0x04.
15–8 TXADD	Transmit address. Number of address bits for the entire TX buffer. Always 0x06.
7–0 TXBURST	Transmit burst. Indicates number of data beats in a burst for transmit DMA data transfers. Always 0x04.

#### **24.3.1.6 Receive Buffer Hardware Parameters Register (HWRXBUF)**

Provides the receive buffer parameters for this implementation of the module.

Address: 0xFC0B\_0014 (HWRXBUF)

Access: User read-only



**Figure 24-8. Receive Buffer Hardware Parameters Register (HWRXBUF)**

**Table 24-10. HWRXBUF Field Descriptions**

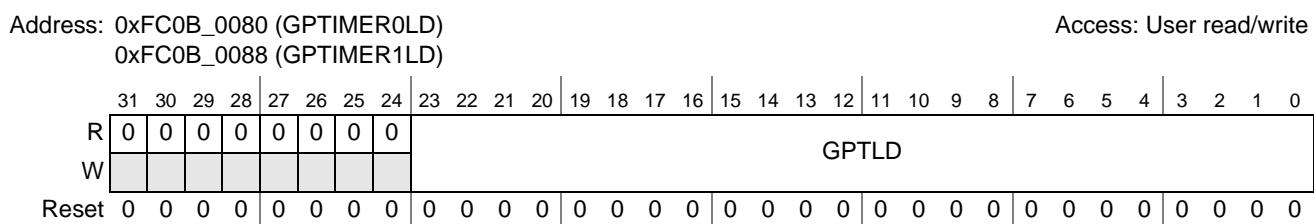
Field	Description
31–16	Reserved.
15–8 RXADD	Receive address. The number of address bits for the entire RX buffer. Always 0x04.
7–0 RXBURST	Receive burst. Indicates the number of data beats in a burst for receive DMA data transfers. Always 0x04.

### 24.3.2 Device/Host Timer Registers

The host/device controller drivers can measure time-related activities using these timer registers, which are not defined by the EHCI specification.

#### 24.3.2.1 General Purpose Timer *n* Load Registers (GPTIMER*n*LD)

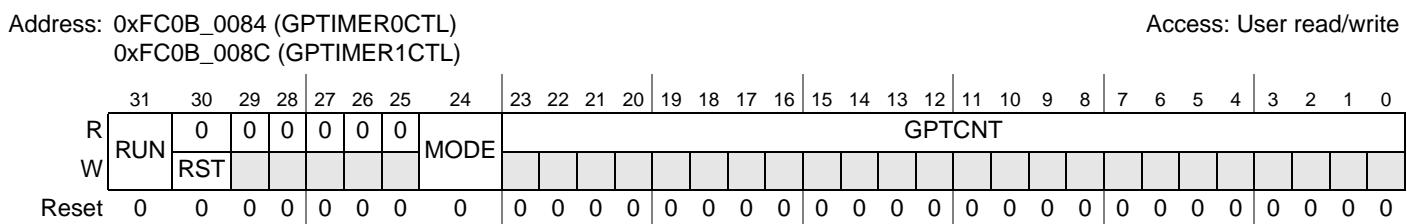
The GPTIMER*n*LD registers contain the timer duration or load value.

**Figure 24-9. General Purpose Timer *n* Load Registers (GPTIMER*n*LD)****Table 24-11. GPTIMER*n*LD Field Descriptions**

Field	Description
31–24	Reserved, must be cleared.
23–0 GPTLD	Specifies the value to be loaded into the countdown timer on a reset. The value in this register represents the time in microseconds minus 1 for the timer duration. For example, for a one millisecond timer, load 1000 – 1 = 999 (0x00_03E7). <b>Note:</b> Maximum value of 0xFF_FFFF or 16.777215 seconds.

#### 24.3.2.2 General Purpose Timer *n* Control Registers (GPTIMER*n*CTL)

The GPTIMER*n*CTL registers control the various functions of the general purpose timers.

**Figure 24-10. General Purpose Timer *n* Control Registers (GPTIMER*n*CTL)**

**Table 24-12. GPTIMERnCTL Field Descriptions**

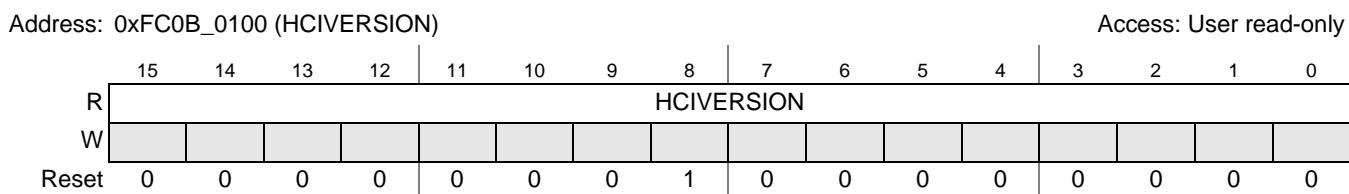
Field	Description
31 RUN	Timer run. Enables the general purpose timer. Setting or clearing this bit does not have an effect on the GPTCNT field. 0 Timer stop 1 Timer run
30 RST	Timer reset. Setting this bit reloads GPTCNT with the value in GPTIMERnLD[GPTLD]. 0 No action 1 Load counter value
29–25	Reserved, must be cleared.
24 MODE	Timer mode. Selects between a single timer countdown and a looped countdown. In one-shot mode, the timer counts down to zero, generates an interrupt, and stops until the counter is reset by software. In repeat mode, the timer counts down to zero, generates an interrupt, and automatically reloads the counter and begins another countdown. 0 One shot 1 Repeat
23–0 GPTCNT	Timer count. Indicates the current value of the running timer.

### 24.3.3 Capability Registers

Specifies software limits, restrictions, and capabilities of the host/device controller implementation. Most of these registers are defined by the EHCI specification. Registers not defined by the EHCI specification are noted in their descriptions.

#### 24.3.3.1 Host Controller Interface Version Register (HCIVERSION)

This is a two-byte register containing a BCD encoding of the EHCI revision number supported by this OTG controller. The most-significant byte of the register represents a major revision; the least-significant byte is the minor revision. [Figure 24-11](#) shows the HCIVERSION register.

**Figure 24-11. Host Controller Interface Version Register (HCIVERSION)****Table 24-13. HCIVERSION Field Descriptions**

Field	Description
15–0 HCIVERSION	EHCI revision number. Value is 0x0100 indicating version 1.0.

### 24.3.3.2 Capability Registers Length Register (CAPLENGTH)

Register is used as an offset to add to the register base address to find the beginning of the operational register space, the location of the USBCMD register.

Address: 0xFC0B_0103 (CAPLENGTH)								Access: User read-only	
CAPLENGTH									
R									
W	0	1	0	0	0	0	0	0	0
Reset:	0	1	0	0	0	0	0	0	0

**Figure 24-12. Capability Registers Length Register (CAPLENGTH)**

**Table 24-14. CAPLENGTH Field Descriptions**

Field	Description
7-0 CAPLENGTH	Capability registers length. Always 0x40.

### 24.3.3.3 Host Controller Structural Parameters Register (HCSPARAMS)

This register contains structural parameters such as the number of downstream ports. [Figure 24-13](#) shows the HCSPARMS register.

Address: 0xFC0B_0104 (HCSPARAMS)																Access: User read-only																
	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	N_TT				N_PTT				0	0	0	PI	N_CC				N_PCC				0	0	0	PPC	N_PORTS			
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	See Note	0	0	0	0	0	0	0	0	0	1	0	0	0	1		

**Note:** Set on USB host module; Cleared on USB OTG.

**Figure 24-13. Host Controller Structural Parameters Register (HCSPARAMS)**

#### **Table 24-15. HCSPARAMS Field Descriptions**

Field	Description
31–28	Reserved, always cleared.
27–24 N_TT	Number of transaction translators. Non-EHCI field. Indicates number of embedded transaction translators associated with host controller. This field is always 0x0. See <a href="#">Section 24.5.5.1, “Embedded Transaction Translator Function,”</a> for more information on embedded transaction translators.
23–20 N_PTT	Ports per transaction translator. Non-EHCI field. Indicates number of ports assigned to each transaction translator within host controller.
19–17	Reserved, always cleared.

**Table 24-15. HCSPARAMS Field Descriptions (continued)**

Field	Description
16 PI	Port indicators. Indicates whether the ports support port indicator control. Always set on USB host, cleared on USB OTG. 0 No port indicator fields. 1 The port status and control registers include a R/W field for controlling the state of the port indicator. See <a href="#">Table 24-3</a> for more information.
15–12 N_CC	Number of companion controllers. Indicates number of companion controllers associated with USB OTG controller. Always cleared.
11–8 N_PCC	Number ports per CC. Indicates number of ports supported per internal companion controller. This field is 0 because no companion controllers are present.
7–5	Reserved, always cleared.
4 PPC	Power port control. Indicates whether host controller supports port power control. Always set. 1 Ports have power port switches.
3–0 N_PORTS	Number of ports. Indicates number of physical downstream ports implemented for host applications. Field value determines how many addressable port registers in the operational register. For the USB host and OTG modules, this is always 0x1.

#### 24.3.3.4 Host Controller Capability Parameters Register (HCCPARAMS)

Identifies multiple mode control (time-base bit functionality) addressing capability.

Address: 0xFC0B\_0108 (HCCPARAMS)

Access: User read-only

	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	0	0	0	0	0	0	0	0	0	0	0	0	0	EECP	IST	0	ASP	PFL	ADC							
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	1	1	0		

**Figure 24-14. Host Controller Capability Parameters Register (HCCPARAMS)****Table 24-16. HCCPARAMS Field Descriptions**

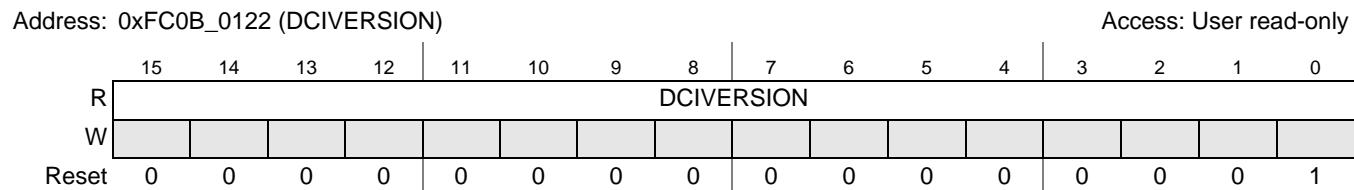
Field	Description
31–16	Reserved, always cleared.
15–8 EECP	EHCI extended capabilities pointer. This optional field indicates the existence of a capabilities list. 0x00 No extended capabilities are implemented. This field is always 0.
7–4 IST	Isochronous scheduling threshold. Indicates where software can reliably update the isochronous schedule, relative to the current position of the executing host controller. This field is always 0. 0 The value of the least significant 3 bits indicates the number of microframes a host controller can hold a set of isochronous data structures (one or more) before flushing the state.
3	Reserved, always cleared.
2 ASP	Asynchronous schedule park capability. Indicates if the host controller supports the park feature for high-speed queue heads in the asynchronous schedule. The feature can be disabled or enabled and set to a specific level by using the asynchronous schedule park mode enable and asynchronous schedule park mode count fields in the USBCMD register. This bit is always set. 0 Park not supported. 1 Park supported.

**Table 24-16. HCCPARAMS Field Descriptions (continued)**

Field	Description
1 PFL	Programmable frame list flag. Indicates that system software can specify and use a frame list length less than 1024 elements. This bit is always set. 1 Frame list size is configured via the USBCMD register frame list size field. The frame list must always be aligned on a 4K-page boundary. This requirement ensures that the frame list is always physically contiguous.
0 ADC	64-bit addressing capability. This field is always 0; 64-bit addressing is not supported. 0 Data structures use 32-bit address memory pointers

### 24.3.3.5 Device Controller Interface Version (DCIVERSION)

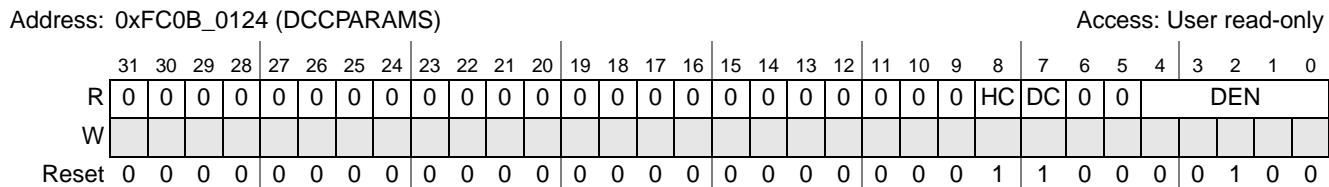
Not defined in the EHCI specification. DCIVERSION is a two-byte register containing a BCD encoding of the device controller interface. The most-significant byte of the register represents a major revision and the least-significant byte is the minor revision.

**Figure 24-15. Device Controller Interface Version Register (DCIVERSION)****Table 24-17. DCIVERSION Field Descriptions**

Field	Description
15–0 DCIVERSION	Device interface revision number.

### 24.3.3.6 Device Controller Capability Parameters (DCCPARAMS)

Not defined in the EHCI specification. Register describes the overall host/device capability of the USB OTG module.

**Figure 24-16. Device Control Capability Parameters (DCCPARAMS)****Table 24-18. DCCPARAMS Field Descriptions**

Field	Description
31–9	Reserved, always cleared.
8 HC	Host capable. Indicates the USB OTG controller can operate as an EHCI compatible USB 2.0 host. Always set.

**Table 24-18. DCCPARAMS Field Descriptions (continued)**

Field	Description
7 DC	Device Capable. Indicates the USB OTG controller can operate as an USB 2.0 device. Always set.
6–5	Reserved, always cleared.
4–0 DEN	Device endpoint number. This field indicates the number of endpoints built into the device controller. Always 0x04.

## 24.3.4 Operational Registers

Comprised of dynamic control or status registers and are defined below.

### 24.3.4.1 USB Command Register (USBCMD)

The module executes the command indicated in this register.

Address: 0xFC0B\_0140 (USBCMD)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	ITC							
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	FS2	ATDTW	SUTW	0	ASPE	0	ASP		0	IAA	ASE	PSE	FS1	FS0	RST	RS
W									0	0	0	0	0	0		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 24-17. USB Command Register (USBCMD)****Table 24-19. USBCMD Field Descriptions**

Field	Description
31–24	Reserved, must be cleared.
23–16 ITC	Interrupt threshold control. System software uses this field to set the maximum rate at which the module issues interrupts. ITC contains maximum interrupt interval measured in microframes. 0x00 Immediate (no threshold) 0x01 1 microframe 0x02 2 microframes 0x04 4 microframes 0x08 8 microframes 0x10 16 microframes 0x20 32 microframes 0x40 64 microframes Else Reserved
15 FS2	See the FS bit description below. This is a non-EHCI bit.

**Table 24-19. USBCMD Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>
14 ATDTW	Add dTD TripWire. This is a non-EHCI bit, that is present on the USB OTG module only. This bit is used as a semaphore when a dTD is added to an active (primed) endpoint. This bit is set and cleared by software. This bit is also cleared by hardware when the state machine is in a hazard region where adding a dTD to a primed endpoint may go unrecognized. More information appears in <a href="#">Section 24.5.3.6.3, “Executing a Transfer Descriptor.”</a>
13 SUTW	Setup TripWire. A non-EHCI bit present on the USB OTG module only. Used as a semaphore to ensure that the setup data payload of 8 bytes is extracted from a QH by driver software without being corrupted. If the setup lockout mode is off (USBMODE[SLOM] = 1) then a hazard exists when new setup data arrives, and the software copies setup from the QH for a previous setup packet. This bit is set and cleared by software and is cleared by hardware when a hazard exists. More information appears in <a href="#">Section 24.5.3.4.4, “Control Endpoint Operation.”</a>
12	Reserved, must be cleared.
11 ASPE	Asynchronous schedule park mode enable. Software uses this bit to enable or disable park mode. 1 Park mode enabled 0 Park mode disabled
10	Reserved, must be cleared.
9–8 ASP	Asynchronous schedule park mode count. Contains a count of the successive transactions the host controller can execute from a high-speed queue head on the asynchronous schedule before continuing traversal of the asynchronous schedule. Valid values are 0x1 to 0x3. Software must not write a zero to this field when ASPE is set as this results in undefined behavior.
7	Reserved, must be cleared.
6 IAA	Interrupt on async advance doorbell. Used as a doorbell by software to tell controller to issue an interrupt the next time it advances the asynchronous schedule. Software must write a 1 to this bit to ring the doorbell. When controller has evicted all appropriate cached schedule states, it sets USBSTS[AAI] register. If the USBINTR[AAE] bit is set, the host controller asserts an interrupt at the next interrupt threshold. The controller clears this bit after it has set the USBSTS[AAI] bit. Software must not write a 1 to this bit when the asynchronous schedule is inactive. Doing so yields undefined results. This bit used only in host mode. Writing a 1 to this bit when the USB OTG module is in device mode has undefined results.
5 ASE	Asynchronous schedule enable. Controls whether the controller skips processing the asynchronous schedule. Only used in host mode. 1 Use the ASYNCLISTADDR register to access asynchronous schedule. 0 Do not process asynchronous schedule.
4 PSE	Periodic schedule enable. Controls whether the controller skips processing periodic schedule. Used only in host mode. 1 Use the PERIODICLISTBASE register to access the periodic schedule. 0 Do not process periodic schedule.
3–2 FS	Frame list size. With bit 15, these bits make the FS[2:0] fields, which specifies the frame list size controlling which bits in the frame index register must be used for the frame list current index. Used only in host mode. <b>Note:</b> Values below 256 elements are not defined in the EHCI specification. 000 1024 elements (4096 bytes) 001 512 elements (2048 bytes) 010 256 elements (1024 bytes) 011 128 elements (512 bytes) 100 64 elements (256 bytes) 101 32 elements (128 bytes) 110 16 elements (64 bytes) 111 8 elements (32 bytes)

**Table 24-19. USBCMD Field Descriptions (continued)**

Field	Description
1 RST	Controller reset. Software uses this bit to reset controller. Controller clears this bit when reset process completes. Clearing this register does not allow software to terminate the reset process early.  Host mode (USB Host and USB OTG): When software sets this bit, the controller resets its internal pipelines, timers, counters, state machines etc. to their initial value. Any transaction in progress on the USB immediately terminates. A USB reset is not driven on downstream ports. Software must not set this bit when the USBSTS[HCH] bit is cleared. Attempting to reset an actively running host controller results in undefined behavior.  Device mode (USB OTG-only): When software sets this bit, the controller resets its internal pipelines, timers, counters, state machines, etc. to their initial value. Setting this bit with the device in the attached state is not recommended because it has an undefined effect on an attached host. To ensure the device is not in an attached state before initiating a device controller reset, all primed endpoints must be flushed and the USBCMD[RS] bit must be cleared.
0 RS	Run/Stop.  Host mode (USB Host and USB OTG): When set, the controller proceeds with the execution of the schedule. The controller continues execution as long as this bit is set. When this bit is cleared, the controller completes the current transaction on the USB and then halts. The USBSTS[HCH] bit indicates when the host controller finishes the transaction and enters the stopped state. Software must not set this bit unless controller is in halted state (USBSTS[HCH] = 1).  Device mode (USB OTG-only): Setting this bit causes the controller to enable a pull-up on DP and initiate an attach event. This control bit is not directly connected to the pull-up enable, as the pull-up becomes disabled upon transitioning into high-speed mode. Software must use this bit to prevent an attach event before the USB OTG controller has properly initialized. Clearing this bit causes a detach event.

#### 24.3.4.2 USB Status Register (USBSTS)

This register indicates various states of each module and any pending interrupts. This register does not indicate status resulting from a transaction on the serial bus. Software clears certain bits in this register by writing a 1 to them.

Address: 0xFC0B_0144 (USBSTS)												Access: User read/write				
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
W	0	0	0	0	0	0	TI1	TI0	0	0	0	0	UPI	UAI	0	NAKI
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	AS	PS	RCL	HCH	0	ULPII	0	SLI	SRI	URI	AAI	SEI	FRI	PCI	UEI	UI
Reset	0	0	0	0	0	0	0	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c

**Figure 24-18. USB Status Register (USBSTS)**

**Table 24-20. USBSTS Field Descriptions**

<b>Field</b>	<b>Description</b>
31–26	Reserved, must be cleared.
25 TI1	General purpose timer 1 interrupt. Set when the counter in the GPTIMER1CTRL register transitions to zero. Writing a one to this bit clears it. 0 No interrupt 1 Interrupt occurred.
24 TI0	General purpose timer 0 interrupt. Set when the counter in the GPTIMER0CTRL register transitions to zero. Writing a one to this bit clears it. 0 No interrupt 1 Interrupt occurred.
23–20	Reserved, must be cleared.
19 UPI	USB host periodic interrupt. Set by the host controller when the cause of an interrupt is a completion of a USB transaction where the transfer descriptor (TD) has an interrupt on complete (IOC) bit set and the TD was from the periodic schedule. This bit is also set by the host controller when a short packet is detected and the packet is on the periodic schedule. A short packet is when the actual number of bytes received was less than the expected number of bytes. <b>Note:</b> This bit is not used by the device controller and is always zero.
18 UAI	USB host asynchronous interrupt. Set by the host controller when the cause of an interrupt is a completion of a USB transaction where the transfer descriptor (TD) has an interrupt on complete (IOC) bit set and the TD was from the asynchronous schedule. This bit is also set by the host controller when a short packet is detected and the packet is on the asynchronous schedule. A short packet is when the actual number of bytes received was less than the expected number of bytes. <b>Note:</b> This bit is not used by the device controller and is always zero.
17	Reserved, must be cleared.
16 NAKI	NAK interrupt. Set by hardware for a particular endpoint when the TX/RX endpoint's NAK bit and the corresponding TX/RX endpoint's NAK enable bit are set. The hardware automatically clears this bit when all the enabled TX/RX endpoint NAK bits are cleared.
15 AS	Asynchronous schedule status. Reports the current real status of asynchronous schedule. Controller is not immediately required to disable or enable the asynchronous schedule when software transitions the USBCMD[ASE] bit. When this bit and the USBCMD[ASE] bit have the same value, the asynchronous schedule is enabled (1) or disabled (0). Used only in host mode. 0 Disabled. 1 Enabled.
14 PS	Periodic schedule status. Reports current real status of periodic schedule. Controller is not immediately required to disable or enable the periodic schedule when software transitions the USBCMD[PSE] bit. When this bit and the USBCMD[PSE] bit have the same value, the periodic schedule is enabled or disabled. Used only in host mode. 0 Disabled. 1 Enabled.
13 RCL	Reclamation. DetectS an empty asynchronous schedule. Used only by the host mode. 0 Non-empty asynchronous schedule. 1 Empty asynchronous schedule.

**Table 24-20. USBSTS Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>
12 HCH	Host controller halted. This bit is cleared when the USBCMD[RS] bit is set. The controller sets this bit after it stops executing because of the USBCMD[RS] bit being cleared, by software or the host controller hardware (for example, internal error). Used only in host mode. 0 Running. 1 Halted.
11	Reserved, must be cleared.
10 ULPII	ULPI interrupt. Set by event completion.
9	Reserved, must be cleared.
8 SLI	Device-controller suspend. Non-EHCI bit present on the USB OTG module only. When a device controller enters a suspend state from an active state, this bit is set. The device controller clears the bit upon exiting from a suspend state. Used only by the device controller. 0 Active. 1 Suspended.
7 SRI	SOF received. This is a non-EHCI status bit. Software writes a 1 to this bit to clear it. Host mode (USB host and USB OTG): In host mode, this bit is set every 125 µs, provided PHY clock is present and running (for example, the port is NOT suspended) and can be used by the host-controller driver as a time base. Device mode (USB OTG-only): When controller detects a start of (micro) frame, bit is set. When a SOF is extremely late, controller automatically sets this bit to indicate an SOF was expected. Therefore, this bit is set roughly every 1 ms in device FS mode and every 125 µsec in HS mode, and it is synchronized to the actual SOF received. Because the controller is initialized to FS before connect, this bit is set at an interval of 1 ms during the prelude to the connect and chirp.
6 URI	USB reset received. A non-EHCI bit present on the USB OTG module only. When the controller detects a USB reset and enters the default state, this bit is set. Software can write a 1 to this bit to clear it. Used only by in device mode. 0 No reset received. 1 Reset received.
5 AAI	Interrupt on async advance. By setting the USBCMD[IAA] bit, system software can force the controller to issue an interrupt the next time the controller advances the asynchronous schedule. This status bit indicates the assertion of that interrupt source. Used only by the host mode. 0 No async advance interrupt. 1 Async advance interrupt.
4 SEI	System error. Set when an error is detected on the system bus. If the system error enable bit (USBINTR[SEE]) is set, interrupt generates. The interrupt and status bits remain set until cleared by writing a 1 to this bit. Additionally, when in host mode, the USBCMD[RS] bit is cleared, effectively disabling controller. An interrupt generates for the USB OTG controller in device mode, but no other action is taken. 0 Normal operation 1 Error
3 FRI	Frame-list rollover. Controller sets this bit when the frame list index (FRINDEX) rolls over from its maximum value to 0. The exact value the rollover occurs depends on the frame list size. For example, if the frame list size (as programmed in the USBCMD[FS] field) is 1024, the frame index register rolls over every time FRINDEX[13] toggles. Similarly, if the size is 512, the controller sets this bit each time FRINDEX[12] toggles. Used only in the host mode.

**Table 24-20. USBSTS Field Descriptions (continued)**

Field	Description
2 PCI	Port change detect. This bit is not EHCI compatible. Host mode (USB host and USB OTG): Controller sets this bit when a connect status occurs on any port, a port enable/disable change occurs, an over-current change occurs, or the force port resume (PORTSCn[FPR]) bit is set as the result of a J-K transition on the suspended port. Device mode (USB OTG only): The controller sets this bit when it enters the full- or high-speed operational state. When it exits the full- or high-speed operation states due to reset or suspend events, the notification mechanisms are URI and SLI bits respectively. The device controller detects resume signaling only.
1 UEI	USB error interrupt. When completion of USB transaction results in error condition, the controller sets this bit. If the TD on which the error interrupt occurred also had its interrupt on complete (IOC) bit set, this bit is set along with the USBINT bit. See Section 4.15.1 in the EHCI specification for a complete list of host error interrupt conditions. See <a href="#">Table 24-58</a> for more information on device error matrix. 0 No error. 1 Error detected.
0 UI	USB interrupt (USBINT). This bit is set by the controller when the cause of an interrupt is a completion of a USB transaction where the TD has an interrupt on complete (IOC) bit set. This bit is also set by the controller when a short packet is detected. A short packet is when the actual number of bytes received was less than the expected number of bytes.

#### 24.3.4.3 USB Interrupt Enable Register (USBINTR)

The interrupts to software are enabled with this register. An interrupt generates when a bit is set and the corresponding interrupt is active. The USB status register (USBSTS) continues to show interrupt sources (even if the USBINTR register disables them), allowing polling of interrupt events by the software.

Address: 0xFC0B\_0148 (USBINTR) Access: User read/write

31 30 29 28				27 26 25 24				23 22 21 20				19 18 17 16					
R	0	0	0	0	0	0	TIE1	TIE0	0	0	0	0	UPIE	UAIE	0	NAKE	
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	ULPIE	0	SLE	SRE	URE	AAE	SEE	FRE	PCE	UEE	UE
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 24-19. USB Interrupt Enable Register (USBINTR)**

**Table 24-21. USBINTR Field Descriptions**

<b>Field</b>	<b>Description</b>
31–26	Reserved, must be cleared.
25 TIE1	General purpose timer 1 interrupt enable. When this bit and USBSTS[GPTINT1] are set, the USB controller issues an interrupt to the processor. The interrupt is acknowledged by clearing GPTINT1. 0 Disabled 1 Enabled
24 TIE0	General purpose timer 0 interrupt enable. When this bit and USBSTS[GPTINT0] are set, the USB controller issues an interrupt to the processor. The interrupt is acknowledged by clearing GPTINT0. 0 Disabled 1 Enabled
23–20	Reserved, must be cleared.
19 UPIE	USB host periodic interrupt enable. When this bit and USBSTS[USBHSTPERINT] are set, the host controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by clearing USBHSTPERINT.
18 UAIE	USB host asynchronous interrupt enable. When this bit and USBSTS[USBHSTASYNCINT] are set, the host controller issues an interrupt at the next interrupt threshold. The interrupt is acknowledged by clearing USBHSTASYNCINT.
17	Reserved, must be cleared.
16 NAKE	NAK interrupt enable. When this bit and the USBSTS[NAKI] bit are set, an interrupt generates. 0 Disabled 1 Enabled
15–11	Reserved, must be cleared.
10 ULPIE	ULPI enable. When this bit and USBSTS[ULPII] are set, controller issues an interrupt. The interrupt is acknowledged by writing a 1 to USBSTS[ULPII].
9	Reserved, must be cleared.
8 SLE	Sleep (DC suspend) enable. A non-EHCI bit present on the OTG module only. When this bit is set and the USBSTS[SLI] bit transitions, USB OTG controller issues an interrupt. Software writing a 1 to the USBSTS[SLI] bit acknowledges the interrupt. Used only in device mode. 0 Disabled 1 Enabled
7 SRE	SOF-received enable. This is a non-EHCI bit. When this bit and the USBSTS[SRI] bit are set, controller issues an interrupt. Software clearing the USBSTS[SRI] bit acknowledges the interrupt. 0 Disabled 1 Enabled
6 URE	USB-reset enable. A non-EHCI bit present on the USB OTG module only. When this bit and the USBSTS[URI] bit are set, device controller issues an interrupt. Software clearing the USBSTS[URI] bit acknowledges the interrupt. Used only in device mode. 0 Disabled 1 Enabled
5 AAE	Interrupt on async advance enable. When this bit and the USBSTS[AAI] bit are set, controller issues an interrupt at the next interrupt threshold. Software clearing the USBSTS[AAI] bit acknowledges the interrupt. Used only in host mode. 0 Disabled 1 Enabled

**Table 24-21. USBINTR Field Descriptions (continued)**

Field	Description
4 SEE	<p>System error enable. When this bit and the USBSTS[SEI] bit are set, controller issues an interrupt. Software clearing the USBSTS[SEI] bit acknowledges the interrupt.</p> <p>0 Disabled 1 Enabled</p>
3 FRE	<p>Frame list rollover enable. When this bit and the USBSTS[FRI] bit are set, controller issues an interrupt. Software clearing the USBSTS[FRI] bit acknowledges the interrupt. Used only in host mode.</p> <p>0 Disabled 1 Enabled</p>
2 PCE	<p>Port change detect enable. When this bit and the USBSTS[PCI] bit are set, controller issues an interrupt. Software clearing the USBSTS[PCI] bit acknowledges the interrupt.</p> <p>0 Disabled 1 Enabled</p>
1 UEE	<p>USB error interrupt enable. When this bit and the USBSTS[UEI] bit are set, controller issues an interrupt at the next interrupt threshold. Software clearing the USBSTS[UEI] bit acknowledges the interrupt.</p> <p>0 Disabled 1 Enabled</p>
0 UE	<p>USB interrupt enable. When this bit is 1 and the USBSTS[UI] bit is set, the USB OTG controller issues an interrupt at the next interrupt threshold. Software clearing the USBSTS[UI] bit acknowledges the interrupt.</p> <p>0 Disabled 1 Enabled</p>

#### **24.3.4.4 Frame Index Register (FRINDEX)**

In host mode, the controller uses this register to index the periodic frame list. The register updates every 125 microseconds (once each microframe). Bits [N-3] select a particular entry in the periodic frame list during periodic schedule execution. The number of bits used for the index depends on the size of the frame list as set by system software in the **USBCMD[FS]** field.

This register must be a longword. Byte writes produce undefined results. This register cannot be written unless the USB OTG controller is in halted state as the USBSTS[HCH] bit indicates. A write to this register while the USBSTS[RS] bit is set produces undefined results. Writes to this register also affect the SOF value.

In device mode (USB OTG-only), this register is read-only, and the USB OTG controller updates the FRINDEX[13–3] bits from the frame number the SOF marker indicates. When the USB bus receives a SOF, FRINDEX[13–3] checks against the SOF marker. If FRINDEX[13–3] is different from the SOF marker, FRINDEX[13–3] is set to the SOF value and FRINDEX[2–0] is cleared (SOF for 1 ms frame). If FRINDEX[13–3] equals the SOF value, FRINDEX[2–0] is incremented (SOF for 125 µsec microframe.)

Address: 0xFC0B\_014C (FRINDEX)

Access: User read/write

**Figure 24-20. Frame Index Register (FRINDEX)**

**Table 24-22. FRINDEX Field Descriptions**

Field	Description
31–14	Reserved, must be cleared.
13–0 FRINDEX	Frame index. The value in this register increments at the end of each time frame (microframe). Bits [N– 3] are for the frame list current index. This means each location of the frame list is accessed 8 times per frame (once each microframe) before moving to the next index. In device mode for the USB OTG module, the value is the current frame number of the last frame transmitted and not used as an index. In either mode, bits 2–0 indicate current microframe.

Table 24-23 illustrates values of N based on the value of the USBCMD[FS] field when used in host mode.

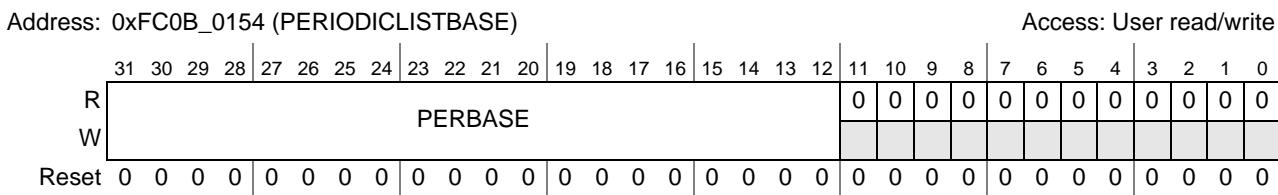
**Table 24-23. FRINDEX N Values**

USBCMD[FS]	Frame List Size	FRINDEX N value
000	1024 elements (4096 bytes)	12
001	512 elements (2048 bytes)	11
010	256 elements (1024 bytes)	10
011	128 elements (512 bytes)	9
100	64 elements (256 bytes)	8
101	32 elements (128 bytes)	7
110	16 elements (64 bytes)	6
111	8 elements (32 bytes)	5

#### 24.3.4.5 Periodic Frame List Base Address Register (PERIODICLISTBASE)

This register contains the beginning address of the periodic frame list in the system memory. The host controller driver loads this register prior to starting the schedule execution by the controller. The memory structure referenced by this physical memory pointer assumes to be 4-Kbyte aligned. The contents combine with the FRINDEX register to enable the controller to step through the periodic frame list in sequence.

On the USB OTG module, the host and device mode functions share this register. In host mode, it is the PERIODICLISTBASE register; in device mode, it is the DEVICEADDR register. See [Section 24.3.4.6, “Device Address Register \(DEVICEADDR\),”](#) for more information.

**Figure 24-21. Periodic Frame List Base Address Register (PERIODICLISTBASE)**

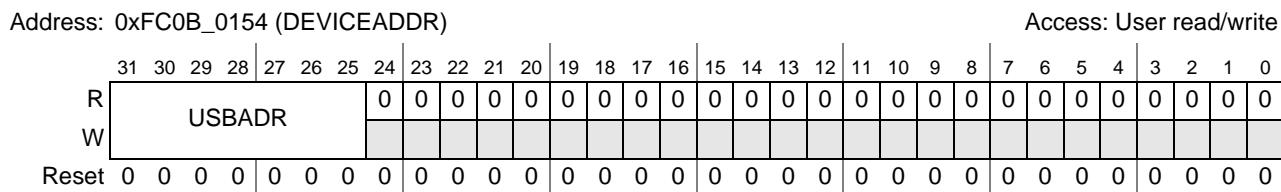
**Table 24-24. PERIODICLISTBASE Field Descriptions**

Field	Description
31–12 PERBASE	Base Address. These bits correspond to memory address signal [31:12]. Used only in the host mode
11–0	Reserved, must be cleared.

#### 24.3.4.6 Device Address Register (DEVICEADDR)

This register is not defined in the EHCI specification. For the USB OTG module in device mode, the upper seven bits of this register represent the device address. After any controller or USB reset, the device address is set to the default address (0). The default address matches all incoming addresses. Software reprograms the address after receiving a SET\_ADDRESS descriptor.

On the USB OTG module, the host and device mode functions share this register. In device mode, it is the DEVICEADDR register; in host mode, it is the PERIODICLISTBASE register. See [Section 24.3.4.5, “Periodic Frame List Base Address Register \(PERIODICLISTBASE\),”](#) for more information.



**Figure 24-22. Device Address Register (DEVICEADDR)**

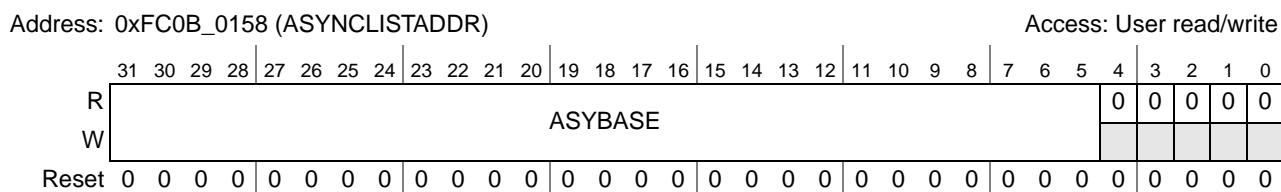
**Table 24-25. DEVICEADDR Field Descriptions**

Field	Description
31–25 USBADR	Device Address. This field corresponds to the USB device address.
24–0	Reserved, must be cleared.

#### 24.3.4.7 Current Asynchronous List Address Register (ASYNCLISTADDR)

The ASYNCLISTADDR register contains the address of the next asynchronous queue head to be executed by the host.

On the USB OTG module, the host and device mode functions share this register. In host mode, it is the ASYNCLISTADDR register; in device mode, it is the EPLISTADDR register. See [Section 24.3.4.8, “Endpoint List Address Register \(EPLISTADDR\),”](#) for more information.



**Figure 24-23. Current Asynchronous List Address Register (ASYNCLISTADDR)**

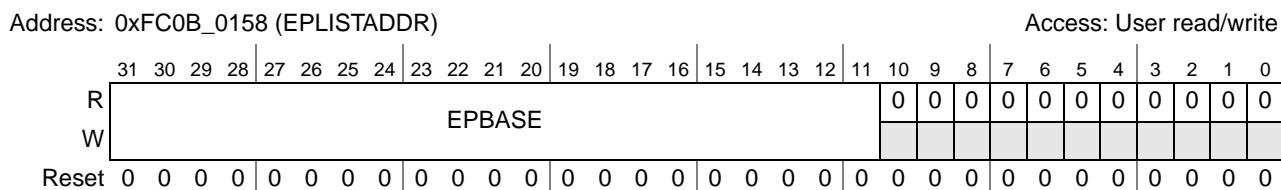
**Table 24-26. ASYNCLISTADDR Field Descriptions**

Field	Description
31–5 ASYBASE	Link pointer low (LPL). These bits correspond to memory address signal [31:5]. This field may only reference a queue head (QH). Used only in host mode.
4–0	Reserved, must be cleared.

#### 24.3.4.8 Endpoint List Address Register (EPLISTADDR)

This register is not defined in the EHCI specification. For the USB OTG module in device mode, this register contains the address of the endpoint list top in system memory. The memory structure referenced by this physical memory pointer assumes to be 64-bytes. The queue head is actually a 48-byte structure, but must be aligned on 64-byte boundary. However, the EPBASE field has a granularity of 2 Kbytes; in practice, the queue head should be 2-Kbyte aligned.

On the USB OTG module, the host and device mode functions share this register. In device mode, it is the EPLISTADDR register; in host mode, it is the ASYNCLISTADDR register. See [Section 24.3.4.7, “Current Asynchronous List Address Register \(ASYNCLISTADDR\),”](#) for more information.

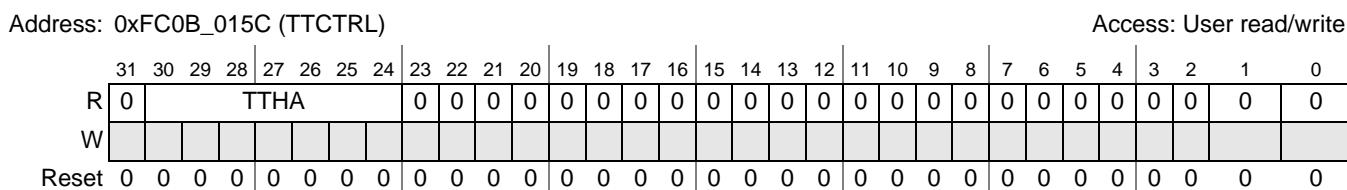


**Figure 24-24. Endpoint List Address Register (EPLISTADDR)**

**Table 24-27. EPLISTADDR Field Descriptions**

Field	Description
31–11 EPBASE	Endpoint list address. Correspond to memory address signals [31:11] References a list of up to 32 queue heads (i.e. one queue head per endpoint and direction). Address of the top of the endpoint list.
10–0	Reserved, must be cleared.

#### 24.3.4.9 Host TT Asynchronous Buffer Control (TTCTRL)



**Figure 24-25. Host TT Asynchronous Buffer Control (TTCTRL)**

**Table 24-28. TTCTRL Field Descriptions**

Field	Description
31	Reserved, must be cleared.
30–24 TTHA	TT Hub Address. This field is used to match against the Hub Address field in a QH or siTD to determine if the packet is routed to the internal TT for directly attached FS/LS devices. If the hub address in the QH or siTD does not match this address then the packet is broadcast on the high speed ports destined for a downstream HS hub with the address in the QH or siTD.
23–0	Reserved, must be cleared.

**24.3.4.10 Master Interface Data Burst Size Register (BURSTSIZE)**

This register is not defined in the EHCI specification. BURSTSIZE dynamically controls the burst size during data movement on the initiator (master) interface.

Address: 0xFC0B_0160 (BURSTSIZE)																Access: User read/write																		
R	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		
W																	TXPBURST				RXPBURST													
Reset	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	0	0	1	0	1	0	1

**Figure 24-26. Master Interface Data Burst Size (BURSTSIZE)****Table 24-29. BURSTSIZE Field Descriptions**

Field	Description
31–16	Reserved, must be cleared.
15–8 TXPBURST	Programable TX burst length. Represents the maximum length of a burst in 32-bit words while moving data from system memory to the USB bus. Must not be set to greater than 16.
7–0 RXPBURST	Programable RX burst length. This register represents the maximum length of a burst in 32-bit words while moving data from the USB bus to system memory. Must not be set to greater than 16.

**24.3.4.11 Transmit FIFO Tuning Control Register (TXFILLTUNING)**

This register is not defined in the EHCI specification. The TXFILLTUNING register controls performance tuning associated with how the module posts data to the TX latency FIFO before moving the data onto the USB bus. The specific areas of performance include how much data to post into the FIFO and an estimate for how long that operation takes in the target system.

Definitions:

$$T_0 = \text{Standard packet overhead}$$

$$T_I = \text{Time to send data payload}$$

$$T_s = \text{Total packet flight time (send-only) packet } (T_s = T_0 + T_I)$$

$$T_{ff} = \text{Time to fetch packet into TX FIFO up to specified level}$$

$$T_p = \text{Total packet time (fetch and send) packet } (T_p = T_{ff} + T_s)$$

Upon discovery of a transmit (OUT/SETUP) packet in the data structures, the host controller checks to ensure  $T_p$  remains before the end of the (micro)frame. If so, it pre-fills the TX FIFO. If at anytime during the pre-fill operation the time remaining the (micro)frame is less than  $T_s$ , packet attempt ceases and tries at a later time. Although this is not an error condition and the module eventually recovers, a mark is made in the scheduler health counter to mark the occurrence of a back-off event. When a back-off event is detected, the partial packet fetched may need to be discarded from the latency buffer to make room for periodic traffic beginning after the next SOF. Too many back-off events can waste bandwidth and power on the system bus and should be minimized (not necessarily eliminated). The TSCHHEALTH ( $T_{ff}$ ) parameter described below can minimize back-offs.

Address: 0xFC0B_0164 (TXFILLTUNING)																																
	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	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																																
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	

Figure 24-27. Transmit FIFO Tuning Controls (TXFILLTUNING)

Table 24-30. TXFILLTUNING Field Descriptions

Field	Description
31–22	Reserved, must be cleared.
21–16 TXFIFOTHRES	FIFO burst threshold. Controls the number of data bursts that are posted to the TX latency FIFO in host mode before the packet begins on the bus. The minimum value is 2 and this value should be as low as possible to maximize USB performance. Systems with unpredictable latency and/or insufficient bandwidth can use a higher value where the FIFO may underrun because the data transferred from the latency FIFO to USB occurs before it can replenish from system memory. This value is ignored if the USBMODE[SDIS] bit is set. When the USBMODE[SDIS] bit is set, the host controller behaves as if TXFIFOTHRES is set to its maximum value.
15–13	Reserved, must be cleared.

**Table 24-30. TXFILLTUNING Field Descriptions (continued)**

Field	Description
12–8 TXSCHHEALTH	Scheduler health counter. These bits increment when the host controller fails to fill the TX latency FIFO to the level programmed by TXFIFOTHRES before running out of time to send the packet before the next SOF. This health counter measures the number of times this occurs to provide feedback to selecting a proper TXSCHOH. Writing to this register clears the counter and this counter stops counting after reaching the maximum of 31.
7–0 TXSCHOH	Scheduler overhead. These bits add an additional fixed offset to the schedule time estimator described as $T_{ff}$ . As an approximation, the value chosen for this register should limit the number of back-off events captured in the TXSCHHEALTH field to less than 10 per second in a highly utilized bus. Choosing a value too high for this register is not desired as it can needlessly reduce USB utilization. The time unit represented in this register is 1.267 $\mu$ s when a device connects in high-speed mode. The time unit represented in this register is 6.333 $\mu$ s when a device connects in low-/full-speed mode.  For most applications, TXSCHOH can be set to 4 or less. A good value to begin with is: $\frac{\text{TXFIFOTHRES} \times (\text{BURSTSIZE} \times 4)}{40 \times \text{TimeUnit}}$ <b>Eqn. 24-1</b> Always rounded to the next higher integer. <i>TimeUnit</i> is 1.267 or 6.333 as noted earlier in this description. For example, if TXFIFOTHRES is 5 and BURSTSIZE is 8, set TXSCHOH to $5 \times (8 \times 4) / (40 \times 1.267)$ equals 4 for a high-speed link. If this value of TXSCHOH results in a TXSCHHEALTH count of 0 per second, low the value by 1 if optimizing performance is desired. If TXSCHHEALTH exceeds 10 per second, raise the value by 1. If streaming mode is disabled via the USBMODE register, treat TXFIFOTHRES as the maximum value for purposes of the TXSCHOH calculation.

#### 24.3.4.12 ULPI Register Access (ULPI\_VIEWPORT)

The register provides indirect access to the ULPI PHY register set. Although the controller modules perform access to the ULPI PHY register set, there may be circumstances where software may need direct access.

##### NOTE

Be advised that writes to the ULPI through the ULPI viewport can substantially harm standard USB operations. Currently no usage model has been defined where software should need to execute writes directly to the ULPI. Executing read operations though the ULPI viewport should have no harmful side effects to standard USB operations. Also, if the ULPI interface is not enabled, this register is always read cleared.

Address: 0xFC0B_0170 (ULPI_VIEWPORT)																Access: User read/write								
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8
W	ULPI_WU	ULPI_RUN	ULPI_RW	0	ULPI_SS	ULPI_PORT	ULPI_ADDR								ULPI_DATRD								ULPI_DATWR	
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

**Figure 24-28. ULPI Register Access (ULPI VIEWPORT)**

**Table 24-31. ULPI VIEWPORT Field Descriptions**

<b>Field</b>	<b>Description</b>
31 ULPI_WU	ULPI wake-up. Setting this bit begins the wake-up operation. This bit automatically clears after the wake-up is complete. After this bit is set, it can not be cleared by software. <b>Note:</b> The driver must never execute a wake-up and a read/write operation at the same time.
30 ULPI_RUN	ULPI run. Setting this bit begins a read/write operation. This bit automatically clears after the read/write is complete. After this bit is set, it can not be cleared by software. <b>Note:</b> The driver must never execute a wake-up and a read/write operation at the same time.
29 ULPI_RW	Read/write. Selects between running a read or write operation to the ULPI. 0 Read 1 Write
28	Reserved, should be cleared.
27 ULPI_SS	Sync state. Represents the state of the ULPI interface. Before reading this bit, the ULPI_PORT field should be set accordingly if used with the multi-port host. Otherwise, this field should always remain 0. 0 Any other state (that is, carkit, serial, low power). 1 Normal sync state.
26–24 ULPI_PORT	Port number. For wake-up or read/write operations this value selects the port number to which the ULPI PHY is attached. Valid values are 0 and 1.
23–16 ULPI_ADDR	Data address. When a read or write operation is commanded, the address of the operation is written to this field.
15–8 ULPI_DATRD	Data read. After a read operation completes, the result is placed in this field.
7–0 ULPI_DATWR	Data write. When a write operation is commanded, the data to be sent is written to this field.

There are two operations that can be performed with the ULPI viewport, wake-up and read/write operations. The wake-up operation is used to put the ULPI interface into normal operation mode and re-enable the clock if necessary. A wake-up operation is required before accessing the registers when the ULPI interface is operating in low power mode, serial mode, or carkit mode. The ULPI state can be determined by reading the sync state bit (ULPI\_SS). If this bit is set, then the ULPI interface is running in normal operating mode and can accept read/write operations. If ULPI\_SS is cleared, then read/write operations are not executed. Undefined behavior results if a read or write operation is performed when ULPI\_SS is cleared. To execute a wake-up operation, write all 32-bits of the ULPI VIEWPORT where ULPI\_PORT is constructed appropriately and the ULPI\_WU bit is set and the ULPI\_RUN bit is cleared. Poll the ULPI VIEWPORT until ULPI\_WU is cleared for the operation to complete.

To execute a read or write operation, write all 32-bits of the ULPI VIEWPORT where ULPI\_DATWR, ULPI\_ADDR, ULPI\_PORT, ULPI\_RW are constructed appropriately and the ULPI\_RUN bit is set. Poll the ULPI VIEWPORT until ULPI\_RUN is cleared for the operation to complete. For read operations, ULPI\_DATRD is valid after ULPI\_RUN is cleared.

The polling method above can be replaced with interrupts using the ULPI interrupt defined in the USBSTS and USBINTR registers. When a wake-up or read/write operation completes, the ULPI interrupt is set.

### 24.3.4.13 Configure Flag Register (CONFIGFLAG)

This EHCI register is not used in this implementation. A read from this register returns a constant of a 0x0000\_0001 to indicate that all port routings default to this host controller.

Address: 0xFC0B_0180 (CONFIGFLAG)																																	
R	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	
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	1		

Figure 24-29. Configure Flag Register (CONFIGFLAG)

Table 24-32. CONFIGFLAG Field Descriptions

Field	Description
31–0	Reserved. (0x0000_0001, all port routings default to this host)

### 24.3.4.14 Port Status and Control Registers (PORTSCn)

Both USB modules contain a single PORTSC register. This register only resets when power is initially applied or in response to a controller reset. Initial conditions of a port are:

- No device connected
- Port disabled

If the port has port power control, this state remains until software applies power to the port by setting port power to one.

For the USB OTG module in device mode, the USB OTG controller does not support power control. Port control in device mode is used only for status port reset, suspend, and current connect status. It is also used to initiate test mode or force signaling, and allows software to place the PHY into low-power suspend mode and disable the PHY clock.

Address: 0xFC0B_0184 (PORTSC1)																																	Access: User read/write							
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	PTC																							
W	PTS	1	0		PSPD	0	PFSC	PHCD	WKOC	WKDS	WLCN		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0						
Reset	1	1	1	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	0	0	0	0	0					

R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	PIC																				
W	PO	PP			LS	HSP	PR	SUSP	FPR	OCC	OCA	PEC	PE	CSC	CCS																						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	w1c	w1c																			

Figure 24-30. Port Status and Control Register (PORTSC1)

**Table 24-33. PORTSC1 Field Descriptions**

<b>Field</b>	<b>Description</b>
31–30 PTS	Port transceiver select. Controls which parallel transceiver interface is selected. This field is read-only for the host and is set to 11 to indicate a FS/LS on-chip transceiver. The following settings apply for the OTG module: 00 Reserved 01 Reserved 10 ULPI parallel interface 11 FS/LS on-chip transceiver This bit is not defined in the EHCI specification.
29	Reserved, must be set.
28	Reserved, must be cleared.
27–26 PSPD	Port speed. This read-only register field indicates the speed the port operates. This bit is not defined in the EHCI specification. 00 Full speed 01 Low speed 10 High speed 11 Undefined
25	Reserved, must be cleared.
24 PFSC	Port force full-speed connect. Disables the chirp sequence that allows the port to identify itself as a HS port. useful for testing FS configurations with a HS host, hub, or device. Not defined in the EHCI specification. 0 Allow the port to identify itself as high speed. 1 Force the port to only connect at full speed. This bit is for debugging purposes.
23 PHCD	PHY low power suspend. This bit is not defined in the EHCI specification. Host mode (USB host and USB OTG): The PHY can be placed into low-power suspend when downstream device is put into suspend mode or when no downstream device connects. Software completely controls low-power suspend. Device mode (USB OTG only): For the USB OTG module in device mode, the PHY can be put into low power suspend when the device is not running (USBCMD[RS] = 0) or suspend signaling is detected on the USB. The PHCD bit is cleared automatically when the resume signaling is detected or when forcing port resumes. 0 Normal PHY operation. 1 Signal the PHY to enter low-power suspend mode Reading this bit indicates the status of the PHY.
22 WKOC	Wake on over-current enable. Enables the port to be sensitive to over-current conditions as wake-up events. This field is 0 if the PP bit is cleared. In host mode, this bit can work with an external power control circuit.
21 WKDS	Wake on disconnect enable. Enables the port to be sensitive to device disconnects as wake-up events. This field is 0 if the PP bit is cleared or the module is in device mode (USB OTG-only). In host mode, this bit can work with an external power control circuit.
20 WLCN	Wake on connect enable. Enables the port to be sensitive to device connects as wake-up events. This field is 0 if the PP bit is cleared or the module is in device mode (USB OTG-only). In host mode, this can work with an external power control circuit.

**Table 24-33. PORTSC1 Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>
19–16 PTC	<p>Port test control. Any value other than 0 indicates the port operates in test mode. Refer to Chapter 7 of the <i>USB Specification Revision 2.0</i> for details on each test mode.</p> <ul style="list-style-type: none"> <li>0000 Not enabled.</li> <li>0001 J_STATE</li> <li>0010 K_STATE</li> <li>0011 SEQ_NAK</li> <li>0100 Packet</li> <li>0101 FORCE_ENABLE_HS</li> <li>0110 FORCE_ENABLE_FS</li> <li>0111 FORCE_ENABLE_LS</li> <li>Else Reserved.</li> </ul> <p><b>Note:</b> The FORCE_ENABLE_FS and FORCE_ENABLE_LS settings are extensions to the test mode support in the EHCI specification. Writing the PTC field to any of the FORCE_ENABLE values forces the port into the connected and enabled state at the selected speed. Then clearing the PTC field allows the port state machines to progress normally from that point.</p>
15–14 PIC	<p>Port indicator control. Controls the link indicator signals and is valid for host mode only. Refers to the <i>USB Specification Revision 2.0</i> for a description on how these bits are used.</p> <ul style="list-style-type: none"> <li>00 Off</li> <li>01 Amber</li> <li>10 Green</li> <li>11 Undefined</li> </ul> <p>This field is output from the module on the USB port control signals for use by an external LED driving circuit. For this device's USB host module, the port indicator signals are implemented as status bits within the CCM. On the USB OTG module this feature is not implemented, therefore this field is read-only and is always cleared.</p>
13 PO	Port owner. Port owner handoff is not implemented in this design, therefore this bit is read-only and is always cleared.
12 PP	<p>Port power. Represents the current setting of the port power control switch (0 equals off, 1 equals on). When power is not available on a port (PP = 0), it is non-functional and does not report attaches, detaches, etc.</p> <p>When an over-current condition is detected on a powered port, the host controller driver from a 1 to a 0 (removing power from the port) transitions the PP bit in each affected port.</p>
11–10 LS	<p>Line status. Reflects current logical levels of the USB DP (bit 11) and DM (bit 10) signal lines. In host mode, the line status by the host controller driver is not necessary (unlike EHCI) because hardware manages the connection of FS and LS. In device mode, LS by the device controller is not necessary.</p> <ul style="list-style-type: none"> <li>00 SE0</li> <li>01 J-state</li> <li>10 K-state</li> <li>11 Undefined</li> </ul>
9 HSP	<p>High speed port. Indicates if the host/device connected is in high speed mode.</p> <ul style="list-style-type: none"> <li>0 FS or LS</li> <li>1 HS</li> </ul> <p><b>Note:</b> This bit is redundant with the PSPD bit field.</p>

**Table 24-33. PORTSC1 Field Descriptions (continued)**

Field	Description												
8 PR	<p>Port reset. This field is cleared if the PP bit is cleared.</p> <p>Host mode (USB host and USB OTG):</p> <p>When software sets this bit the bus-reset sequence as defined in the <i>USB Specification Revision 2.0</i> starts. This bit automatically clears after the reset sequence completes. This behavior is different from EHCI where the host controller driver is required to clear this bit after the reset duration is timed in the driver.</p> <p>Device mode (USB OTG only):</p> <p>This bit is a read-only status bit. Device reset from the USB bus is also indicated in the USBSTS register.</p> <ul style="list-style-type: none"> <li>0 Port is not in reset.</li> <li>1 Port is in reset.</li> </ul>												
7 SUSP	<p>Suspend</p> <ul style="list-style-type: none"> <li>0 Port not in suspend state.</li> <li>1 Port in suspend state.</li> </ul> <p>Host mode (USB host and USB OTG):</p> <p>The PE and SUSP bits define the port state as follows:</p> <table border="1" data-bbox="633 728 1106 939"> <thead> <tr> <th data-bbox="687 728 763 777">PE</th><th data-bbox="763 728 866 777">SUSP</th><th data-bbox="866 728 1106 777">Port State</th></tr> </thead> <tbody> <tr> <td data-bbox="687 777 763 825">0</td><td data-bbox="763 777 866 825">x</td><td data-bbox="866 777 1106 825">Disable</td></tr> <tr> <td data-bbox="687 825 763 874">1</td><td data-bbox="763 825 866 874">0</td><td data-bbox="866 825 1106 874">Enable</td></tr> <tr> <td data-bbox="687 874 763 922">1</td><td data-bbox="763 874 866 922">1</td><td data-bbox="866 874 1106 922">Suspend</td></tr> </tbody> </table> <p>When in suspend state, downstream propagation of data is blocked on this port, except for port reset. The blocking occurs at the end of the current transaction if a transaction was in progress when this bit was set. In the suspend state, the port is sensitive to resume detection. The bit status does not change until the port is suspended and there may be a delay in suspending a port if there is a transaction currently in progress on the USB.</p> <p>The module unconditionally clears this bit when software clears the FPR bit. The host controller ignores clearing this bit. If host software sets this bit when the port is not enabled (PE = 0), the results are undefined.</p> <p>This field is cleared if the PP bit is cleared in host mode.</p> <p>Device mode (USB OTG only):</p> <p>In device mode, this bit is a read-only status bit.</p>	PE	SUSP	Port State	0	x	Disable	1	0	Enable	1	1	Suspend
PE	SUSP	Port State											
0	x	Disable											
1	0	Enable											
1	1	Suspend											
6 FPR	<p>Force Port Resume. This bit is not-EHCI compatible.</p> <ul style="list-style-type: none"> <li>0 No resume (K-state) detected/driven on port.</li> <li>1 Resume detected/driven on port.</li> </ul> <p>Host mode (USB host and USB OTG):</p> <p>Software sets this bit to drive resume signaling. The controller sets this bit if a J-to-K transition is detected while the port is in suspend state (PE = SUSP = 1), which in turn sets the USBSTS[PCI] bit. This bit automatically clears after the resume sequence is complete. This behavior is different from EHCI where the host controller driver is required to clear this bit after the resume duration is timed in the driver.</p> <p>When the controller owns the port, the resume sequence follows the defined sequence documented in the <i>USB Specification Revision 2.0</i>. The resume signaling (full-speed K) is driven on the port as long as this bit remains set. This bit remains set until the port switches to the high-speed idle. Clearing this bit has no affect because the port controller times the resume operation to clear the bit the port control state switches to HS or FS idle.</p> <p>This field is cleared if the PP bit is cleared in host mode.</p> <p>Device mode (USB OTG only):</p> <p>After the device is in suspend state for 5 ms or more, software must set this bit to drive resume signaling before clearing. The device controller sets this bit if a J-to-K transition is detected while port is in suspend state, which in turn sets the USBSTS[PCI] bit. The bit is cleared when the device returns to normal operation.</p>												

**Table 24-33. PORTSC1 Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>
5 OCC	Over-current change. Indicates a change to the OCA bit. Software clears this bit by writing a 1. For host mode, the user can provide over-current detection to the USBn_PWRFAULT signal for this condition. For device-only implementations (USB OTG only), this bit must always be cleared. 0 No over-current. 1 Over-current detect.
4 OCA	Over-current active. This bit automatically transitions from 1 to 0 when the over-current condition is removed. For host/OTG implementations, the user can provide over-current detection to the USBn_PWRFAULT signal for this condition. For device-only implementations (USB OTG only), this bit must always be cleared. 0 Port not in over-current condition. 1 Port currently in over-current condition.
3 PEC	Port enable/disable change. For the root hub, this bit gets set only when a port is disabled due to disconnect on the port or due to the appropriate conditions existing at the EOF2 point (See Chapter 11 of the <i>USB Specification</i> ). Software clears this by writing a 1 to it. In device mode (USB OTG only), the device port is always enabled. (This bit is zero). 0 No change. 1 Port disabled. This field is cleared if the PP bit is cleared.
2 PE	Port enabled/disabled. Host mode (USB host and USB OTG): Ports can only be enabled by the controller as a part of the reset and enable sequence. Software cannot enable a port by setting this bit. A fault condition (disconnect event or other fault condition) or host software can disable ports. The bit status does not change until the port state actually changes. There may be a delay in disabling or enabling a port due to other host and bus events. When the port is disabled, downstream propagation of data is blocked except for reset. This field is cleared if the PP bit is cleared in host mode. Device mode (USB USB OTG only): The device port is always enabled. (This bit is set).
1 CSC	Connect change status. Host mode (USB host and USB OTG): This bit indicates a change occurred in the port's current connect status. The controller sets this bit for all changes to the port device connect status, even if system software has not cleared an existing connect status change. For example, the insertion status changes twice before system software has cleared the changed condition; hub hardware is setting an already-set bit (i.e., the bit remains set). Software clears this bit by writing a 1 to it. This field is cleared if the PP bit is cleared. 0 No change. 1 Connect status has changed. In device mode (USB USB OTG only), this bit is undefined.
0 CCS	Current connect status. Indicates that a device successfully attaches and operates in high speed or full speed as indicated by the PSPD bit. If clear, the device did not attach successfully or forcibly disconnects by the software clearing the USBCMD[RUN] bit. It does not state the device disconnected or suspended. This field is cleared if the PP bit is cleared in host mode. 0 No device present (host mode) or attached (device mode) 1 Device is present (host mode) or attached (device mode)

#### 24.3.4.15 On-the-Go Status and Control Register (OTGSC)

This register is not defined in the EHCI specification. The host controller implements one OTGSC register corresponding to port 0 of the host controller.

The OTGSC register has four sections:

- OTG interrupt enables (read/write)
- OTG interrupt status (read/write to clear)
- OTG status inputs (read-only)
- OTG controls (read/write)

The status inputs de-bounce using a 1 ms time constant. Values on the status inputs that do not persist for more than 1 ms do not cause an update of the status inputs or an OTG interrupt.

Address: 0xFC0B_01A4 (OTGSC)																Access: User read/write							
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16							
W	0	DPIE	1MSE	BSEIE	BSVIE	ASVIE	AVVIE	IDIE	0	DPIS	1MSS	BSEIS	BSVIS	ASVIS	AVVIS	IDIS							
Reset	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0							
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0							
W	0	DPS	1MST	BSE	BSV	ASV	AVV	ID	0	0	IDPU	DP	OT	0	VC	VD							
Reset	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0							

Figure 24-31. On-the-Go Status and Control Register (OTGSC)

Table 24-34. OTGSC Field Descriptions

Field	Description
31	Reserved, must be cleared.
30 DPIE	Data pulse interrupt enable. 0 Disable 1 Enable
29 1MSE	1 millisecond timer interrupt enable. 0 Disable 1 Enable
28 BSEIE	B session end interrupt enable. 0 Disable 1 Enable
27 BSVIE	B session valid interrupt enable. 0 Disable 1 Enable
26 ASVIE	A session valid interrupt enable. 0 Disable 1 Enable
25 AVVIE	A VBUS valid interrupt enable. 0 Disable 1 Enable

**Table 24-34. OTGSC Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>
24 IDIE	USB ID interrupt enable. 0 Disable 1 Enable
23	Reserved, must be cleared.
22 DPIS	Data pulse interrupt status. Indicates when data bus pulsing occurs on DP or DM. Data bus pulsing only detected when USBMODE[CM] equals 11 and PORTSC0[PP] is cleared. Software must write a 1 to clear this bit.
21 1MSS	1 millisecond timer interrupt status. This bit is set once every millisecond. Software must write a 1 to clear this bit.
20 BSEIS	B session end interrupt status. Indicates when VBUS falls below the B session end threshold. Software must write a 1 to clear this bit.
19 BSVIS	B session valid interrupt status. Indicates when VBUS rises above or falls below the B session valid threshold (0.8 VDC). Software must write a 1 to clear this bit.
18 ASVIS	A session valid interrupt status. Indicates when VBUS rises above or falls below the A session valid threshold (0.8 VDC). Software must write a 1 to clear this bit.
17 AVVIS	A VBUS valid interrupt status. Indicates when VBUS rises above or falls below the VBUS valid threshold (4.4 VDC) on an A device. Software must write a 1 to clear this bit.
16 IDIS	USB ID interrupt status. Indicates when a change on the ID input is detected. Software must write a 1 to clear this bit.
15	Reserved, must be cleared.
14 DPS	Data bus pulsing status. 0 No pulsing on port. 1 Pulsing detected on port.
13 1MST	1 millisecond timer toggle. This bit toggles once per millisecond.
12 BSE	B session end. 0 VBus is above B session end threshold. 1 VBus is below B session end threshold.
11 BSV	B Session valid. 0 VBus is below B session valid threshold. 1 VBus is above B session valid threshold.
10 ASV	A Session valid. 0 VBus is below A session valid threshold. 1 VBus is above A session valid threshold.
9 AVV	A VBus valid. 0 VBus is below A VBus valid threshold. 1 VBus is above A VBus valid threshold.
8 ID	USB ID. 0 A device. 1 B device.
7–6	Reserved, must be cleared.

**Table 24-34. OTGSC Field Descriptions (continued)**

Field	Description
5 IDPU	ID Pull-up. Provides control over the ID pull-up resistor. 0 Disable pull-up. ID input not sampled. 1 Enable pull-up.
4 DP	Data pulsing. 0 The pull-up on DP is not asserted. 1 The pull-up on DP is asserted for data pulsing during SRP.
3 OT	OTG Termination. This bit must be set with the OTG module in device mode. 0 Disable pull-down on DM. 1 Enable pull-down on DM.
2	Reserved, must be cleared.
1 VC	VBUS charge. Setting this bit causes the VBUS line to charge. This is used for VBus pulsing during SRP.
0 VD	VBUS discharge. Setting this bit causes VBUS to discharge through a resistor.

#### 24.3.4.16 USB Mode Register (USBMODE)

This register is not defined in the EHCI specification. It controls the operating mode of the module.

Address: 0xFC0B_01A8 (USBMODE)																Access: User read/write			
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16			
W																			
Reset	0	0	0	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	0	0	0	0	0	0	0	0	0	0	0	SDIS	SLOM	ES	CM			
W																			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 24-32. USB Mode Register (USBMODE)**

**Table 24-35. USBMODE Field Descriptions**

Field	Description
31–5	Reserved, must be cleared.
4 SDIS	<p>Stream disable.            0 Inactive.            1 Active.</p> <p>Host mode (USB host and USB OTG):            Setting this bit ensures that overruns/underruns of the latency FIFO are eliminated for low bandwidth systems where the RX and TX buffers are sufficient to contain the entire packet. Enabling stream disable also has the effect of ensuring the TX latency fills to capacity before the packet launches onto the USB.            Time duration to pre-fill the FIFO becomes significant when stream disable is active. See TXFILLTUNING to characterize the adjustments needed for the scheduler when using this feature.            Also, in systems with high system bus utilization, setting this bit ensures no overruns or underruns during operation at the expense of link utilization. SDIS can be left clear and the rules under the description of the TXFILLTUNING register can limit underruns/overruns for those who desire optimal link performance.</p> <p>Device mode (USB OTG only):            Setting this bit disables double priming on RX and TX for low bandwidth systems. This mode ensures that when the RX and TX buffers are sufficient to contain an entire packet that the standard double buffering scheme is disabled to prevent overruns/underruns in bandwidth limited systems.            In high-speed mode, all packets received are responded to with a NYET handshake when stream disable is active.</p>
3 SLOM	<p>Setup lockout mode. For the module in device mode, this bit controls behavior of the setup lock mechanism. See <a href="#">Section 24.5.3.4.4, “Control Endpoint Operation.”</a></p> <p>0 Setup lockouts on.            1 Setup lockouts off (software requires use of the USBCMD[SUTW] bit).</p>
2 ES	<p>Endian select. Controls the byte ordering of the transfer buffers to match the host microprocessor bus architecture. The bit fields in the register interface and the DMA data structures (including the setup buffer within the device QH) are unaffected by the value of this bit, because they are based upon 32-bit words.</p> <p>0 Little endian. First byte referenced in least significant byte of 32-bit word.            1 Big endian. First byte referenced in most significant byte of 32-bit word.</p> <p><b>Note:</b> For proper operation, this bit must be set for this ColdFire device.</p>
1–0 CM	<p>Controller mode. This register can be written only once after reset. If necessary to switch modes, software must reset the controller by writing to the USBCMD[RST] bit before reprogramming this register.</p> <p>00 Idle (default for the USB OTG module)            01 Reserved            10 Device controller            11 Host controller (default for the USB host module)</p> <p><b>Note:</b> The USB OTG module must be initialized to the desired operating mode after reset.</p>

#### 24.3.4.17 Endpoint Setup Status Register (EPSETUPSR)

This register is not defined in the EHCI specification. This register contains the endpoint setup status and is used only by the USB OTG module in device mode.

Address: 0xFC0B\_01AC (EPSETUPSR)

Access: User read/write

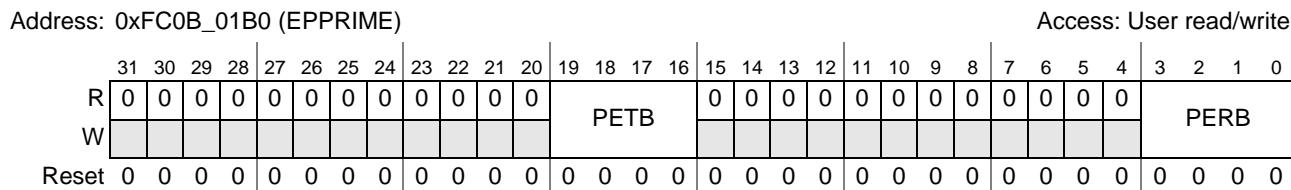
**Figure 24-33. Endpoint Setup Status Register (EPSETUPSR)**

**Table 24-36. EPSETUPSR Field Descriptions**

Field	Description
31–4	Reserved, must be cleared.
3–0 EPSETUPSTAT	Setup endpoint status. For every setup transaction received, a corresponding bit in this field is set. Software must clear or acknowledge the setup transfer by writing a 1 to a respective bit after it has read the setup data from the queue head. The response to a setup packet, as in the order of operations and total response time, is crucial to limit bus time outs while the setup lockout mechanism engages.

#### 24.3.4.18 Endpoint Initialization Register (EPPRIME)

This register is not defined in the EHCI specification. This register is used to initialize endpoints and is used only by the USB OTG module in device mode.

**Figure 24-34. Endpoint Initialization Register (EPPRIME)****Table 24-37. EPPRIME Field Descriptions**

Field	Description
31–20	Reserved, must be cleared.
19–16 PETB	Prime endpoint transmit buffer. For each endpoint, a corresponding bit requests that a buffer be prepared for a transmit operation to respond to a USB IN/INTERRUPT transaction. Software must write a 1 to the corresponding bit when posting a new transfer descriptor to an endpoint. Hardware automatically uses this bit to begin parsing for a new transfer descriptor from the queue head and prepare a transmit buffer. Hardware clears this bit when associated endpoint(s) is (are) successfully primed. <b>Note:</b> These bits are momentarily set by hardware during hardware re-priming operations when a dTD retires, and the dQH updates.
15–4	Reserved, must be cleared.
3–0 PERB	Prime endpoint receive buffer. For each endpoint, a corresponding bit requests that a buffer be prepared for a receive operation to respond to a USB OUT transaction. Software must write a 1 to the corresponding bit when posting a new transfer descriptor to an endpoint. Hardware automatically uses this bit to begin parsing for a new transfer descriptor from the queue head and prepare a receive buffer. Hardware clears this bit when associated endpoint(s) is (are) successfully primed. <b>Note:</b> These bits are momentarily set by hardware during hardware re-priming operations when a dTD retires, and the dQH updates.

#### 24.3.4.19 Endpoint Flush Register (EPFLUSH)

This register is not defined in the EHCI specification. This register used only by the USB OTG module in device mode.

Address: 0xFC0B_01B4 (EPFLUSH)																Access: User read/write												
R	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
W	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
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
	FETB																0	0	0	0	0	0	0	0	0	0	0	0
	FERB																0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-35. Endpoint Flush Register (EPFLUSH)

Table 24-38. EPFLUSH Field Descriptions

Field	Description
31–20	Reserved, must be cleared.
19–16 FETB	Flush endpoint transmit buffer. Writing a 1 to a bit in this field causes the associated endpoint to clear any primed buffers. If a packet is in progress for an associated endpoint, that transfer continues until completion. Hardware clears this register after the endpoint flush operation is successful.
15–4	Reserved, must be cleared.
3–0 FERB	Flush endpoint receive buffer. Writing a 1 to a bit in this field causes the associated endpoint to clear any primed buffers. If a packet is in progress for an associated endpoint, that transfer continues until completion. Hardware clears this register after the endpoint flush operation is successful. FERB[3] corresponds to endpoint 3.

#### 24.3.4.20 Endpoint Status Register (EPSR)

This register is not defined in the EHCI specification. This register is only used by the USB OTG module in device mode.

Address: 0xFC0B_01B8 (EPSR)																Access: User read-only												
R	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
W	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
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
	ETBR																0	0	0	0	0	0	0	0	0	0	0	0
	ERBR																0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-36. Endpoint Status Register (EPSR)

Table 24-39. EPSR Field Descriptions

Field	Description
31–20	Reserved, must be cleared.
19–16 ETBR	Endpoint transmit buffer ready. One bit for each endpoint indicates status of the respective endpoint buffer. The hardware sets this bit in response to receiving a command from a corresponding bit in the EPPRIME register. A constant delay exists between setting a bit in the EPPRIME register and endpoint indicating ready. This delay time varies based upon the current USB traffic and the number of bits set in the EPPRIME register. USB reset, USB DMA system, or EPFLUSH register clears the buffer ready. ETBR[3] (bit 19) corresponds to endpoint 3. <b>Note:</b> Hardware momentarily clears these bits during hardware endpoint re-priming operations when a dTD is retired, and the dQH is updated.

**Table 24-39. EPSR Field Descriptions (continued)**

Field	Description
15–4	Reserved, must be cleared.
3–0 ERBR	Endpoint receive buffer ready. One bit for each endpoint indicates status of the respective endpoint buffer. The hardware sets this bit in response to receiving a command from a corresponding bit in the EPPRIME register. A constant delay exists between setting a bit in the EPPRIME register and endpoint indicating ready. This delay time varies based upon the current USB traffic and the number of bits set in the EPPRIME register. USB reset, USB DMA system, or EPFLUSH register clears the buffer ready. ERBR[3] (bit 19) corresponds to endpoint 3. <b>Note:</b> Hardware momentarily clears these bits during hardware endpoint re-priming operations when a dTD is retired, and the dQH is updated.

#### 24.3.4.21 Endpoint Complete Register (EPCOMPLETE)

This register is not defined in the EHCI specification. This register is used only by the USB OTG module in device mode.

Address: 0xFC0B_01BC (EPCOMPLETE)																				Access: User read/write							
R																				ETCE							
W																				w1c							
Reset																				ERCE							

**Figure 24-37. Endpoint Complete Register (EPCOMPLETE)****Table 24-40. EPCOMPLETE Field Descriptions**

Field	Description
31–20	Reserved, must be cleared.
19–16 ETCE	Endpoint transmit complete event. Each bit indicates a transmit event (IN/INTERRUPT) occurs and software must read the corresponding endpoint queue to determine the endpoint status. If the corresponding IOC bit is set in the transfer descriptor, this bit is set simultaneously with the USBINT. Writing a 1 clears the corresponding bit in this register. ETCE[3] (bit 19) corresponds to endpoint 3.
15–4	Reserved, must be cleared
3–0 ERCE	Endpoint receive complete event. Each bit indicates a received event (OUT/SETUP) occurs and software must read the corresponding endpoint queue to determine the transfer status. If the corresponding IOC bit is set in the transfer descriptor, this bit is set simultaneously with the USBINT. Writing a 1 clears the corresponding bit in this register. ERCE[3] corresponds to endpoint 3.

### 24.3.4.22 Endpoint Control Register 0 (EPCR0)

This register is not defined in the EHCI specification. Every device implements endpoint 0 as a control endpoint.

Address: 0xFC0B\_01C0 (EPCR0)

Access: User 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	0	0	0	0	0	0	0	0	TXE	0	0	0	TXT	0	TXS	0	0	0	0	0	0	0	0	0	RXE	0	0	0	RXT	0	RXS	
W																																

Figure 24-38. Endpoint Control 0 (EPCR0)

Table 24-41. EPCR0 Field Descriptions

Field	Description
31–24	Reserved, must be cleared.
23 TXE	TX endpoint enable. Endpoint zero is always enabled. 1 Enable
22–20	Reserved, must be cleared.
19–18 TXT	TX endpoint type. Endpoint zero is always a control endpoint. 00 Control
17	Reserved, must be cleared.
16 TXS	TX endpoint stall. Software can write a 1 to this bit to force the endpoint to return a STALL handshake to the host. It continues returning STALL until software clears the bit or it automatically clears upon receipt of a new SETUP request. 0 Endpoint OK 1 Endpoint stalled
15–8	Reserved, must be cleared.
7 RXE	RX endpoint enable. Endpoint zero is always enabled. 1 Enabled.
6–4	Reserved, must be cleared.
3–2 RXT	RX endpoint type. Endpoint zero is always a control endpoint. 00 Control
1	Reserved, must be cleared.
0 RXS	RX endpoint stall. Software can write a 1 to this bit to force the endpoint to return a STALL handshake to the host. It continues returning STALL until software clears the bit or it automatically clears upon receipt of a new SETUP request. 0 Endpoint OK 1 Endpoint stalled

### 24.3.4.23 Endpoint Control Register *n* (EPCR*n*)

These registers are not defined in the EHCI specification. There is an EPCR*n* register for each endpoint in a device.

Address: 0xFC0B_01C4 (EPCR1) 0xFC0B_01C8 (EPCR2) 0xFC0B_01CA (EPCR3)												Access: User read/write				
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
W									TXE	0	TXI	0	TXT	TXD	TXS	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W									RXE	0	RXI	0	RXT	RXD	RXS	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 24-39. Endpoint Control Registers (EPCR*n*)

Table 24-42. EPCR*n* Field Descriptions

Field	Description
31–24	Reserved, must be cleared.
23 TXE	TX endpoint enable. 0 Disabled 1 Enabled
22 TXR	TX data toggle reset. When a configuration event is received for this Endpoint, software must write a 1 to this bit to synchronize the data PID's between the host and device. This bit is self-clearing.
21 TXI	TX data toggle inhibit. This bit is used only for test and should always be written as 0. Writing a 1 to this bit causes this endpoint to ignore the data toggle sequence and always transmit DATA0 for a data packet. 0 PID sequencing enabled. 1 PID sequencing disabled.
20	Reserved, must be cleared.
19–18 TXT	TX endpoint type. 00 Control 01 Isochronous 10 Bulk 11 Interrupt <b>Note:</b> When only one endpoint (RX or TX, but not both) of an endpoint pair is used, the unused endpoint should be configured as a bulk type endpoint.
17 TXD	TX endpoint data source. This bit should always be written as 0, which selects the dual port memory/DMA engine as the source.

**Table 24-42. EPCR<sub>n</sub> Field Descriptions (continued)**

Field	Description
16 TXS	TX endpoint stall. This bit sets automatically upon receipt of a SETUP request if this endpoint is not configured as a control endpoint. It clears automatically upon receipt of a SETUP request if this endpoint is configured as a control endpoint. Software can write a 1 to this bit to force the endpoint to return a STALL handshake to the host. It continues returning STALL until software clears this bit or automatically clears as above. 0 Endpoint OK 1 Endpoint stalled
15–8	Reserved, must be cleared.
7 RXE	RX endpoint enable. 0 Disabled 1 Enabled
6 RXR	RX data toggle reset. When a configuration event is received for this endpoint, software must write a 1 to this bit to synchronize the data PIDs between the host and device. This bit is self-clearing.
5 RXI	RX data toggle inhibit. This bit is only for testing and should always be written as 0. Writing a 1 to this bit causes this endpoint to ignore the data toggle sequence and always accept data packets regardless of their data PID. 0 PID sequencing enabled 1 PID sequencing disabled
4	Reserved, must be cleared.
3–2 RXT	RX endpoint type. 00 Control 01 Isochronous 10 Bulk 11 Interrupt <b>Note:</b> When only one endpoint (RX or TX, but not both) of an endpoint pair is used, the unused endpoint should be configured as a bulk type endpoint.
1 RXD	RX endpoint data sink. This bit should always be written as 0, which selects the dual port memory/DMA engine as the sink.
0 RXS	RX endpoint stall. This bit sets automatically upon receipt of a SETUP request if this endpoint is not configured as a control endpoint. It clears automatically upon receipt of a SETUP request if this endpoint is configured as a control endpoint. Software can write a 1 to this bit to force the endpoint to return a STALL handshake to the host. It continues returning STALL until software clears this bit or automatically clears as above, 0 Endpoint OK 1 Endpoint stalled

## 24.4 Functional Description

Each module (USB host and USB OTG) can be broken down into functional sub-blocks as described below.

### 24.4.1 System Interface

The system interface block contains all the control and status registers to allow a core to interface to the module. These registers allow the processor to control the configuration and ascertain the capabilities of the module and, they control the module's operation.

## 24.4.2 DMA Engine

Both USB modules contain local DMA engines. It is responsible for moving all of the data transferred over the USB between the module and system memory. Like the system interface block, the DMA engine block uses a simple synchronous bus signaling protocol.

The DMA controllers must access control information and packet data from system memory. Control information is contained in link list based queue structures. The DMA controllers have state machines able to parse data structures defined in the EHCI specification. In host mode, the data structures are EHCI compliant and represent queues of transfers performed by the host controller, including the split-transaction requests that allow an EHCI controller to direct packets to FS and LS speed devices. In device mode (USB OTG module only), data structures are similar to those in the EHCI specification and used to allow device responses to be queued for each of the active pipes in the device.

## 24.4.3 FIFO RAM Controller

The FIFO RAM controller is used for context information and to control FIFOs between the protocol engine and the DMA controller. These FIFOs decouple the system processor/memory bus requests from the extremely tight timing required by USB.

The use of the FIFO buffers differs between host and device mode operation. In host mode, a single data channel maintains in each direction through the buffer memory. In device mode (USB OTG module only), multiple FIFO channels maintain for each of the active endpoints in the system.

In host mode, the USB host and USB OTG modules use 16-byte transmit buffers and 16-byte receive buffers. For the USB OTG module, device operation uses a single 16-byte receive buffer and a 16-byte transmit buffer for each endpoint.

## 24.4.4 Physical Layer (PHY) Interface

Readers should familiarize themselves with chapter 7 of the *Universal Serial Bus Specification, Revision 2.0*. The USB host and OTG modules contain an on-chip digital to analog transceiver (XCVR) for DP and DN USB network communication. The USB module defaults to FS XCVR operation and can communicate in LS. The USB OTG module may interface to any ULPI compatible PHY as well.

Due to pin-count limitations the USB modules only support certain combinations of PHY interfaces and USB functionality. Refer to the [Table 24-43](#) for more information.

**Table 24-43. USB Network Speed and Required Physical Interface**

USB Mode and Speed	DP and DN On-Chip Analog XCVR Active	I <sup>2</sup> C	FEC	External Integrated Circuit Required
USB Host FS/LS	Yes	No	Yes	See <a href="#">Section 24.4.4.1, “USB On-Chip Transceiver Required External Components”</a>
USB Device FS	Yes	No	Yes	See <a href="#">Section 24.4.4.1, “USB On-Chip Transceiver Required External Components”</a>
Host/Device ULPI HS/FS	No	Yes	No	Maxim

#### 24.4.4.1 USB On-Chip Transceiver Required External Components

USB system operation does not require external components. However, the recommended method ensures driver output impedance, eye diagram, and V<sub>BUS</sub> cable fault tolerance requirements are met. The recommended method is for the DM and DP I/O pads to connect through series resistors (approximately 33 Ω each) to the USB connector on the application printed circuit board (PCB). Additionally, signal quality optimizes when these 33 Ω resistors are mounted close to the processor rather than closer to the USB board level connector.

##### NOTE

Internal pull-down resistors are included that keep the DP and DM ports in a known quiescent state when the USB port is not used or when a USB cable is not connected.

Also included is an internal 1.5k Ω pull-up resistor on DP controlled by the CCM. (See [Chapter 10, “Chip Configuration Module \(CCM\),”](#) for more details.) This allows the OTG module to operate in full-speed device operation. Host operation requires this internal resistor to be disabled via the CCM, and 15k Ω external resistors to connect from DP and DM signals to ground.

### 24.5 Initialization/Application Information

#### 24.5.1 Host Operation

Enhanced Host Controller Interface (EHCI) Specification defines the general operational model for the USB modules in host mode. The EHCI specification describes the register-level interface for a host controller for USB Revision 2.0. It includes a description of the hardware/software interface between system software and host controller hardware. The next section has information about the initialization of the USB modules; however, full details of the EHCI specification are beyond the scope of this document.

##### 24.5.1.1 Host Controller Initialization

After initial power-on or module reset (via the USBCMD[RST] bit), all of the operational registers are at default values, as illustrated in the register memory map in [Table 24-4](#).

To initialize the host controller, software must:

1. Optionally set streaming disable in the USBMODE[SDIS] bit.
2. Optionally modify the BURSTSIZE register.
3. Program the PORTSC1[PTS] field if using a non-ULPI PHY.
4. Optionally write the appropriate value to the USBINTR register to enable the desired interrupts.
5. Set the USBMODE[CM] field to enable host mode, and set the USBMODE[ES] bit for big endian operation.
6. Write the USBCMD register to set the desired interrupt threshold, frame list size (if applicable), and turn the controller on by setting the USBCMD[RS] bit.

7. Enable external VBUS supply. The exact steps required for initialization depend on the external hardware used to supply the 5V VBUS power.
8. Set the PORTSC[PP] bit.

At this point, the host controller is up and running and the port registers begin reporting device connects. System software can enumerate a port through the reset process (port is in the enabled state).

To communicate with devices via the asynchronous schedule, system software must write the ASYNCLISTADDR register with the address of a control or bulk queue head. Software must then enable the asynchronous schedule by setting the asynchronous schedule enable (ASE) bit in the USBCMD register. To communicate with devices via the periodic schedule, system software must enable the periodic schedule by setting the periodic schedule enable (PSE) bit in the USBCMD register. Schedules can be turned on before the first port is reset and enabled.

Any time the USBCMD register is written, system software must ensure the appropriate bits are preserved, depending on the intended operation.

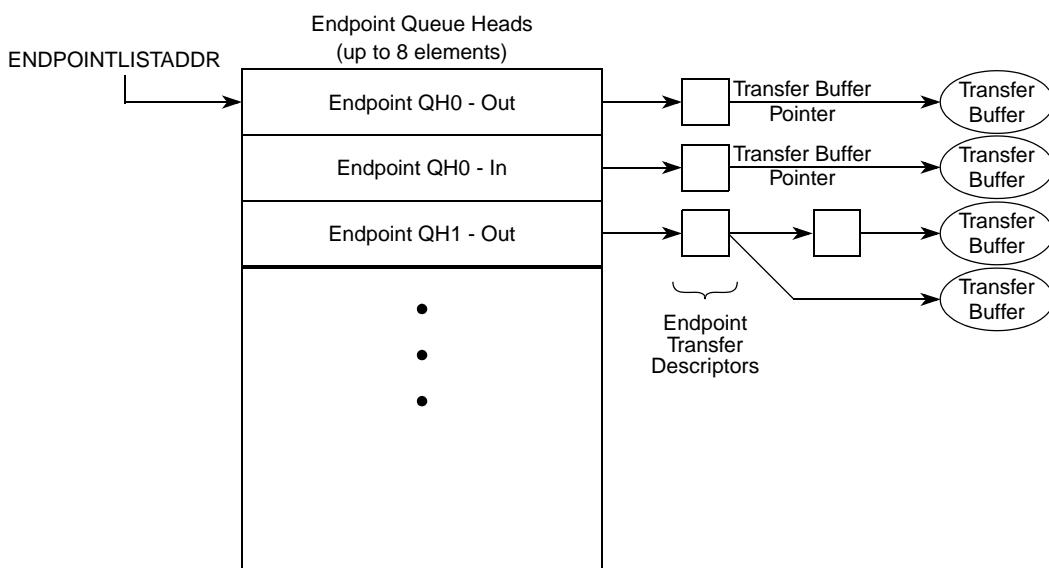
### 24.5.2 Device Data Structures

This section defines the interface data structures used to communicate control, status, and data between device controller driver (DCD) software and the device controller. The interface consists of device queue heads and transfer descriptors.

#### NOTE

Software must ensure that data structures do not span a 4K-page boundary.

The USB OTG uses an array of device endpoint queue heads to organize device transfers. As shown in [Figure 24-40](#), there are two endpoint queue heads in the array for each device endpoint—one for IN and one for OUT. The EPLISTADDR provides a pointer to the first entry in the array.



**Figure 24-40. End Point Queue Head Organization**

### 24.5.2.1 Endpoint Queue Head

All transfers are managed in the device endpoint queue head (dQH). The dQH is a 48-byte data structure, but must align on 64-byte boundaries. During priming of an endpoint, the dTD (device transfer descriptor) copies into the overlay area of the dQH, which starts at the nextTD pointer longword and continues through the end of the buffer pointers longwords. After a transfer is complete, the dTD status longword updates in the dTD pointed to by the currentTD pointer. While a packet is in progress, the overlay area of the dQH acts as a staging area for the dTD so the device controller can access needed information with minimal latency.

[Figure 24-41](#) shows the endpoint queue head structure.

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	offset
Mult	ZLT	0	0																												0x00	
																															0x04	
																															0x08 <sup>1</sup>	
0	0																														0xC <sup>1</sup>	
																															0x10 <sup>1</sup>	
																															0x14 <sup>1</sup>	
																															0x18 <sup>1</sup>	
																															0x1C <sup>1</sup>	
																															0x20 <sup>1</sup>	
																															0x24	
																															0x28	
																															0x2C	



Device controller read/write; all others read-only.

**Figure 24-41. Endpoint Queue Head Layout**

<sup>1</sup> Offsets 0x08 through 0x20 contain the transfer overlay.

#### 24.5.2.1.1 Endpoint Capabilities/Characteristics (Offset = 0x0)

This longword specifies static information about the endpoint. In other words, this information does not change over the lifetime of the endpoint. DCD software must not attempt to modify this information while the corresponding endpoint is enabled.

**Table 24-44. Endpoint Capabilities/Characteristics**

<b>Field</b>	<b>Description</b>
31–30 Mult	Mult. This field indicates the number of packets executed per transaction description as given by: 00 Execute N Transactions as demonstrated by the USB variable length packet protocol where N computes using the Maximum Packet Length (dQH) and the Total Bytes field (dT) 01 Execute 1 Transaction. 10 Execute 2 Transactions. 11 Execute 3 Transactions.  <b>Note:</b> Non-ISO endpoints must set Mult equal to 00. ISO endpoints must set Mult equal to 01, 10, or 11 as needed.
29 ZLT	Zero length termination select. This bit is ignored in isochronous transfers. Clearing this bit enables the hardware to automatically append a zero length packet when the following conditions are true: <ul style="list-style-type: none"> <li>• The packet transmitted equals maximum packet length</li> <li>• The dTD has exhausted the field Total Bytes</li> </ul> After this the dTD retires. When the device is receiving, if the last packet length received equals the maximum packet length and the total bytes is zero, it waits for a zero length packet from the host to retire the current dTD.  Setting this bit disables the zero length packet. When the device is transmitting, the hardware does not append any zero length packet. When receiving, it does not require a zero length packet to retire a dTD whose last packet was equal to the maximum packet length packet. The dTD is retired as soon as Total Bytes field goes to zero, or a short packet is received.  0 Enable zero length packet (default). 1 Disable the zero length packet. <b>Note:</b> Each transfer is defined by one dTD, so the zero length termination is for each dTD. In some software application cases, the logic transfer does not fit into only one dTD, so it does not make sense to add a zero length termination packet each time a dTD is consumed. On those cases we recommend to disable the ZLT feature, and use software to generate the zero length termination.
28–27	Reserved. Reserved for future use and must be cleared.
26–16 Maximum Packet Length	Maximum packet length. This directly corresponds to the maximum packet size of the associated endpoint (wMaxPacketSize). The maximum value this field may contain is 0x400 (1024).
15 IOS	Interrupt on setup (IOS). This bit used on control type endpoints indicates if USBSTS[UI] is set in response to a setup being received.
14–0	Reserved. Reserved for future use and must be cleared.

#### 24.5.2.1.2 Current dTD Pointer (Offset = 0x4)

The device controller uses the current dTD pointer to locate transfer in progress. This word is for USB OTG (hardware) use only and should not be modified by DCD software.

**Table 24-45. Current dTD Pointer**

<b>Field</b>	<b>Description</b>
31–5 Current dtd	Current dtd. This field is a pointer to the dTD represented in the transfer overlay area. This field is modified by the device controller to next dTD pointer during endpoint priming or queue advance.
4–0	Reserved. Reserved for future use and must be cleared.

### 24.5.2.1.3 Transfer Overlay (Offset = 0x8–0x20)

The seven longwords in the overlay area represent a transaction working space for the device controller. The general operational model is that the device controller can detect whether the overlay area contains a description of an active transfer. If it does not contain an active transfer, it does not read the associated endpoint.

After an endpoint is readied, the dTD is copied into this queue head overlay area by the device controller. Until a transfer expires, software must not write the queue head overlay area or the associated transfer descriptor. When the transfer is complete, the device controller writes the results back to the original transfer descriptor and advance the queue.

See [Section 24.5.2.2, “Endpoint Transfer Descriptor \(dTD\),”](#) for a description of the overlay fields.

### 24.5.2.1.4 Setup Buffer (Offset = 0x28–0x2C)

The set-up buffer is dedicated storage for the 8-byte data that follows a set-up PID. Refer to [Section 24.5.3.4.4, “Control Endpoint Operation”](#) for information on the procedure for reading the setup buffer

#### NOTE

Each endpoint has a TX and an RX dQH associated with it, and only the RX queue head receives setup data packets.

**Table 24-46. Multiple Mode Control**

longword	Field	Description
1	31–0 Setup Buffer 0	Setup Buffer 0. This buffer contains bytes 3 to 0 of an incoming setup buffer packet and is written by the device controller software reads.
2	31–0 Setup Buffer 1	Setup Buffer 1. This buffer contains bytes 7 to 4 of an incoming setup buffer packet and is written by the device controller software reads.

### 24.5.2.2 Endpoint Transfer Descriptor (dTD)

The dTD describes to the device controller the location and quantity of data sent/received for a given transfer. The DCD software should not attempt to modify any field in an active dTD except the next dTD pointer, which must be modified only as described in [Section 24.5.3.6, “Managing Transfers with Transfer Descriptors.”](#)

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	offset
Next dTD Pointer																									0	0	0	0	T	0x00		
0	Total Bytes				ioc	0	0	0	MultO	0	0	Status												0x04								
Buffer Pointer (Page 0)												Current Offset																	0x08			
Buffer Pointer (Page 1)												0	Frame Number												0x0C							
Buffer Pointer (Page 2)												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x10	
Buffer Pointer (Page 3)												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x14	
Buffer Pointer (Page 4)												0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x18	



Device controller read/write; all others read-only.

**Figure 24-42. Endpoint Transfer Descriptor (dTD)**

#### 24.5.2.2.1 Next dTD Pointer (Offset = 0x0)

The next dTD pointer is used to point the device controller to the next dTD in the linked list.

**Table 24-47. Next dTD Pointer**

Field	Description
31–5 Next dTD pointer	Next dTD pointer. This field contains the physical memory address of the next dTD to be processed. The field corresponds to memory address signals [31:5], respectively.
4–1	Reserved. Reserved for future use and must be cleared.
0 T	Terminate. This bit indicates to the device controller no more valid entries exist in the queue. 0=Pointer is valid (points to a valid transfer element descriptor). 1=pointer is invalid.

#### 24.5.2.2.2 dTD Token (Offset = 0x4)

The dTD token is used to specify attributes for the transfer including the number of bytes to read or write and the status of the transaction.

**Table 24-48. dTD Token**

Field	Description
31	Reserved. Reserved for future use and must be cleared.
30–16 Total Bytes	<p>Total bytes. This field specifies the total number of bytes moved with this transfer descriptor. This field decrements by the number of bytes actually moved during the transaction and only on the successful completion of the transaction.</p> <p>The maximum value software may store in the field is 5*4K(0x5000). This is the maximum number of bytes 5 page pointers can access. Although possible to create a transfer up to 20K, this assumes the first offset into the first page is 0. When the offset cannot be predetermined, crossing past the fifth page can be guaranteed by limiting the total bytes to 16K**. Therefore, the maximum recommended transfer is 16K (0x4000).</p> <p><b>Note:</b> Larger transfer sizes can be supported, but require disabling ZLT and using multiple dTDs.</p> <p>If the value of the field is 0 when the host controller fetches this transfer descriptor (and the active bit is set), the device controller executes a zero-length transaction and retires the transfer descriptor.</p> <p>For IN transfers it is not a requirement for total bytes to transfer be an even multiple of the maximum packet length. If software builds such a transfer descriptor for an IN transfer, the last transaction is always less than maximum packet length.</p> <p>For OUT transfers the total bytes must be evenly divisible by the maximum packet length.</p>
15 IOC	Interrupt on complete. Indicates if USBSTS[UI] is set in response to device controller finished with this dTD.
14–12	Reserved. Reserved for future use and must be cleared.
11–10 MultO	<p>Multiplier Override. This field can possibly transmit-ISOs (ISO-IN) to override the multiplier in the QH. This field must be 0 for all packet types not transmit-ISO.</p> <p>For example, if QH.MULT equals 3; Maximum packet size equals 8; Total Bytes equals 15; MultO equals 0 [default], then three packets are sent: {Data2(8); Data1(7); Data0(0)}.</p> <p>If QH.MULT equals 3; Maximum packet size equals 8; Total Bytes equals 15; MultO equals 2, then two packets are sent: {Data1(8); Data0(7)}</p> <p>For maximal efficiency, software must compute MultO equals greatest integer of (Total Bytes / Max. Packet Size) except for the case when Total Bytes equals 0; then MultO must be 1.</p> <p><b>Note:</b> Non-ISO and Non-TX endpoints must set MultO equals 00.</p>
9–8	Reserved. Reserved for future use and must be cleared.

**Table 24-48. dTD Token (continued)**

<b>Field</b>	<b>Description</b>	
7–0 Status	Status. Device controller communicates individual command execution states back to the DCD software. This field contains the status of the last transaction performed on this dTD. The bit encodings are:	
	<b>Bit</b>	
	<b>Status Field Description</b>	
7	Active. Set by software to enable the execution of transactions by the device controller.	
6	Halted. Set by the device controller during status updates to indicate a serious error has occurred at the device/endpoint addressed by this dTD. Any time a transaction results in the halted bit being set, the active bit is also cleared.	
5	Data Buffer Error. Set by the device controller during status update to indicate the device controller is unable to maintain the reception of incoming data (overrun) or is unable to supply data fast enough during transmission (under run).	
4	Reserved.	
3	Transaction Error. Set by the device controller during status update in case the device did not receive a valid response from the host (time-out, CRC, bad PID).	
2–0	Reserved.	

#### 24.5.2.2.3 dTD Buffer Page Pointer List (Offset = 0x8–0x18)

The last five longwords of a device element transfer descriptor are an array of physical memory address pointers. These pointers reference the individual pages of a data buffer.

**Table 24-49. Buffer Page Pointer List**

<b>Field</b>	<b>Description</b>
31–12 Buffer Pointer	Buffer Pointer. Selects the page offset in memory for the packet buffer. Non virtual memory systems typically set the buffer pointers to a series of incrementing integers.
0;11–0 Current Offset	Current Offset. Offset into the 4kB buffer where the packet begins.
1;10–0 Frame Number	Frame Number. Written by the device controller to indicate the frame number a packet finishes in. Typically correlates relative completion times of packets on an ISO endpoint.

### 24.5.3 Device Operation

The device controller performs data transfers using a set of linked list transfer descriptors pointed to by a queue head. The next sections explain the use of the device controller from the device controller driver (DCD) point-of-view and further describe how specific USB bus events relate to status changes in the device controller programmer's interface.

#### 24.5.3.1 Device Controller Initialization

After hardware reset, USB OTG is disabled until the run/stop bit in the USBCMD register is set. At minimum, it is necessary to have the queue heads set up for endpoint 0 before the device attach occurs. Shortly after the device is enabled, a USB reset occurs followed by setup packet arriving at endpoint 0. A queue head must be prepared so the device controller can store the incoming setup packet.

To initialize a device, the software must:

1. Optionally set streaming disable in the USBMODE[SDIS] bit.
2. Optionally modify the BURSTSIZE register.
3. Program the PORTSC1[PTS] field if using a non-ULPI PHY.
4. Write the appropriate value to the USBINTR to enable the desired interrupts. For device operation, setting UE, UEE, PCE, URE, and SLE is recommended.  
For a list of available interrupts, refer to [Section 24.3.4.3, “USB Interrupt Enable Register \(USBINTR\),”](#) and [Section 24.3.4.2, “USB Status Register \(USBSTS\).”](#)
5. Set the USBMODE[CM] field to enable device mode, and set the USBMODE[ES] bit for big endian operation.
6. Optionally write the USBCMD register to set the desired interrupt threshold.
7. Set USBMODE[SLOM] to disable setup lockouts.
8. Initialize the EPLISTADDR.
9. Create two dQHs for endpoint 0—one for IN transactions and one for OUT transactions.  
For information on device queue heads, refer to [Section 24.5.2.1, “Endpoint Queue Head.”](#)
10. Set the CCM’s UOCSR[BVLD] bit to allow device to connect to a host.
11. Set the USBCMD[RS] bit.

After the run/stop bit is set, a device reset occurs. The DCD must monitor the reset event and set the DEVICEADDR and EPCRn registers, and adjust the software state as described in [Section 24.5.3.2.1, “Bus Reset.”](#)

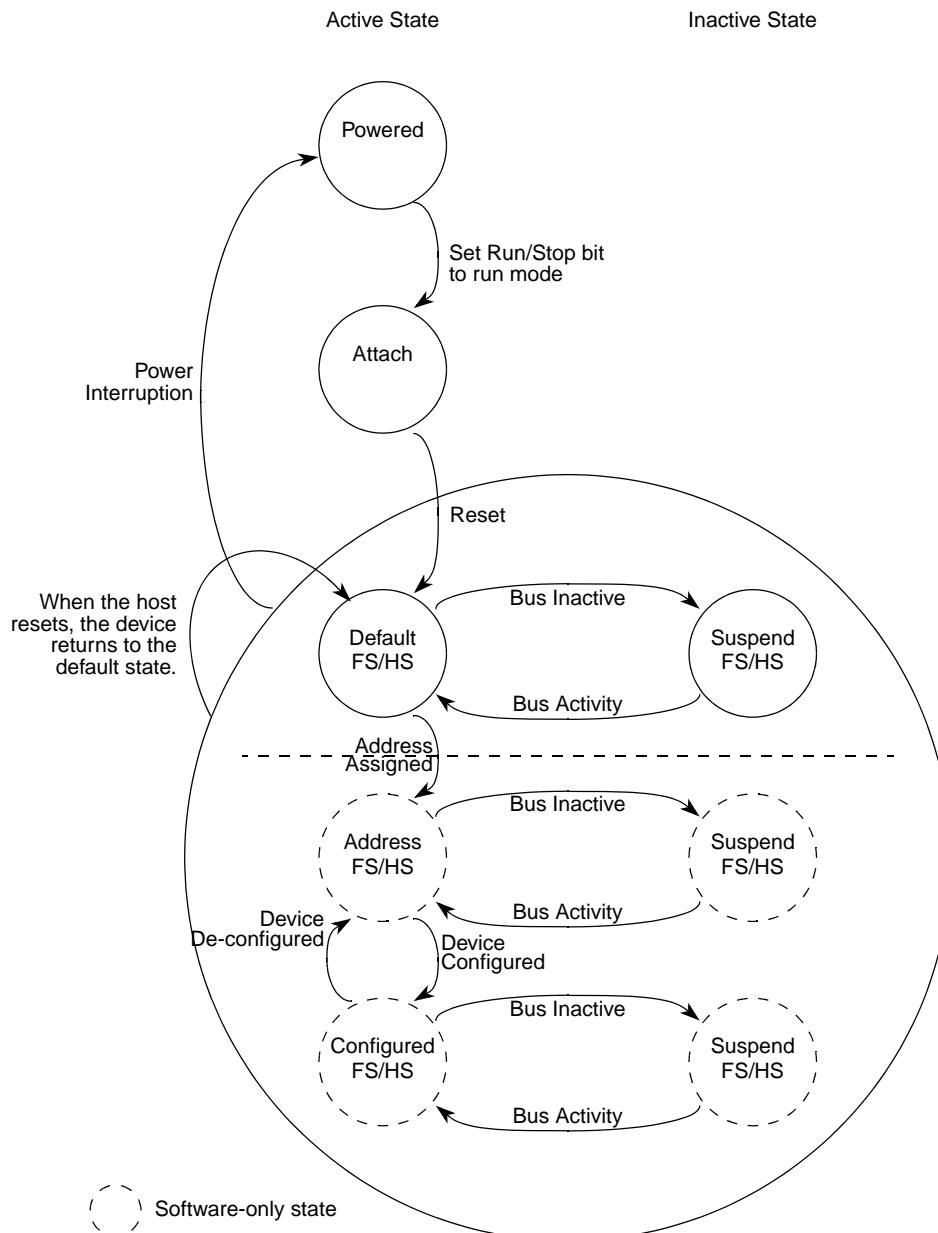
#### NOTE

Endpoint 0 is a control endpoint only and does not need to be configured using the EPCR0 register.

It is not necessary to initially prime endpoint 0 because the first packet received is always a setup packet. The contents of the first setup packet requires a response in accordance with USB device framework command set.

### 24.5.3.2 Port State and Control

From a chip or system reset, the USB OTG module enters the powered state. A transition from the powered state to the attach state occurs when the USBCMD[RS] bit is set. After receiving a reset on the bus, the port enters the defaultFS or defaultHS state in accordance with the protocol reset described in Appendix C.2 of the *Universal Serial Bus Specification, Revision 2.0*. [Figure 24-43](#) depicts the state of a USB 2.0 device.

**Figure 24-43. USB 2.0 Device States**

States powered, attach, defaultFS/HS, suspendFS/HS are implemented in the USB OTG, and they are communicated to the DCD using these status bits:

**Table 24-50. Device Controller State Information Bits**

Bit	Register
DC Suspend (SLI)	USBSTS
USB Reset Received (URI)	USBSTS

**Table 24-50. Device Controller State Information Bits (continued)**

Bit	Register
Port Change Detect (PCI)	USBSTS
High-Speed Port (PSPD)	PORTSC $n$

DCD software must maintain a state variable to differentiate between the defaultFS/HS state and the address/configured states. Change of state from default to the address and configured states is part of the enumeration process described in the device framework section of the USB 2.0 specification.

As a result of entering the address state, the DCD must program the device address register (DEVICEADDR).

Entry into the configured state indicates that all endpoints to be used in the operation of the device have been properly initialized by programming the EPCR $n$  registers and initializing the associated queue heads.

#### 24.5.3.2.1 Bus Reset

The host uses a bus reset to initialize downstream devices. When a bus reset is detected, USB OTG controller renegotiates its attachment speed, resets the device address to 0, and notifies the DCD by interrupt (assuming the USB reset interrupt enable is set). After a reset is received, all endpoints (except endpoint 0) are disabled and the device controller cancels any primed transactions. The concept of priming is clarified below, but when a reset is received, the DCD must perform:

1. Clear all setup token semaphores by reading the EPSETUPSR register and writing the same value back to the EPSETUPSR register.
2. Clear all the endpoint complete status bits by reading the EPCOMPLETE register and writing the same value back to the EPCOMPLETE register.
3. Cancel all primed status by waiting until all bits in the EPPRIME are 0 and then writing 0xFFFF\_FFFF to EPFLUSH.
4. Read the reset bit in the PORTSC $n$  register and make sure it remains active. A USB reset occurs for a minimum of 3 ms and the DCD must reach this point in the reset clean-up before end of the reset occurs, otherwise a hardware reset of the device controller is recommended (rare).
  - a) Setting USBCMD[RST] bit can perform a hardware reset.

#### NOTE

A hardware reset causes the device to detach from the bus by clearing the USBCMD[RS] bit. Therefore, the DCD must completely re-initialize the USB OTG after a hardware reset.

5. Free all allocated dTDs because the device controller no longer executes them. If this is the first time the DCD processes a USB reset event, it is likely w3a4no dTDs have been allocated.
6. At this time, the DCD may release control back to the OS because no further changes to the device controller are permitted until a port change detect is indicated.

7. After a port change detect, the device has reached the default state and the DCD can read the PORTSCn register to determine if the device operates in FS or HS mode. At this time, the device controller has reached normal operating mode and DCD can begin enumeration according to the chapter 9 Device Framework of the USB specification.

In some applications, it may not be possible to enable one or more pipes while in FS mode. Beyond the data rate issue, there is no difference in DCD operation between FS and HS modes.

#### **24.5.3.2.2 Suspend/Resume**

To conserve power, USB OTG module automatically enters the suspended state when no bus traffic is observed for a specified period. When suspended, the module maintains any internal status, including its address and configuration. Attached devices must be prepared to suspend any time they are powered, regardless if they are assigned a non-default address, are configured, or neither. Bus activity may cease due to the host entering a suspend mode of its own. In addition, a USB device shall also enter the suspended state when the hub port it is attached to is disabled.

The USB OTG module exits suspend mode when there is bus activity. It may also request the host to exit suspend mode or selective suspend by using electrical signaling to indicate remote wake-up. The ability of a device to signal remote wake-up is optional. The USB OTG is capable of remote wake-up signaling. When the USB OTG is reset, remote wake-up signaling must be disabled.

#### **Suspend Operational Model**

The USB OTG moves into the suspend state when suspend signaling is detected or activity is missing on the upstream port for more than a specific period. After the device controller enters the suspend state, an interrupt notifies the DCD (assuming device controller suspend interrupt is enabled, USBINTR[SLE] is set). When the PORTSCn[SUSP] is set, the device controller is suspended.

DCD response when the device controller is suspended is application specific and may involve switching to low power operation. Find information on the bus power limits in suspend state in USB 2.0 specification.

#### **Resume**

If the USB OTG is suspended, its operation resumes when any non-idle signaling is received on its upstream facing port. In addition, the USB OTG can signal the system to resume operation by forcing resume signaling to the upstream port. Setting the PORTSCn[FPR] bit while the device is in suspend state sends resume signaling upstream. Sending resume signal to an upstream port should cause the host to issue resume signaling and bring the suspended bus segment (one more devices) back to the active condition.

#### **NOTE**

Before use of resume signaling, the host must enable it by using the set feature command defined in chapter 9 Device Framework of the USB 2.0 specification.

### 24.5.3.3 Managing Endpoints

The USB 2.0 specification defines an endpoint (also called a device endpoint or an address endpoint) as a uniquely addressable portion of a USB device that can source or sink data in a communications channel between the host and the device. Combination of the endpoint number and the endpoint direction specifies endpoint address.

The channel between the host and an endpoint at a specific device represents a data pipe. Endpoint 0 for a device is always a control type data channel used for device discovery and enumeration. Other types of endpoints are supported by USB include bulk, interrupt, and isochronous. Each endpoint type has specific behavior related to packet response and error managing. Find more detail on endpoint operation in the USB 2.0 specification.

The USB OTG supports up to four endpoint specified numbers. The DCD can enable, disable, and configure each endpoint.

Each endpoint direction is essentially independent and can have differing behavior in each direction. For example, the DCD can configure endpoint 1-IN to be a bulk endpoint and endpoint 1-OUT to be an isochronous endpoint. This helps to conserve the total number of endpoints required for device operation. The only exception is that control endpoints must use both directions on a single endpoint number to function as a control endpoint. Endpoint 0, for example, is always a control endpoint and uses both directions.

Each endpoint direction requires a queue head allocated in memory. If the maximum is four endpoint numbers (one for each endpoint direction used by the device controller), eight queue heads are required. The operation of an endpoint and use of queue heads are described later in this document.

#### 24.5.3.3.1 Endpoint Initialization

After hardware reset, all endpoints except endpoint 0 are uninitialized and disabled. The DCD must configure and enable each endpoint by writing to the appropriate EPCR $n$  register. Each EPCR $n$  is split into an upper and lower half. The lower half of EPCR $n$  configures the receive or OUT endpoint, and the upper half configures the corresponding transmit or IN endpoint. Control endpoints must be configured the same in the upper and lower half of the EPCR $n$  register; otherwise, behavior is undefined. [Table 24-51](#) shows how to construct a configuration word for endpoint initialization.

**Table 24-51. Device Controller Endpoint Initialization**

Field	Value
Data Toggle Reset (TXR, RXR)	1 Synchronize the data PIDs
Data Toggle Inhibit (TXI, RXI)	0 PID sequencing disabled
Endpoint Type (TXT, RXT)	00 Control 01 Isochronous 10 Bulk 11 Interrupt
Endpoint Stall (TXS, RXS)	0 Not stalled

### 24.5.3.3.2 Stalling

There USB OTG has two occasions it may need to return to the host a STALL:

- The first is the functional stall, a condition set by the DCD as described in the USB 2.0 Device Framework chapter. A functional stall is used only on non-control endpoints and can be enabled in the device controller by setting the endpoint stall bit in the EPCR $n$  register associated with the given endpoint and the given direction. In a functional stall condition, the device controller continues to return STALL responses to all transactions occurring on the respective endpoint and direction until the endpoint stall bit is cleared by the DCD.
- A protocol stall, unlike a function stall, is used on control endpoints and automatically cleared by the device controller at the start of a new control transaction (setup phase). When enabling a protocol stall, DCD must enable the stall bits as a pair (TXS and RXS bits). A single write to the EPCR $n$  register can ensure both stall bits are set at the same instant.

#### NOTE

Any write to the EPCR $n$  register during operational mode must preserve the endpoint type field (perform a read-modify-write).

**Table 24-52. Device Controller Stall Response Matrix**

USB Packet	Endpoint Stall Bit	Effect on Stall bit	USB Response
SETUP packet received by a non-control endpoint.	N/A	None	STALL
IN/OUT/PING packet received by a non-control endpoint.	1	None	STALL
IN/OUT/PING packet received by a non-control endpoint.	0	None	ACK/NAK/NYET
SETUP packet received by a control endpoint.	N/A	Cleared	ACK
IN/OUT/PING packet received by a control endpoint	1	None	STALL
IN/OUT/PING packet received by a control endpoint.	0	None	ACK/NAK/NYET

### 24.5.3.3.3 Data Toggle

Data toggle maintains data coherency between host and device for any given data pipe. For more information on data toggle, refer to the USB 2.0 specification.

#### Data Toggle Reset

The DCD may reset the data toggle state bit and cause the data toggle sequence to reset in the device controller by setting the data toggle reset bit in the EPCR $n$  register. This should only happen when configuring/initializing an endpoint or returning from a STALL condition.

## Data Toggle Inhibit

This feature is for test purposes only and must never be used during normal device controller operation.

Setting the data toggle inhibit bit causes the USB OTG module to ignore the data toggle pattern normally sent and accepts all incoming data packets regardless of the data toggle state.

In normal operation, the USB OTG checks the DATA0/DATA1 bit against the data toggle to determine if the packet is valid. If the data PID does not match the data toggle state bit maintained by the device controller for that endpoint, the data toggle is considered not valid. If the data toggle is not valid, the device controller assumes the packet was already received and discards the packet (not reporting it to the DCD). To prevent the USB OTG from re-sending the same packet, the device controller responds to the error packet by acknowledging it with an ACK or NYET response.

### 24.5.3.4 Packet Transfers

The host initiates all transactions on the USB bus and in turn, the device must respond to any request from the host within the turnaround time stated in the USB 2.0 specification.

A USB host sends requests to the USB OTG in an order that can not be precisely predicted as a single pipeline, so it is not possible to prepare a single packet for the device controller to execute. However, the order of packet requests is predictable when the endpoint number and direction is considered. For example, if endpoint 3 (transmit direction) is configured as a bulk pipe, expect the host to send IN requests to that endpoint. This USB OTG module prepares packets for each endpoint/direction in anticipation of the host request. The process of preparing the device controller to send or receive data in response to host initiated transaction on the bus is referred to as priming the endpoint. This term appears throughout the documentation to describe the USB OTG operation so the DCD is built properly. Further, the term flushing describes the action of clearing a packet queued for execution.

#### 24.5.3.4.1 Priming Transmit Endpoints

Priming a transmit endpoint causes the device controller to fetch the device transfer descriptor (dTD) for the transaction pointed to by the device queue head (dQH). After the dTD is fetched, it is stored in the dQH until the device controller completes the transfer described by the dTD. Storing the dTD in the dQH allows the device controller to fetch the operating context needed to manage a request from the host without the need to follow the linked list, starting at the dQH when the host request is received.

After the device has loaded the dTD, the leading data in the packet is stored in a FIFO in the device controller. This FIFO splits into virtual channels so the leading data can be stored for any endpoint up to the maximum number of endpoints configured at device synthesis time.

After a priming request is complete, an endpoint state of primed is indicated in the EPSR register. For a primed transmit endpoint, the device controller can respond to an IN request from the host and meet the stringent bus turnaround time of high-speed USB.

#### 24.5.3.4.2 Priming Receive Endpoints

Priming receives endpoints identical to priming of transmit endpoints from the point of view of the DCD. The major difference in the operational model at the device controller is no data movement of the leading packet data because the data is to be received from the host.

As part of the architecture, the FIFO for the receive endpoints is not partitioned into multiple channels like the transmit FIFO. Thus, the size of the RX FIFO does not scale with the number of endpoints.

#### 24.5.3.4.3 Interrupt/Bulk Endpoint Operation

The behaviors of the device controller for interrupt and bulk endpoints are identical. All valid IN and OUT transactions to bulk pipes handshake with a NAK unless the endpoint is primed. After the endpoint is primed, data delivery commences.

A dTD is retired by the device controller when the packets described in the transfer descriptor are completed. Each dTD describes N packets to transfer according to the USB variable length transfer protocol. The formula below and [Table 24-53](#) describe how the device controller computes the number and length of the packets sent/received by the USB vary according to the total number of bytes and maximum packet length. See [Section 24.5.2.1.1, “Endpoint Capabilities/Characteristics \(Offset = 0x0\),”](#) for details on the ZLT bit.

With zero-length termination (ZLT) cleared:

$$N = \text{INT}(\text{number of bytes}/\text{max. packet length}) + 1$$

With zero-length termination (ZLT) set:

$$N = \text{MAXINT}(\text{number of bytes}/\text{max. packet length})$$

**Table 24-53. Variable Length Transfer Protocol Example (ZLT=0)**

Bytes (dTD)	Max. Packet Length (dQH)	N	P1	P2	P3
511	256	2	256	255	—
512	256	3	256	256	0
512	512	2	512	0	—

**Table 24-54. Variable Length Transfer Protocol Example (ZLT=1)**

Bytes (dTD)	Max. Packet Length (dQH)	N	P1	P2	P3
511	256	2	256	255	—
512	256	2	256	256	—
512	512	1	512	—	—

#### NOTE

The MULT field in the dQH must be set to 00 for bulk, interrupt, and control endpoints.

TX-dTD is complete when:

- All packets described in the dTD successfully transmit. Total bytes in dTD equal 0 when this occurs.

RX-dTD is complete when:

- All packets described in the dTD are successfully received. Total bytes in dTD equal 0 when this occurs.
- A short packet (number of bytes < maximum packet length) was received. This is a successful transfer completion; DCD must check the total bytes field in the dTD to determine the number of bytes remaining. From the total bytes remaining in the dTD, the DCD can compute the actual bytes received.
- A long packet was received (number of bytes > maximum packet size) or (total bytes received > total bytes specified). This is an error condition. The device controller discards the remaining packet and set the buffer error bit in the dTD. In addition, the endpoint flushes and the USBERR interrupt becomes active.

#### **NOTE**

Disabling zero-length packet termination allows transfers larger than the total bytes field spanning across two or more dTDs.

Upon successful completion of the packet(s) described by the dTD, the active bit in the dTD is cleared and the next pointer is followed when the terminate bit is clear. When the terminate bit is set, USB OTG flushes the endpoint/direction and ceases operations for that endpoint/direction.

Upon unsuccessful completion of a packet (see long packet above), the dQH is left pointing to the dTD in error. To recover from this error condition, DCD must properly re-initialize the dQH by clearing the active bit and update the nextTD pointer before attempting to re-prime the endpoint.

#### **NOTE**

All packet level errors, such as a missing handshake or CRC error, are retried automatically by the device controller. There is no required interaction with the DCD for managing such errors.

**Table 24-55. Interrupt/Bulk Endpoint Bus Response Matrix**

Token Type	Stall	Not Primed	Primed	Underflow	Overflow
<b>Setup</b>	Ignore	Ignore	Ignore	N/A	N/A
<b>In</b>	STALL	NAK	Transmit	BS Error <sup>1</sup>	N/A
<b>Out</b>	STALL	NAK	Receive + NYET/ACK <sup>2</sup>	N/A	NAK
<b>Ping</b>	STALL	NAK	ACK	N/A	N/A
<b>Invalid</b>	Ignore	Ignore	Ignore	Ignore	Ignore

<sup>1</sup> Force bit stuff error

<sup>2</sup> NYET/ACK — NYET unless the transfer descriptor has packets remaining according to the USB variable length protocol then ACK.

#### 24.5.3.4.4 Control Endpoint Operation

##### Setup Phase

All requests to a control endpoint begin with a setup phase followed by an optional data phase and a required status phase.

Setup packet managing:

- Disable setup lockout by setting the setup lockout mode bit (USBMODE[SLOM]), once at initialization. Setup lockout is not necessary when using the tripwire as described below.

##### NOTE

Leaving the setup lockout mode cleared results in a potential compliance issue.

- After receiving an interrupt and inspecting EPSETUPSR to determine a setup packet was received on a particular pipe:
  1. Write 1 to clear corresponding bit in EPSETUPSR.
  2. Set the setup tripwire bit (USBCMD[SUTW]).
  3. Duplicate contents of dQH.SetupBuffer into local software byte array.
  4. Read the USBCMD[SUTW] bit. If set, continue; if cleared, goto 2)
  5. Clear the USBCMD[SUTW] bit.
  6. Poll until the EPSETUPSR bit clears.
  7. Process setup packet using the local software byte array copy and execute status/handshake phases.

##### NOTE

After receiving a new setup packet, status and/or handshake phases may remain pending from a previous control sequence. These should be flushed and de-allocated before linking a new status and/or handshake dTD for the most recent setup packet.

##### Data Phase

Following the setup phase, the DCD must create a device transfer descriptor for the data phase and prime the transfer.

After priming the packet, the DCD must verify a new setup packet is not received by reading the EPSETUPSR register immediately verifying that the prime had completed. A prime completes when the associated bit in the EPPRIME register is cleared and the associated bit in the EPSR register is set. If the EPPRIME bit goes to 0 and the EPSR bit is not set, the prime fails. This can only happen because of improper setup of the dQH, dTD, or a setup arriving during the prime operation. If a new setup packet is indicated after the EPPRIME bit is cleared, then the transfer descriptor can be freed and the DCD must re-interpret the setup packet.

Should a setup arrive after the data stage is primed, the device controller automatically clears the prime status (EPSR) to enforce data coherency with the setup packet.

### NOTE

Error managing of data phase packets is the same as bulk packets described previously.

### Status Phase

Similar to the data phase, the DCD must create a transfer descriptor (with byte length equal zero) and prime the endpoint for the status phase. The DCD must also perform the same checks of the EPSETUPSR as described above in the data phase.

### NOTE

Error managing of status phase packets is the same as bulk packets described previously.

### Control Endpoint Bus Response Matrix

[Table 24-56](#) shows the device controller response to packets on a control endpoint according to the device controller state.

**Table 24-56. Control Endpoint Bus Response Matrix**

Token Type	Endpoint State					Setup Lockout
	Stall	Not Primed	Primed	Underflow	Overflow	
Setup	ACK	ACK	ACK	N/A	SYSERR <sup>1</sup>	
In	STALL	NAK	Transmit	BS Error <sup>2</sup>	N/A	N/A
Out	STALL	NAK	Receive + NYET/ACK <sup>3</sup>	N/A	NAK	N/A
Ping	STALL	NAK	ACK	N/A	N/A	N/A
Invalid	Ignore	Ignore	Ignore	Ignore	Ignore	Ignore

<sup>1</sup> SYSERR — System error must never occur when the latency FIFOs are correctly sized and the DCD is responsive.

<sup>2</sup> Force bit stuff error

<sup>3</sup> NYET/ACK — NYET unless the transfer descriptor has packets remaining according to the USB variable length protocol then ACK.

### 24.5.3.4.5 Isochronous Endpoint Operation

Isochronous endpoints used for real-time scheduled delivery of data, and their operational model is significantly different than the host throttled bulk, interrupt, and control data pipes. Real time delivery by the USB OTG is accomplished by:

- Exactly MULT packets per (micro)frame are transmitted/received.

## NOTE

MULT is a two-bit field in the device queue head. Isochronous endpoints do not use the variable length packet protocol.

- NAK responses are not used. Instead, zero length packets are sent in response to an IN request to unprimed endpoints. For unprimed RX endpoints, the response to an OUT transaction is to ignore the packet within the device controller.
- Prime requests always schedule the transfer described in the dTD for the next (micro)frame. If ISO-dTD remains active after that frame, ISO-dTD holds ready until executed or canceled by the DCD.

The USB OTG in host mode uses the periodic frame list to schedule data exchanges to isochronous endpoints. The operational model for device mode does not use such a data structure. Instead, the same dTD used for control/bulk/interrupt endpoints is also used for isochronous endpoints. The difference is in the managing of the dTD.

The first difference between bulk and ISO-endpoints is that priming an ISO-endpoint is a delayed operation such that an endpoint becomes primed only after a SOF is received. After the DCD writes the prime bit, the prime bit clears as usual to indicate to software that the device controller completed a priming the dTD for transfer. Internal to the design, the device controller hardware masks that prime start until the next frame boundary. This behavior is hidden from the DCD, but occurs so the device controller can match the dTD to a specific (micro)frame.

Another difference with isochronous endpoints is that the transaction must wholly complete in a (micro)frame. After an ISO transaction is started in a (micro)frame, it retires the corresponding dTD when MULT transactions occur or the device controller finds a fulfillment condition.

The transaction error bit set in the status field indicates a fulfillment error condition. When a fulfillment error occurs, the frame after the transfer failed to complete wholly, and the device controller retires the current ISO-dTD and move to the next ISO-dTD.

Fulfillment errors are only caused due to partially completed packets. If no activity occurs to a primed ISO-dTD, the transaction stays primed indefinitely. This means it is up to software must discard transmit ISO-dTDs that pile up from a failure of the host to move the data.

Finally, the last difference with ISO packets is in the data level error managing. When a CRC error occurs on a received packet, the packet is not retried similar to bulk and control endpoints. Instead, the CRC is noted by setting the transaction error bit and the data is stored as usual for the application software to sort out.

- TX packet retired:
  - MULT counter reaches zero.
  - Fulfillment error (transaction error bit is set):
    - # packets occurred > 0 AND # packets occurred < MULT

**NOTE**

For TX-ISO, MULT counter can be loaded with a lesser value in the dTD multiplier override field. If the multiplier override field is zero, the MULT counter initializes to the multiplier in the QH.

- RX packet retired:
  - MULT counter reaches zero.
  - Non-MDATA data PID is received
  - Overflow error:
    - Packet received is > maximum packet length. (Buffer Error bit is set)
    - Packet received exceeds total bytes allocated in dTD. (Buffer Error bit is set)
  - Fulfillment error (Transaction Error bit is set):
    - # packets occurred > 0 AND # packets occurred < MULT
  - CRC error (Transaction Error bit is set)

**NOTE**

For ISO, when a dTD is retired, the next dTD is primed for the next frame. For continuous (micro)frame to (micro)frame operation, DCD must ensure the dTD linked-list is out ahead of the device controller by at least two (micro)frames.

**Isochronous Pipe Synchronization**

When it is necessary to synchronize an isochronous data pipe to the host, the (micro)frame number (FRINDEX register) can act as a marker. To cause a packet transfer to occur at a specific (micro)frame number (N), the DCD must interrupt on SOF during frame N-1. When the FRINDEX equals N-1, the DCD must write the prime bit. The USB OTG primes the isochronous endpoint in (micro)frame N-1 so the device controller executes delivery during (micro)frame N.

**CAUTION**

Priming an endpoint towards the end of (micro)frame N-1 does not guarantee delivery in (micro)frame N. The delivery may actually occur in (micro)frame N+1 if the device controller does not have enough time to complete the prime before the SOF for packet N is received.

**Isochronous Endpoint Bus Response Matrix****Table 24-57. Isochronous Endpoint Bus Response Matrix**

<b>Token Type</b>	<b>Stall</b>	<b>Not Primed</b>	<b>Primed</b>	<b>Underflow</b>	<b>Overflow</b>
<b>Setup</b>	STALL	STALL	STALL	N/A	N/A
<b>In</b>	NULL <sup>1</sup> Packet	NULL Packet	Transmit	BS Error <sup>2</sup>	N/A

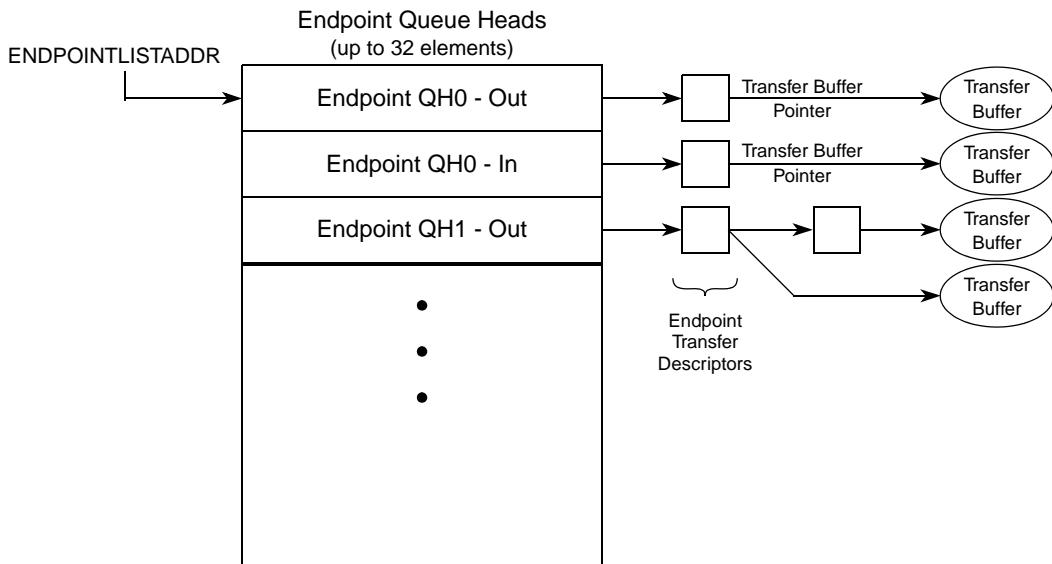
**Table 24-57. Isochronous Endpoint Bus Response Matrix (continued)**

Token Type	Stall	Not Primed	Primed	Underflow	Overflow
<b>Out</b>	Ignore	Ignore	Receive	N/A	Drop Packet
<b>Ping</b>	Ignore	Ignore	Ignore	Ignore	Ignore
<b>Invalid</b>	Ignore	Ignore	Ignore	Ignore	Ignore

<sup>1</sup> Zero length packet<sup>2</sup> Force bit stuff error

### 24.5.3.5 Managing Queue Heads

The device queue head (dQH) points to the linked list of transfer tasks, each depicted by the device transfer descriptor (dT). An area of memory pointed to by EPLISTADDR contains a group of all dQH's in a sequential list (Figure 24-44). The even elements in the list of dQH's receive endpoints (OUT/SETUP) and the odd elements transmit endpoints (IN/INTERRUPT). Device transfer descriptors are linked head to tail starting at the queue head and ending at a terminate bit. After the dTD retires, it is no longer part of the linked list from the queue head. Therefore, software is required to track all transfer descriptors because pointers no longer exist within the queue head after the dTD is retired (see Section 24.5.3.6.1, “Software Link Pointers”).

**Figure 24-44. Endpoint Queue Head Diagram**

In addition to current and next pointers and the dTD overlay examined in Section 24.5.3.4, “Packet Transfers,” the dQH also contains the following parameters for the associated endpoint: multiplier, maximum packet length, and interrupt on setup. The next section includes demonstration of complete initialization of the dQH including these fields.

#### 24.5.3.5.1 Queue Head Initialization

One pair of device queue heads must be initialized for each active endpoint. To initialize a device queue head:

- Write the wMaxPacketSize field as required by the USB specification chapter 9 or application specific protocol.
- Write the multiplier field to 0 for control, bulk, and interrupt endpoints. For ISO endpoints, set the multiplier to 1,2, or 3 as required for bandwidth with the USB specification chapter 9 protocol. In FS mode, the multiplier field can only be 1 for ISO endpoints.
- Set the next dTD terminate bit field.
- Clear the active bit in the status field.
- Clear the halt bit in the status field.

#### **NOTE**

The DCD must only modify dQH if the associated endpoint is not primed and there are no outstanding dTDs.

#### 24.5.3.5.2 Setup Transfers Operation

As discussed in [Section 24.5.3.4.4, “Control Endpoint Operation,”](#) setup transfers require special treatment by the DCD. A setup transfer does not use a dTD, but instead stores the incoming data from a setup packet in an 8-byte buffer within the dQH.

Upon receiving notification of the setup packet, the DCD should manage the setup transfer by:

1. Copying setup buffer contents from dQH-RX to software buffer.
2. Acknowledging setup backup by writing a 1 to the corresponding bit in the EPSETUPSR register.

#### **NOTE**

The acknowledge must occur before continuing to process the setup packet.

After acknowledge occurs, DCD must not attempt to access the setup buffer in dQH-RX. Only local software copy should be examined.

3. Checking for pending data or status dTD's from previous control transfers and flushing if any exist as discussed in [Section 24.5.3.6.5, “Flushing/De-priming an Endpoint.”](#)

#### **NOTE**

It is possible for the device controller to receive setup packets before previous control transfers complete. Existing control packets in progress must be flushed and the new control packet completed.

4. Decoding setup packet and prepare data phase (optional) and status phase transfer as required by the USB specification chapter 9 or application specific protocol.

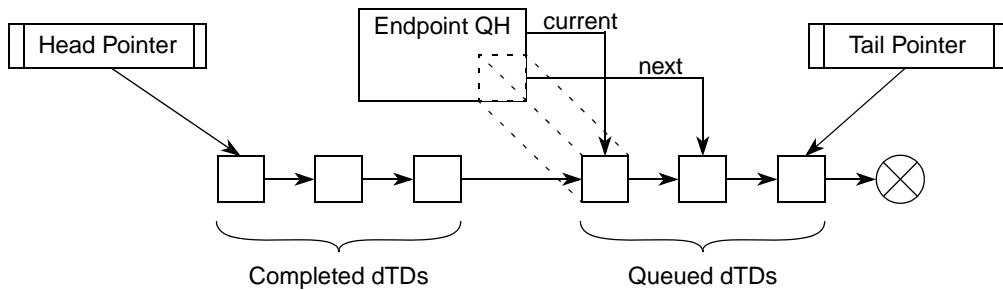
## 24.5.3.6 Managing Transfers with Transfer Descriptors

### 24.5.3.6.1 Software Link Pointers

It is necessary for the DCD software to maintain head and tail pointers for the linked list of dTDs for each respective queue head. This is necessary because the dQH only maintains pointers to the current working dTD and the next dTD executed. The operations described in the next section for managing dTDs assumes DCD can reference the head and tail of the dTD linked list.

#### NOTE

To conserve memory, the reserved fields at the end of the dQH can be used to store the head and tail pointers, but DCD must continue maintaining the pointers.



**Figure 24-45. Software Link Pointers**

#### NOTE

Check the status of each dTD to determine completed status.

### 24.5.3.6.2 Building a Transfer Descriptor

Before a transfer can be executed from the linked list, a dTD must be built to describe the transfer. Use the following procedure for building dTDs.

Allocate an 8-longword dTD block of memory aligned to 8-longword boundaries. The last 5 bits of the address must equal 00000.

Write the following fields:

1. Initialize the first 7 longwords to 0.
2. Set the terminate bit.
3. Fill in total bytes with transfer size.
4. Set the interrupt on complete bit if desired.
5. Initialize the status field with the active bit set, and all remaining status bits cleared.
6. Fill in buffer pointer page 0 and the current offset to point to the start of the data buffer.
7. Initialize buffer pointer page 1 through page 4 to be one greater than each of the previous buffer pointers.

### 24.5.3.6.3 Executing a Transfer Descriptor

To safely add a dTD, the DCD must follow this procedure that manages the event where the device controller reaches the end of the dTD list. At the same time, a new dTD is added to the end of the list.

Determine whether the linked list is empty:

Check the DCD driver to see if the pipe is empty (internal representation of the linked list should indicate if any packets are outstanding)

Case 1: Link list is empty

1. Write dQH next pointer AND dQH terminate bit to 0 as a single longword operation.
2. Clear active and halt bit in dQH (in case set from a previous error).
3. Prime endpoint by writing 1 to the correct bit position in the EPPRIME register.

Case 2: Link list is not empty

1. Add dTD to end of the linked list.
2. Read correct prime bit in EPPRIME - if set, DONE.
3. Set the USBCMD[ATDTW] bit.
4. Read correct status bit in EPSR, and store in a temporary variable for later.
5. Read the USBCMD[ATDTW] bit:
  - If clear, go to 3.
  - If set, continue to 6.
6. Clear the USBCMD[ATDTW] bit.
7. If status bit read in step 4 is 1 DONE.
8. If status bit read in step 4 is 0 then go to case 1, step 1.

### 24.5.3.6.4 Transfer Completion

After a dTD is initialized and the associated endpoint is primed, the device controller executes the transfer upon the host-initiated request. The DCD is notified with a USB interrupt if the interrupt-on-complete bit was set, or alternatively, the DCD can poll the endpoint complete register to determine when the dTD had been executed. After a dTD is executed, DCD can check the status bits to determine success or failure.

#### CAUTION

Multiple dTDs can be completed in a single endpoint complete notification.

After clearing the notification, the DCD must search the dTD linked list and retire all finished (active bit cleared) dTDs.

By reading the status fields of the completed dTDs, the DCD can determine if the transfers completed successfully. Success is determined with the following combination of status bits:

- Active = 0, Halted = 0, Transaction error = 0, Data buffer error = 0

Should any combination other than the one shown above exist, the DCD must take proper action. Transfer failure mechanisms are indicated in [Section 24.5.3.6.6, “Device Error Matrix.”](#)

In addition to checking the status bit, the DCD must read the transfer bytes field to determine the actual bytes transferred. When a transfer is complete, the total bytes transferred decrements by the actual bytes transferred. For transmit packets, a packet is only complete after the actual bytes reaches zero. However, for receive packets, the host may send fewer bytes in the transfer according the USB variable length packet protocol.

#### 24.5.3.6.5 Flushing/De-priming an Endpoint

It is necessary for the DCD to flush or de-prime endpoints during a USB device reset or during a broken control transfer. There may also be application specific requirements to stop transfers in progress. The DCD can use this procedure to stop a transfer in progress:

1. Set the corresponding bit(s) in the EPFLUSH register.
2. Wait until all bits in the EPFLUSH register are cleared.

#### NOTE

This operation may take a large amount of time depending on the USB bus activity. It is not desirable to have this wait loop within an interrupt service routine.

3. Read the EPSR register to ensure that for all endpoints commanded to be flushed, that the corresponding bits are now cleared. If the corresponding bits are set after step #2 has finished, flush failed as described below:

In very rare cases, a packet is in progress to the particular endpoint when commanded to flush using EPFLUSH. A safeguard is in place to refuse the flush to ensure that the packet in progress completes successfully. The DCD may need to repeatedly flush any endpoints that fail to flush by repeating steps 1-3 until each endpoint successfully flushes.

#### 24.5.3.6.6 Device Error Matrix

The following table summarizes packet errors not automatically managed by the USB OTG module.

**Table 24-58. Device Error Matrix**

Error	Direction	Packet Type	Data Buffer Error Bit	Transaction Error Bit
Data Buffer Overflow	RX	Any	1	0
ISO Packet Error	RX	ISO	0	1
ISO Fulfillment Error	Both	ISO	0	1

The device controller manages all errors on bulk/control/interrupt endpoints except for a data buffer overflow. However, for ISO endpoints, errors packets are not retried and errors are tagged as indicated.

**Table 24-59. Error Descriptions**

Overflow	Number of bytes received exceeded max. packet size or total buffer length.  <b>Note:</b> This error also sets the halt bit in the dQH, and if there are dTDs remaining in the linked list for the endpoint, those are not executed.
ISO Packet Error	CRC error on received ISO packet. Contents not guaranteed correct.
ISO Fulfillment Error	Host failed to complete the number of packets defined in the dQH mult field within the given (micro)frame. For scheduled data delivery, DCD may need to readjust the data queue because a fulfillment error causes the device controller to cease data transfers on the pipe for one (micro)frame. During the dead (micro)frame, the device controller reports error on the pipe and primes for the following frame.

## 24.5.4 Servicing Interrupts

The interrupt service routine must understand there are high frequency, low frequency, and error operations to order accordingly.

### 24.5.4.1 High Frequency Interrupts

In particular, high frequency interrupts must be managed in the order below. The most important of these is listed first because the DCD must acknowledge a setup buffer in the timeliest manner possible.

**Table 24-60. Interrupt Managing Order**

Execution Order	Interrupt	Action
1a	USB Interrupt <sup>1</sup> EPSETUPSR	Copy contents of setup buffer and acknowledge setup packet (as indicated in <a href="#">Section 24.5.3.5, “Managing Queue Heads”</a> ). Process setup packet according to USB specification chapter 9 or application specific protocol.
1b	USB Interrupt EPCOMPLETE	Manage completion of dTD as indicated in <a href="#">Section 24.5.3.5, “Managing Queue Heads.”</a>
2	SOF Interrupt	Action as deemed necessary by application. This interrupt may not have a use in all applications.

<sup>1</sup> It is likely multiple interrupts stack up on any call to the interrupt service routine and during interrupt service routine.

#### 24.5.4.1.1 Low Frequency Interrupts

The low frequency events include the following interrupts. These interrupts can be managed in any order because they do not occur often in comparison to the high-frequency interrupts.

**Table 24-61. Low Frequency Interrupt Events**

Interrupt	Action
Port Change	Change software state information.
Sleep Enable (Suspend)	Change software state information. Low power managing as necessary.
Reset Received	Change software state information. Abort pending transfers.

#### 24.5.4.1.2 Error Interrupts

Error interrupts are least frequent and should be placed last in the interrupt service routine.

**Table 24-62. Error Interrupt Events**

Interrupt	Action
USB Error Interrupt.	This error is redundant because it combines USB interrupt and an error status in the dTD. The DCD more aptly manages packet-level errors by checking the dTD status field upon receipt of USB interrupt (w/ EPCOMPLETE).
System Error	Unrecoverable error. Immediate reset of module; free transfers buffers in progress and restart the DCD.

#### 24.5.5 Deviations from the EHCI Specifications

The host mode operation of the USB host and OTG modules is nearly EHCI-compatible with a few minor differences. For the most part, the modules conform to the data structures and operations described in Section 3, “Data Structures,” and Section 4, “Operational Model,” in the EHCI specification. The particulars of the deviations occur in the following areas:

- Embedded transaction translator (USB host and OTG modules)—Allows direct attachment of FS and LS devices in host mode without the need for a companion controller.
- Device operation (USB OTG module only)—In host mode, the device operational registers are generally disabled; therefore, device mode is mostly transparent when in host mode. However, there are a couple exceptions documented in the following sections.
- Embedded design interface—The modules do not have a PCI Interface and therefore the PCI configuration registers described in the EHCI specification are not applicable.

For the purposes of the USB OTG implementing a dual-role host/device controller with support for OTG applications, it is necessary to deviate from the EHCI specification. Device and OTG operation are not specified in the EHCI specification, and thus the implementation supported in the USB OTG module is proprietary.

#### 24.5.5.1 Embedded Transaction Translator Function

The USB host mode supports directly connected full- and low-speed devices without requiring a companion controller by including the capabilities of a USB 2.0 high-speed hub transaction translator. Although there is no separate transaction translator block in the system, the transaction translator function normally associated with a high-speed hub is implemented within the DMA and protocol engine blocks. The embedded transaction translator function is an extension to EHCI interface, but makes use of the standard data structures and operational models existing in the EHCI specification to support full- and low-speed devices.

##### 24.5.5.1.1 Capability Registers

These additions to the capability registers support the embedded Transaction translator function:

- N\_TT added to HSCPARAMS - Host Controller Structural Parameters
- N\_PTT added to HSCPARAMS - Host Controller Structural Parameters

See [Section 24.3.3.3, “Host Controller Structural Parameters Register \(HCSPARAMS\)”](#) for usage information.

### 24.5.5.1.2 Operational Registers

These additions to the operational registers support the embedded TT:

- Addition of the TTCTRL register.
- Addition of a two-bit port speed (PSPD) field to the PORTSC $n$  register.

### 24.5.5.1.3 Discovery

In a standard EHCI controller design, the EHCI host controller driver detects a full-speed (FS) or low-speed (LS) device by noting if the port enable bit is set after the port reset operation. The port enable is set only in a standard EHCI controller implementation after the port reset operation and when the host and device negotiate a high-speed connection (chirp completes successfully).

The module always sets the port enable bit after the port reset operation regardless of the result of the host device chirp result, and the resulting port speed is indicated by the PORTSC $n$ [PSPD] field. Therefore, the standard EHCI host controller driver requires an alteration to manage directly connected full- and low-speed devices or hubs. The change is a fundamental one summarized in [Table 24-63](#).

**Table 24-63. Functional Differences Between EHCI and EHCI with Embedded TT**

Standard EHCI	EHCI with embedded Transaction Translator
After port enable bit is set following a connection and reset sequence, the device/hub is assumed to be HS.	After port enable bit is set following a connection and reset sequence, the device/hub speed is noted from PORTSC $n$ .
FS and LS devices are assumed to be downstream from a HS hub. Therefore, all port-level control performs through the hub class to the nearest hub.	FS and LS device can be downstream from a HS hub or directly attached. When the FS/LS device is downstream from a HS hub, port-level control acts using the hub class through the nearest hub. When a FS/LS device is directly attached, then port-level control is accomplished using PORTSC $n$ .
FS and LS devices are assumed to be downstream from a HS hub with HubAddr equal to X. [where HubAddr > 0 and HubAddr is the address of the hub where the bus transitions from HS to FS/LS (split target hub)]	FS and LS device can be downstream from a HS hub with HubAddr equal to X [HubAddr > 0] or directly attached [where HubAddr equals 0 and HubAddr is the address of the root hub where the bus transitions from HS to FS/LS (split target hub is the root hub)]

### 24.5.5.1.4 Data Structures

The same data structures used for FS/LS transactions though a HS hub are also used for transactions through the root hub. It is demonstrated here how hub address and endpoint speed fields should be set for directly attached FS/LS devices and hubs:

1. QH (for direct attach FS/LS) – asynchronous (bulk/control endpoints) periodic (interrupt)
  - Hub address equals 0
  - Transactions to direct attached device/hub.

- QH.EPS equals port speed
- Transactions to a device downstream from direct attached FS hub.
  - QH.EPS equals downstream device speed

#### **NOTE**

When QH.EPS equals 01 (LS) and PORTSC $n$ [PSPD] equals 00 (FS), a LS-pre-PID is sent before transmitting LS traffic.

Maximum packet size must equal 64 or less to prevent undefined behavior.

2. siTD (for direct attach FS) – Periodic (ISO endpoint)

- All FS ISO transactions:

- Hub address equals 0
- siTD.EPS equals 00 (full speed)

Maximum packet size must equal to 1023 or less to prevent undefined behavior.

#### **24.5.5.1.5 Operational Model**

The operational models are well defined for the behavior of the transaction translator (see USB 2.0 specification) and for the EHCI controller moving packets between system memory and a USB-HS hub. Because the embedded transaction translator exists within the USB host controller, no physical bus between EHCI host controller driver and the USB FS/LS bus. These sections briefly discuss the operational model for how the EHCI and transaction translator operational models combine without the physical bus between. The following sections assume the reader is familiar with the EHCI and USB 2.0 transaction translator operational models.

#### **Microframe Pipeline**

The EHCI operational model uses the concept of H-frames and B-frames to describe the pipeline between the host (H) and the bus (B). The embedded transaction translator uses the same pipeline algorithms specified in the USB 2.0 specification for a hub-based transaction translator.

All periodic transfers always begin at B-frame 0 (after SOF) and continue until the stored periodic transfers are complete. As an example of the microframe pipeline implemented in the embedded transaction translator, all periodic transfers that are tagged in EHCI to execute in H-frame 0 are ready to execute on the bus in B-frame 0.

When programming the S-mask and C-masks in the EHCI data structures to schedule periodic transfers for the embedded transaction translator, the EHCI host controller driver must follow the same rules specified in EHCI for programming the S-mask and C-mask for downstream hub-based transaction translators.

After periodic transfers are exhausted, any stored asynchronous transfer is moved. Asynchronous transfers are opportunistic because they execute when possible and their operation is not tied to H-frame and B-frame boundaries with the exception that an asynchronous transfer cannot babble through the SOF (start of B-frame 0.)

## Split State Machines

The start and complete-split operational model differs from EHCI slightly because there is no bus medium between the EHCI controller and the embedded transaction translator. Where a start or complete-split operation would occur by requesting the split to the HS hub, the start/complete-split operation is simple an internal operation to the embedded transaction translator. [Table 24-64](#) summarizes the conditions where handshakes are emulated from internal state instead of actual handshakes to HS split bus traffic.

**Table 24-64. Emulated Handshakes**

Condition	Emulate TT Response
<b>Start-Split:</b> All asynchronous buffers full	NAK
<b>Start-Split:</b> All periodic buffers full	ERR
<b>Start-Split:</b> Success for start of async. transaction	ACK
<b>Start-Split:</b> Start periodic transaction	No handshake (Ok)
<b>Complete-Split:</b> Failed to find transaction in queue	Bus time-out
<b>Complete-Split:</b> Transaction in queue is busy	NYET
<b>Complete-Split:</b> Transaction in queue is complete	Actual handshake from FS/LS device

## Asynchronous Transaction Scheduling and Buffer Management

The following USB 2.0 specification items are implemented in the embedded Transaction Translator:

- USB 2.0 – 11.17.3
  - Sequencing is provided and a packet length estimator ensures no full-/low-speed packet babbles into SOF time.
- USB 2.0 – 11.17.4
  - • Transaction tracking for 2 data pipes.
- USB 2.0 – 11.17.5
  - • Clear\_TT\_Buffer capability provided though the use of the TTCTRL register.

## Periodic Transaction Scheduling and Buffer Management

The following USB 2.0 specification items are implemented in the embedded transaction translator:

- USB 2.0 – 11.18.6.[1-2]
  - Abort of pending start-splits
    - EOF (and not started in microframes 6)
    - Idle for more than 4 microframes
  - Abort of pending complete-splits
    - EOF
    - Idle for more than 4 microframes
- USB 2.0 - 11.18.[7-8]
  - Transaction tracking for up to 4 data pipes.

- No more than 4 periodic transactions (interrupt/isochronous) can be scheduled through the embedded TT per frame.
- Complete-split transaction searching.

#### **NOTE**

There is no data schedule mechanism for these transactions other than the microframe pipeline. The embedded TT assumes the number of packets scheduled in a frame does not exceed the frame duration (1 ms) or else undefined behavior may result.

#### **24.5.5.2 Device Operation**

The co-existence of a device operational controller within the USB OTG module has little effect on EHCI compatibility for host operation. However, given that the USB OTG controller initializes in neither host nor device mode, the **USBMODE** register must be programmed for host operation before the EHCI host controller driver can begin EHCI host operations.

#### **24.5.5.3 Non-Zero Fields in the Register File**

Some of the reserved fields and reserved addresses in the capability registers and operational registers have use in device mode. Adhere to these steps:

- Write operations to all EHCI reserved fields (some of which are device fields in the USB OTG module) in the operation registers should always be written to zero. This is an EHCI requirement of the device controller driver that must be adhered to.
- Read operations by the module must properly mask EHCI reserved fields (some of which are device fields in the USB OTG module registers).

#### **24.5.5.4 SOF Interrupt**

The SOF interrupt is a free running 125  $\mu$ s interrupt for host mode. EHCI does not specify this interrupt, but it has been added for convenience and as a potential software time base. The free running interrupt is shared with the device mode start-of-frame interrupt. See [Section 24.3.4.2, “USB Status Register \(USBSTS\),”](#) and [Section 24.3.4.3, “USB Interrupt Enable Register \(USBINTR\),”](#) for more information.

#### **24.5.5.5 Embedded Design**

This is an embedded USB host controller as defined by the EHCI specification; therefore, it does not implement the PCI configuration registers.

##### **24.5.5.5.1 Frame Adjust Register**

Given that the optional PCI configuration registers are not included in this implementation, there is no corresponding bit level timing adjustments like those provided by the frame adjust register in the PCI configuration registers. Starts of microframes are timed precisely to 125  $\mu$ s using the transceiver clock as a reference clock or a 60 Mhz transceiver clock for 8-bit physical interfaces and full-speed serial interfaces.

## 24.5.5.6 Miscellaneous Variations from EHCI

### 24.5.5.6.1 Programmable Physical Interface Behavior

The modules support multiple physical interfaces that can operate in different modes when the module is configured with the software programmable physical interface modes. The control bits for selecting the PHY operating mode are added to the PORTSC $n$  register providing a capability not defined by the EHCI specification.

### 24.5.5.6.2 Discovery

#### Port Reset

The port connect methods specified by EHCI require setting the port reset bit in the PORTSC $n$  register for a duration of 10 ms. Due to the complexity required to support the attachment of devices not high speed, a counter is present in the design that can count the 10 ms reset pulse to alleviate the requirement of the software to measure this duration. Therefore, the basic connection is summarized as:

- Port change interrupt—Port connect change occurs to notify the host controller driver that a device has attached.
- Software shall set the PORTSC $n$ [PR] bit to reset the device.
- Software shall clear the PORTSC $n$ [PR] bit after 10 ms.
  - This step, necessary in a standard EHCI design, may be omitted with this implementation. Should the EHCI host controller driver attempt to write a 0 to the reset bit while a reset is in progress, the write is ignored and the reset continues until completion.
- Port change interrupt—Port enable change occurs to notify the host controller that the device is now operational and at this point the port speed is determined.

#### Port Speed Detection

After the port change interrupt indicates that a port is enabled, the EHCI stack should determine the port speed. Unlike the EHCI implementation, which re-assigns the port owner for any device that does not connect at high speed, this host controller supports direct attach of non-HS devices. Therefore, the following differences are important regarding port speed detection:

- Port owner hand-off is not implemented. Therefore, PORTSC $n$ [PO] bit is read-only and always reads 0.
- A 2-bit port speed indicator field has been added to PORTSC $n$  to provide the current operating speed of the port to the host controller driver.
- A 1-bit high-speed indicator bit has been added to PORTSC $n$  to signify that the port is in HS vs. FS/LS.
  - This information is redundant with the 2-bit port speed indicator field above.

# Chapter 25

## Enhanced Secure Digital Host Controller

### 25.1 Overview

The enhanced Secure Digital host controller (eSDHC) provides an interface between the host system and several types of memory cards:

- MultiMediaCard (MMC)

MMC is a universal low-cost data storage and communication medium designed to cover a wide area of applications including mobile video and gaming, which are available from pre-loaded MMC cards or downloadable from cellular phones, WLAN, or other wireless networks. Old MMC cards are based on a seven-pin serial bus with a single data pin, while the new high-speed MMC communication is based on an advanced 11-pin serial bus designed to operate in a low voltage range.

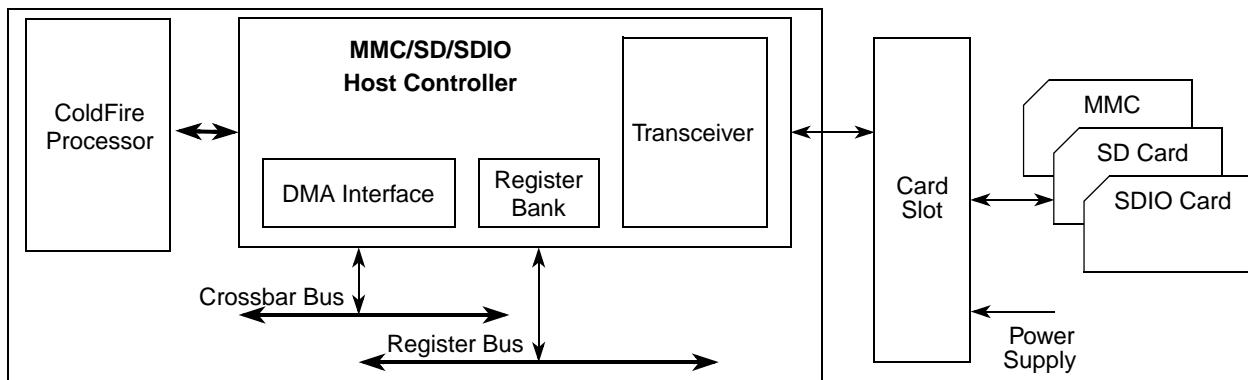
- Secure Digital (SD) card

The Secure Digital (SD) card is an evolution of MMC technology. It is specifically designed to meet the security, capacity, performance, and environment requirements inherent in the emerging audio and video consumer electronic devices. The physical form factor, pin assignments, and data transfer protocol are forward-compatible with the old MMC.

- SDIO

Under the SD protocol, the SD cards can be categorized as a memory card, I/O card, or combo card. The memory card invokes a copyright protection mechanism that complies with the security of the SDMI standard. The I/O card provides high-speed data I/O with low power consumption for mobile electronic devices. The combo card has both memory and I/O functions. For the sake of simplicity, [Figure 25-1](#) does not show cards with reduced sizes.

The eSDHC acts as a bridge, passing host bus transactions to SD/SDIO/MMC cards by sending commands and performing data accesses to or from the cards. It handles the SD/SDIO/MMC protocol at the transmission level. [Figure 25-1](#) shows connection of the eSDHC.



**Figure 25-1. System Connection of the eSDHC**

## 25.1.1 Block Diagram

Figure 25-2 is a block diagram of the eSDHC.

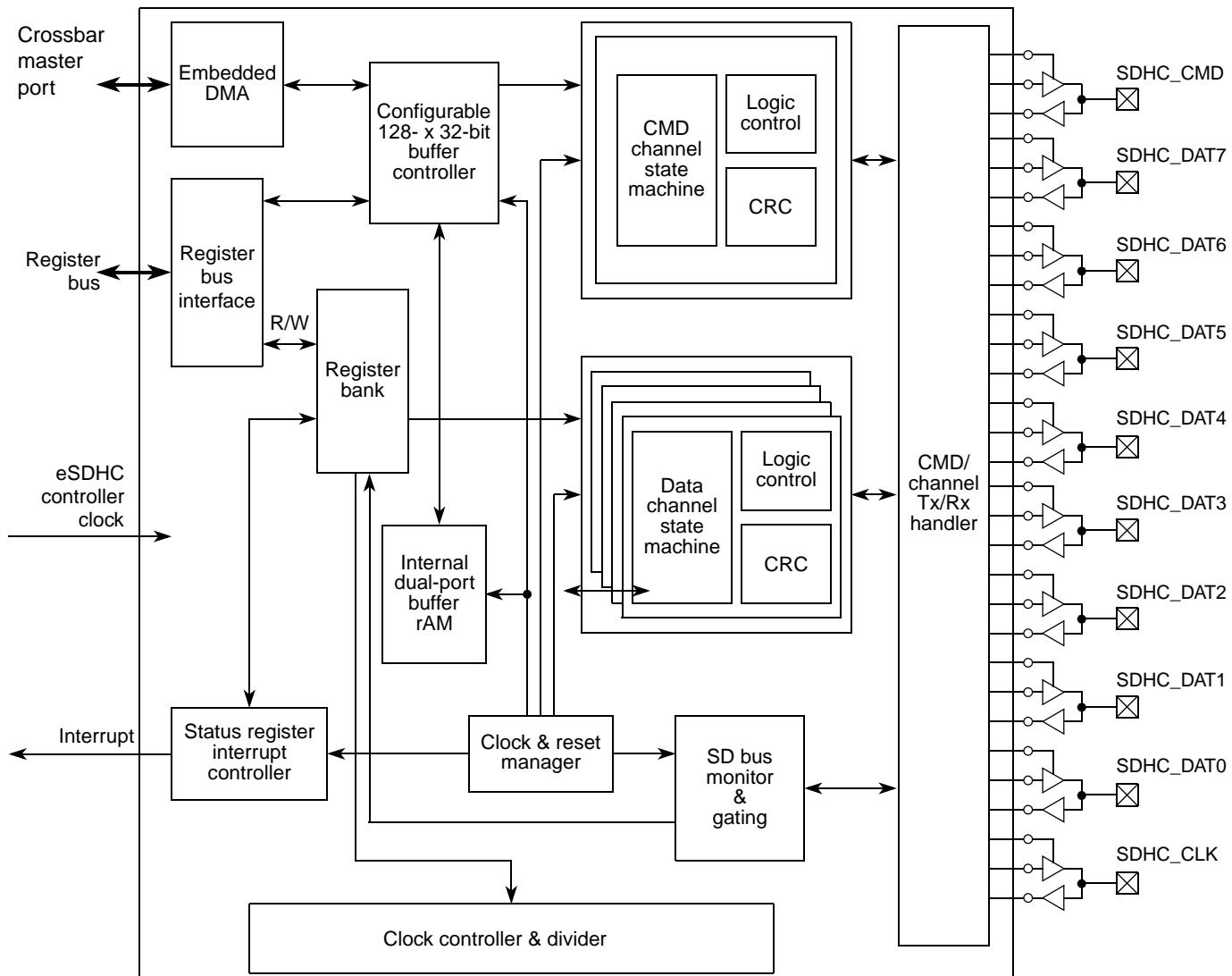


Figure 25-2. eSDHC Block Diagram

## 25.1.2 Features

The eSDHC includes the following features:

- Compatible with the following specifications:
  - *SD Host Controller Standard Specification, Version 2.0* (<http://www.sdcard.org>) with test event register and advanced DMA support
  - *MultiMediaCard System Specification, Version 4.2* (<http://www.mmca.org>)
  - *SD Memory Card Specification, Version 2.0* (<http://www.sdcard.org>), supporting high capacity SD memory cards

- SDIO Card Specification, Version 2.0 (<http://www.sdcard.org>)
- Designed to work with SD Memory, miniSD Memory, SDIO, miniSDIO, SD Combo, MMC, MMCplus, and RS-MMC cards
- SD bus clock frequency up to 25 MHz
- Supports 1-/4-bit SD and SDIO modes, 1-/4-bit MMC modes
  - Up to 200 Mbps data transfer for SD/SDIO/MMC cards using four parallel data lines
- Single- and multi-block read and write
- Write-protection switch for write operations
- Synchronous and asynchronous abort
- Pause during the data transfer at a block gap
- SDIO read wait and suspend/resume operations
- Auto CMD12 for multi-block transfer
- Host can initiate non-data transfer commands while the data transfer is in progress
- Allows cards to interrupt the host in 1- and 4-bit SDIO modes
- Supports interrupt period, defined in the SDIO standard
- Fully configurable  $128 \times 32$ -bit FIFO for read/write data
- Internal DMA capabilities
- Supports advanced DMA to perform linked memory access

### 25.1.3 Data Transfer Modes

The eSDHC can select the following modes for data transfer:

- SD 1-bit
- SD 4-bit
- MMC 1-bit
- MMC 4-bit
- Full-speed mode (up to 25 MHz)

## 25.2 External Signal Description

The eSDHC contains the following chip I/O signals:

- SDHC\_CLK is the internally generated clock signal that drives the MMC, SD, or SDIO card
- SDHC\_CMD I/O sends commands and receive responses from the card
- SDHC\_DAT3–SDHC\_DAT0 perform data transfers between the eSDHC and the card

[Table 25-1](#) shows the properties of the eSDHC I/O signals.

**Table 25-1. Signal Properties**

Name	Port	Function	Reset State	Pull up
SDHC_CLK	O	Clock for MMC/SD/SDIO card	0	N/A
SDHC_CMD	I/O	Command line to card	High Z	Pull up
SDHC_DAT3	I/O	<b>4-bit mode:</b> DAT3 line or configured as card detection pin <b>1-bit mode:</b> May be configured as card detection pin	High Z	Board should have 100K pull down. The card drives 50K pull up as required by the SD card specification.
SDHC_DAT2	I/O	<b>4-bit mode:</b> DAT2 line or read wait <b>1-bit mode:</b> Read wait	High Z	Pull up
SDHC_DAT1	I/O	<b>4-bit mode:</b> DAT1 line or interrupt detect <b>1-bit mode:</b> Interrupt detect	High Z	Pull up
SDHC_DAT0	I/O	DAT0 line or busy-state detect	High Z	Pull up

## 25.3 Memory Map/Register Definition

Table 25-2 shows the register memory map. All registers can only be accessed based on a 32-bit width.

**Table 25-2. eSDHC Memory Map**

Address	Register	Width (bits)	Access	Reset	Section/Page
0xFC0C_C000	DMA system address (DSADDR)	32	R/W	0x0000_0000	<a href="#">25.3.1/25-6</a>
0xFC0C_C004	Block attributes (BLKATTR)	32	R/W	0x0001_0000	<a href="#">25.3.2/25-7</a>
0xFC0C_C008	Command argument (CMDARG)	32	R/W	0x0000_0000	<a href="#">25.3.3/25-8</a>
0xFC0C_C00C	Command transfer type (XFERTYP)	32	R/W	0x0000_0000	<a href="#">25.3.4/25-8</a>
0xFC0C_C010	Command response0 (CMDRSP0)	32	R	0x0000_0000	<a href="#">25.3.5/25-11</a>
0xFC0C_C014	Command response1 (CMDRSP1)	32	R	0x0000_0000	<a href="#">25.3.5/25-11</a>
0xFC0C_C018	Command response2 (CMDRSP2)	32	R	0x0000_0000	<a href="#">25.3.5/25-11</a>
0xFC0C_C01C	Command response3 (CMDRSP3)	32	R	0x0000_0000	<a href="#">25.3.5/25-11</a>
0xFC0C_C020	Data buffer access port (DATPORT)	32	R/W	0x0000_0000	<a href="#">25.3.6/25-13</a>
0xFC0C_C024	Present state (PRSSTAT)	32	R	0xFF88_00F8	<a href="#">25.3.7/25-13</a>
0xFC0C_C028	Protocol control (PROCTL)	32	R/W	0x0000_0020	<a href="#">25.3.8/25-17</a>
0xFC0C_C02C	System control (SYSCTL)	32	R/W	0x0000_8008	<a href="#">25.3.9/25-20</a>
0xFC0C_C030	Interrupt status (IRQSTAT)	32	R/W	0x0000_0000	<a href="#">25.3.10/25-22</a>
0xFC0C_C034	Interrupt status enable (IRQSTATEN)	32	R/W	0x117F_013F	<a href="#">25.3.11/25-26</a>
0xFC0C_C038	Interrupt signal enable (IRQSIGEN)	32	R/W	0x0000_0000	<a href="#">25.3.12/25-29</a>
0xFC0C_C03C	Auto CMD12 status (AUTOC12ERR)	32	R	0x0000_0000	<a href="#">25.3.13/25-30</a>
0xFC0C_C040	Host controller capabilities (HOSTCAPBLT)	32	R	0x07F3_0000	<a href="#">25.3.14/25-33</a>
0xFC0C_C044 <sup>1</sup>	Watermark level (WML)	32	R/W	0x0810_0810	<a href="#">25.3.15/25-34</a>

**Table 25-2. eSDHC Memory Map (continued)**

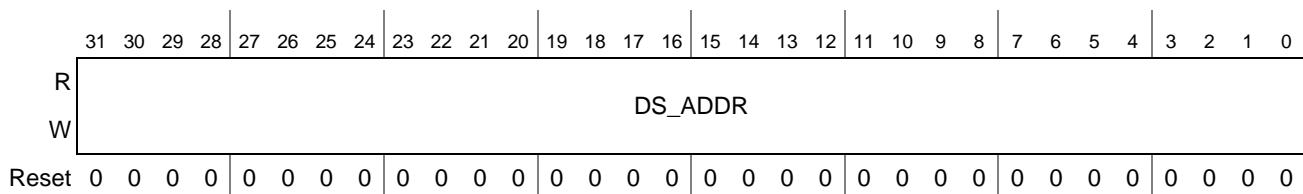
Address	Register	Width (bits)	Access	Reset	Section/Page
0xFC0C_C050	Force event (FEVT)	32	W	0x0000_0000	<a href="#">25.3.16/25-34</a>
0xFC0C_C054	ADMA error status (ADMAESR)	32	R	0x0000_0000	<a href="#">25.3.17/25-36</a>
0xFC0C_C058	ADMA system address (ADMASAR)	32	R/W	0x0000_0000	<a href="#">25.3.18/25-37</a>
0xFC0C_C0C0	Vendor specific register (VENDOR)	32	R/W	0x0000_0001	<a href="#">25.3.19/25-38</a>
0xFC0C_C0FC	Host controller version (HOSTVER)	32	R	0x0000_1201	<a href="#">25.3.20/25-39</a>

<sup>1</sup> The addresses following 0x044, except 0x050, 0x0FC and 0x40C, are reserved and read as all 0s. Writes to these registers are ignored.

### 25.3.1 DMA System Address Register (DSADDR)

The DMA system address register contains the physical system memory address used for DMA transfers. Only access this register when no transactions are executing (after transactions have stopped). The host driver must wait until PRSSTAT[DLA] is cleared.

Address: 0xFC0C\_C000 (DSADDR) Access: Read/Write



**Figure 25-3. DMA System Address Register (DSADDR)**

**Table 25-3. DSADDR Field Descriptions**

Field	Description
31–0 DS_ADDR	<p>DMA system address. When the eSDHC stops a DMA transfer, this register points to the system address of the next contiguous data position.</p> <p><b>Note:</b> The DS_ADDR must be aligned to a four-byte boundary; the two least-significant bits must be cleared.</p> <p><b>Note:</b> Since this register supports dynamic address reflecting, when TC bit is set, it automatically alters the value of internal address counter. So, software cannot change this register when the IRQSTAT[TC] bit is set.</p>

### 25.3.2 Block Attributes Register (BLKATTR)

The block attributes register configures the number of data blocks and the number of bytes in each block. Only access this register when no transactions are executing (after transactions have stopped). The host driver must wait until PRSSTAT[DLA] is cleared.

<sup>1</sup> Since at reset XFERTYP[MSBSEL] is cleared, this field reads 0x0001 after reset

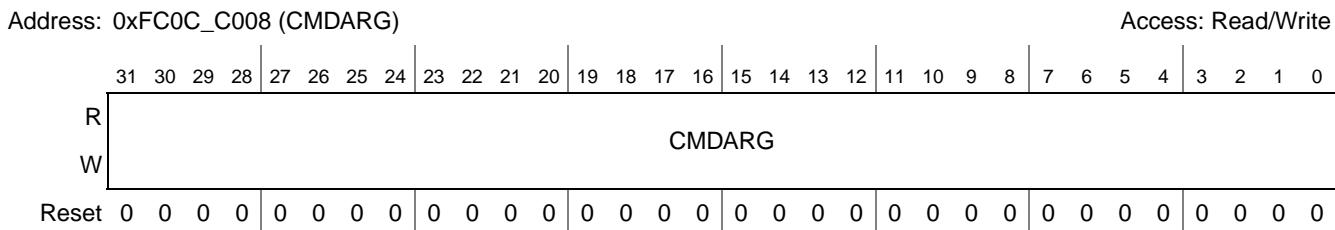
**Figure 25-4. Block Attributes Register (BLKATTR)**

## Table 25-4. BLKATTR Field Descriptions

Field	Description
31–16 BLKCNT	<p>Block count for current transfer. This field is enabled when XFERTYP[BCEN] is set and is valid only for multiple block transfers. The host driver should set this field to a value between 1 and the maximum block count. The eSDHC decrements the block count after each block transfer and stops when the count reaches zero. Clearing this field results in no data blocks being transferred.</p> <p>When saving transfer context as a result of a suspend command, this field indicates the number of blocks yet to be transferred. When restoring transfer context prior to issuing a resume command, the host driver should write the previously saved block count.</p> <ul style="list-style-type: none"> <li>0000 Stop count</li> <li>0001 1 block</li> <li>0002 2 blocks</li> <li>...</li> <li>FFFF 65,535 blocks</li> </ul> <p><b>Note:</b> When XFERTYP[MSBSEL] is cleared, this field always reads 0x0001.</p>
15–13	Reserved
12–0 BLKSIZE	<p>Transfer block size. Specifies the block size for block data transfers. Values can range from one byte up to the maximum buffer size. The DMA always writes at least four bytes to memory. Thus, software should allocate a buffer space rounded up to a 4-byte aligned size in order to avoid data corruption.</p> <ul style="list-style-type: none"> <li>0000 No data transfer</li> <li>0001 1 byte</li> <li>0002 2 bytes</li> <li>0003 3 bytes</li> <li>0004 4 bytes</li> <li>...</li> <li>01FF 511 bytes</li> <li>0200 512 bytes</li> <li>...</li> <li>0800 2048 bytes</li> <li>1000 4096 bytes</li> </ul>

### 25.3.3 Command Argument Register (CMDARG)

The command argument register contains the SD/MMC command argument.



**Figure 25-5. Command Argument Register (CMDARG)**

**Table 25-5. CMDARG Field Descriptions**

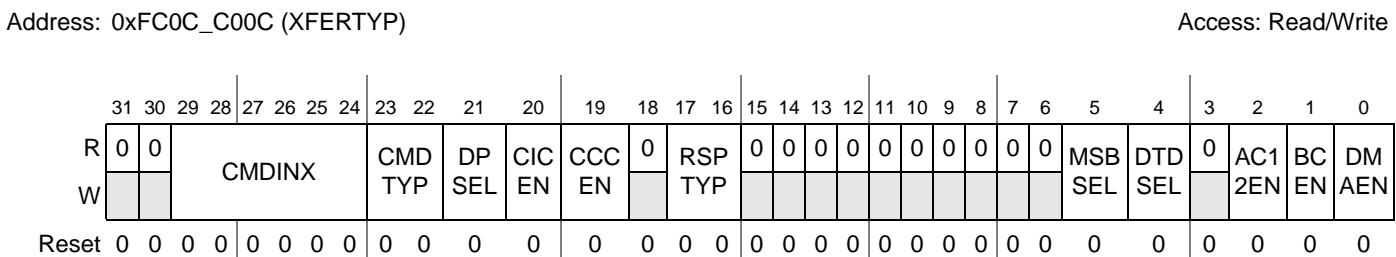
Field	Description
31–0 CMDARG	Command argument. The SD/MMC command argument is specified as bits 39–8 of the command format in the SD or MMC Specification. If PRSSTAT[CMD] is set, this register is write-protected.

#### 25.3.4 Transfer Type Register (XFERTYP)

The transfer type register controls the operation of data transfers. The host driver should set this register before issuing a command followed by a data transfer, or before issuing a resume command. To prevent data loss, the eSDHC prevents a write to the bits that are involved in the data transfer of this register while the data transfer is active.

The host driver should check PRSSTAT[CDIHB] and PRSSTAT[CIHB] before writing to this register.

- If PRSSTAT[CDIHB] is set, any attempt to send a command with data by writing to this register is ignored.
  - If PRSSTAT[CIHB] is set, any write to this register is ignored.



**Figure 25-6. Transfer Type Register (XFERTYP)**

**Table 25-6. XFERTYP Field Descriptions**

Field	Description
31–30	Reserved, must be cleared.
29–24 CMDINX	Command index. These bits should be set to the command number (CMD0–63, ACMD0–63) that is specified in bits 45–40 of the command format in the <i>SD Memory Card Physical Layer Specification</i> and <i>SDIO Card Specification</i> .

**Table 25-6. XFERTYP Field Descriptions (continued)**

Field	Description
23–22 CMDTYP	<p>Command type. There are three types of special commands: suspend, resume, and abort. Clear this bit field for all other commands.</p> <ul style="list-style-type: none"> <li>• Suspend command. If the suspend command succeeds, the eSDHC assumes the SD bus has been released and it is possible to issue the next command which uses the SDHC_DAT line. The eSDHC de-asserts read wait for read transactions and stops checking busy for write transactions. In 4-bit mode, the interrupt cycle starts. If the suspend command fails, the eSDHC maintains its current state, and the host driver should restart the transfer by setting PROCTL[CREQ]. The eSDHC does not check if the suspend command succeeds or not. It is the host driver's responsibility to issue a normal CMD52 marked as suspend command when the suspend request is accepted by the card, so that eSDHC can be informed that the SD bus is released and de-assert read wait during read operation.</li> <li>• Resume command. The host driver restarts the data transfer by restoring the registers saved before sending the suspend command and sends the resume command. The eSDHC checks for pending busy state before starting write transfers.</li> <li>• Abort command. If this command is set when executing a read transfer, the eSDHC stops reads to the buffer. If this command is set when executing a write transfer, the eSDHC stops driving the SDHC_DAT line. After issuing the abort command, the host driver should issue a software reset. (Abort transaction)</li> </ul> <p>00 Normal—other commands 01 Suspend—CMD52 for writing bus suspend in the common card control register (CCCR), defined in the SDIO specification 10 Resume—CMD52 for writing function select in CCCR 11 Abort—CMD12, CMD52 for writing I/O abort in CCCR</p>
21 DPSEL	<p>Data present select. Set to indicate that data is present and should be transferred using the SDHC_DAT line. It is cleared for the following:</p> <ul style="list-style-type: none"> <li>• Commands using only the SDHC_CMD line (e.g. CMD52)</li> <li>• Commands with no data transfer but using busy signal on the SDHC_DAT[0] line (R1b or R5b, e.g. CMD38)</li> </ul> <p><b>Note:</b> In resume command, this bit should be set while the other bits in this register should be set the same as when the transfer initially launched.</p> <p>0 No data present 1 Data present</p>
20 CICEN	<p>Command index check enable.</p> <p>0 Disable. The index field is not checked. 1 Enable. The eSDHC checks the index field in the response to see if it has the same value as the command index. If it is not, it is reported as a command index error.</p>
19 CCcen	<p>Command CRC check enable. The number of bits checked by the CRC field value changes according to the length of the response. (Refer to RSPTYP[1:0] and <a href="#">Table 25-8</a>.)</p> <p>0 Disable. The CRC field is not checked. 1 Enable. The eSDHC checks the CRC field in the response if it contains the CRC field. If an error is detected, it is reported as a command CRC error.</p>
18	Reserved, must be cleared.
17–16 RSPTYP	<p>Response type select.</p> <p>00 No response 01 Response length 136 10 Response length 48 11 Response length 48 check busy after response</p>
15–6	Reserved, must be cleared.

**Table 25-6. XFERTYP Field Descriptions (continued)**

Field	Description
5 MSBSEL	Multi/single block select. Enables multiple block SDHC_DAT line data transfers. For any other commands, this bit should be cleared. If this bit is cleared, it is not necessary to set the block count register. (Refer to <a href="#">Table 25-7</a> .) 0 Single block 1 Multiple blocks
4 DTDSEL	Data transfer direction select. Defines the direction of SDHC_DAT line data transfers. The bit is set by the host driver to transfer data from the SD card to the eSDHC and it is cleared for all other commands. 0 Write (host to card) 1 Read (card to host)
3	Reserved, must be cleared.
2 AC12EN	Auto CMD12 enable. Multiple block transfers for memory require CMD12 to stop the transaction. If this bit is set, the eSDHC issues CMD12 automatically when the last block transfer is completed. The host driver should not set this bit to issue commands that do not require CMD12 to stop a multiple block data transfer. In particular, secure commands defined in the Part 3 File Security specification do not require CMD12. In a single block transfer, the eSDHC ignores this bit. 0 Disable 1 Enable
1 BCEN	Block count enable. Enables the block attributes register, which is only relevant for multiple block transfers. When this bit is cleared, the block attributes register is disabled, which is useful in executing an infinite transfer. 0 Disable 1 Enable
0 DMAEN	DMA enable. Enables DMA functionality as described in <a href="#">Section 25.4.2, “DMA Crossbar Switch Interface.”</a> If this bit is set, a DMA operation should begin when the host driver writes to the CMDINX field of the transfer type register. 0 Disable 1 Enable

[Table 25-7](#) shows how register settings determine the data transfer type.

**Table 25-7. Determination of Transfer Type**

Multi/Single Block Select XFERTYP[MSBSEL]	Block Count Enable XFERTYP[BCEN]	Block Count BLKATTR[BLKCNT]	Function
0	—	—	Single transfer
1	0	—	Infinite transfer
1	1	Non-zero	Multiple transfer
1	1	0	No data transfer

[Table 25-8](#) shows how the response type can be determined by the command index check enable, command CRC check enable, and response type bits.

**Table 25-8. Relation Between Parameters and Name of Response Type**

<b>Response Type XFERTYP[RSPTYP]</b>	<b>Index Check Enable XFERTYP[CICEN]</b>	<b>CRC Check Enable XFERTYP[CCCEN]</b>	<b>Response Type</b>
00	0	0	No response
01	0	1	R2
10	0	0	R3, R4
10	1	1	R1, R5, R6
11	1	1	R1b, R5b

## **NOTE**

In the SDIO specification, response type notation of R5b is not defined. R5 includes R5b in the SDIO specification. But, R5b is defined in this specification to specify the eSDHC checks busy status after receiving a response. For example, usually CMD52 is used as R5 but the I/O abort command should be used as R5b.

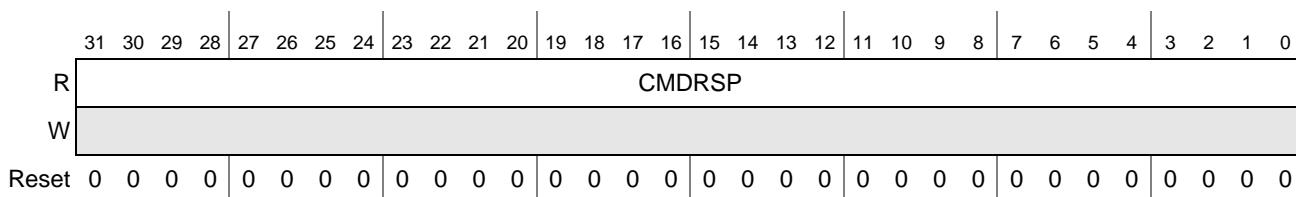
The CRC field for R3 and R4 is expected to be all ones. The CRC check should be disabled for these response types.

### **25.3.5 Command Response 0–3 (CMDRSP0–3)**

The command response registers stores the four parts of the response bits from the card.

Address: 0xFC0C\_C010 (CMDRSP0)  
0xFC0C\_C014 (CMDRSP1)  
0xFC0C\_C018 (CMDRSP2)  
0xFC0C\_C01C (CMDRSP3)

### Access: Read



**Figure 25-7. Command Response 0–3 Register (CMDRSPn)**

**Table 25-9** describes the mapping of command responses from the SD bus to the command response registers for each response type. In the table, R[ ] refers to a bit range within the response data as transmitted on the SD bus.

**Table 25-9. Response Bit Definition for Each Response Type**

<b>Response Type</b>	<b>Meaning of Response</b>	<b>Response Field</b>	<b>Response Register</b>
R1,R1b (normal response)	Card status	R[39:8]	CMDRSP0
R1b (Auto CMD12 response)	Card status for Auto CMD12	R[39:8]	CMDRSP3
R2 (CID, CSD register)	CID/CSD register [127:8]	R[127:8]	{CMDRSP3[23:0], CMDRSP2, CMDRSP1, CMDRSP0}
R3 (OCR register)	OCR register for memory	R[39:8]	CMDRSP0
R4 (OCR register)	OCR register for I/O etc.	R[39:8]	CMDRSP0
R5, R5b	SDIO response	R[39:8]	CMDRSP0
R6 (publish RCA)	New published RCA[31:16] and card status[15:0]	R[39:9]	CMDRSP0

This table shows that:

- Most responses with a length of 48 (R[47:0]) have 32 bits of the response data (R[39:8]) stored in the CMDRSP0 register.
- Responses of type R1b (Auto CMD12 responses) have response data bits R[39:8] stored in the CMDRSP3 register.
- Responses with length 136 (R[135:0]) have 120 bits of the response data (R[127:8]) stored in the CMDRSP0, 1, 2, and 3 registers.

To be able to read the response status efficiently, the eSDHC only stores part of the response data in the command response registers. This enables the host driver to efficiently read 32 bits of response data in one read cycle on a 32-bit bus system. Parts of the response, the index field, and the CRC are checked by the eSDHC (as specified by XFERTYP[CICEN, CCCEN]) and generate an error interrupt if any error is detected. The bit range for the CRC check depends on the response length. If the response length is 48, the eSDHC checks R[47:1], and if the response length is 136, the eSDHC checks R[119:1].

Since the eSDHC may have a multiple block data transfer executing concurrently with a CMD\_wo\_DAT command, the eSDHC stores the Auto CMD12 response in the CMDRSP3 register and the CMD\_wo\_DAT response is stored in CMDRSP0. This allows the eSDHC to avoid overwriting the Auto CMD12 response with the CMD\_wo\_DAT and vice versa. When the eSDHC modifies part of the command response registers it preserves the unmodified bits.

### 25.3.6 Buffer Data Port Register (DATPORT)

The buffer data port register is a 32-bit data port register used to access the internal buffer.

Address: 0xFC0C_C020 (DATPORT)																															Access: Read/Write			
R	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		
W	DATCONT																																	
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	0	0

Figure 25-8. Buffer Data Port Register (DATPORT)

Table 25-10. DATPORT Field Descriptions

Field	Description
31–0 DATCONT	Data content. The buffer data port register is for 32-bit data access by the CPU or an external DMA. When the internal DMA is enabled, any write to this register is ignored, and a read from this register always yields 0.

### 25.3.7 Present State Register (PRSSTAT)

PRSSTAT indicates the status of the eSDHC to the host driver.

Address: 0xFC0C_C024 (PRSSTAT)																															Access: Read			
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	CLSL	0	0	0	1	0	0	0	1	0	0	0	CINS					
W					DDSL																													
Reset	1	1	1	1	1	1	1	1	1	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	SD OFF	PER OFF	HCK OFF	IPG OFF	SDSTB	DLA	CDIHB	CIHB										
W					BREN	BWEN	RTA	WTA																										
Reset	0	0	0	0	0	0	0	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	

Figure 25-9. Present State Register (PRSSTAT)

**Table 25-11. PRSSTAT Field Descriptions**

<b>Field</b>	<b>Description</b>										
31–28	Reserved, must be cleared.										
27–24 DDSL	SDHC_DAT[3:0] line signal level. These bits are used to check the SDHC_DAT line level to recover from errors, and for debugging. This is especially useful in detecting the busy signal level from SDHC_DAT[0]. The reset value is affected by the external pull resistors. By default, read value of this bit field after reset is 0111, when SDHC_DAT[3] is pull-down and other lines are pull-up.  <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th><b>PRSSTAT Bit</b></th><th><b>SDHC_DAT<math>n</math></b></th></tr> </thead> <tbody> <tr> <td>27</td><td>3</td></tr> <tr> <td>26</td><td>2</td></tr> <tr> <td>25</td><td>1</td></tr> <tr> <td>24</td><td>0</td></tr> </tbody> </table>	<b>PRSSTAT Bit</b>	<b>SDHC_DAT<math>n</math></b>	27	3	26	2	25	1	24	0
<b>PRSSTAT Bit</b>	<b>SDHC_DAT<math>n</math></b>										
27	3										
26	2										
25	1										
24	0										
23 CLSL	SDHC_CMD line signal level. This status is used to check the SDHC_CMD line level to recover from errors, and for debugging. The reset value is affected by the external pull resistor, by default, read value of this bit after reset is 1, when the command line is pull-up.										
16 CINS	Card inserted. Indicates if a card has been inserted. The eSDHC debounces this signal so that the host driver does not need to wait for it to stabilize. Changing from 0 to 1 generates a card-insertion interrupt in the interrupt status register and changing from 1 to 0 generates a card removal interrupt in the interrupt status register. A write to the force event register does not affect this bit. The software reset for all in the system control register does not affect this bit. A software reset does not affect this bit. 0 Power-on-reset or no card 1 Card inserted										
15–12	Reserved, must be cleared.										
11 BREN	Buffer read enable. This status is used for non-DMA read transfers. The eSDHC may implement multiple buffers to transfer data efficiently. This read-only flag indicates that a burst-length of valid data exists in the host-side buffer. When the buffer is read, this bit is cleared. When a burst length of data is ready in the buffer, this bit is set and a buffer read ready interrupt is generated (if the interrupt is enabled). 0 Buffer read disable 1 Buffer read enable										
10 BWEN	Buffer write enable. This status is used for non-DMA write transfers. The eSDHC can implement multiple buffers to transfer data efficiently. This read-only flag indicates if space is available for a burst length of write data. When the buffer is written, this bit is cleared. When a burst length of data is written to the buffer, this bit is set and a buffer write ready interrupt is generated (if the interrupt is enabled). 0 Buffer write disable 1 Buffer write enable										
9 RTA	Read transfer active. This status is used for detecting completion of a read transfer. This bit is set for either of the following conditions: <ul style="list-style-type: none"> <li>• After the end bit of the read command</li> <li>• When writing a 1 to PROCTL[CREQ] to restart a read transfer</li> </ul> This bit is cleared for either of the following conditions: <ul style="list-style-type: none"> <li>• When the last data block as specified by block length is transferred to the system</li> <li>• When all valid data blocks have been transferred to the system and no current block transfers are being sent as a result of PROCTL[SABGREQ] being set. A transfer complete interrupt is generated when this bit changes to 0. 0 No valid data 1 Transferring data</li> </ul>										

**Table 25-11. PRSSTAT Field Descriptions (continued)**

Field	Description
8 WTA	<p>Write transfer active. This status indicates a write transfer is active. If this bit is 0, it means no valid write data exists in eSDHC.</p> <p>This bit is set in either of the following cases:</p> <ul style="list-style-type: none"> <li>• After the end bit of the write command.</li> <li>• When writing a 1 to PROCTL[CREQ] to restart a write transfer.</li> </ul> <p>This bit is cleared in either of the following cases:</p> <ul style="list-style-type: none"> <li>• After getting the CRC status of the last data block, as specified by the transfer count (single and multiple)</li> <li>• After getting the CRC status of any block where data transmission is about to be stopped by a stop-at-block-gap request.</li> </ul> <p>During a write transaction, a IRQSTAT[BGE] interrupt is generated when this bit is changed to 0, as result of PROCTL[SABGREQ] being set. This status is useful for the host driver in determining when to issue commands during write busy.</p> <p>0 No valid data 1 Transferring data</p>
7 SDOFF	<p>SDHC clock gated off internally. Indicates the SDHC clock is internally gated off because of a buffer overrun, buffer underrun, a read pause without read-wait assertion, or SYSCTL[SDCLKEN] is cleared to stop the SD clock. This bit is for the host driver to debug data transaction on SD bus.</p> <p>0 SD clock is active 1 SD clock is gated off</p>
6 PEROFF	<p>Peripheral clock gated off internally. The host driver uses this bit to debug a transaction on SD bus. When SYSCTL[NITA] is set and the eSDHC is sending 80 clock cycles to the card, SYSCTL[SDCLKEN] must be set to enable the output card clock. Otherwise, the peripheral clock is never gated off.</p>
5 HCKOFF	<p>Crossbar switch master clock internally gated off. The host driver uses this bit to debug a data transfer.</p>
4 IPGOFF	<p>Controller clock gated off internally. Indicates that the controller clock is internally gated off. This bit is for the host driver to debug.</p>
3 SDSTB	<p>SD clock stable. Indicates that the internal card clock is stable. This bit is for the host driver to poll clock status when changing the clock frequency. It is recommended to clear SYSCTL[SDCLKEN] to remove glitches on the card clock when the frequency is changing.</p> <p>0 Clock is changing frequency and not stable 1 Clock is stable</p>

**Table 25-11. PRSSTAT Field Descriptions (continued)**

Field	Description
2 DLA	<p>Data line active. Indicates whether one of the SDHC_DAT line on SD bus is in use.</p> <p>For read transactions, this bit indicates if a read transfer is executing on the SD bus. Clearing this bit from 1 to 0 between data blocks generates a block gap event interrupt.</p> <p>This bit is set in either of the following cases:</p> <ul style="list-style-type: none"> <li>• After the end bit of the read command</li> <li>• When writing a 1 to PROCTL[CREQ] to restart a read transfer</li> </ul> <p>This bit is cleared in either of the following cases:</p> <ul style="list-style-type: none"> <li>• When the end bit of the last data block is sent from the SD bus to the eSDHC</li> <li>• When beginning a read wait transfer initiated by a stop at block gap request</li> </ul> <p>The eSDHC waits at the next block gap by driving read wait at the start of the interrupt cycle. If the read-wait signal is already driven (data buffer cannot receive data), the eSDHC can wait for current block gap by continuing to drive the read-wait signal. It is necessary to support read wait in order to use the suspend/resume function.</p> <p>For write transactions, this bit indicates that a write transfer is executing on the SD bus. Clearing this bit from 1 to 0 generates a transfer complete interrupt.</p> <p>This bit is set in any of the following cases:</p> <ul style="list-style-type: none"> <li>• After the end bit of the write command</li> <li>• When writing a 1 to PROCTL[CREQ] to continue a write transfer</li> </ul> <p>This bit is cleared in any of the following cases:</p> <ul style="list-style-type: none"> <li>• When the SD card releases write-busy of the last data block, the eSDHC also detects if output is not busy. If the SD card does not drive the busy signal after CRC status is received, the eSDHC should consider the card drive not busy.</li> <li>• When the SD card releases write-busy prior to waiting for write transfer as a result of a stop at block gap request</li> </ul> <p>0 SDHC_DAT line inactive 1 SDHC_DAT line active</p>
1 CDIHB	<p>Command inhibit (SDHC_DAT). This bit is set if the SDHC_DAT line is active, the read transfer active is set, or read wait is asserted. If this bit is cleared, it indicates the eSDHC can issue the next SD/MMC command. Commands with busy signal belong to command inhibit (SDHC_DAT) (e.g. R1b and R5b type). Clearing from 1 to 0 generates a transfer complete interrupt.</p> <p><b>Note:</b> The SD host driver can save registers for a suspend transaction after this bit has cleared from 1 to 0.</p> <p>0 Can issue command which uses the SDHC_DAT line 1 Cannot issue command which uses the SDHC_DAT line</p>
0 CIHB	<p>Command inhibit (SDHC_CMD). This bit is cleared, if the SDHC_CMD line is not in use and the eSDHC can issue a SD/MMC command using the SDHC_CMD line.</p> <p>This bit is set immediately after the XFERTYP register is written. This bit is cleared when the command response is received. Even if the CDIHB bit is set, commands using only the SDHC_CMD line can be issued if this bit is cleared. Clearing from 1 to 0 generates a command complete interrupt.</p> <p>If the eSDHC cannot issue the command because of a command conflict error (refer to command CRC error) or Auto CMD12 error, this bit remains set and IRQSTAT[CC] is not set.</p> <p>0 Can issue command using only SDHC_CMD line 1 Cannot issue command</p>

## NOTE

The host driver can issue CMD0, CMD12, CMD13 (for memory) and CMD52 (for SDIO) when the SDHC\_DAT lines are busy during a data transfer. These commands can be issued when PRSSTAT[CIHB] is cleared. Other commands should be issued when PRSSTAT[CDIHB] is cleared. Possible changes to the SD physical specification may add other commands to this list in the future.

### 25.3.8 Protocol Control Register (PROCTL)

The protocol control register is shown in Figure 25-10.

Address: 0xFC0C_C028 (PROCTL)																Access: Read/Write				
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	IABG	RW CTL	CREQ	SABG REQ
	0	0	0	0	0	WE CRM	WE CINS	WE CINT	0	0	0	0								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	D3CD	DTW	0	
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 25-10. Protocol Control Register (PROCTL)

Table 25-12. PROCTL Field Descriptions

Field	Description
31–27	Reserved, must be cleared.
26 WECRM	Wake-up event enable on SD card removal. This bit enables wakeup event via card removal assertion in the IRQSTAT register. FN_WUS (wake-up support) in CIS does not affect this bit. 0 Disable 1 Enable
25 WECINS	Wake-up event enable on SD card insertion. This bit enables wakeup event via card insertion assertion in the IRQSTAT register. FN_WUS (wake-up support) in CIS does not affect this bit. 0 Disable 1 Enable
24 WECINT	Wake-up event enable on card interrupt. This bit enables wakeup event via card interrupt assertion in the IRQSTAT register. This bit can be set to 1 if FN_WUS (wake-up support) in CIS is set to 1. 0 Disable 1 Enable
23–20	Reserved, must be cleared.

**Table 25-12. PROCTL Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>
19 IABG	Interrupt at block gap. This bit is valid only in 4-bit mode of the SDIO card and selects a sample point in the interrupt cycle. If the SDIO card cannot signal an interrupt during a multiple block transfer, this bit should be cleared to avoid an inadvertent interrupt. When the host driver detects an SDIO card insertion, it should set this bit according to the CCCR of the card. 0 Disable interrupt detection during a multiple block transfer. 1 Enable interrupt detection at the block gap for a multiple block transfer.
18 RWCTL	Read wait control. The read wait function is optional for SDIO cards. If the card supports read wait, set this bit to enable the read wait protocol to stop read data using the SDHC_DAT[2] line. Otherwise, the eSDHC has to stop the SD clock to hold read data, which restricts command generation. When the host driver detects an SDIO card insertion, it should set this bit according to the CCCR of the card. If the card does not support read wait, this bit should never be set otherwise an SDHC_DAT line conflict may occur. If this bit is cleared, a stop-at-block-gap-during-read operation is also supported, but the eSDHC stops the SD clock to pause the reading operation. 0 Disable read-wait control, and stop SD clock at block gap when the SABGREQ bit is set 1 Enable read-wait control, and assert read wait without stopping the SD clock at block gap when PROCTL[SABGREQ] is set
17 CREQ	Continue request. Restarts a transaction which was stopped using the stop-at-block-gap request. To cancel the request, clear SABGREQ and set this bit to restart the transfer. The eSDHC automatically clears this bit in either of the following cases: <ul style="list-style-type: none"><li>• For a read transaction, the PRSSTAT[DLA] bit changes from 0 to 1 as a read transaction restarts.</li><li>• For a write transaction, the PRSSTAT[WTA] bit changes from 0 to 1 as the write transaction restarts.</li></ul> Therefore, it is not necessary for the host driver to clear. If SABGREQ and this bit are set, the continue request is ignored. 0 No effect 1 Restart
16 SABGREQ	Stop at block gap request. Stops executing a transaction at the next block gap for both DMA and non-DMA transfers. Until the TC bit is set, indicating a transfer completion, the host driver should leave this bit set. Clearing SABGREQ and CREQ does not cause the transaction to restart. Read wait is used to stop the read transaction at the block gap. The eSDHC honors stop-at-block-gap request for write transfers. But for read transfers it requires that the SDIO card support read wait. Therefore, the host driver should not set this bit during read transfers unless the SDIO card supports read wait and has set read wait control to 1. Otherwise, the eSDHC stops the SD bus clock to pause the read operation during the block gap.  For write transfers in which the host driver writes data to the data port register, the host driver should set this bit after all block data is written. If this bit is set, the host driver should not write data to the DATPORT register after a block is sent. When this bit is set, the host driver should not clear this bit before IRQSTAT[TC] is set. Otherwise, the eSDHC behavior is undefined. Confirm that IRQSTAT[TC] is enabled. This bit affects PRSSTAT[RTA, WTA, DLA, CIHB]. 0 Transfer 1 Stop or not resume yet
15–10	Reserved, must be cleared.
9–8 DMAS	DMA select. Selects the DMA operation when DMA is enabled. 00 Simple DMA 01 Advanced DMA 1 (ADMA1) 10 Advanced DMA 2 (ADMA2) 11 Reserved

**Table 25-12. PROCTL Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>
7 CDSS	Card detect signal selection. Selects the source for card detection. 0 Card detection level is selected (for normal purpose) 1 Card detection test level is selected (for test purpose)
6 CDTL	Card detect test level. Determines card insertion status when CDSS is set. 0 No card in the slot 1 Card is inserted
5–4 EMODE	Endian mode. eSDHC supports only address-invariant mode in data transfer. 00 Reserved 01 Reserved 10 Address-invariant mode. Each byte location in the main memory is mapped to the same byte location in the MMC/SD card. 11 Reserved
3 D3CD	SDHC_DAT3 as card detection pin. If this bit is set, SDHC_DAT3 should be pulled down to act as a card detection pin. Be cautious when using this feature, because SDHC_DAT3 is chip-select for SPI mode, and a pull-down on this pin and CMD0 may set the card into SPI mode, which the eSDHC does not support. <b>Note:</b> On this device there is no separate card-detection signal. To use card detection this bit must be set. 0 SDHC_DAT3 does not monitor card insertion 1 SDHC_DAT3 is card0detection pin
2–1 DTW	Data transfer width. Selects the data width of the SD bus. The host driver should set it to match the data width of the card. 00 1-bit mode 01 4-bit mode 10 Reserved 11 Reserved
1	Reserved, must be cleared.

There are three ways to restart the transfer after a stop at the block gap. The appropriate method depends on whether the eSDHC issues a suspend command or the SD card accepts the suspend command:

- If the host driver does not issue a suspend command, the continue request should be used to restart the transfer.
- If the host driver issues a suspend command and the SD card accepts it, a resume command should be used to restart the transfer.
- If the host driver issues a suspend command and the SD card does not accept it, PROCTL[CREQ] should be used to restart the transfer.

Any time PROCTL[SABGREQ] stops the data transfer, the host driver should wait for IRQSTAT[TC] before attempting to restart the transfer. When restarting the data transfer by continue request, the host driver should clear PROCTL[SABGREQ] before or simultaneously.

### 25.3.9 System Control Register (SYSCTL)

The system control register is shown in [Figure 25-11](#).

Address: 0xFC0C_C02C (SYSCTL)																Access: Read/write																
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	DTOCV															
W					INITA	0	0	0	0	0	0	0	0	0	0	0	DTOCV															
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			
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	SDCLKFS	DVS	SDCLK EN	PEREN	HCKEN	IPGEN										
W																																
Reset	1	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			

**Figure 25-11. System Control Register (SYSCTL)**

**Table 25-13. SYSCTL Field Descriptions**

Field	Description
31–28	Reserved, must be cleared.
27 INITA	Initialization active. When this bit is set, 80 SD clocks are sent to the card. After the 80 clocks are sent, this bit is self-cleared. This bit is very useful during the card power-up period when 74 SD clocks are needed and clock auto-gating feature is enabled. Writing one to this bit when it is already set has no effect. Clearing this bit at any time does not affect it. When PRSSTAT[CIHB] or PRSSTAT[CDIHB] is set, writing a one to this bit is ignored. That is, when the command line or data line is active, writing to this bit is not allowed.
26 RSTD	Software reset for SDHC_DAT line. The DMA and part of the data circuit are reset. The following registers and bits are cleared by this bit: <ul style="list-style-type: none"> <li>• DATPORT register</li> <li>• Buffer is cleared and initialized; PRSSTAT register</li> <li>• PRSSTAT[BREN, BWEN, RTA, WTA, DLA, CDIHB]</li> <li>• PROCTL[CREQ, SABGREQ]</li> <li>• IRQSTAT[BRR, BWR, DINT, BGE, TC]</li> </ul> 0 Work 1 Reset
25 RSTC	Software reset for SDHC_CMD line. Only part of the command circuit is reset. The following bits are cleared by this bit: <ul style="list-style-type: none"> <li>• PRSSTAT[CIHB]</li> <li>• IRQSTAT[CC]</li> </ul> 0 Work 1 Reset

**Table 25-13. SYSCTL Field Descriptions (continued)**

Field	Description
24 RSTA	Software reset for all. This reset affects the entire host controller except for the card-detection circuit. Register bits of type ROC, RW, RW1C, and RWAC are cleared. During its initialization, the host driver should set this bit to reset the eSDHC. The eSDHC should clear this bit when capabilities registers are valid and the host driver can read them. Additional use of the this bit does not affect the value of the capabilities registers. After this bit is set, it is recommended the host driver reset the external card and re-initialize it. 0 Work 1 Reset
23–20	Reserved, must be cleared.
19–16 DTOCV	Data timeout counter value. Determines the interval by which SDHC_DAT line timeouts are detected. Refer to the data timeout error <a href="#">Section 25.3.10, “Interrupt Status Register (IRQSTAT)”,</a> for information on factors that dictate timeout generation. Timeout clock frequency is generated by dividing the base clock SDHC_CLK value by this value. When setting this register, prevent inadvertent timeout events by clearing IRQSTATEN[DTOESEN]. 0000 SDHC_CLK x 2 <sup>13</sup> 0001 SDHC_CLK x 2 <sup>14</sup> ... 1110 SDHC_CLK x 2 <sup>27</sup> 1111 Reserved
15–8 SDCLKFS	SDHC_CLK frequency select. This field, together with DVS, selects the frequency of SDHC_CLK pin. This bit holds the prescaler of the base clock frequency. Only the following settings are allowed: 0x01 Base clock divided by 2 0x02 Base clock divided by 4 0x04 Base clock divided by 8 0x08 Base clock divided by 16 0x10 Base clock divided by 32 0x20 Base clock divided by 64 0x40 Base clock divided by 128 0x80 Base clock divided by 256 Multiple bits must not be set or the behavior of this prescaler is undefined. The maximum SD clock frequency is 25 MHz, and should never exceed this limit. The frequency of SDHC_CLK is set by the following formula: $\text{clock frequency} = (\text{base clock}) / [(\text{SDCLKFS} \times 2) \times (\text{DVS} + 1)]$ Eqn. 25-1 For example, if the base clock frequency is 96 MHz, and the target frequency is 25 MHz, then choosing the prescaler value of 0x1 and divisor value of 0x1 yields 24 MHz, which is the nearest frequency less than or equal to the target. Similarly, to approach a clock value of 400 KHz, the prescaler value of 0x04 and divisor value of 0xE yields the exact clock value of 400 KHz. The reset value of this bit field is 0x80. So, if the input base clock is about 96 MHz, the default SD clock after reset is 375 KHz.
7–4 DVS	Divisor. Provides a more exact divisor to generate the desired SD clock frequency. The settings are as follows: 0x0 Divide by 1 0x1 Divide by 2 ... 0xE Divide by 15 0xF Divide by 16

**Table 25-13. SYSCTL Field Descriptions (continued)**

Field	Description
3 SDCLKEN	SDHC_CLK enable. The host controller stops the SDHC_CLK when this bit is cleared. Only change the SDHC_CLK frequency when this bit is cleared. If PRSSTAT[CINS] is cleared, this bit should be cleared by the host driver to save power. 0 SDHC_CLK disabled 1 SDHC_CLK enabled
2 PEREN	Peripheral clock enable. If set, the peripheral clock is always active and no automatic gating is applied, thus SDHC_CLK is active only except auto gating-off during buffer danger. If cleared, the peripheral clock is automatically off when no transaction is on the SD bus. Clearing this bit does not stop SDHC_CLK immediately. The peripheral clock will be internally gated off, if none of the following factors are met: <ul style="list-style-type: none"><li>• Command part is reset</li><li>• Data part is reset</li><li>• Soft reset</li><li>• Command is about to send</li><li>• Clock divisor is just updated</li><li>• Continue request is just set</li><li>• This bit is set</li><li>• Card insertion is detected</li><li>• Card removal is detected</li><li>• Card external interrupt is detected</li><li>• 80 clocks for initialization phase is ongoing</li></ul> 0 The peripheral clock is internally gated off 1 The peripheral clock is not automatically gated off
1 HCKEN	Crossbar switch master clock enable. If set, the clock is always active and no automatic gating is applied. If cleared, the clock is automatically off when no data transfer is on SD bus. 0) Clock is internally gated off 1) Clock is not automatically gated off
0 IPGEN	Controller clock enable. If this bit is set, the controller clock is always active and no automatic gating is applied. The controller clock is internally gated off, if neither the following factors is met: <ul style="list-style-type: none"><li>• Command part is reset</li><li>• Data part is reset</li><li>• Soft reset</li><li>• Command is about to send</li><li>• Clock divisor is just updated</li><li>• Continue request is just set</li><li>• This bit is set</li><li>• Card insertion is detected</li><li>• Card removal is detected</li><li>• Card external interrupt is detected</li><li>• The controller clock is not gated off</li></ul> <b>Note:</b> The controller clock is not auto-gated off if the peripheral clock is not gated off. So, clearing this bit only does not take effect if SYSCTL[PEREN] is not cleared. 0 The controller clock is internally gated off 1 The controller clock is not automatically gated off

### 25.3.10 Interrupt Status Register (IRQSTAT)

An interrupt is generated when one of the status bits and its corresponding interrupt enable bit are set. For all bits, writing one to a bit clears it, while writing zero keeps the bit unchanged. More than one status can

be cleared with a single register write. For a card interrupt (IRQSTAT[CINT]), the card must stop asserting the interrupt before writing one to clear. Otherwise, the CINT bit is set again.

Address: 0xFC0C_C030 (IRQSTAT)												Access: Read/write					
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
R	0	0	0	DMAE	0	0	0	AC12E	0	DEBE	DCE	DTOE	CIE	CEBE	CCE	CTOE	
W				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	
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R	0	0	0	0	0	0	0	CINT	CRM	CINS	BRR	BWR	DINT	BGE	TC	CC	
W								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 25-12. Interrupt Status Register (IRQSTAT)

Table 25-14. IRQSTAT Field Descriptions

Field	Description
31–29	Reserved, must be cleared.
28 DMAE	DMA error. Occurs when internal DMA (simple or advanced) transfer failed. This bit is set when some error occurs in the data transfer. The value in the DMA system address register is the next fetch address where the error occurs. Since any error corrupts the entire data block, the host driver should restart the transfer from the corrupted block boundary. The address of the block boundary can be calculated from the current DS_ADDR value or the remaining number of blocks and the block size. 0 No Error 1 Error
27–25	Reserved, must be cleared.
24 AC12E	Auto CMD12 error. Occurs when one of the bits in AUTOC12ERR is set. This bit is also set when Auto CMD12 is not executed due to a previous command error. 0 No Error 1 Error
23	Reserved, must be cleared.
22 DEBE	Data end bit error. Occurs when detecting 0 at the end bit position of read data on the SDHC_DAT line or at the end bit position of the CRC. 0 No Error 1 Error <b>Note:</b> When DEBE and CINT are set, the software should ignore DEBE. But, it must not ignore the other status bits. The software should also clear this bit by writing 1 to it. It is highly recommended to clear this bit before the next transfer.
21 DCE	Data CRC error. Occurs when detecting CRC error when transferring read data on the SDHC_DAT line or when detecting the write CRC status having a value other than 0b010. 0 No Error 1 Error

**Table 25-14. IRQSTAT Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>
20 DTOE	Data timeout error. Occurs during one of following timeout conditions: <ul style="list-style-type: none"> <li>• Busy timeout for R1b and R5b types</li> <li>• Busy timeout after write CRC status</li> <li>• Read data timeout</li> </ul> 0 No error 1 Timeout
19 CIE	Command index error. Occurs if a command index error occurs in the command response. 0 No error 1 Timeout
18 CEBE	Command end bit error. Occurs when the end bit of a command response is 0. 0 No error 1 End bit error generated
17 CCE	Command CRC error. A command CRC error is generated in two cases: <ul style="list-style-type: none"> <li>• If a response is returned and IRQSTAT[CTOE] is cleared (indicating no timeout), this bit is set when detecting a CRC error in the command response.</li> <li>• The eSDHC detects a SDHC_CMD line conflict by monitoring the SDHC_CMD line when a command is issued. If the eSDHC drives the SDHC_CMD line to 1, but detects 0 on the SDHC_CMD line at the next SDHC_CLK edge, then the eSDHC aborts the command (stop driving SDHC_CMD line) and sets this bit. The CTOE bit is also set to distinguish the SDHC_CMD line conflict.</li> </ul> 0 No error 1 CRC error generated
16 CTOE	Command timeout error. Occurs if no response is returned within 64 SDHC_CLK cycles from the end bit of the command. Also, if eSDHC detects a SDHC_CMD line conflict, this bit is set along with IRQSTAT[CCE] as shown in <a href="#">Table 25-31</a> . 0 No error 1 Time out
15–9	Reserved, must be cleared.
8 CINT	Card interrupt. <ul style="list-style-type: none"> <li>• In 1-bit mode, the eSDHC detects the card interrupt without the SD clock to support wakeup.</li> <li>• In 4-bit mode, the card interrupt signal is sampled during the interrupt cycle. So, there are some sample delays between the interrupt signal from the SD card and the interrupt to the host system.</li> </ul> Writing 1 clears this bit. But, if the interrupt source from the SD card is not cleared, this bit is set again. To clear this bit, the SD card interrupt source must be cleared followed by writing 1 to this bit.            When this bit is set and the host driver needs to start the interrupt service, IRQSIGEN[CINTIEN] should be cleared to stop driving the interrupt signal to the host system. After completing the card interrupt service, write 1 to clear this bit, set IRQSIGEN[CINTIEN], and start sampling the interrupt signal again.            0 No card interrupt 1 Generate card interrupt
7 CRM	Card removal. This bit is set if PRSSTAT[CINS] changes from 1 to 0. When the host driver writes 1 to this bit to clear it, the status of PRSSTAT[CINS] should be confirmed. Because the card-detect state may be changed when the host driver clears this bit, an interrupt event may not be generated.            When this bit is cleared, it is set again if no card is inserted. To leave it cleared, clear IRQSTATEN[CRMSEN]. 0 Card state unstable or inserted 1 Card removed

**Table 25-14. IRQSTAT Field Descriptions (continued)**

Field	Description
6 CINS	<p>Card insertion. This bit is set if PRSSTAT[CINS] changes from 0 to 1. When the host driver writes 1 to this bit to clear it, the status of PRSSTAT[CINS] should be confirmed. Because the card-detect state may be changed when the host driver clears this bit, an interrupt event may not be generated.</p> <p>When this bit is cleared, it is set again if a card has been inserted. To leave it cleared, clear IRQSTATEN[CINSEN].</p> <ul style="list-style-type: none"> <li>0 Card state unstable or removed</li> <li>1 Card inserted</li> </ul>
5 BRR	<p>Buffer read ready. This bit is set if PRSSTAT[BREN] changes from 0 to 1.</p> <ul style="list-style-type: none"> <li>0 Not ready to read buffer</li> <li>1 Ready to read buffer</li> </ul>
4 BWR	<p>Buffer write ready. This bit is set if PRSSTAT[BWEN] changes from 0 to 1.</p> <ul style="list-style-type: none"> <li>0 Not ready to write buffer</li> <li>1 Ready to write buffer</li> </ul>
3 DINT	<p>DMA interrupt. Occurs when the internal DMA (simple or advanced) finishes the data transfer successfully. If errors occur during data transfer, this bit is not set. Instead, the DMAE bit is set.</p> <ul style="list-style-type: none"> <li>0 No DMA interrupt</li> <li>1 DMA interrupt is generated</li> </ul>
2 BGE	<p>Block gap event. If PROCTL[SABGREQ] is set, this bit is set when a read or write transaction is stopped at a block gap. If PROCTL[SABGREQ] is cleared, this bit is not set.</p> <p>During a read transaction, this bit is set at the falling edge of the SDHC_DAT line active status (when the transaction is stopped at SD bus timing). Read wait must be supported to use this function.</p> <p>During a write transaction, this bit is set at the falling edge of PRSSTAT[WTA] (after reading the CRC status at SD bus timing).</p> <ul style="list-style-type: none"> <li>0 No block gap event</li> <li>1 Transaction stopped at block gap</li> </ul>
1 TC	<p>Transfer complete. This bit is set when a read or write transfer is completed.</p> <p>For a read transaction, this bit is set at the falling edge of PRSSTAT[WTA]. There are two cases in which this interrupt is generated:</p> <ul style="list-style-type: none"> <li>• When a data transfer is completed, as specified by data length (after the last data has been read to the host system).</li> <li>• When data has stopped at the block gap and completed the data transfer by setting PROCTL[SABGREQ] (after valid data has been read to the host system).</li> </ul> <p>For a write transaction, this bit is set at the falling edge of PRSSTAT[DLA]. There are two cases in which this interrupt is generated:</p> <ul style="list-style-type: none"> <li>• When the last data is written to the SD card, as specified by data length and the busy signal is released.</li> <li>• When data transfers are stopped at the block gap by setting PROCTL[SABGREQ] and data transfers have completed (after valid data is written to the SD card and the busy signal is released).</li> </ul>
0 CC	<p>Command complete. This bit is set when the end bit of the command response is received (except Auto CMD12). Refer to PRSSTAT[CIHB].</p> <ul style="list-style-type: none"> <li>0 No command complete</li> <li>1 Command complete</li> </ul>

Table 25-15 below shows that command timeout error has higher priority than command complete. If both bits are set, it can be assumed that the response was not received correctly.

**Table 25-15. Relation Between Command Timeout Error and Command Complete Status**

Command Complete	Command Timeout Error	Meaning of the Status
0	0	—
Don't Care	1	Response not received within 64 SDHC_CLK cycles
1	0	Response received

Table 25-16 below shows that transfer complete has higher priority than data timeout error. If both bits are set, the data transfer can be considered complete.

**Table 25-16. Relation Between Data Timeout Error and Transfer Complete Status**

Transfer Complete	Data Timeout Error	Meaning of the Status
0	0	—
0	1	Timeout occur during transfer
1	X	Data transfer complete

The relation between command CRC error and command timeout error is shown in Table 25-17 below.

**Table 25-17. Relation Between Command CRC Error and Command Timeout Error**

Command CRC Error	Command Timeout Error	Meaning of the Status
0	0	No error
0	1	Response Timeout Error
1	0	Response CRC Error
1	1	SDHC_CMD line conflict

### 25.3.11 Interrupt Status Enable Register (IRQSTATEN)

Setting the bits of IRQSTATEN enables the corresponding interrupt status bit to be set by the specified event. If any bit is cleared, the corresponding IRQSTAT bit is also cleared and is never set.

Address: 0xFC0C\_C034 (IRQSTATEN)

Access: Read/Write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	DMAE SEN	0	0	0	AC12E SEN	0	DEBE SEN	DCE SEN	DTOE SEN	CIE SEN	CEBE SEN	CCE SEN	CTOE SEN
W																
Reset	0	0	0	1	0	0	0	1	0	1	1	1	1	1	1	1
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	CINT SEN	CRM SEN	CINS SEN	BRR SEN	BWR SEN	DINT SEN	BGE SEN	TC SEN	CC SEN
W																
Reset	0	0	0	0	0	0	0	1	0	0	1	1	1	1	1	1

Figure 25-13. Interrupt Status Enable Register (IRQSTATEN)

Table 25-18. IRQSTATEN Field Descriptions

Field	Description
31–29	Reserved, must be cleared.
28 DMAESEN	DMA error status enable 0 Masked 1 Enabled
27–25	Reserved, must be cleared.
24 AC12ESEN	Auto CMD12 error status enable 0 Masked 1 Enabled
23	Reserved, must be cleared.
22 DEBESEN	Data end bit error status enable 0 Masked 1 Enabled
21 DCESEN	Data CRC error status enable 0 Masked 1 Enabled
20 DTOESEN	Data timeout error status enable 0 Masked 1 Enabled
19 CIESEN	Command index error status enable 0 Masked 1 Enabled
18 CEBESEN	Command end bit error status enable 0 Masked 1 Enabled
17 CCESEN	Command CRC error status enable 0 Masked 1 Enabled

**Table 25-18. IRQSTATEN Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>
16 CTOESEN	Command timeout error status enable 0 Masked 1 Enabled
15–9	Reserved, must be cleared.
8 CINTSEN	Card interrupt status enable. If this bit is cleared, the eSDHC clears the interrupt request to the system. The card interrupt detection is stopped when this bit is cleared and restarted when this bit is set. To prevent inadvertent interrupts, the host driver should clear this bit before servicing the card interrupt and should set this bit again after all interrupt requests from the card are cleared. 0 Masked 1 Enabled
7 CRMSEN	Card removal status enable 0 Masked 1 Enabled
6 CINSEN	Card insertion status enable 0 Masked 1 Enabled
5 BRRSEN	Buffer read ready status enable 0 Masked 1 Enabled
4 BWRSEN	Buffer write ready status enable 0 Masked 1 Enabled
3 DINTSEN	DMA interrupt status enable 0 Masked 1 Enabled
2 BGESEN	Block gap event status enable 0 Masked 1 Enabled
1 TCSEN	Transfer complete status enable 0 Masked 1 Enabled
0 CCSEN	Command complete status enable 0 Masked 1 Enabled

**NOTE**

The eSDHC may sample the card interrupt signal during the interrupt period and hold its value in the flip-flop. As a result of synchronization, there is a delay in the card interrupt (which is asserted from the card) to the time the host system is informed.

To detect a SDHC\_CMD line conflict, the host driver must set both CTOESEN and CCSEN bits.

### 25.3.12 Interrupt Signal Enable Register (IRQSIGEN)

IRQSIGEN selects which interrupt status is indicated to the host system as the interrupt. These status bits all share the same interrupt line. Setting any of these bits enables an interrupt generation. The corresponding status register bit generates an interrupt when the corresponding interrupt signal enable bit is set.

Address: 0xFC0C_C038 (IRQSIGEN)																Access: Read/Write							
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16							
R	0	0	0	DMAE IEN	0	0	0	AC12E IEN	0	DEBE IEN	DCE IEN	DTOE IEN	CIE IEN	CEBE IEN	CCE IEN	CTOE IEN							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0							
	0	0	0	0	0	0	0	CINT IEN	CRM IEN	CINS IEN	BRR IEN	BWR IEN	DINT IEN	BGE IEN	TC IEN	CC IEN							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							

Figure 25-14. Interrupt Signal Enable Register (IRQSIGEN)

Table 25-19. IRQSIGEN Field Descriptions

Field	Description
31–29	Reserved, must be cleared.
28 DMAEIEN	DMA error interrupt enable 0 Masked 1 Enabled
27–25	Reserved, must be cleared.
24 AC12EIEN	Auto CMD12 error interrupt enable 0 Masked 1 Enabled
23	Reserved, must be cleared.
22 DEBEIEN	Data end bit error interrupt enable 0 Masked 1 Enabled
21 DCEIEN	Data CRC error interrupt enable 0 Masked 1 Enabled
20 DTOEIEN	Data timeout error interrupt enable 0 Masked 1 Enabled
19 CIEIEN	Command index error interrupt enable 0 Masked 1 Enabled

**Table 25-19. IRQSIGEN Field Descriptions (continued)**

Field	Description
18 CEBEIEN	Command end bit error interrupt enable 0 Masked 1 Enabled
17 CCEIEN	Command CRC error interrupt enable 0 Masked 1 Enabled
16 CTOEIEN	Command timeout error interrupt enable 0 Masked 1 Enabled
15–9	Reserved, must be cleared.
8 CINTIEN	Card interrupt signal enable 0 Masked 1 Enabled
7 CRMIEN	Card removal interrupt enable 0 Masked 1 Enabled
6 CINIEN	Card insertion interrupt enable 0 Masked 1 Enabled
5 BRIEN	Buffer read ready interrupt enable 0 Masked 1 Enabled
4 BWRIEN	Buffer write ready interrupt enable 0 Masked 1 Enabled
3 DINTIEN	DMA interrupt enable 0 Masked 1 Enabled
2 BGEIEN	Block gap event interrupt enable 0 Masked 1 Enabled
1 TCIEN	Transfer complete interrupt enable 0 Masked 1 Enabled
0 CCIEN	Command complete interrupt enable 0 Masked 1 Enabled

### 25.3.13 Auto CMD12 Error Status Register (AUTOC12ERR)

When IRQSTAT[AC12E] is set, the host driver checks this register to identify what kind of error Auto CMD12 indicated. This register is valid only when IRQSTAT[AC12E] is set.

Address: 0xFC0C\_C03C (AUTOC12ERR)

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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CNIB AC12E	0	0	AC12 IE	AC12 CE	AC12 EBC	AC12 TOE	AC12 NE
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		

Figure 25-15. Auto CMD12 Error Status Register (AUTOC12ERR)

Table 25-20. AUTOC12ERR Field Descriptions

Field	Description
31–8	Reserved, must be cleared.
7 CNIBAC12E	Command not issued by Auto CMD12 error. This bit is set when CMD_wo_DAT is not executed due to an Auto CMD12 error (D04–D01). 0 No error 1 Not Issued
6–5	Reserved, must be cleared.
4 AC12IE	Auto CMD12 index error. Occurs if the command index error occurs in response to a command. 0 No error 1 Error, the CMD index in response is not CMD12
3 AC12CE	Auto CMD12 CRC error. Occurs when detecting a CRC error in the command response. 0 No CRC error 1 CRC error met in Auto CMD12 response
2 AC12EBE	Auto CMD12 end bit error. Occurs when detecting that the end bit of command response is 0 when it should be 1. 0 No error 1 End bit error generated
1 AC12TOE	Auto CMD12 timeout error. Occurs if no response is returned within 64 SDHC_CLK cycles from the end bit of the command. If this bit is set, the other error status bits (2–4) are meaningless. 0 No error 1 Time out
0 AC12NE	Auto CMD12 not executed. If a memory multiple block data transfer is not started due to command error, this bit is not set because it is not necessary to issue Auto CMD12. Setting this bit means eSDHC cannot issue Auto CMD12 to stop the memory multiple block data transfer due to some error. If this bit is set, the other error status bits (1–4) are meaningless. 0 Executed 1 Not executed

Table 25-21. Relationship Between Command CRC Error and Command Timeout Error for Auto CMD12

Auto CMD12 CRC Error	Auto CMD12 Timeout Error	Types of Error
0	0	No error

**Table 25-21. Relationship Between Command CRC Error and Command Timeout Error for Auto CMD12 (continued)**

Auto CMD12 CRC Error	Auto CMD12 Timeout Error	Types of Error
0	1	Response timeout error
1	0	Response CRC error
1	1	SDHC_CMD line conflict

There are three scenarios when AUTOC12ERR can be changed:

1. When eSDHC is going to issue Auto CMD12
  - Set AC12NE if Auto CMD12 cannot be issued due to an error in the previous command.
  - Clear AC12NE if Auto CMD12 is issued.
2. At the end bit of an Auto CMD12 response
  - Check received responses by checking the error bits 1–4.
  - Set if error is detected.
  - Clear if error is not detected.
3. Before reading AUTOC12ERR[CNIBAC12E]
  - Set CNIBAC12E if there is a command that cannot be issued
  - Clear CNIBAC12E if there is no command to issue

The timing of generating the Auto CMD12 error and writing to the command register is asynchronous. The command may be blocked by any Auto CMD12 error causing CNIBAC12E to be set. Therefore, it is suggested to read this register only when IRQSTAT[AC12E] is set. An Auto CMD12 error interrupt is generated when one of the error bits 0–4 is set1. The CNIBAC12E error bit does not generate an interrupt.

### 25.3.14 Host Controller Capabilities (HOSTCAPBLT)

The host controller capabilities provides the host driver with information specific to the eSDHC implementation. The value in this register does not change in a software reset, and any write to this register is ignored.

Address: 0xFC0C_C040 (HOSTCAPBLT)																Access: Read			
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16			
R	0	0	0	0	0	VS18	VS30	VS33	SRS	DMAS	HSS	ADMAS	0	MBL					
W																			
Reset	0	0	0	0	0	1	1	1	1	1	1	1	0	0	1	1			
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	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 25-16. Host Capabilities Register (HOSTCAPBLT)

Table 25-22. HOSTCAPBLT Field Descriptions

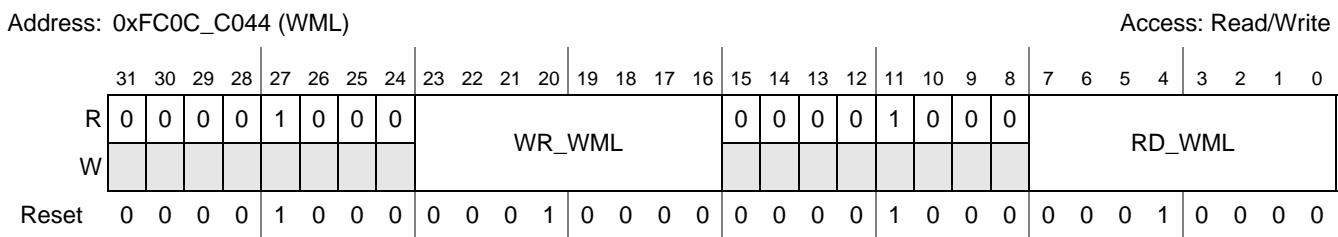
Field	Description
31–27	Reserved, must be cleared.
26 VS18	Voltage support 1.8 V. This bit depends on the host system ability. 0 1.8 V not supported 1 1.8 V supported
25 VS30	Voltage support 3.0 V. This bit depends on the host system ability. 0 3.0 V not supported 1 3.0 V supported
24 VS33	Voltage support 3.3 V. This bit depends on the host system ability. 0 3.3 V not supported 1 3.3 V supported
23 SRS	Suspend/resume support. Indicates if eSDHC supports suspend/resume functionality. If this bit is 0, the suspend and resume mechanism, as well as the read wait, are not supported and the host driver should not issue suspend or resume commands. 0 Not supported 1 Supported
22 DMAS	DMA support. Indicates if eSDHC is capable of using internal DMA to transfer data between system memory and the data buffer directly. 0 DMA not supported 1 DMA supported
21 HSS	High speed support. Indicates if the eSDHC supports high speed mode and the host system can supply the SD clock frequency from 25 to 50 MHz. 0 High speed not supported 1 High speed supported

**Table 25-22. HOSTCAPBLT Field Descriptions (continued)**

Field	Description
20 ADMAS	ADMA support. Indicates if the eSDHC supports advanced DMA feature. 0) Advanced DMA not supported 1) Advanced DMA supported
19	Reserved, must be cleared.
18–16 MBL	Max block length. Indicates the maximum block size that the host driver can read and write to the buffer in the eSDHC. The buffer should transfer block size without wait cycles. 000 512 bytes 001 1024 bytes 010 2048 bytes 011 4096 bytes
15–0	Reserved, must be cleared.

### 25.3.15 Watermark Level Register (WML)

Both write and read watermark levels are configurable. The value can be any number from 1–128 words.

**Figure 25-17. Watermark Level Register (WML)****Table 25-23. WML Field Descriptions**

Field	Description
31–24	Reserved, must be cleared.
23–16 WR_WML	Write watermark level. Number of 32-bit words of watermark level in DMA write operation. Also, the number of words of write burst length.
15–8	Reserved, must be cleared.
7–0 RD_WML	Read watermark level. Number of 32-bit words of watermark level in DMA read operation. Also, the number of words of read burst length. <b>Note:</b> The maximum value for RD_WML is 0x10, which means 16 words (64 bytes). Setting RD_WML to a higher value results in non-predicted behavior.

### 25.3.16 Force Event Register (FEVT)

The force event register is not a physically implemented register. Rather, it is an address to which the IRQSTAT register can be written if the corresponding bit of IRQSTATEEN is set. Therefore, this register is a write-only register and writing zero has no effect. Writing 1 to this register sets the corresponding bit of IRQSTAT. Reading from this register always returns zeroes.

Forcing a card interrupt generates a short pulse on the SDHC\_DAT[1] line, and the driver may treat this interrupt as normal. The interrupt service routine may skip polling the card-interrupt source as the interrupt is self-cleared.

Address: 0xFC0C\_C050 (FEVT)

Access: Write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W	FEVT CINT			FEVT DMAE				FEVT AC12E		FEVT DEBE	FEVT DCE	FEVTD TOE	FEVT CIE	FEVT CEBE	FEVT CCE	FEVT CTOE
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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W									FEVTCNI BAC12E			FEVT AC12IE	FEVTA C12EBE	FEVTA C12CE	FEVTA C12TOE	FEVTA C12NE
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 25-18. Force Event Register (FEVT)

Table 25-24. FEVT Field Descriptions

Field	Description
31 FEVTCINT	Force event card interrupt. Writing 1 to this bit generates a low-level short pulse on the internal SDHC_DAT[1] line, which imitates a self-clearing interrupt from the external card. If enabled, IRQSTAT[CINT] is set and the interrupt service routine may treat this interrupt as a normal interrupt from the external card.
30–29	Reserved, must be cleared.
28 FEVTDMAE	Force event DMA error. Forces IRQSTAT[DMAE] to set.
27–25	Reserved, must be cleared.
24 FEVTAC12E	Force event Auto CMD12 error. Forces IRQSTAT[AC12E] to set.
23	Reserved, must be cleared.
22 FEVTDEBE	Force event data end bit error. Forces IRQSTAT[DEBE] to set.
21 FEVTDCE	Force event data CRC error. Forces IRQSTAT[DCE] to set.
20 FEVTDTOE	Force event data time out error. Forces IRQSTAT[DTOE] to set.
19 FEVTCIE	Force event command index error. Forces IRQSTAT[CCE] to set.
18 FEVTCEBE	Force event command end bit error. Forces IRQSTAT[CEBE] to set.

**Table 25-24. FEVT Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>
17 FEVTCCE	Force event command CRC error. Forces IRQSTAT[CCE] to set.
16 FEVTCCE	Force event command time out error. Forces IRQSTAT[CTOE] to set.
15–8	Reserved, must be cleared.
7 FEVTCNIBAC12E	Force event command not executed by Auto CMD12 error. Forces AUTOC12ERR[CNIBAC12E] to set.
6–5	Reserved, must be cleared.
4 FEVTAC12IE	Force event Auto CMD12 index error. Forces AUTOC12ERR[AC12IE] to set.
3 FEVTAC12EBE	Force event Auto CMD12 end bit error. Forces AUTOC12ERR[AC12EBE] to set.
2 FEVTAC12CE	Force event Auto CMD12 CRC error. Forces AUTOC12ERR[AC12CE] to set.
1 FEVTAC12TOE	Force event Auto CMD12 time out error. Forces AUTOC12ERR[AC12TOE] to set.
0 FEVTAC12NE	Force event Auto CMD12 not executed. Forces AUTOC12ERR[AC12NE] to set.

### 25.3.17 ADMA Error Status Register

When an ADMA error interrupt occurs, this register contains the ADMA state. For recovering from the error, the host driver uses the ADMA state to identify the error descriptor address as follows:

- ST\_STOP — The previous location set in the ADMA system address register is the error descriptor address
- ST\_FDS — The current location set in the ADMA system address register is the error descriptor address
- ST\_CADR — This state is never set because it only increments the descriptor pointer and does not generate an ADMA error
- ST\_TFR — The previous location set in the ADMA system address register is the error descriptor address

In a write operation, the host driver should use ACMD22 to get the number of written blocks rather than using this information, since unwritten data may exist in the host controller.

The host controller generates an ADMA error interrupt when it detects invalid descriptor data (Valid=0) at the ST\_FDS state. The host driver distinguishes this error by reading the valid bit of the error descriptor.

Address: 0xFC0C\_C054 (ADMAESR)

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	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	DCE	LME	ES				
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		

Figure 25-19. ADMA Error Status Register (ADMAESR)

Table 25-25. ADMAESR Field Descriptions

Field	Description																	
31–4	Reserved, must be cleared.																	
3 DCE	Descriptor error. Occurs when an invalid descriptor is fetched by the ADMA. 0 No error 1 Error																	
2 LME	ADMA length mismatch error. This error occurs when either: <ul style="list-style-type: none"><li>If block count enable is set, the total data length specified by the descriptor table is different than that specified by the block count and block length.</li><li>Total data length is not evenly divisible by the block length.</li></ul> 0) No error 1) Error																	
1–0 ES	ADMA error state. Indicates the state of ADMA when an error occurred during ADMA data transfer.  <table border="1"><thead><tr><th>ES</th><th>ADMA Error State</th><th>Contents of ADMASAR</th></tr></thead><tbody><tr> <td>00</td><td>ST_STOP (Stop DMA)</td><td>Points to the next descriptor</td></tr><tr> <td>01</td><td>ST_FDS (Fetch descriptor)</td><td>Points to the error descriptor</td></tr><tr> <td>10</td><td>ST_CADR (Change address)</td><td>No ADMA error is generated</td></tr><tr> <td>11</td><td>ST_TFR (Transfer data)</td><td>Points to the next descriptor</td></tr></tbody></table>			ES	ADMA Error State	Contents of ADMASAR	00	ST_STOP (Stop DMA)	Points to the next descriptor	01	ST_FDS (Fetch descriptor)	Points to the error descriptor	10	ST_CADR (Change address)	No ADMA error is generated	11	ST_TFR (Transfer data)	Points to the next descriptor
ES	ADMA Error State	Contents of ADMASAR																
00	ST_STOP (Stop DMA)	Points to the next descriptor																
01	ST_FDS (Fetch descriptor)	Points to the error descriptor																
10	ST_CADR (Change address)	No ADMA error is generated																
11	ST_TFR (Transfer data)	Points to the next descriptor																

### 25.3.18 ADMA System Address Register

This register contains the physical system memory address used for ADMA transfers. The lower two bits of this register is tied to zero so that the ADMA address is always word-aligned.

Address: 0xFC0C\_C058 (ADMASAR)

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																																
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		

Figure 25-20. ADMA System Address Register (ADMASAR)

**Table 25-26. ADMASAR Field Descriptions**

Field	Description
31–2 ADDR	ADMA system address. Contains the word address of the executing command of the descriptor table. At the start of ADMA, the host driver sets the start address of the descriptor table. The ADMA engine increments this register address every descriptor command fetch. When the ADMA is stopped at the block gap, this register indicates the address of the next executable descriptor command. If an ADMA error interrupt is generated, this register contains a valid descriptor address depending on the ADMA state. <b>Note:</b> Since this register supports dynamic address reflecting, when TC bit is set, it automatically alters the value of internal address counter. So, software cannot change this field when IRQSTAT[TC] is set.
1–0	Reserved, must be cleared

### 25.3.19 Vendor Specific Register

This register contains the vendor specific control/status bits.

Address: 0xFC0C_C0C0 (VENDOR)																Access: Read/write				
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	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					
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	R	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				1	

**Figure 25-21. Vendor Specific Register (VENDOR)****Table 25-27. VENDOR Field Descriptions**

Field	Description
31–1	Reserved
0 EXT_DMA_EN	External DMA Request Enable Enable the request to external DMA. When the internal DMA (either Simple DMA or Advanced DMA) is not in use and this bit is set, the eSDHC sends a DMA request when the internal buffer is ready. This bit is particularly useful when transferring data by CPU polling mode, and it is not allowed to send out the external DMA request. By default, this bit is set. 0) In any scenario, eSDHC does not send out external DMA request 1) When internal DMA is not active, the external DMA request will be sent out

### 25.3.20 Host Controller Version Register (HOSTVER)

The host controller version register contains the version for the vendor and the host controller. All the bits are read-only.

Address: 0xFC0C_C0FC (HOSTVER)																Access: Read									
R				VVN								SVN													
W																									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1

**Figure 25-22. Host Controller Version Register (HOSTVER)**

**Table 25-28. HOSTVER Field Descriptions**

Field	Description										
31–16	Reserved, must be cleared.										
15–8 VVN	<p>Vendor version number. The host driver should not use this status. The upper and the lower 4-bits indicate the version.</p> <table> <tr> <td>0x00</td> <td>Freescale eSDHC version 1.0</td> </tr> <tr> <td>0x10</td> <td>Freescale eSDHC version 2.0</td> </tr> <tr> <td>0x11</td> <td>Freescale eSDHC version 2.1</td> </tr> <tr> <td>0x12</td> <td>Freescale eSDHC version 2.2</td> </tr> <tr> <td>others</td> <td>Reserved</td> </tr> </table>	0x00	Freescale eSDHC version 1.0	0x10	Freescale eSDHC version 2.0	0x11	Freescale eSDHC version 2.1	0x12	Freescale eSDHC version 2.2	others	Reserved
0x00	Freescale eSDHC version 1.0										
0x10	Freescale eSDHC version 2.0										
0x11	Freescale eSDHC version 2.1										
0x12	Freescale eSDHC version 2.2										
others	Reserved										
7–0 SVN	<p>Specification version number. Indicates for the host controller specification version. The upper and the lower 4-bits indicate the version.</p> <table> <tr> <td>0x00</td> <td>SD Host Specification Version 1.0</td> </tr> <tr> <td>0x01</td> <td>SD Host Specification Version 2.0, supports the test event register.</td> </tr> <tr> <td>others</td> <td>Reserved</td> </tr> </table>	0x00	SD Host Specification Version 1.0	0x01	SD Host Specification Version 2.0, supports the test event register.	others	Reserved				
0x00	SD Host Specification Version 1.0										
0x01	SD Host Specification Version 2.0, supports the test event register.										
others	Reserved										

## 25.4 Functional Description

The following sections provide a brief functional description of the major system blocks, including the data buffer, DMA crossbar switch interface, register bank, register bus interface, dual-port memory wrapper, data/command controller, clock and reset manager, and clock generator.

### 25.4.1 Data Buffer

The eSDHC uses one configurable data buffer so that data can be transferred between the internal system bus and the SD card in an optimized manner to maximize throughput between the two clock domains (the peripheral clock and the crossbar switch master clock). See [Figure 25-23](#) for an illustration of the buffer scheme.

The buffer is used as temporary storage for data being transferred between the host system and the card. The burst lengths for read and write are both configurable and can be any value between 1 and 128 words.

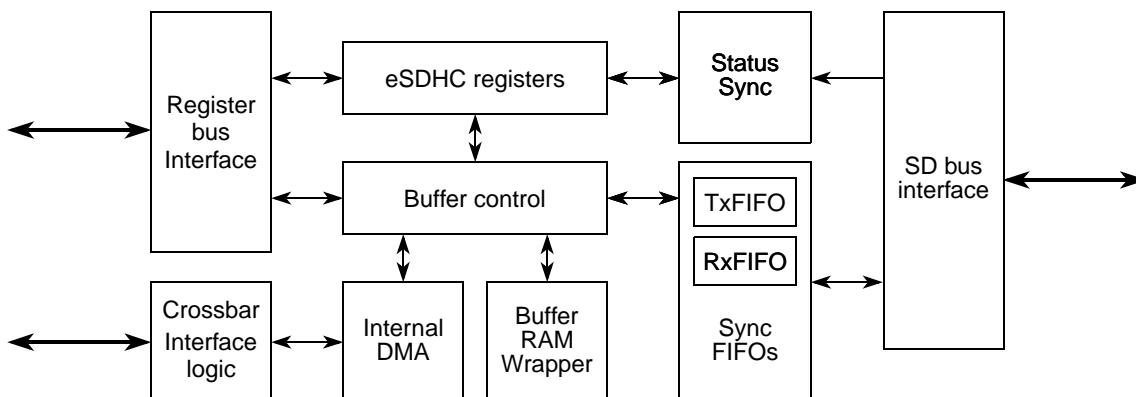


Figure 25-23. eSDHC Buffer Scheme

For a host read operation, when the amount of data exceeds the RD\_WML value, the eSDHC sets PRSSTAT[BREN] and either:

- Issues a DMA request to inform the system to read the data
- Issues a DMA interrupt to inform the system to read the data
- When granted crossbar access permission, the internal DMA burst-reads RD\_WML number of words

Conversely, for a host write operation, when the amount of buffer spaces exceeds the WR\_WML value, the eSDHC sets PRSSTAT[BWEN] and either:

- Issues a DMA request to inform the system to write data to the buffer
- Issues a DMA interrupt to inform the system to write data to the buffer
- When granted crossbar access permission, the internal DMA burst-writes WR\_WML number of words into the buffer

#### 25.4.1.1 Write Operation Sequence

There are three ways to write data into the buffer when the user transfers data to the card.

- The external DMA through the eSDHC DMA request signal
- The processor core polling IRQSTAT[BWR] (interrupt or polling)
- The internal DMA

When the internal DMA is not used (XFERTYP[DMAEN] is not set when the command is sent), the eSDHC asserts an external DMA request when more than WML[WR\_WML] number of empty buffer word slots are available and ready for receiving new data. At the same time, the eSDHC sets IRQSTAT[BWR]. The buffer write ready interrupt is generated if it is enabled by software.

When the internal DMA is used, the eSDHC does not inform the system before all the required number of bytes are transferred and no error is encountered. When an error occurs during the data transfer, the eSDHC aborts the data transfer and abandons the current block. The host driver should read the content of the DMA system address register to obtain the start address of abandoned data block. If the current data transfer is in multi-block mode, the eSDHC does not automatically send CMD12 even though XFERTYP[AC12EN] is set. Therefore, in this scenario, the host driver should send CMD12 and restart

the write operation from that address. It is recommended that a software reset for data is applied before the transfer is restarted after error recovery.

The eSDHC does not start data transmission until the WML[WR\_WML] number of words of data can be held in the buffer. If the buffer is empty and the host system does not write data in time, the eSDHC stops the SDHC\_CLK to avoid a data buffer underrun situation.

### **25.4.1.2 Read Operation Sequence**

There are three ways to read data from the buffer when transferring data to the card.

- The external DMA through the eSDHC DMA request signal
- The processor core polling IRQSTAT[BRR] (interrupt or polling)
- The internal DMA

When the internal DMA is not used (XFERTYP[DMAEN] is not set when the command is sent), the eSDHC asserts a DMA request when more than WML[RD\_WML] number of words are available and ready for the system to fetch the data. At the same time, the eSDHC sets the IRQSTAT[BRR] bit. The buffer read ready interrupt is generated if it is enabled by software.

When the internal DMA is used, the eSDHC does not inform the system before all the required number of bytes are transferred and no error is encountered. When an error occurs during the data transfer, the eSDHC aborts the data transfer and abandons the current block. The host driver should read the content of the DMA system address register to obtain the start address of abandoned data block. If the current data transfer is in multi-block mode, the eSDHC does not automatically send CMD12 even though XFERTYP[AC12EN] is set. Therefore, in this scenario, the host driver should send CMD12 and restart the read operation from that address. It is recommended that a software reset for data is applied before the transfer is restarted after error recovery.

The eSDHC does not start data transmission until the WML[RD\_WML] number of words of data are in the buffer. If the buffer is full and the host system does not read the data in time, the eSDHC stops the SDHC\_CLK to avoid a data buffer overrun situation.

### **25.4.1.3 Data Buffer Size**

To use the buffer in the most optimized way, the buffer size must be known. In the eSDHC the data buffer can hold up to 128 32-bit words, and the read and write watermark levels are each configurable from 1–128 words. The host driver may configure the values according to the system situation and requirements.

During multi-block data transfer, the maximum block length is 4096 bytes, which can satisfy all the requirements from MMC, SD, and SDIO cards. Any block length less than this value is also allowed. The only restriction is from the external card since it may not support such a large block or a partial block access that is not an integer multiple of 512 bytes.

### **25.4.1.4 Dividing Large Data Transfer**

This SDIO command CMD53 definition limits the maximum data size of data transfers according to the following formula:

$$\text{Maximum data size} = (\text{block size}) \times (\text{block count})$$

**Eqn. 25-2**

The length of a multiple block transfer must be in block size units. If the total data length cannot be divided evenly to a multiple of the block size, then there are two ways to transfer the data depending on the function and card design.

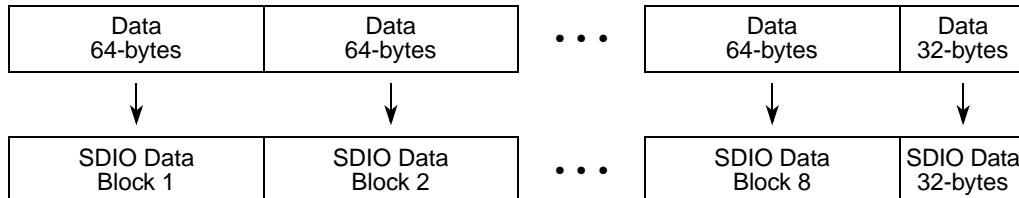
- The card driver splits the transaction. The remainder of block size data is then transferred using a single block command at the end.
- Add dummy data in the last block to fill the block size. The card must remove the dummy data.

See [Figure 25-24](#) for an example of dividing large data transfers. Although the eSDHC supports a block size of up to 4096 bytes, the example below illustrates a maximum of 64 bytes where the data must be divided.

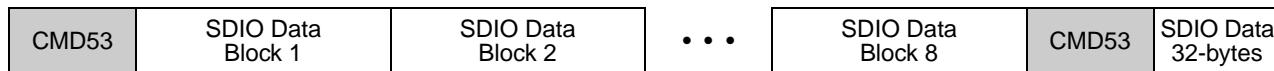
544-bytes WLAN Frame



*WLAN Frame is divided equally into 64-byte blocks plus the remainder 32-bytes*



*Eight 64-byte blocks are sent in Block Transfer Mode and the remainder 32-bytes are sent in Byte Transfer Mode*

**Figure 25-24. Example of Dividing a Large Data Transfer**

### 25.4.1.5 External DMA Request

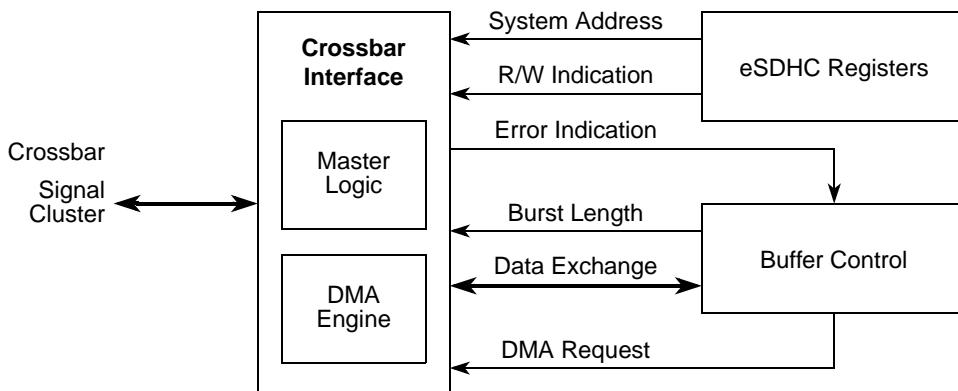
When the internal DMA is not enabled, the data buffer generates a DMA request to the system. During a write operation, when the number of WR\_WML words can be held in the buffer, a DMA request is asserted to the processor for a DMA burst write. IRQSTAT[BWR] is also set if IRQSTATEN[BWRSEN] is set. The DMA request is immediately deasserted when an access on the data port register is made. If another write burst is allowed, the DMA request is asserted again after a cycle.

Likewise, during a read operation, when the number of RD\_WML words are in the buffer, a DMA request is asserted to the processor for a DMA burst read. IRQSTAT[BRR] is also set if IRQSTATEN[BRRSEN] is set. The DMA request is immediately deasserted when an access on the data port register is made. If another read burst is allowed, the DMA request is asserted again after a cycle.

Since the DMA burst length cannot change during a data transfer, the read or write burst length must be a divisor of the block size. For example, if the block size is 512 bytes, the burst length must be 1, 2, 4, 8, 16..., or 128 words.

## 25.4.2 DMA Crossbar Switch Interface

The internal DMA implements a DMA engine and crossbar switch master. When the internal DMA is enabled (XFERTYP[DMAEN] is set), the buffer interrupt status bits are still set if they are enabled. To avoid setting them, clear IRQSTATEN[BWRSEN, BRRSEN]. See [Figure 25-25](#) for illustration of the DMA crossbar switch interface block.



**Figure 25-25. DMA Crossbar Switch Interface Block**

### 25.4.2.1 Internal DMA Request

If the watermark level is met in the data transfer and the internal DMA is enabled, the data buffer block sends a DMA request to this block. Meanwhile, the external DMA request is disabled. The delay of response from the internal DMA engine depends on the system bus loading and the priority assigned to eSDHC. The DMA engine does not respond to the request during its burst transfer, and is available as soon as the burst is over. The data buffer deasserts the request once an access on the buffer is made. Upon access to the buffer by the internal DMA, the data buffer updates its internal buffer pointer and when the watermark level is satisfied, another DMA request is sent.

The data transfer is in the block unit and the last watermark level is always set to the remaining number of words. For instance, for a multi-block data read with each block size of 31 bytes, the burst length is set at six words. After the first burst transfer, if there are more than seven bytes in the buffer, which might be partly some data of the next block, another DMA read request is sent because the remaining number of words to send for the current block is  $(31 - 6 \times 4) \div 4 = 2$ , and eSDHC reads two words out of the buffer, with seven valid bytes and one stuff byte automatically added by eSDHC.

### 25.4.2.2 DMA Burst Length

Just like the CPU polling access, the DMA burst length for the internal DMA engine does not have a restriction other than the maximum size. The burst length for read or write can be 1–128 words. The actual burst length for the DMA depends on which is smaller: configured watermark level or the remaining words of current block.

Take the example in [Section 25.4.2.1, “Internal DMA Request,”](#) again. After six words are read, the burst length is two words to complete the 31-byte block. The burst length then changes back to six words to prepare for the next 31-byte block. The host driver writer may take this variable burst length into account.

It is also acceptable to configure the burst length as the divisor of block size so that each time the burst length is the same.

### 25.4.2.3 ADMA Engine

The *SD Host Controller Standard Specification, Version 2.0* defines a new DMA transfer algorithm called ADMA (advanced DMA). For simple DMA, when a page boundary is reached, a DMA interrupt is generated and the new system address is programmed by the host driver. The advanced DMA defines a programmable descriptor table in the system memory. The host driver can calculate system address at the page boundary and program the descriptor table before executing ADMA. This reduces the frequency of interrupts to the host system. Therefore, a higher speed DMA transfer is realized since host MCU intervention is not needed during a long DMA-based data transfer.

There are two types of ADMA in the host controller: ADMA1 and ADMA2. ADMA1 supports data transfer of 4 KByte-aligned data in system memory. ADMA2 improves the restriction so that data of any location and size can be transferred in system memory. Their descriptor table formats are different.

#### 25.4.2.3.1 ADMA Concept and Descriptor Format

ADMA can recognize many descriptors defined in *SD Host Controller Standard Specification, Version 2.0*, including:

- Valid/Invalid descriptor
- NOP descriptor
- Set data length descriptor
- Set data address descriptor
- Link descriptor
- Interrupt and end flags in descriptor
- (ADMA2 only) Rsv descriptor

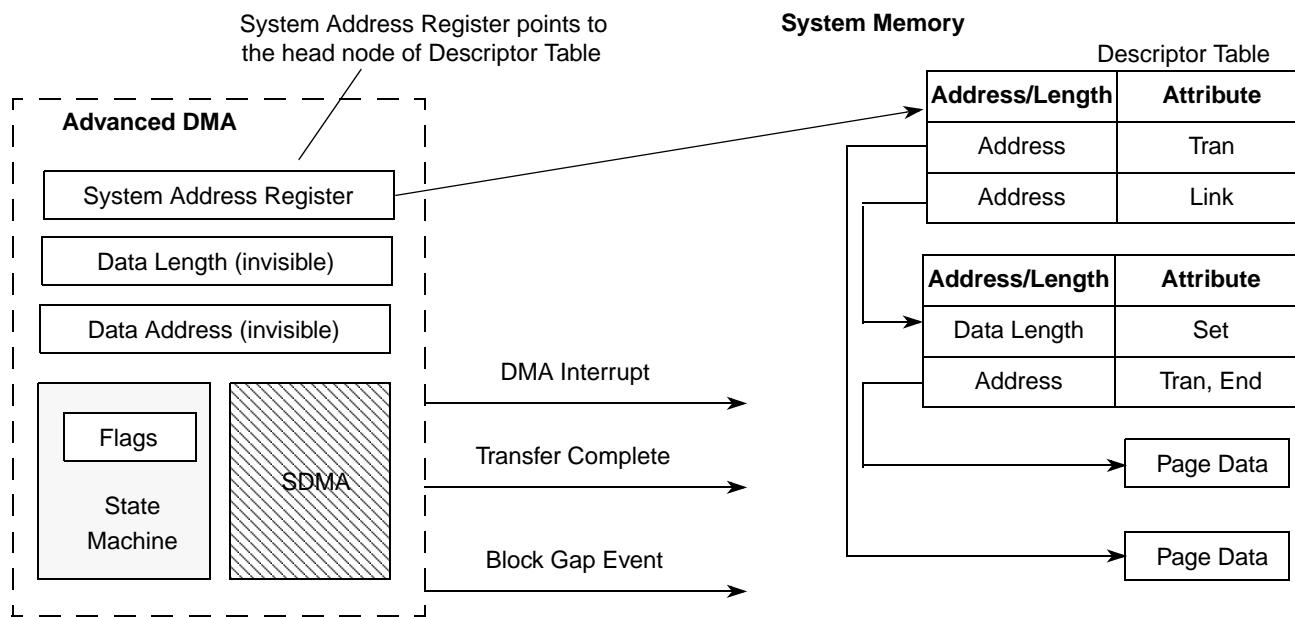
The ADMA1 format descriptor table is shown in [Figure 25-26](#), and its concept is shown in [Figure 25-27](#).

Address/Page Field																Attribute Field							
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							
Address or Data Length																0 0 0 0   0 0 ACT2 ACT1   0 INT END VALID							

**Figure 25-26. ADMA1 Descriptor Table Format**

**Table 25-29. ADMA1 Descriptor Table Field Descriptions**

Field	Description						
Address or data field, ACT2, ACT1	ACT[2:1]	Symbol	Command	31–28	27–12		
	00	NOP	No operation	Don't Care			
	01	Set	Set data length	0000	Data Length		
	10	Tran	Transfer data	Data Address			
	11	Link	Link descriptor	Descriptor Address			
Valid	0	Generate ADMA error interrupt and stop ADMA					
	1	This line of descriptor is effective					
End	0	Not the end of the transfer					
	1	Terminate transfer and generate a transfer complete interrupt when this transfer is completed					
Int	0	Do not generate interrupt					
	1	Generates DMA interrupt when this transfer completes					

**Figure 25-27. Concept and Access Method of ADMA1 Descriptor Table**

The ADMA2 format descriptor table is shown in [Figure 25-28](#), and its concept is shown in [Figure 25-29](#).

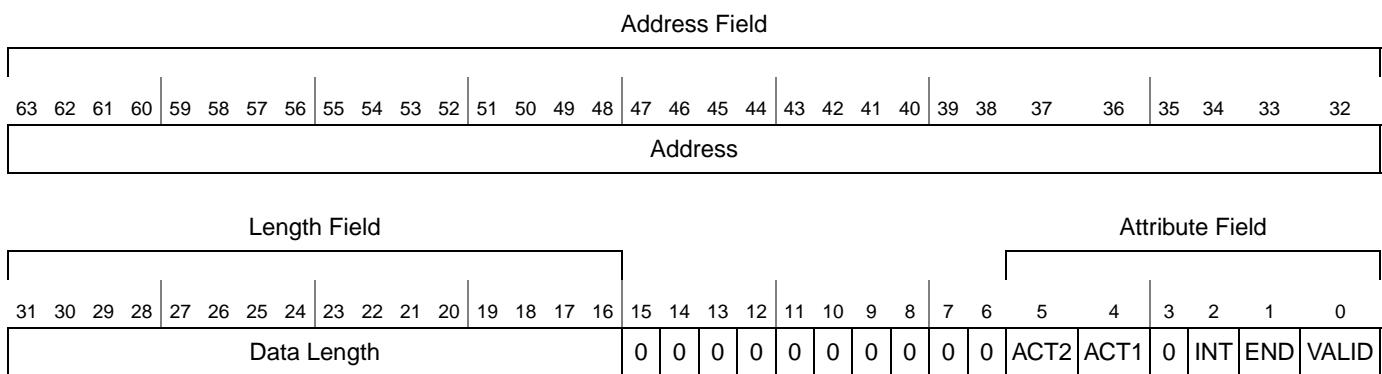
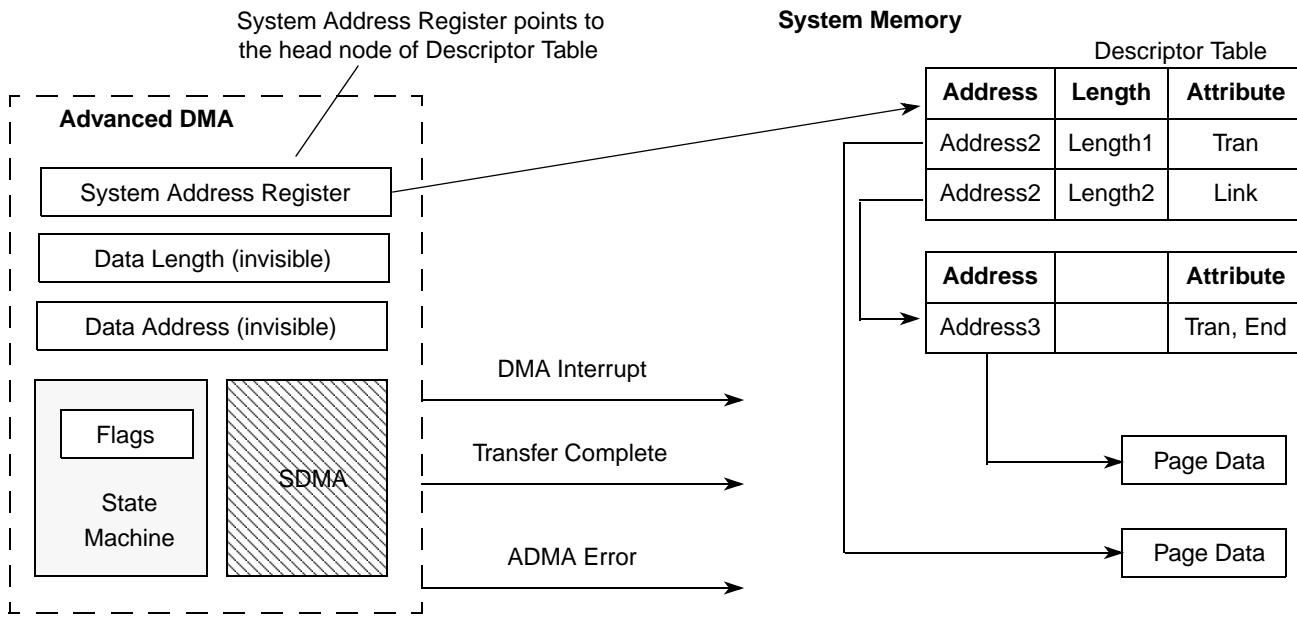


Figure 25-28. ADMA2 Descriptor Table Format

Table 25-30. ADMA2 Descriptor Table Field Descriptions

Field	Description			
Address or data field, ACT2, ACT1	ACT[2:1]	Symbol	Command	Operation
	00	NOP	No operation	Don't Care
	01	Rsv	Reserved	Same as NOP. Read this line and go to next one
	10	Tran	Transfer data	Transfer data with address and length set in this descriptor line
	11	Link	Link descriptor	Link to another descriptor
Valid	0	Generate ADMA error interrupt and stop ADMA		
	1	This line of descriptor is effective		
End	0	Not the end of the transfer		
	1	Terminate transfer and generate a transfer complete interrupt when this transfer is completed		
Int	0	Do not generate interrupt		
	1	Generates DMA interrupt when this transfer completes		



**Figure 25-29. Concept and Access Method of ADMA2 Descriptor Table**

ADMA2 deals with the lower 32-bits first, then the higher 32-bits. If the Valid flag of the descriptor is cleared, it ignores the high 32-bits. The address field must be word-aligned (lower 2-bit is always cleared). The data length is in bytes.

ADMA starts a read/write operation after it reaches the Tran state, using the data length and data address analyzed from the most recent descriptors.

- ADMA1 — The valid data length is specified in the last Set descriptor before a Tran descriptor. Every Tran type triggers a transfer, and the transfer data length is extracted from the most recent Set descriptor. If there is no Set descriptor after the previous Tran descriptor, the data length is the value from the previous transfer, or zero if there has been no Set descriptor.
- ADMA2 — A Tran descriptor contains the data length and transfer data address. So, only a Tran type descriptor starts a data transfer

#### 25.4.2.3.2 ADMA Interrupts

ADMA1 generates four types of interrupts:

- Set type descriptor — interrupt is generated when data length is set
- Tran type descriptor — interrupt is generated when this transfer is complete
- Link type descriptor — interrupt is generated when new descriptor address is set
- Nop type descriptor — interrupt is generated just after this descriptor is fetched

ADMA2 generates three types of interrupts:

- Tran type descriptor — interrupt is generated when this transfer is complete
- Link type descriptor — interrupt is generated when new descriptor address is set

- Nop/Rsv type descriptor — interrupt is generated just after this descriptor is fetched

### 25.4.2.3.3 ADMA Errors

The ADMA stops when any error is encountered. See [Section 25.5.3.4.3, “ADMA Error”](#), for details on these errors.

### 25.4.2.4 Crossbar Switch Master Interface

It is possible that the internal DMA engine fails during the data transfer. When an error occurs, the DMA engine stops the transfer and goes to the idle state, while the internal data buffer stops working, too. IRQSTAT[DMAE] is set to inform the driver.

Once the IRQSTAT[DMAE] interrupt is received, software should send CMD12 to abort the current transfer and read DSADDR[DS\_ADDR] to obtain the start address of the corrupted block. After the DMA error is fixed, the software should apply a data reset and restart the transfer from this address to recover the corrupted block.

## 25.4.3 SD Protocol Unit

The SD protocol unit deals with all SD protocol affairs and performs the following:

- Acts as the bridge between internal buffer and the SD bus
- Sends the command data and its argument serially
- Stores the serial response bit stream into corresponding registers
- Detects bus state on SDHC\_DAT[0] line
- Monitors interrupt from the SDIO card
- Asserts read wait signal
- Gates off SD clock when the buffer announces danger status
- Detects write-protect state
- And other functions

It consists of four submodules: SD transceiver, SD clock and monitor, command agent and data agent.

### 25.4.3.1 SD Transceiver

In the SD protocol unit, the transceiver is the main control module. It consists of a FSM and the control module, from which the control signals for all other three modules are generated.

### 25.4.3.2 SD Clock & Monitor

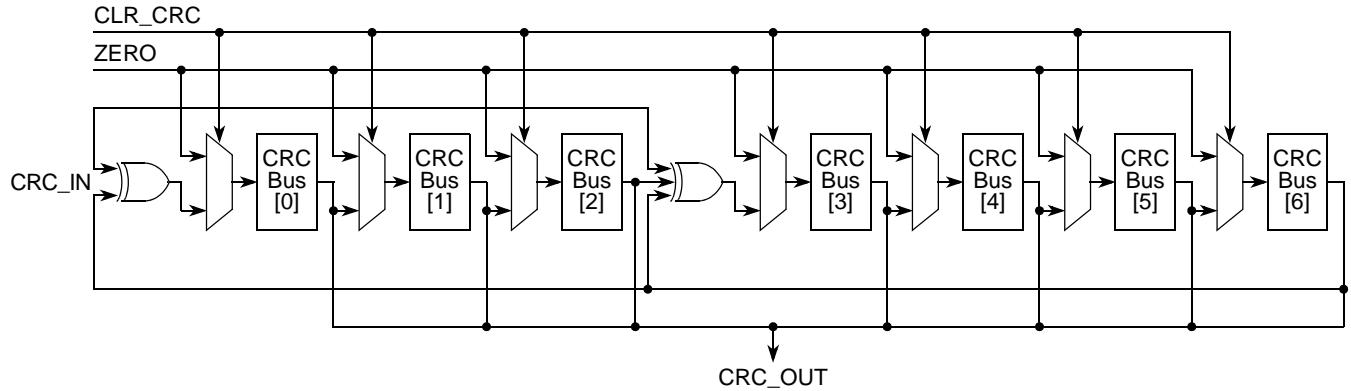
This module monitors the signal level on all four data lines and the command lines, directly route the level values into the register bank for the driver to debug with.

The transceiver reports the card insertion state according to the signal level on SDHC\_DAT[3] line when PROCTL[D3CD] is set.

If the internal data buffer is in danger and the SD clock must be gated off to avoid buffer over/underrun, this module asserts the gate of output SD clock to shut the clock off. When the buffer danger is eliminated when system access of the buffer catches up, the clock gate of this module is open and the SD clock is active again.

### 25.4.3.3 Command Agent

The command agent deals with the transactions on SDHC\_CMD line. See [Figure 25-30](#) for illustration of the structure for the command CRC shift register.



**Figure 25-30. Command CRC Shift Register**

The CRC polynomials for the SDHC\_CMD are as follows:

$$\text{Generator polynomial: } G(x) = x^7 + x^3 + 1$$

$$M(x) = (\text{first bit}) \times x^n + (\text{second bit}) \times x^{n-1} + \dots + (\text{last bit}) \times x^0$$

$$\text{CRC}[6:0] = \text{Remainder } [(M(x) \times x^7) \div G(x)]$$

### 25.4.3.4 Data Agent

The data agent handles the transactions on the eight data lines. Moreover, this module also detects the busy state from on SDHC\_DAT[0] line, and generates read wait state by the request from the transceiver. The CRC polynomials for the SDHC\_DAT are as follows:

$$\text{Generator polynomial: } G(x) = x^{16} + x^{12} + x^5 + 1$$

$$M(x) = (\text{first bit}) \times x^n + (\text{second bit}) \times x^{n-1} + \dots + (\text{last bit}) \times x^0$$

$$\text{CRC}[15:0] = \text{Remainder } [(M(x) \times x^{16}) \div G(x)]$$

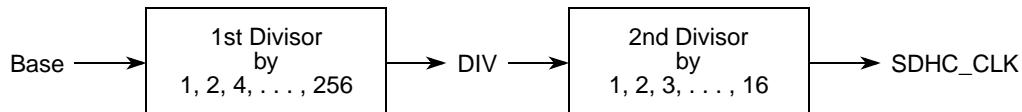
### 25.4.4 Clock & Reset Manager

This module controls all the reset signals within the eSDHC. There are four types of reset signals within eSDHC: hardware reset, software reset for all, software reset for data, and software reset for command. All these signals are fed into this module and stable signals are generated to reset all other modules.

This module also gates off all the inside signals. The module monitors the activities of all other modules, supplies the clocks for them, and when enabled, automatically gates off the corresponding clocks.

## 25.4.5 Clock Generator

The clock generator generates the SDHC\_CLK by dividing the internal bus clock into two stages. Refer to [Figure 25-31](#) for the structure of the divider, in which the term base represents the frequency of the internal bus clock.



**Figure 25-31. Two Stages of the Clock Divider**

The first stage is a prescaler. The frequency of clock output from this stage, DIV, can be base, base/2, base/4, ..., or base/256.

The second stage outputs the actual clock, SDHC\_CLK, as the driving clock for all sub-modules of SD protocol unit, and the sync FIFOs in [Figure 25-23](#) to synchronize with the data rate from the internal data buffer. It can be div, div/2, div/3,..., or div/16. Thus, the highest frequency of SDHC\_CLK generated by the internal bus clock is base, while the lowest frequency is base/4096.

## 25.4.6 SDIO Card Interrupt

### 25.4.6.1 Interrupts in 1-bit Mode

In this case the SDHC\_DAT[1] pin is dedicated to providing the interrupt function. An interrupt is asserted by pulling the SDHC\_DAT[1] low from the SDIO card, until the interrupt service is finished to clear the interrupt.

### 25.4.6.2 Interrupt in 4-bit Mode

Since the interrupt and data line 1 share pin 8 in four-bit mode, an interrupt is only sent by the card and recognized by the host during a specific time. This is known as the interrupt period. The eSDHC only samples the level on pin 8 during the interrupt period. At all other times, the host ignores the level on pin 8 and treats it as the data signal. The definition of the interrupt period is different for operations with single- and multiple-block data transfers.

For normal single data block transmissions, the interrupt period becomes active two clock cycles after the completion of a data packet. This interrupt period lasts until after the card receives the end bit of the next command that has a data block transfer associated with it.

For multiple block data transfers in 4-bit mode there is only a limited period of time that the interrupt period can be active due to the limited period of data line availability between the multiple blocks of data. This requires a more strict definition of the interrupt period. For this case, the interrupt period is limited to two clock cycles, which begins two clocks after the end bit of the previous data block. During this two-clock cycle interrupt period if an interrupt is pending, the SDHC\_DAT[1] line is held low for one clock cycle with the last clock cycle pulling SDHC\_DAT[1] high. On completion of the interrupt period, the card releases the SDHC\_DAT[1] line into the high-Z state. The eSDHC samples the SDHC\_DAT[1] during the interrupt period when PROCTL[IABG] is set.

Refer to *SDIO Card Specification v1.10f* for further information about the SDIO card interrupt.

### 25.4.6.3 Card Interrupt Handling

When IRQSIGEN[CINTIEN] is cleared, the eSDHC clears the interrupt request to the host system. The host driver should clear this bit before servicing the SDIO interrupt and should set this bit again after all interrupt requests from the card are cleared to prevent inadvertent interrupts.

If enabled by IRQSTATEN[CINTSEN], the IRQSTAT[CINT] bit can only be cleared by resetting the SDIO interrupt source and then writing one to this bit. Merely writing to this bit has no effect.

In 1-bit mode, the eSDHC detects the SDIO interrupt with or without SD clock (to support wakeup). In 4-bit mode, the interrupt signal is sampled during the interrupt period, so there are some sample delays between the interrupt signal from the SDIO card and the interrupt to the host system interrupt controller. When IRQSTAT[CINT] is set and the host driver needs to start this interrupt service, IRQSTATEN[CINTSEN] is cleared in order to clear IRQSTAT[CINT] that is latched in the eSDHC and to stop driving the interrupt signal to the processor's interrupt controller. The host driver must issue a CMD52 to clear the interrupts at the card. After completion of the card interrupt service, IRQSTATEN[CINTSEN] is set, and the eSDHC can start sampling the interrupt signal again.

See the following illustrations:

- [Figure 25-32](#) (a) for an illustration of the SDIO card interrupt scheme
- [Figure 25-32](#) (b) for the sequences of software and hardware events that take place during card interrupt handling procedure

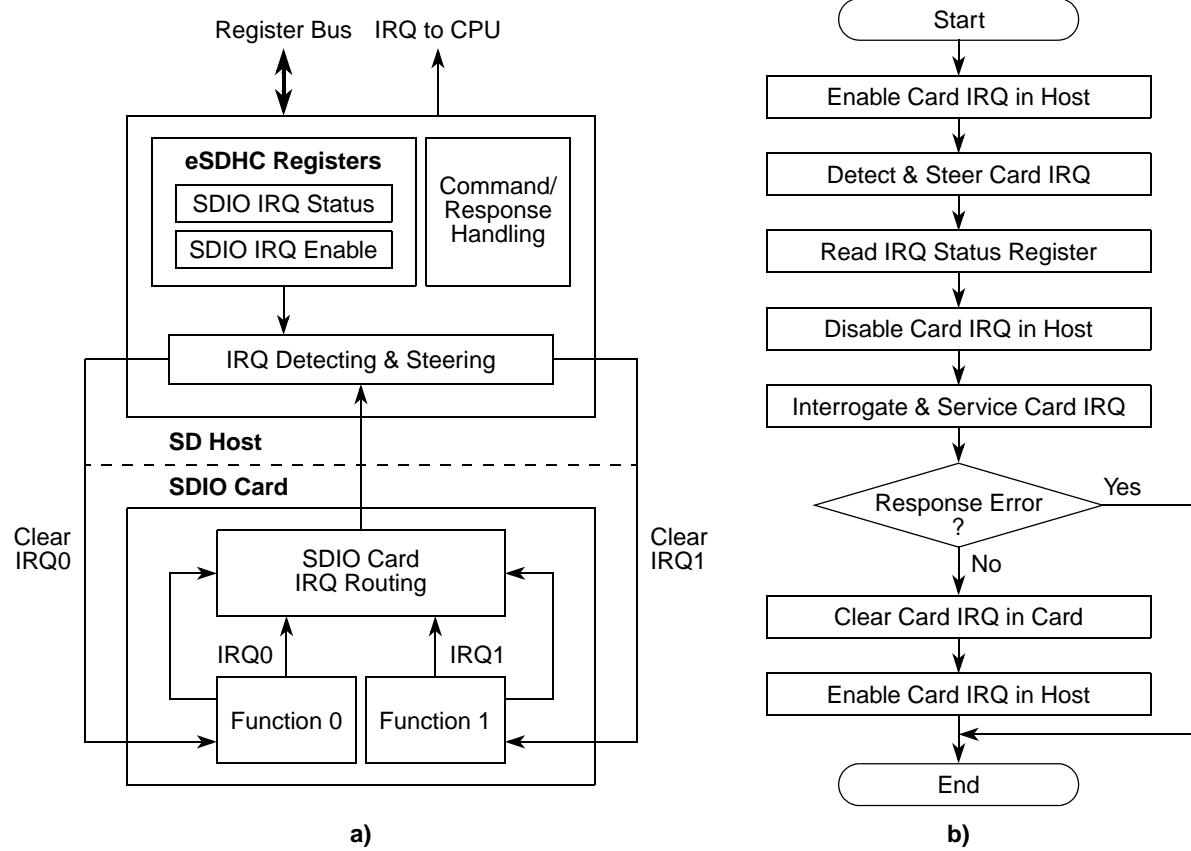


Figure 25-32. a) Card Interrupt Scheme; b) Card Interrupt Detection and Handling Procedure

### 25.4.7 Card Insertion and Removal Detection

The eSDHC uses the SDHC\_DAT[3] pin to detect card insertion or removal. When SDHC\_DAT[3] pin is used for card detection, the chip level integration needs to pull-down this pad as a default state. When there is no card on the MMC/SD bus, the SDHC\_DAT[3] is pulled to a low voltage level by default. When any card is inserted to or removed from the socket, the eSDHC detects the logic value changes on the SDHC\_DAT[3] pin and generates an interrupt.

### 25.4.8 Power Management and Wake-Up Events

When there is no operation between eSDHC and the card through SD bus, you can completely disable the internal clocks in the chip level clock control module to save power. When you need to use the eSDHC to communicate with the card, it can enable the clock and start the operation. This can be done by clearing the appropriate bit in the PPMR register as described in Chapter 9, “Power Management”.

In some circumstances, when the clocks to eSDHC are disabled, or when system is in low power mode, there are some events when you need to enable the clock and handle the event. These events are called wakeup interrupts. The eSDHC can generate these interrupts even there are no clocks enabled. The three interrupts which can be used as wake-up events are:

- Card removal interrupt

- Card insertion interrupt
- SDIO card interrupt

These three wake-up events (or wake-up interrupts) can also wake up the system from low-power modes.

#### 25.4.8.1 Setting Wake Up Events

For the eSDHC to respond to a wake up event, the software must set the respective wake up enable bit before the CPU enters sleep mode. Refer to [Section 25.3.8, “Protocol Control Register \(PROCTL\),”](#) for more information on the wakeup enable bits.

Before the software disables the host clock, it should ensure that all of the following conditions have been met:

- No read or write transfer is active
- Data and command lines are not active
- No interrupts are pending
- Internal data buffer is empty

### 25.5 Initialization/Application Information

All communication between system and cards are controlled by the host. The host sends commands of two types: broadcast and addressed (point-to-point) commands.

Broadcast commands are intended for all cards, such as GO\_IDLE\_STATE, SEND\_OP\_COND, ALL\_SEND\_CID, etc. In broadcast mode, all cards are in the open-drain mode to avoid bus contention. Refer to [Section 25.5.5, “Commands for MMC/SD/SDIO,”](#) for the commands of bc and bcr categories.

After the broadcast command CMD3 is issued, the cards enter standby mode. Addressed type commands are used from this point. In this mode, the SDHC\_CMD/SDHC\_DAT I/O pads turn to push-pull mode, to have the driving capability for maximum frequency operation. Refer to [Section 25.5.5, “Commands for MMC/SD/SDIO,”](#) for the commands of ac and adtc categories.

#### 25.5.1 Command Send and Response Receive Basic Operation

Assuming data type WORD is an unsigned 32-bit integer, the below flow is a guideline for sending a command to the card(s):

```
send_command(cmd_index, cmd_arg, other requirements)
{
    WORD wCmd; // 32-bit integer to make up the data to write into the XFERTYP register, it is
               // recommended to implement in a bit-field manner
    wCmd = (<cmd_index> & 0x3f) >> 24; // set the first 8 bits as '00'+<cmd_index>
    set CMDTYP, DPSEL, CICEN, CCCEN, RSTTYP, and DTDSEL according to the command index;
               // XFERTYP register bits
    if (internal DMA is used) wCmd |= 0x1;
    if (multi-block transfer) {
        set XFERTYP[MSBSEL] bit;
        if (finite block number) {
            set XFERTYP[BCEN] bit;
            if (auto12 command is to use) set XFERTYP[AC12EN] bit;
        }
    }
}
```

```

    }
}

write_reg(CMDARG, <cmd_arg>); // configure the command argument
write_reg(XFERTYP, wCmd); // set XFERTYP register as wCmd value to issue the command
}
wait_for_response(cmd_index)
{
while (IRQSTAT[CC] is not set); // wait until command complete bit is set
read IRQSTAT register and check if any error bits about command are set;
if (any error bits are set) report error;
write 1 to clear IRQSTAT[CC] and all command error bits;
}

```

For the sake of simplicity, the function `wait_for_response` is implemented here by means of polling. For an effective and formal way, the response is usually checked after the command complete interrupt is received. By doing this, ensure the corresponding interrupt status bits are enabled.

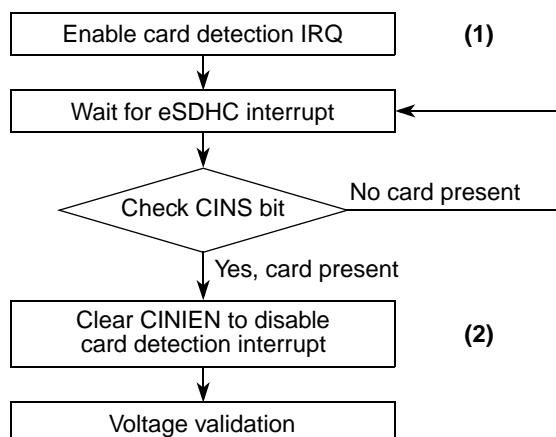
For some scenarios, the response timeout is expected. For instance, after all cards respond to CMD3 and go to the standby state, no response to the host when CMD2 is sent. The host driver should manage false errors similar to this with caution.

## 25.5.2 Card Identification Mode

When a card is inserted to the socket or the card was reset by the host, the host needs to validate the operation voltage range, identify the cards, and request the cards to publish the relative card address (RCA) or to set the RCA for the MMCs.

### 25.5.2.1 Card Detect

See [Figure 25-33](#) for a flow diagram showing the detection of MMC, SD, and SDIO cards using the eSDHC.



**Figure 25-33. Flow Diagram for Card Detection**

- Set IRQSIGEN[CINIEN] to enable card detection interrupt.
- When an interrupt from eSDHC is received, check IRQSTAT[CINS] to see if it is caused by card insertion.

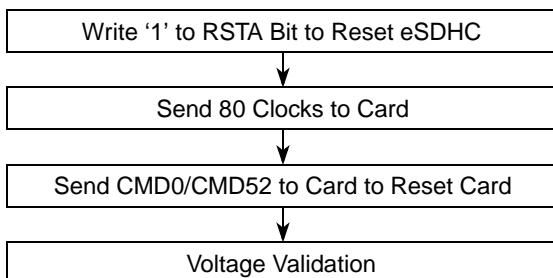
- Clear the IRQSIGEN[CINIEN] to disable card detection interrupt and ignore all card insertion interrupt afterwards.

### 25.5.2.2 Reset

The host consists of three types of reset:

- Hardware reset (card and host) which is driven by POR (power on reset).
- Software reset (host only) is proceeded by the write operation on the SYSCTL[RSTD], SYSCTL[RSTC], or SYSCTL[RSTA] bits to reset the data part, command part, or all parts of the host controller, respectively.
- Card reset (card only). The command CMD0, GO\_IDLE\_STATE, is the software reset command for all types of MMCs and SD memory cards. This command sets each card into idle state regardless of the current card state. For an SDIO card, CMD52 is used to write I/O reset in CCCR. The cards are initialized with a default relative card address (RCA = 0x0000) and with a default driver stage register setting (lowest speed, highest driving current capability).

After the card is reset, the host needs to validate the voltage range of the card. See [Figure 25-34](#) for the software flow to reset the eSDHC and card.



**Figure 25-34. Flow Chart for Reset of eSDHC and SD I/O Card**

```

software_reset()
{
    set_bit(SYSCTL, RSTA);           // software reset the host
    set_SYSCTL[DTOCV and SDCLKFS];   // get the SDHC_CLK of frequency around 400 KHz
    configure I/O pad;              // set the voltage of external card to around 3.0 V
    poll PRSSTAT[CIHB and CDIHB];   // wait until both bits are cleared
    set_bit(SYSCTRL, INTIA);        // send 80 clock ticks for card to power-up
    send_command(CMD_GO_IDLE_STATE, <other parameters>); // reset the card with CMD0
    or send_command(CMD_IO_RW_DIRECT, <other parameters>);
}
  
```

### 25.5.2.3 Voltage Validation

All cards should be able to establish communication with the host using any operation voltage in the maximum allowed voltage range specified in this standard. However, the supported minimum and maximum values for  $V_{DD}$  are defined in the operation conditions register (OCR) and may not cover the whole range. Cards that store the CID (card identification) and CSD data in the preloaded memory are only able to communicate these information under data transfer  $V_{DD}$  conditions. This means that if the host and

card have different V<sub>DD</sub> ranges, the card is not able to complete the identification cycle, nor is it able to send CSD data.

Therefore, a special command is available:

- SEND\_OP\_CONT (CMD1 for MMC),
- SD\_SEND\_OP\_CONT (ACMD41 for SD Memory), and
- IO\_SEND\_OP\_CONT (CMD5 for SD I/O).

The voltage validation procedure is designed to provide a mechanism to identify and reject cards which do not match the V<sub>DD</sub> range(s) desired by the host. This is accomplished by the host sending the desired V<sub>DD</sub> voltage window as the operand of this command. Cards that can not perform data transfer in the specified range must discontinue any further bus operations and enter the inactive state. By omitting the voltage range in the command, the host can query each card and determine the common voltage range before sending out-of-range cards into the inactive state. This query should be used if the host is able to select a common voltage range or if a notification should be sent to the system when a non-usuable cards in the stack is detected.

The following steps illustrate how to perform voltage validation when a card is inserted:

```
voltage_validation(voltage_range_arguement)
{
label the card as UNKNOWN;
send_command(IO_SEND_OP_COND, 0x0, <other parameters are omitted>);
    // CMD5, check SDIO operation voltage, command argument is zero
if (RESP_TIMEOUT != wait_for_response(IO_SEND_OP_COND)) { // SDIO command is accepted
    if (0 < number of IO functions) {
        label the card as SDIO;
        IORDY = 0;
        while (!(IORDY in IO OCR response)) { // set voltage range for each IO function
            send_command(IO_SEND_OP_COND, <voltage range>, <other parameter>);
            wait_for_response(IO_SEND_OP_COND);
        } // end of while ...
    } // end of if (0 < ...)
    if (memory part is present inside SDIO card) label the card as SDCombo;
        // this is an SD-Combo card
} // end of if (RESP_TIMEOUT...)
if (the card is labeled as SDIO card) return;
    // card type is identified and voltage range is set, so exit the function;
send_command(APP_CMD, 0x0, <other parameters are omitted>);
    // CMD55, application specific CMD prefix
if (no error calling wait_for_response(APP_CMD, <...>)) { // CMD55 is accepted
    send_command(SD_APP_OP_COND, <voltage range>, <...>);
        // ACMD41, to set voltage range for memory part or SD card
    wait_for_response(SD_APP_OP_COND); // voltage range is set
    if (card type is UNKNOWN) label the card as SD;
    return;
} // end of if (no error ...
else if (errors other than timeout occur) { // command/response pair is corrupted
    respond to it by program specific manner;
} // of else if (response timeout)
else { // CMD55 is refused, it must be MMC or CE-ATA card
    if (card is already labeled as SD Combo) { // change label
        re-label the card as SDIO;
        ignore the error or report it;
    }
}
```

```

        return; // card is identified as SDIO card
    } // of if (card is ...)
    send_command(SEND_OP_COND, <voltage range>, <...>);
    if (RESP_TIMEOUT == wait_for_response(SEND_OP_COND)) {
        // CMD1 is not accepted, either
        label the card as UNKNOWN;
        return;
    } // of if (RESP_TIMEOUT...)
    if (check for CE-ATA signature succeeded) { // the card is CE-ATA
        store CE-ATA specific info from the signature;
        label the card as CE-ATA;
    } // of if (check for CE-ATA...)
    else label the card as MMC;
} // of else
}

```

#### 25.5.2.4 Card Registry

Card registry on MMC and SD/SDIO/SD Combo cards are different.

For the SD card, the identification process starts at a clock rate lower than 400 KHz and the power voltage higher than 2.7 V, as defined by the card specification. At this time, the SDHC\_CMD line output drivers are push-pull drivers instead of open-drain. After the bus is activated, the host requests the card to send their valid operation conditions. The response to ACMD41 is the operation condition register of the card. The same command should be sent to all of the new cards in the system. Incompatible cards are placed into the inactive state. The host then issues the command, ALL\_SEND\_CID (CMD2), to each card to get its CID. Cards that are currently unidentified (that is, in ready state), send their CID number as the response. After the CID is sent by the card, the card goes into the identification state.

The host then issues Send\_Relative\_Addr (CMD3), requesting the card to publish a new relative card address (RCA) that is shorter than CID. This RCA is used to address the card for future data transfer operations. Once the RCA is received, the card changes its state to the standby state. At this point, if the host wants the card to have an alternative RCA number, it may ask the card to publish a new number by sending another Send\_Relative\_Addr command to the card. The last published RCA is the actual RCA of the card.

The host repeats the identification process with CMD2 and CMD3 for each card in the system until the last CMD2 gets no response from any of the cards in system.

For MMC operation, the host starts the card identification process in open-drain mode with the identification clock rate lower than 400 KHz, the power voltage higher than 2.7 V. The open-drain driver stages on the SDHC\_CMD line allow parallel card operation during card identification. After the bus is activated the host requests the cards to send their valid operation conditions (CMD1). The response to CMD1 is the wired-OR operation on the condition restrictions of all cards in the system. Incompatible cards are sent into inactive state. The host then issues the broadcast command All\_Send\_CID (CMD2), asking all cards for their unique CID number. All unidentified cards (the cards in ready state) simultaneously start sending their CID numbers serially, while bit-wise monitoring their outgoing bit stream. Those cards, whose outgoing CID bits do not match the corresponding bits on the command line in any one of the bit periods, stop sending their CID immediately and must wait for the next identification cycle. Since the CID is unique for each card, only one card can successfully send its full CID to the host. This card then goes into identification state. Thereafter, the host issues Set\_Relative\_Addr (CMD3) to

assign to this card a relative card address (RCA). Once the RCA is received, the card state changes to the stand-by state, and the card does not react in further identification cycles, and its output driver switches from open-drain to push-pull. The host repeats the process, namely CMD2 and CMD3, until the host receives a time-out condition to recognize completion of the identification process.

```

card_registry()
{
do { // decide RCA for each card until response timeout
    if(card is labeled as SD Combo or SDIO) { // for SDIO card like device
        send_command(SET_RELATIVE_ADDR, 0x00, <...>);
        // ask SDIO card to publish its RCA
        retrieve RCA from response;
    } // end if (card is labeled as SD Combo...)
    else if (card is labeled as SD) { // for SD card
        send_command(ALL_SEND_CID, <...>);
        if (RESP_TIMEOUT == wait_for_response(ALL_SEND_CID)) break;
        send_command(SET_RELATIVE_ADDR, <...>);
        retrieve RCA from response;
    } // else if (card is labeled as SD ...)
    else if (card is labeled as MMC or CE-ATA) { // treat CE-ATA as MMC
        send_command(ALL_SEND_CID, <...>);
        rca = 0x1; // arbitrarily set RCA, 1 here for example, this RCA is also the
        // relative address to access the CE-ATA card
        send_command(SET_RELATIVE_ADDR, 0x1 << 16, <...>);
        // send RCA at upper 16 bits
    } // end of else if (card is labeled as MMC...)
} while (response is not timeout);
}

```

## 25.5.3 Card Access

These sections describe the supported access modes with external cards.

### 25.5.3.1 Block Write

This section describes the process of writing data to external cards in block mode.

#### 25.5.3.1.1 Normal Write

During block write (CMD24–27), one or more blocks of data are transferred from the host to the card with a CRC appended to the end of each block by the host. If the CRC fails, the card should indicate the failure on the SDHC\_DAT line (see below). The transferred data is discarded and not written, and all further transmitted blocks (in multi-block write mode) are ignored.

If the host uses partial blocks whose accumulated length is not block-aligned and block misalignment is not allowed (CSD parameter WRITE\_BLK\_MISALIGN is not set), the card detects the block misalignment error and aborts programming before the beginning of the first misaligned block. The card sets the ADDRESS\_ERROR error bit in the status register, defined in the MMC/SD Specification, and then waits in the receive-data state for a stop command while ignoring all further data transfers. The write operation is also aborted if the host attempts to write over a write-protected area.

For MMC and SD cards, programming the CID and CSD registers does not require a previous block length setting. The transferred data is also CRC protected. If a part of the CSD or CID register is stored in the

ROM, this unchangeable section must match the corresponding section of the receive buffer. If this match fails, then the card reports an error and does not change any register contents.

Some cards may require a long and unpredictable period of time to write a block of data. After receiving a block of data and completing the CRC check, the card begins writing. If its write buffer is full and unable to accept new data from a new WRITE\_BLOCK command, the card holds the SDHC\_DAT line low. The host may poll the status of the card with a SEND\_STATUS command (CMD13) or other means for SDIO cards, at any time and the card responds with its status. The card status indicates whether the card can accept new data or if the write process is still in progress. The host may deselect the card by issuing CMD7 (to select a different card) to change the card into the standby state and release the SDHC\_DAT line without interrupting the write operation. When re-selecting the card, it reactivates the busy indication by pulling SDHC\_DAT low if programming is still in progress and the write buffer is unavailable.

For simplicity, the software flow described below incorporates the internal DMA, and the write operation is a multi-block write with Auto CMD12 enabled. For the other two methods (external DMA or CPU polling status) and different transfer nature, the internal DMA part of the procedure should be removed and alternative steps inserted.

1. Check the card status and wait until the card is ready for data.
2. Set the card block length.
  - MMC/SD cards — use SET\_BLOCKLEN (CMD16)
  - SDIO cards or the I/O portion of SD Combo cards — IO\_RW\_DIRECT (CMD52) to set I/O block size bit field in the CCCR register (for function 0) or FBR (for functions 1–7)
3. Set eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in step 2.
4. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.
5. Disable the buffer write ready interrupt, configure the DMA setting, and enable the eSDHC DMA when sending the command with data transfer. Set XFERTYP[AC12EN].
6. Wait for the transfer complete interrupt.
7. Check the status bit to see if a read CRC error or any other errors occurred between sending Auto CMD12 and receiving the response.

### 25.5.3.1.2 Write with Pause

The write operation can be paused during the transfer. Instead of stopping the SDHC\_CLK at any time to pause all the operations which is also inaccessible to the host driver, the driver can set PROCTL[SABGREQ] to pause the transfer between the data blocks. Since there is no timeout condition in a write operation during the data blocks, a write operation to the cards can be paused in this way and if line SDHC\_DAT0 is not required to de-assert to release busy state, no suspend command is needed.

Similar to the flow described in [Section 25.5.3.1.1, “Normal Write,”](#) the write with pause is shown with the same type of write operations:

1. Check the card status and wait until card is ready for data.
2. Set the card block length.
  - MMC/SD cards — use SET\_BLOCKLEN (CMD16)

- SDIO cards or the I/O portion of SD Combo cards — use IO\_RW\_DIRECT (CMD52) to set the I/O block size bit field in CCCR register (for function 0) or FBR (for functions 1–7)
3. Set the eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in step 2.
  4. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.
  5. Disable the buffer write ready interrupt, configure the DMA setting, and enable the eSDHC DMA when sending the command with data transfer. Set XFERTYP[AC12EN].
  6. Set PROCTL[SABGREQ].
  7. Wait for the transfer complete interrupt.
  8. Clear PROCTL[SABGREQ].
  9. Check the status bit to see if a read CRC error occurred.
  10. Set PROCTL[CREQ] to continue the read operation.
  11. Wait for the transfer complete interrupt.
  12. Check the status bit to see if a read CRC error or any other errors occurred between sending Auto CMD12 and receiving the response.

The number of blocks left during the data transfer is accessible by reading the content of BLKATTR[BLKCNT]. Due to the data transfers and setting PROCTL[SABGREQ] are concurrent, along with the delay of register read and the register setting, the actual number of blocks left may not be the same as the value read earlier. The driver should read the value of BLKATTR[BLKCNT] after the transfer is paused and the transfer complete interrupt is received.

It is also possible that the transfer of the last block begins when the stop-at-block-gap request is sent to the buffer. In this case, the next block gap is the actual end of the transfer, and therefore, the request is ignored. The driver should treat this as a non-pause transfer and a common write operation.

When the write operation is paused, the data transfer inside the host system does not stop and the transfer remains active until the data buffer is full. Therefore, avoid using the suspend command for the SDIO card. When such command is sent, the eSDHC assumes the system switches to another function of the SDIO card and flushes the data buffer. The eSDHC reads the resume command as a normal command with a data transfer, and it is the driver's responsibility to set all the relevant registers before the transfer is resumed. If there is only one block to send when the transfer is resumed, XFERTYP[MSBSEL, BCEN, AC12EN] are set. However, the eSDHC automatically sends CMD12 to mark the end of multi-block transfer.

### 25.5.3.2 Block Read

#### 25.5.3.2.1 Normal Read

For block reads, the basic unit of a data transfer is a block whose maximum size is stored in areas defined in corresponding card specifications. A CRC is appended to the end of each block, ensuring data transfer integrity. CMD17, CMD18, CMD53, and so on, can initiate a block read. After completing the transfer, the card returns to the transfer state.

For multi-block reads, data blocks are continuously transferred until a stop command is issued. If the host uses partial blocks whose accumulated length is not block aligned and block misalignment is not allowed,

the card which does not support partial block length, should detect the block misalignment at the beginning of the first misaligned block and report the error, depending on its card type.

For simplicity, the software flow described below incorporates the internal DMA, and the read operation is a multi-block read with Auto CMD12 enabled. For the other two methods (external DMA or CPU polling status) and different transfer nature, the internal DMA part should be removed and the alternative steps are straightforward.

1. Check the card status and wait until the card is ready for data.
2. Set the card block length.
  - MMC/SD cards — use SET\_BLOCKLEN (CMD16)
  - SDIO cards or the I/O portion of SD Combo cards — use IO\_RW\_DIRECT (CMD52) to set IO block size bit field in CCCR register (for function 0) or FBR (for functions 1–7)
3. Set eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in step 2.
4. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.
5. Disable the buffer read ready interrupt, configure the DMA setting, and enable the eSDHC DMA when sending the command with data transfer. Set XFERTYP[AC12EN].
6. Wait for the transfer complete interrupt.
7. Check the status bit to see if a read CRC error or any other errors occurred between sending Auto CMD12 and receiving the response.

### 25.5.3.2.2 Read with Pause

In general, the read operation is not able to pause. Only the SDIO card (and SD Combo card working under I/O mode) supporting the read wait feature can pause during the read operation. If the SDIO card supports read wait (CCCR[SRW] = 1), the driver can set PROCTL[SABGREQ] to pause the transfer between the data blocks. Before setting SABGREQ, PROCTL[RWCTL] must be set. Otherwise, the eSDHC does not assert the read wait signal during the block gap and data corruption occurs. It is recommended to set the RWCTL bit once the read wait capability of the SDIO card is recognized.

Similar to the flow described in [Section 25.5.3.2.1, “Normal Read,”](#) the read with pause is shown with the same type of read operations:

1. Check CCCR[SRW] in the SDIO card to confirm the card supports read wait.
2. Set PROCTL[RWCTL].
3. Check the card status and wait until the card is ready for data.
4. Set the card block length.
  - MMC/SD cards — use SET\_BLOCKLEN (CMD16)
  - SDIO cards or the I/O portion of SD Combo cards — use IO\_RW\_DIRECT (CMD52) to set IO block size bit field in CCCR register (for function 0) or FBR (for functions 1–7)
  - .
5. Set eSDHC BLKATTR[BLKSIZE] to the same as the block length set in the card in Step 2.
6. Set eSDHC BLKATTR[BLKCNT] with the number of blocks to send.

7. Disable the buffer read ready interrupt, configure the DMA setting, and enable the eSDHC DMA when sending the command with data transfer. Set XFERTYP[AC12EN].
8. Set PROCTL[SABGREQ].
9. Wait for the transfer complete interrupt.
10. Clear PROCTL[SABGREQ].
11. Check the status bit to see if a read CRC error occurred.
12. Set PROCTL[CREQ] to continue the read operation.
13. Wait for the transfer complete interrupt.
14. Check the status bit to see if a read CRC error or any other errors occurred between sending Auto CMD12 and receiving the response.

Similar to the write operation, it is possible to meet the ending block of the transfer when paused. In this case, the eSDHC ignores the stop-at-block-gap request and treats it as a command read operation.

Unlike the write operation, there is no remaining data inside the buffer when the transfer is paused. All data received before the pause is transferred to the host system. Whether or not a suspend command is sent, the internal data buffer is not flushed.

If the suspend command is sent and the transfer is later resumed by means of the resume command, the eSDHC takes the command as a normal one accompanied with data transfer, and it is left for the driver to set all the relevant registers before the transfer is resumed. If there is only one block to send when the transfer is resumed, XFERTYP[MSBSEL, BCEN] and IRQSTT[AC12EN] are set. However, the eSDHC automatically sends CMD12 to mark the end of a multi-block transfer.

### 25.5.3.3 ADMA Usage

To use the ADMA in a data transfer, the host driver must prepare the correct descriptor chain prior to sending the read/write command. To do this:

1. Create a descriptor to set the data length that the current descriptor group is about to transfer. The data length should be even numbers of the block size.
2. Create another descriptor to transfer the data from the address in this descriptor. The data address must be at a page boundary (4kB address aligned).
3. If necessary, create a link descriptor containing the address of the next descriptor. The descriptor group is created in steps 1–3.
4. Repeat steps 1–3 until all descriptors are created.
5. In the last descriptor, set the end flag and ensure the total length of all descriptors matches the product of the block size and block number configured in the block attribute register.
6. Set the ADMA system address register to the address of the first descriptor and set PROCTL[DMAS] to 01 to select the ADMA.
7. Issue a write or read command with XFERTYP[DMAEN] set.

Since steps 1–5 are independent of step 6, step 6 can finish before steps 1–5. Regarding the descriptor configuration, it is recommended not to use the link descriptor as it requires extra system memory access.

## 25.5.3.4 Transfer Error

### 25.5.3.4.1 CRC Error

At the end of a block transfer, a write CRC status error or read CRC error may occur. For this type of error, the last block received should be discarded because the integrity of the data block is not guaranteed. It is recommended to discard the following data blocks and re-transfer the block from the corrupted one. For a multi-block transfer, the host driver should issue CMD12 to abort the current process and start the transfer by a new data command. In this scenario, even when the XFERTYP[AC12EN, BCEN] are set, the eSDHC does not automatically send CMD12 because the last block is not transferred. On the other hand, if it is within the last block that CRC error occurs, Auto CMD12 is sent by the eSDHC. In this case, the driver should resend or re-obtain the last block with a single block transfer.

### 25.5.3.4.2 Internal DMA Error

During the data transfer with the internal DMA, if the DMA engine encounters an error on the CSB bus, the DMA operation is aborted and a DMA error interrupt is sent to the host system. When acknowledged by such an interrupt, the driver should calculate the start address of the data block where the error occurred. The start address can be calculated by either of the following methods:

- Read the DSADDR[DSADDR] field. The error occurs during the previous burst. Therefore, by taking the block size, the previous burst length, and the start address of the next burst transfer into account, one can obtain the start address of the corrupted block.
- Read the BLKATTR[BLKCNT] field. The start address of the corrupted block can be calculated by the number of blocks left, the total number to transfer, the start address of transfer, and the size of each block. However, if BCEN is not set, the contents of the block attribute register does not change and this method does not work.

When a DMA error occurs, it is recommended to abort the current transfer by means of CMD12 (for multi-block transfer), apply a reset for data, and restart the transfer from the corrupted block to recover the error.

### 25.5.3.4.3 ADMA Error

There are three possible ADMA errors: AHB transfer, invalid descriptor, and data-length mismatch errors. When these errors occur, the DMA transfer stops and the corresponding error status bit is set. To acknowledge the status, the host driver should recover the error as shown below and re-transfer from the place of interruption.

- AHB transfer error — Such errors may occur during a data transfer or descriptor fetch. For either scenario, it is recommended to retrieve the transfer context, reset the data part and either re-transfer the block that was corrupted or transfer the next block if no block is corrupted.
- Invalid descriptor error — It is recommended to retrieve the transfer context, reset the data part and re-create the descriptor chain from the invalid descriptor and issue a new transfer. Since the data to transfer now may be less than the previous setting, configure the data length in the new descriptor chain to match the new value.
- Data-length mismatch error — Similar to the invalid descriptor error, the host driver polls related registers to retrieve the transfer context, resets the data part, configures a new descriptor chain, and

makes another transfer if there is data left. Like the previous scenario of the invalid descriptor error, the data length must match the new transfer.

#### 25.5.3.4.4 Auto CMD12 Error

After the last block of a multi-block transfer is sent or received and XFERTYP[AC12EN] is set when the data transfer is initiated by the data command, the eSDHC automatically sends CMD12 to the card to stop the transfer. When an error occurs at this point, it is recommended that the host driver responds by:

1. Auto CMD12 response timeout. It is not certain whether the command has been accepted by the card or not. The driver should clear the Auto CMD12 error status bits and resend CMD12 until it is accepted by the card.
2. Auto CMD12 response CRC error. Since CMD12 has been received by the card, the card aborts the transfer. The driver may ignore the error and clear the error status bit.
3. Auto CMD12 conflict error or not sent. The command was not sent. Therefore, the driver should send CMD12 manually.

#### 25.5.3.5 Card Interrupt

The external cards can inform the host controller through the use of special signals. For SDIO cards, it can be the low level on the SDHC\_DAT[1] line during a specific period. It is possible some other external interrupt behaviors can be defined. The eSDHC only monitors the SDHC\_DAT[1] line and supports SDIO interrupts.

When an SDIO interrupt is captured by the eSDHC and the host system is informed by the eSDHC asserting its interrupt line, the interrupt service of host driver is requested.

As the interrupt source is controlled by the external card, the interrupt from the SDIO card must be served before the CINT bit is cleared. Refer to [Section 25.4.6.3, “Card Interrupt Handling,”](#) for the card interrupt handling flow.

### 25.5.4 Switch Function

MMCs transferring data with a bus width other than one-bit wide is a new feature added to the MMC specification. The high-speed timing mode for all card devices is also newly-defined in recent various card specifications. To enable these new features, a type of switch command should be issued by the host driver.

#### NOTE

High-speed mode is not supported on this device.

For SDIO cards, the high speed mode is enabled by writing to CCCR[EHS] after the CCCR[SHS] bit is confirmed. For SD cards, the high-speed mode is queried and enabled by CMD6 (with the mnemonic symbol as SWICH\_FUNC); for MMCs, the high-speed mode is queried by CMD8 and enabled by CMD6 (with the mnemonic symbol as SWITCH).

The 4-bit and 8-bit bus width of MMC is also enabled by the SWITCH command, but with a different argument.

These new functions can also be disabled by software reset (for SDIO card, by setting RES bit in CCCR register; for other cards, by issuing CMD0), but such manner of restoring to normal mode is not recommended because a complete identification process is needed before the card is ready for data transfer.

For simplicity, the following flowcharts do not show a current capability check, which is recommended in the function switch process.

#### 25.5.4.1 Query, Enable and Disable SDIO High Speed Mode

```
enable_sdio_high_speed_mode(void)
{
send CMD52 to query bit SHS at address 0x13;
if (SHS bit is '0')
{
    report the SDIO card does not support high speed mode and return;
}
send CMD52 to set bit EHS at address 0x13 and read after write to confirm EHS bit is set;
change clock divisor value or configure the system clock feeding into eSDHC to generate the
card_clk of around 50MHz;
(data transactions like normal peers)
}
disable_sdio_high_speed_mode(void)
{
send CMD52 to clear bit EHS at address 0x13 and read after write to confirm EHS bit is cleared;
change clock divisor value or configure the system clock feeding into eSDHC to generate the
card_clk of the desired value below 25MHz;
(data transactions like normal peers)
}
```

#### 25.5.4.2 Query, Enable and Disable SD High Speed Mode

```
enable_sd_high_speed_mode(void)
{
    set BLKATTR[BLKCNT] to 1 (block), set BLKATTR[BLKSIZE] to 64 (bytes);
    send CMD6, with argument 0xFFFFF1 and read 64 bytes of data accompanying the R1
        response;
    wait data transfer done bit is set;
    check if the bit 401 of received 512 bit is set;
    if (bit 401 is '0') report the SD card does not support high speed mode and return;
    send CMD6, with argument 0x80FFFFF1 and read 64 bytes of data accompanying the R1
        response;
    check if the bit field 379~376 is 0xF;
    if (the bit field is 0xF) report the function switch failed and return;
    change clock divisor value or configure the system clock feeding into eSDHC to generate
        the card_clk of around 50MHz;
    (data transactions like normal peers)
}
disable_sd_high_speed_mode(void)
{
    set BLKCNT field to 1 (block), set BLKSIZE field to 64 (bytes);
    send CMD6, with argument 0x80FFFFFF0 and read 64 bytes of data accompanying the R1
        response;
    check if the bit field 379~376 is 0xF;
    if (the bit field is 0xF) report the function switch failed and return;
```

```

        change clock divisor value or configure the system clock feeding into eSDHC to generate
        the card_clk of the desired value below 25MHz;
        (data transactions like normal peers)
}

```

### 25.5.4.3 Query, Enable and Disable MMC High Speed Mode

```

enable_mmc_high_speed_mode(void)
{
    send CMD9 to get CSD value of MMC;
    check if the value of SPEC_VER field is 4 or above;
    if (SPEC_VER value is less than 4) report the MMC does not support high speed mode and
        return;
    set BLKCNT field to 1 (block), set BLKSIZE field to 512 (bytes);
    send CMD8 to get EXT_CSD value of MMC;
    extract the value of CARD_TYPE field to check the 'high speed mode' in this MMC is
        26MHz or 52MHz;
    send CMD6 with argument 0x1B90100;
    send CMD13 to wait card ready (busy line released);
    send CMD8 to get EXT_CSD value of MMC;
    check if HS_TIMING byte (byte number 185) is 1;
    if (HS_TIMING is not 1) report MMC switching to high speed mode failed and return;
    change clock divisor value or configure the system clock feeding into eSDHC to generate
        the card_clk of around 26MHz or 52MHz according to the CARD_TYPE;
    (data transactions like normal peers)
}

disable_mmc_high_speed_mode(void)
{
    send CMD6 with argument 0x2B90100;
    set BLKCNT field to 1 (block), set BLKSIZE field to 512 (bytes);
    send CMD8 to get EXT_CSD value of MMC;
    check if HS_TIMING byte (byte number 185) is 0;
    if (HS_TIMING is not 0) report the function switch failed and return;
    change clock divisor value or configure the system clock feeding into eSDHC to generate
        the card_clk of the desired value below 20MHz;
    (data transactions like normal peers)
}

```

### 25.5.4.4 Set MMC Bus Width

```

change_mmc_bus_width(void)
{
    send CMD9 to get CSD value of MMC;
    check if the value of SPEC_VER field is 4 or above;
    if (SPEC_VER value is less than 4) report the MMC does not support multiple bit width
        and return;
    send CMD6 with argument 0x3B70x00; (8-bit, x=2; 4-bit, x=1; 1-bit, x=0)
    send CMD13 to wait card ready (busy line released);
    (data transactions like normal peers)
}

```

### 25.5.4.5 ADMA1 Operation

```

Set_admal_descriptor
{

```

```

if (to start data transfer) {
    // Ensure the address is 4KB aligned.
    Set 'Set' type descriptor;
    {
        Set Act bits to 01;
        Set [31:12] bits data length (byte unit);
    }
    Set 'Tran' type descriptor;
    {
        Set Act bits to 10;
        Set [31:12] bits address (4KB align);
    }
}
else if (to fetch descriptor at non-continuous address) {
    Set Act bits to 11;
    Set [31:12] bits the next descriptor address (4KB align);
}
else { // other types of descriptor
    Set Act bits accordingly
}
if (this descriptor is the last one) {
    Set End bit to 1;
}
if (to generate interrupt for this descriptor) {
    Set Int bit to 1;
}
Set Valid bit to 1;
}

```

#### 25.5.4.6 ADMA2 Operation

```

Set_adma2_descriptor
{
    if (to start data transfer) {
        // Ensure the address is 32-bit boundary (lower 2-bit are always '00').
        Set higher 32-bit of descriptor for this data transfer initial address;
        Set [31:16] bits data length (byte unit);
        Set Act bits to '10';
    }
    else if (to fetch descriptor at non-continuous address) {
        Set Act bits to '11';
        // Ensure the address is 32-bit boundary (lower 2-bit are always set to '00').
        Set higher 32-bit of descriptor for the next descriptor address;
    }
    else { // other types of descriptor
        Set Act bits accordingly
    }
    if (this descriptor is the last one) {
        Set 'End' bit '1';
    }
    if (to generate interrupt for this descriptor) {
        Set 'Int' bit '1';
    }
    Set the 'Valid' bit to '1';
}

```

## 25.5.5 Commands for MMC/SD/SDIO

See [Table 25-31](#) for the list of commands for the MMC/SD/SDIO cards. Refer to the corresponding specifications for details about the command information.

Four kinds of commands control the MMC:

1. Broadcast commands (bc)—no response
2. Broadcast commands with response (bcr)—response from all cards simultaneously
3. Addressed (point-to-point) commands (ac)—no data transfer on SDHC\_DAT
4. Addressed (point-to-point) data transfer commands (ADTC)

**Table 25-31. Commands for MMC/SD/SDIO**

CMD INDEX	Type	Argument	Resp	Abbreviation	Description <sup>1</sup>
CMD0	bc	[31:0] stuff bits	—	GO_IDLE_STATE	Resets all MMC and SD memory cards to idle state.
CMD1	bcr	[31:0] OCR without busy	R3	SEND_OP_COND	Asks all MMCs and SD memory cards in idle state to send their operation conditions register contents in the response on the SDHC_CMD line.
CMD2	bcr	[31:0] stuff bits	R2	ALL_SEND_CID	Asks all cards to send their CID numbers on the SDHC_CMD line.
CMD3 <sup>(1)</sup>	ac	[31:6] RCA [15:0] stuff bits	R1 R6(SDIO)	SET/SEND_RELATIVE_ADDRESS	Assigns relative address to the card.
CMD4	bc	[31:0] DSR [15:0] stuff bits	—	SET_DSR	Programs the DSR of all cards.
CMD5	bc	[31:0] OCR without busy	R4	IO_SEND_OP_COND	Asks all SDIO cards in idle state to send their operation conditions register contents in the response on the SDHC_CMD line.
CMD6 <sup>(2)</sup>	adtc	[31] Mode 0: Check function 1: Switch function [30:8] Reserved for function groups 6 ~ 3 (All 0 or 0xFFFF) [7:4] Function group1 for command system [3:0] Function group2 for access mode	R1	SWITCH_FUNC	Checks switch ability (mode 0) and switch card function (mode 1). Refer to SD Physical Specification version 1.1 for details.
CMD6 <sup>(3)</sup>	ac	[31:26] Set to 0 [25:24] Access [23:16] Index [15:8] Value [7:3] Set to 0 [2:0] Cmd Set	R1b	SWITCH	Switches the mode of operation of the selected card or modifies the EXT_CSD registers. Refer to the MultiMediaCard System Specification version 4.0 final draft 2 for details.

**Table 25-31. Commands for MMC/SD/SDIO (continued)**

CMD INDEX	Type	Argument	Resp	Abbreviation	Description <sup>1</sup>
CMD7	ac	[31:6] RCA [15:0] stuff bits	R1b	SELECT/DESELECT_CARD	Command toggles a card between the stand-by and transfer states or between the programming and disconnect states. In both cases, the card is selected by its own relative address and gets deselected by any other address; address 0 deselects all.
CMD8	adtc	[31:0] stuff bits	R1	SEND_EXT_CSD	The card sends its EXT_CSD register as a block of data, with block size of 512 bytes.
CMD9	ac	[31:6] RCA [15:0] stuff bits	R2	SEND_CSD	Addressed card sends its card-specific data (CSD) on the SDHC_CMD line.
CMD10	ac	[31:6] RCA [15:0] stuff bits	R2	SEND_CID	Addressed card sends its card-identification (CID) on the SDHC_CMD line.
CMD11	adtc	[31:0] data address	R1	READ_DAT_UNTIL_STOP	Reads data stream from the card starting at the given address until STOP_TRANSMISSION is received.
CMD12	ac	[31:0] stuff bits	R1b	STOP_TRANSMISSION	Forces the card to stop transmission.
CMD13	ac	[31:6] RCA [15:0] stuff bits	R1	SEND_STATUS	Addressed card sends its status register.
CMD14		Reserved			
CMD15	ac	[31:6] RCA [15:0] stuff bits	—	GO_INACTIVE_STATE	Sets the card to inactive state in order to protect the card stack against communication breakdowns.
CMD16	ac	[31:0] block length	R1	SET_BLOCKLEN	Sets the block length (in bytes) for all following block commands (read and write). Default block length is specified in the CSD.
CMD17	adtc	[31:0] data address	R1	READ_SINGLE_BLOCK	Reads a block of the size selected by the SET_BLOCKLEN command.
CMD18	adtc	[31:0] data address	R1	READ_MULTIPLE_BLOCK	Continuously transfers data blocks from card to host until interrupted by a stop command.
CMD19		Reserved			
CMD20	adtc	[31:0] data address	R1	WRITE_DAT_UNTIL_STOP	Writes data stream from the host starting at the given address until the STOP_TRANSMISSION command is received.
CMD21–23		Reserved			
CMD24	adtc	[31:0] data address	R1	WRITE_BLOCK	Writes a block of the size selected by the SET_BLOCKLEN command.

**Table 25-31. Commands for MMC/SD/SDIO (continued)**

<b>CMD INDEX</b>	<b>Type</b>	<b>Argument</b>	<b>Resp</b>	<b>Abbreviation</b>	<b>Description<sup>1</sup></b>
CMD25	adtc	[31:0] data address	R1	WRITE_MULTIPLE_BLOCK	Continuously writes blocks of data until the STOP_TRANSMISSION command is received.
CMD26	adtc	[31:0] stuff bits	R1	PROGRAM_CID	Programming of the card identification register. This command should be issued only once per card. The card contains hardware to prevent this operation after the first programming. Normally this command is reserved for the manufacturer.
CMD27	adtc	[31:0] stuff bits	R1	PROGRAM_CSD	Programming of the programmable bits of the CSD.
CMD28	ac	[31:0] data address	R1b	SET_WRITE_PROT	If the card has write-protection features, this command sets the write protection bit of the addressed group. The properties of write protection are coded in the card-specific data (WP_GRP_SIZE).
CMD29	ac	[31:0] data address	R1b	CLR_WRITE_PROT	If the card provides write-protection features, this command clears the write protection bit of the addressed group.
CMD30	adtc	[31:0] write protect data address	R1	SEND_WRITE_PROT	If the card provides write-protection features, this command asks the card to send the status of the write-protection bits.
CMD31				Reserved	
CMD32	ac	[31:0] data address	R1	TAG_SECTOR_START	Sets the address of the first sector of the erase group.
CMD33	ac	[31:0] data address	R1	TAG_SECTOR_END	Sets the address of the last write block of the continuous range to be erased.
CMD34	ac	[31:0] data address	R1	UNTAG_SECTOR	Removes one previously selected sector from the erase selection.
CMD35	ac	[31:0] data address	R1	TAG_ERASE_GROUP_START	Sets the address of the first erase group within a range to be selected for erase.
CMD36	ac	[31:0] data address	R1	TAG_ERASE_GROUP_END	Sets the address of the last erase group within a continuous range to be selected for erase.
CMD37	ac	[31:0] data address	R1	UNTAG_ERASE_GROUP	Removes one previously selected erase group from the erase selection.
CMD38	ac	[31:0] stuff bits	R1b	ERASE	Erase all previously selected sectors.

**Table 25-31. Commands for MMC/SD/SDIO (continued)**

CMD INDEX	Type	Argument	Resp	Abbreviation	Description <sup>1</sup>
CMD39	ac	[31:0] RCA [15] register write flag [14:8] register address [7:0] register data	R4	FAST_IO	Used to write and read 8-bit (register) data fields. The command address a card and a register and provides the data for writing if the write flag is set. The R4 response contains data read from the address register. This command accesses application dependent registers which are not defined in the MMC standard.
CMD40	bcr	[31:0] stuff bits	R5	GO_IRQ_STATE	Sets the system into interrupt mode.
CMD41				Reserved	
CDM42	adtc	[31:0] stuff bits	R1b	LOCK_UNLOCK	Used to set/reset the password or lock/unlock the card. The size of the data block is set by the SET_BLOCK_LEN command.
CMD43–51				Reserved	
CMD52	ac	[31:0] stuff bits	R5	IO_RW_DIRECT	Access a single register within the total 128 Kbytes of register space in any I/O function.
CMD53	ac	[31:0] stuff bits	R5	IO_RW_EXTENDED	Access a multiple I/O register with a single command, it allows the reading or writing of a large number of I/O registers.
CMD54				Reserved	
CMD55	ac	[31:16] RCA [15:0] stuff bits	R1	APP_CMD	Indicates to the card that the next command is an application specific command rather than a standard command.
CMD56	adtc	[31:1] stuff bits [0]: RD/WR	R1b	GEN_CMD	Used either to transfer a data block to the card or to get a data block from the card for general-purpose or application-specific commands. The size of the data block is set by the SET_BLOCK_LEN command.
ACMDs should be preceded with the APP_CMD command (Commands listed below are for SD cards only. Other SD commands not listed below are not supported by this module)					
ACMD6	ac	[31:2] stuff bits [1:0] bus width	R1	SET_BUS_WIDTH	Defines the data bus width (00 = 1 bit or 10 = 4 bit bus) to be used for data transfer. The allowed data bus widths are given in SCR register.
ACMD13	adtc	[31:0] stuff bits	R1	SD_STATUS	Send the SD memory card status.
ACMD22	adtc	[31:0] stuff bits	R1	SEND_NUM_WR_SECTORS	Send the number of the written (without errors) sectors. Responds with 32 bit + CRC data block.

**Table 25-31. Commands for MMC/SD/SDIO (continued)**

CMD INDEX	Type	Argument	Resp	Abbreviation	Description <sup>1</sup>
ACMD23	ac	—	R1	SET_WR_BLK_ERASE_COUNT	—
ACMD41	bcr	[31:0] OCR	R3	SD_APP_OP_COND	Asks the accessed card to send its operating condition register (OCR) content in the response on the SDHC_CMD line.
ACMD42	ac	—	R1	SET_CLR_CARD_DETECT	—
ACMD51	adtc	[31:0] stuff bits	R1	SEND_SCR	Reads the SD Configuration Register (SCR)

<sup>1</sup> Registers mentioned in this table are SD card registers.

### NOTE

- CMD3 differs for MMC and SD cards  
For MMC cards, CMD3 is referred to as SET\_RELATIVE\_ADDR and has a response type R1  
For SD cards, CMD3 is referred to as SEND\_RELATIVE\_ADDR and has a response type R6, with RCA inside
- Command SWITCH is for high-speed MMC cards. The index field can contain any value from 0–255, but only values 0–191 are valid. If the index value is in the 192–255 range, the card does not perform any modification and the status bit EXT\_CSD[SWITCH\_ERROR] is set.  
The access bits are shown in [Table 25-32](#):

**Table 25-32. EXT\_CSD Access Modes**

Bits	Access Name	Operation
00	Command set	The command set is changed according to the command set field of the argument
01	Set bits	The bits in the pointed byte are set, according to the set bits in the value field.
10	Clear bits	The bits in the pointed byte are cleared, according to the set bits in the value field.
11	Write byte	The value field is written into the pointed byte.

## 25.5.6 Software Restrictions

### 25.5.6.1 Initialization Active

The driver cannot set SYSCTL[INITA] when any of the command or data lines are active, so the driver must ensure both CDIHB and CIHB bits are cleared. To auto clear the INITA bit, the SDCLKEN bit must be set; otherwise, no clocks can go to the card and INITA never clears.

### 25.5.6.2 Software Polling Procedure

When polling read or write, once the software begins a buffer read or write, it must access exactly the number of times as set in the watermark level register. Moreover, if the block size is not the value in watermark level register (read and write respectively), the software must access exactly the remaining number of words at the end of each block.

For example, for a read operation, if the RD\_WML is 4, indicating the watermark level is 16 bytes, block size is 40 bytes, and the block number is 2, then the access times for the burst sequence in the whole transfer process must be 4, 4, 2, 4, 4, 2.

### 25.5.6.3 Suspend Operation

To suspend the data transfer, the software must inform the eSDHC that the suspend command is successfully accepted. To achieve this, after the suspend command is accepted by the SDIO card, software must send another normal command marked as suspend command (XFERTYP[CMDTYP] = 01) to inform the eSDHC that the transfer is suspended.

If software must resume the suspended transfer, it should read the value in BLKCNT to save the remaining number of blocks before sending the normal command marked as suspend. Otherwise, on sending the suspend command, the eSDHC regards the current transfer as aborted and changes BLKCNT to its original value, instead of keeping the remained number of blocks.

### 25.5.6.4 Data Length Setting

For an ADMA (ADMA1 or ADMA2) transfer, the data in the data buffer must be word aligned. So, the data length set in the descriptor must be a multiple of 4.

### 25.5.6.5 DMA Address Setting

To configure ADMA/DMA address register, when TC bit is set, the register always updates itself with the internal address value to support dynamic address synchronization. Software must ensure TC bit is cleared prior to configuring the ADMA/DMA address register.

### 25.5.6.6 Data Port Access

The data port does not support parallel access. For example, during an external DMA access, it is not allowed to write any data to the data port by CPU. During a CPU read operation, it is also prohibited to write any data to the data port, by either CPU or external DMA. Otherwise, the data is corrupted inside the eSDHC buffer.

### 25.5.6.7 Change Clock Frequency

The eSDHC does not automatically gate off the card clock when the host driver changes the clock frequency. To remove a possible glitch on the card clock, clear SYSCTL[SDCLKEN] when changing the clock divisor value, and set SDCLKEN after PRSSTAT[SDSTB] sets.



# Chapter 26

## Cryptographic Acceleration Unit (CAU)

### 26.1 Introduction

The cryptographic acceleration unit (CAU) is a ColdFire coprocessor implementing a set of specialized operations in hardware to increase the throughput of software-based encryption and hashing functions.

#### 26.1.1 Block Diagram

Figure 26-1 shows a simplified block diagram of the CAU.

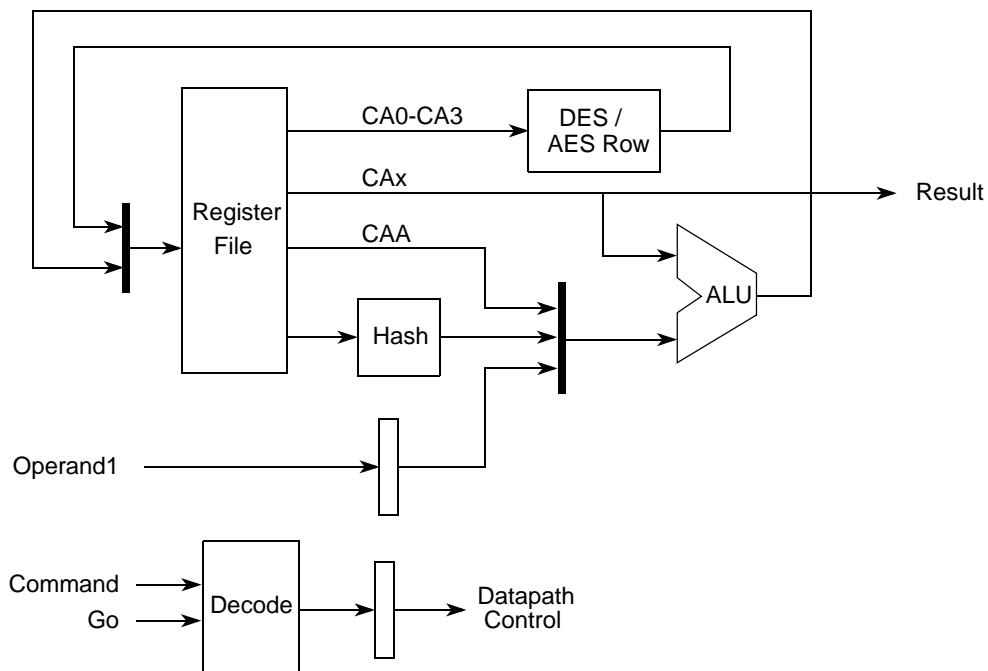


Figure 26-1. Top Level CAU Block Diagram

#### 26.1.2 Overview

The CAU supports acceleration of the following algorithms:

- DES
- 3DES
- AES
- MD5

- SHA-1
- SHA-256

This selection of algorithms provides excellent support for network security standards (SSL, IPsec). Additionally, using the CAU efficiently permits the implementation of any higher level functions or modes of operation (HMAC, CBC, etc.) based on the supported algorithm.

The CAU is an instruction-level ColdFire coprocessor. The cryptographic algorithms are implemented partially in software with only functions critical to increasing performance implemented in hardware. The ColdFire coprocessor allows for efficient, fine-grained partitioning of functions between hardware and software.

- Implement the innermost round functions by using the coprocessor instructions
- Implement higher-level functions in software by using the standard ColdFire instructions

This partitioning of functions is key to minimizing size of the CAU while maintaining a high level of throughput. Using software for some functions also simplifies the CAU design. The CAU implements a set of 22 coprocessor commands that operate on a register file of eight 32-bit registers. It is tightly coupled to the ColdFire core and there is no local memory or external interface.

### 26.1.3 Features

The CAU includes these distinctive features:

- Supports DES, 3DES, AES, MD5, SHA-1, and SHA-256 algorithms
- Simple, flexible programming model

## 26.2 Memory Map/Register Definition

The CAU only supports longword operations and register accesses. All registers support read, write, and ALU operations. However, only bits 1–0 of the CASR are writeable. Bits 31–2 of the CASR must be written as 0 for compatibility with future versions of the CAU.

**Table 26-1. CAU Memory Map**

Code	Register	DES	AES	SHA-1	SHA-256	MD5	Access	Reset Value	Section/Page
0	CAU status register (CASR)	—	—	—	—	—	R/W	0x2000_0000	<a href="#">26.2.1/26-3</a>
1	CAU accumulator (CAA)	—	—	T	T	a	R	0x0000_0000	<a href="#">26.2.2/26-4</a>
2	General purpose register 0 (CA0)	C	W0	A	A	—	R	0x0000_0000	<a href="#">26.2.3/26-4</a>
3	General purpose register 1 (CA1)	D	W1	B	B	b	R	0x0000_0000	<a href="#">26.2.3/26-4</a>
4	General purpose register 2 (CA2)	L	W2	C	C	c	R	0x0000_0000	<a href="#">26.2.3/26-4</a>
5	General purpose register 3 (CA3)	R	W3	D	D	d	R	0x0000_0000	<a href="#">26.2.3/26-4</a>
6	General purpose register 4 (CA4)	—	—	E	E	—	R	0x0000_0000	<a href="#">26.2.3/26-4</a>
7	General purpose register 5 (CA5)	—	—	W	F	—	R	0x0000_0000	<a href="#">26.2.3/26-4</a>

**Table 26-1. CAU Memory Map (continued)**

Code	Register	DES	AES	SHA-1	SHA-256	MD5	Access	Reset Value	Section/Page
8	General purpose register 6 (CA6)	—	—	—	G	—	R	0x0000_0000	26.2.3/26-4
9	General purpose register 7 (CA7)	—	—	—	H	—	R	0x0000_0000	26.2.3/26-4
10	General purpose register 8 (CA8)	—	—	—	W/T <sub>1</sub>	—	R	0x0000_0000	26.2.3/26-4

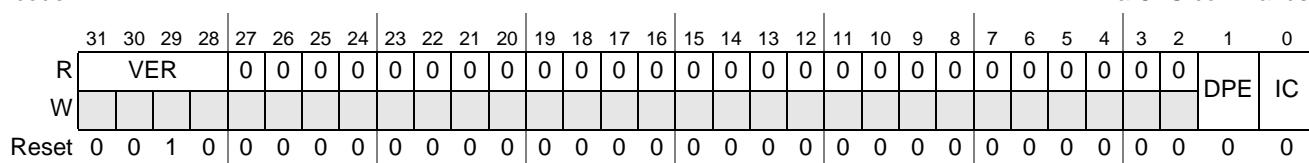
## 26.2.1 CAU Status Register (CASR)

CASR contains the status and configuration for the CAU.

Register 0x0 (CASR)

code:

Access: Read/write  
via CAU commands

**Figure 26-2. CAU Status Register (CASR)****Table 26-2. CASR Field Descriptions**

Field	Description
31–28 VER	CAU version. Indicates CAU version 0x1 Initial CAU version 0x2 Second version, added support for SHA-256 algorithm (This is the value on this device)
27–2	Reserved, must be cleared.
1 DPE	DES parity error. 0 No error detected 1 DES key parity error detected
0 IC	Illegal command. Indicates an illegal instruction not found in Section 26.3.3, “CAU Commands,” has been executed. 0 No illegal commands issued 1 Illegal coprocessor command issued

## 26.2.2 CAU Accumulator (CAA)

CAU commands use the CAU accumulator for storage of results and as an operand for the cryptographic algorithms.

Register 0x1 (CAA)  
code:

Access: Read/write  
via CAU commands

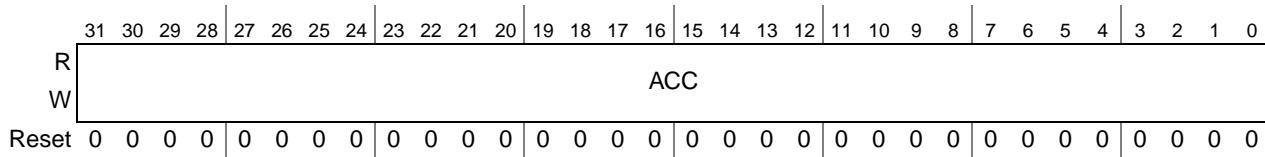


Figure 26-3. CAU Accumulator Register (CAA)

Table 26-3. CAA Field Descriptions

Field	Description
31–0 ACC	Accumulator. Stores results of various CAU commands.

## 26.2.3 CAU General Purpose Registers (CAn)

The nine CAU general purpose registers are used in the CAU commands for storage of results and as operands for the various cryptographic algorithms.

Register 0x2 (CA0)  
code: 0x3 (CA1)  
0x4 (CA2)  
0x5 (CA3)  
0x6 (CA4)  
0x7 (CA5)

0x8 (CA6)  
0x9 (CA7)  
0xA (CA8)

Access: Read/write  
via CAU commands

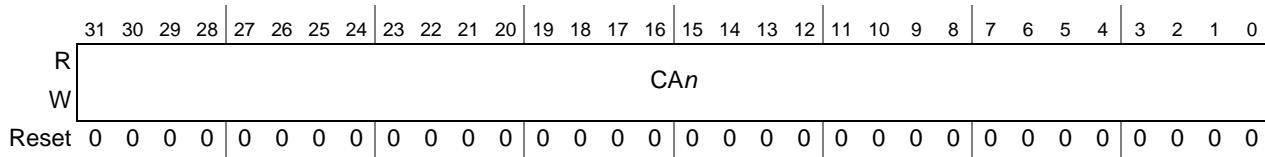


Figure 26-4. CAU General Purpose Registers (CAn)

Table 26-4. CAn Field Descriptions

Field	Description
31–0 CAn	General purpose registers. Used by the CAU commands. Some cryptographic operations work with specific registers.

## 26.3 Functional Description

### 26.3.1 Programming Model

The CAU is an instruction-level coprocessor. It has a dedicated register file, a specialized ALU, and specialized units for performing cryptographic operations. The CAU design uses a simple, flexible

accumulator-based architecture. Most commands, including load and store, can specify any register in the register file. Some cryptographic operations work with specific registers.

### 26.3.2 Coprocessor Instructions

Operation of the CAU is controlled via standard ColdFire coprocessor load (cp0ld) and store (cp0st) instructions. The CAU has a dedicated register file accessed using these instructions. The load instruction loads CAU registers and specifies CAU operations. The store instruction stores CAU registers. The example assembler syntax for the CAU is:

```
cp0ld.1      <ea>, <CMD> ; coprocessor load
cp0st.1      <ea>, <CMD> ; coprocessor store
```

The <ea> field specifies the source operand (operand1) for load instructions and destination (result) for store instructions. The basic ColdFire addressing modes {Rn, (An), -(An), (An)+, (d16,An)} are supported for this field. The <CMD> field is a 9-bit value that specifies the CAU command for an instruction.

[Table 26-5](#) shows how the CAU supports a single command (STR) for store instructions and 21 commands for the load instructions. The CAU only supports longword operations. A CAU command can be issued every clock cycle.

### 26.3.3 CAU Commands

The CAU supports the commands shown in [Table 26-5](#). All other encodings are reserved. The CASR[IC] bit is set if an undefined command is issued. A specific illegal command (ILL) is defined to allow software self-checking. Reserved commands should not be issued to ensure compatibility with future implementations.

The CMD field specifies the CAU command for the instruction.

**Table 26-5. CAU Commands**

Inst Type	Command Name	Description	CMD									Operation
			8	7	6	5	4	3	2	1	0	
cp0ld	CNOP	No Operation	0x000									—
cp0ld	LDR	Load Reg	0x01				CAx				Op1 → CAx	
cp0st	STR	Store Reg	0x02				CAx				CAx → Result	
cp0ld	ADR	Add	0x03				CAx				CAx + Op1 → CAx	
cp0ld	RADR	Reverse and Add	0x04				CAx				CAx + ByteRev(Op1) → CAx	
cp0ld	ADRA	Add Reg to Acc	0x05				CAx				CAx + CAA → CAA	
cp0ld	XOR	Exclusive Or	0x06				CAx				CAx ^ Op1 → CAx	
cp0ld	ROTL	Rotate Left	0x07				CAx				CAx <<< Op1 → CAx	
cp0ld	MVRA	Move Reg to Acc	0x08				CAx				CAA → CAx	
cp0ld	MVAR	Move Acc to Reg	0x09				CAx				CAA → CAx	
cp0ld	AESS	AES Sub Bytes	0x0A				CAx				SubBytes(CAx) → CAx	

**Table 26-5. CAU Commands (continued)**

Inst Type	Command Name	Description	CMD									Operation
			8	7	6	5	4	3	2	1	0	
cp0ld	AESIS	AES Inv Sub Bytes	0x0B									InvSubBytes(CAx) → CAx
cp0ld	AESC	AES Column Op	0x0C									MixColumns(CAx)^Op1 → CAx
cp0ld	AESIC	AES Inv Column Op	0x0D									InvMixColumns(CAx^Op1) → CAx
cp0ld	AESR	AES Shift Rows	0x0E0									ShiftRows(CA0-CA3) → CA0-CA3
cp0ld	AESIR	AES Inv Shift Rows	0x0F0									InvShiftRows(CA0-CA3) → CA0-CA3
cp0ld	DESR	DES Round	0x10		IP	FP	KS[1:0]					DES Round(CA0-CA3) → CA0-CA3
cp0ld	DESK	DES Key Setup	0x11		0	0	CP	DC				DES Key Op(CA0-CA1) → CA0-CA1 Key Parity Error & CP → CASR[1]
cp0ld	HASH	Hash Function	0x12		0	HF[2:0]						Hash Func(CA1-CA3)+CAA → CAA
cp0ld	SHS	Secure Hash Shift	0x130									CAA << 5 → CAA, CAA → CA0, CA0 → CA1, CA1 << 30 → CA2, CA2 → CA3, CA3 → CA4
cp0ld	MDS	Message Digest Shift	0x140									CA3 → CAA, CAA → CA1, CA1 → CA2, CA2 → CA3,
cp0ld	SHS2	Secure Hash Shift 2	0x150									CAA → CA0, CA0 → CA1, CA1 → CA2, CA2 → CA3, CA3 + CA8 → CA4, CA4 → CA5, CA5 → CA6, CA6 → CA7
cp0ld	ILL	Illegal Command	0x1F0									0x1 → CASR[0]

[Section 26.4.2, “Assembler Equate Values,”](#) contains a set of assembly constants used in the command descriptions here. If supported by the assembler, macros can also be created for each instruction. The value CAx should be interpreted as any CAU register (CASR, CAA, CAn) and the <ea> field as one of the supported ColdFire addressing modes {Rn, (An), -(An), (An)+, (d16,An)}. For example, the instruction to add the value from the core register D1 to the CAU register CA0 is:

```
cp0ld.1      %d1, #ADR+CA0 ; CA0=CA0+d1
```

### 26.3.3.1 Coprocessor No Operation (CNOP)

```
cp0ld.1      #CNOP
```

The CNOP command is the coprocessor no-op defined by the ColdFire coprocessor definition for synchronization. It is not actually issued to the coprocessor from the core.

### 26.3.3.2 Load Register (LDR)

```
cp0ld.1      <ea>, #LDR+CAx
```

The LDR command loads CAx with the source data specified by <ea>.

### 26.3.3.3 Store Register (STR)

```
cp0st.1 <ea>, #STR+CAx
```

The STR command stores the value from CAx to the destination specified by <ea>.

### 26.3.3.4 Add to Register (ADR)

```
cp0ld.1 <ea>, #ADR+CAx
```

The ADR command adds the source operand specified by <ea> to CAx and stores the result in CAx.

### 26.3.3.5 Reverse and Add to Register (RADR)

```
cp0ld.1 <ea>, #RADR+CAx
```

The RADR command performs a byte reverse on the source operand specified by <ea>, adds that value to CAx, and stores the result in CAx. [Table 26-6](#) shows an example.

**Table 26-6. RADR Command Example**

Operand	CAx Before	CAx After
0x0102_0304	0xA0B0_C0D0	0xA4B3_C2D1

### 26.3.3.6 Add Register to Accumulator (ADRA)

```
cp0ld.1 #ADRA+CAx
```

The ADRA command adds CAx to CAA and stores the result in CAA.

### 26.3.3.7 Exclusive Or (XOR)

```
cp0ld.1 <ea>, #XOR+CAx
```

The XOR command performs an exclusive-or of the source operand specified by <ea> with CAx and stores the result in CAx.

### 26.3.3.8 Rotate Left (ROTL)

```
cp0ld.1 <ea>, #ROTL+CAx
```

ROTL rotates the CAx bits to the left with the result stored back to CAx. The number of bits to rotate is the value specified by <ea> modulo 32.

### 26.3.3.9 Move Register to Accumulator (MVRA)

```
cp0ld.1 #MVRA+CAx
```

The MVRA command moves the value from the source register CAx to the destination register CAA.

### 26.3.3.10 Move Accumulator to Register (MVAR)

```
cp0ld.1 #MVAR+CAx
```

The MVAR command moves the value from source register CAA to the destination register CAx.

### 26.3.3.11 AES Substitution (AESS)

```
cp0ld.1 #AESS+CAx
```

The AESSION command performs the AES byte substitution operation on CAx and stores the result back to CAx.

### 26.3.3.12 AES Inverse Substitution (AESIS)

```
cp0ld.1 #AESIS+CAx
```

The AESIS command performs the AES inverse byte substitution operation on CAx and stores the result back to CAx.

### 26.3.3.13 AES Column Operation (AESC)

```
cp0ld.1 <ea>, #AESC+CAx
```

The AESC command performs the AES column operation on the contents of CAx then performs an exclusive-or of that result with the source operand specified by <ea> and stores the result in CAx.

### 26.3.3.14 AES Inverse Column Operation (AESIC)

```
cp0ld.1 <ea>, #AESIC+CAx
```

The AESIC command performs an exclusive-or operation of the source operand specified by <ea> on the contents of CAx followed by the AES inverse mix column operation on that result and stores the result back in CAx.

### 26.3.3.15 AES Shift Rows (AESR)

```
cp0ld.1 #AESR
```

The AESR command performs the AES shift rows operation on registers CA0, CA1, CA2, and CA3. [Table 26-7](#) shows an example.

**Table 26-7. AESR Command Example**

Register	Before	After
CA0	0x0102_0304	0x0106_0B00
CA1	0x0506_0708	0x050A_0F04
CA2	0x090A_0B0C	0x090E_0308
CA3	0x0D0E_0F00	0x0D02_070C

### 26.3.3.16 AES Inverse Shift Rows (AESIR)

```
cp0ld.1 #AESIR
```

The AESIR command performs the AES inverse shift rows operation on registers CA0, CA1, CA2 and CA3. [Table 26-8](#) has an example.

**Table 26-8. AESIR Command Example**

Register	Before	After
CA0	01060B00	01020304
CA1	050A0F04	05060708
CA2	090E0308	090A0B0C
CA3	0D02070C	0D0E0F00

### 26.3.3.17 DES Round (DESR)

```
cp0ld.1 #DESR+{IP}+{FP}+{KSx}
```

The DESR command performs a round of the DES algorithm and a key schedule update with the following source and destination designations: CA0=C, CA1=D, CA2=L, CA3=R. If the IP bit is set, DES initial permutation performs on CA2 and CA3 before the round operation. If the FP bit is set, DES final permutation (inverse initial permutation) performs on CA2 and CA3 after the round operation. The round operation uses the source values from registers CA0 and CA1 for the key addition operation. The KSx field specifies the shift for the key schedule operation to update the values in CA0 and CA1. [Table 26-9](#) defines the specific shift function performed based on the KSx field.

**Table 26-9. Key Shift Function Codes**

Ks <sub>x</sub> Code	Ks <sub>x</sub> Define	Shift Function
0	KSL1	Left 1
1	KSL2	Left 2
2	KSR1	Right 1
3	KSR2	Right 2

### 26.3.3.18 DES Key Setup (DESK)

```
cp0ld.1 #DESK+{CP}+{DC}
```

The DESK command performs the initial key transformation (permuted choice 1) defined by the DES algorithm on CA0 and CA1 with CA0 containing bits 1–32 of the key and CA1 containing bits 33–64 of the key<sup>1</sup>. If the DC bit is set, no shift operation performs and the values C<sub>0</sub> and D<sub>0</sub> store back to CA0 and CA1 respectively. The DC bit should be set for decrypt operations. If the DC bit is not set, a left shift by one also occurs and the values C<sub>1</sub> and D<sub>1</sub> store back to CA0 and CA1 respectively. The DC bit should be cleared for encrypt operations. If the CP bit is set and a key parity error is detected, CASR[DPE] bit is set; otherwise, it is cleared.

1.The DES algorithm numbers the most significant bit of a block as bit 1 and the least significant as bit 64.

### 26.3.3.19 Hash Function (HASH)

cp0ld.1 #HASH+HFx

The HASH command performs a hashing operation on a set of registers and adds that result to the value in CAA and stores the result in CAA. The specific hash function performed is based on the HFx field as defined in [Table 26-10](#).

**Table 26-10. Hash Function Codes**

HFx Code	HFx Define	Hash Function	Hash Logic
0	HFF	MD5 F()	(CA1 & CA2)   ( $\overline{CA1}$ & CA3)
1	HFG	MD5 G()	(CA1 & CA3)   (CA2 & $\overline{CA3}$ )
2	HFH	MD5 H(), SHA Parity()	CA1 ^ CA2 ^ CA3
3	HFI	MD5 I()	CA2 ^ (CA1   $\overline{CA3}$ )
4	HFC	SHA Ch()	(CA1 & CA2) ^ ( $\overline{CA1}$ & CA3)
5	HFM	SHA Maj()	(CA1 & CA2) ^ (CA1 & CA3) ^ (CA2 & CA3)
6	HF2C	SHA-256 Ch()	(CA4 & CA5) ^ ( $\overline{CA4}$ & CA6)
7	HF2M	SHA-256 Maj()	(CA0 & CA1) ^ (CA0 & CA2) ^ (CA1 & CA2)
8	HF2S	SHA-256 Sigma 0	ROTR <sup>2</sup> (CA0) ^ ROTR <sup>13</sup> (CA0) ^ ROTR <sup>22</sup> (CA0)
9	HF2T	SHA-256 Sigma 1	ROTR <sup>6</sup> (CA4) ^ ROTR <sup>11</sup> (CA4) ^ ROTR <sup>25</sup> (CA4)
A	HF2U	SHA-256 sigma 0	ROTR <sup>7</sup> (CA8) ^ ROTR <sup>18</sup> (CA8) ^ SHR <sup>3</sup> (CA8)
B	HF2V	SHA-256 sigma 1	ROTR <sup>17</sup> (CA8) ^ ROTR <sup>19</sup> (CA8) ^ SHR <sup>10</sup> (CA8)

### 26.3.3.20 Secure Hash Shift (SHS)

cp0ld.1 #SHS

The SHS command does a set of parallel register-to-register move and shift operations for implementing SHA-1. The following source and destination assignments are made: CAA=CAA<<<5, CA0=CAA, CA1=CA0, CA2=CA1<<<30, CA3=CA2, CA4=CA3.

### 26.3.3.21 Message Digest Shift (MDS)

cp0ld.1 #MDS

The MDS command does a set of parallel register-to-register move operations for implementing MD5. The following source and destination assignments are made: CAA=CA3, CA1=CAA, CA2=CA1, CA3=CA2.

### 26.3.3.22 Secure Hash Shift 2 (SHS2)

cp0ld.1 #SHS2

The SHS2 command does an addition and a set of register to register moves in parallel for implementing SHA-256. The following source and destination assignments are made: CA0=CAA, CA1=CA0, CA2=CA1, CA3=CA2, CA4=CA3+CA8, CA5=CA4, CA6=CA5, CA7=CA6.

### 26.3.3.23 Illegal Command (ILL)

```
cp0ld.1    #ILL
```

The ILL command is a specific illegal command that sets CASR[IC]. All other illegal commands are reserved for use in future implementations.

## 26.4 Application/Initialization Information

### 26.4.1 Code Example

A code fragment is shown below as an example of how the CAU is used. This example shows the round function of the AES algorithm. Core register A0 is pointing to the key schedule.

```
cp0ld.1    #AESS+CA0          ; sub bytes w0
cp0ld.1    #AESS+CA1          ; sub bytes w1
cp0ld.1    #AESS+CA2          ; sub bytes w2
cp0ld.1    #AESS+CA3          ; sub bytes w3
cp0ld.1    #AESR              ; shift rows
cp0ld.1    (%a0)+,#AESC+CA0  ; mix col, add key w0
cp0ld.1    (%a0)+,#AESC+CA1  ; mix col, add key w1
cp0ld.1    (%a0)+,#AESC+CA2  ; mix col, add key w2
cp0ld.1    (%a0)+,#AESC+CA3  ; mix col, add key w3
```

### 26.4.2 Assembler Equate Values

The following equates ease programming of the CAU.

```
; CAU Registers (CAx)
.set      CASR,0x0
.set      CAA,0x1
.set      CA0,0x2
.set      CA1,0x3
.set      CA2,0x4
.set      CA3,0x5
.set      CA4,0x6
.set      CA5,0x7
.set      CA6,0x8
.set      CA7,0x9
.set      CA8,0xA

; CAU Commands
.set      CNOP,0x000
.set      LDR,0x010
.set      STR,0x020
.set      ADR,0x030
.set      RADR,0x040
.set      ADRA,0x050
.set      XOR,0x060
.set      ROTL,0x070
.set      MVRA,0x080
.set      MVAR,0x090
.set      AECC,0x0A0
.set      AESIS,0x0B0
.set      AESC,0x0C0
```

## Cryptographic Acceleration Unit (CAU)

```
.set AESIC,0x0D0
.set AESR,0x0E0
.set AESIR,0x0F0
.set DESR,0x100
.set DESK,0x110
.set HASH,0x120
.set SHS,0x130
.set MDS,0x140
.set SHS2,0x150
.set ILL,0x1F0

; DESR Fields
.set IP,0x08      ; initial permutation
.set FP,0x04      ; final permutation
.set KSL1,0x00    ; key schedule left 1 bit
.set KSL2,0x01    ; key schedule left 2 bits
.set KSR1,0x02    ; key schedule right 1 bit
.set KSR2,0x03    ; key schedule right 2 bits

; DESK Field
.set DC,0x01      ; decrypt key schedule
.set CP,0x02      ; check parity

; HASH Functions Codes
.set HFF,0x0          ; MD5 F() CA1&CA2 | ~CA1&CA3
.set HFG,0x1          ; MD5 G() CA1&CA3 | CA2&~CA3
.set HFH,0x2          ; MD5 H(), SHA Parity() CA1^CA2^CA3
.set HFI,0x3          ; MD5 I() CA2^(CA1|~CA3)
.set HFC,0x4          ; SHA Ch() CA1&CA2 ^ ~CA1&CA3
.set HFM,0x5          ; SHA Maj() CA1&CA2 ^ CA1&CA3 ^ CA2&CA3
.set HF2C,0x6          ; SHA-256 Ch() CA4&CA5 ^ ~CA4&CA6
.set HF2M,0x7          ; SHA-256 Maj() CA0&CA1 ^ CA0&CA2 ^ CA1&CA2
.set HF2S,0x8          ; SHA-256 Sigma 0 ROTR2(CA0)^ROTR13(CA0)^ROTR22(CA0)
.set HF2T,0x9          ; SHA-256 Sigma 1 ROTR6(CA4)^ROTR11(CA4)^ROTR25(CA4)
.set HF2U,0xA          ; SHA-256 sigma 0 ROTR7(CA8)^ROTR18(CA8)^SHR3(CA8)
.set HF2V,0xB          ; SHA-256 sigma 1 ROTR17(CA8)^ROTR19(CA8)^SHR10(CA8)
```

# Chapter 27

## Random Number Generator (RNG)

### 27.1 Introduction

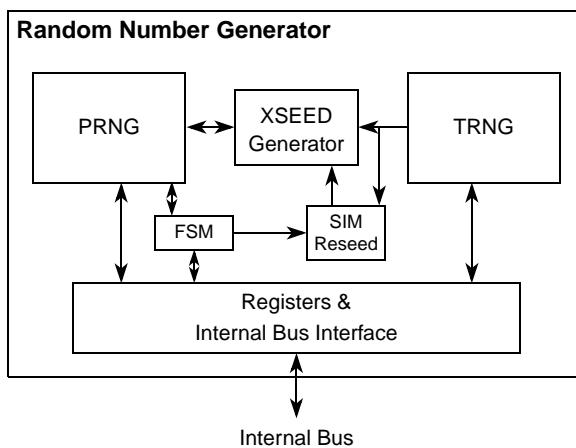
This chapter describes the random number generator (RNG), including a programming model, functional description, and application information.

#### NOTE

The MCF54410, MCF54415, and MCF54417 do not contain cryptography modules. Refer to [Table 1-1](#) for details on device configurations.

#### 27.1.1 Block Diagram

[Figure 27-1](#) shows the RNG's three main blocks: PRNG, TRNG, and XSEED generator. [Section 27.4, “Functional Description,”](#) describes these blocks in more detail.



**Figure 27-1. RNG Block Diagram**

#### 27.1.2 Overview

The purpose of the RNG is to generate cryptographically strong random data. It uses a true random number generator (TRNG) and a pseudo-random number generator (PRNG) to achieve true randomness and cryptographic strength. The RNG generates random numbers for secret keys, per message secrets, random challenges, and other similar quantities used in cryptographic algorithms.

### 27.1.3 Features

The RNG includes these distinctive features:

- National Institute of Standards and Technology (NIST)-approved pseudo-random number generator
  - <http://csrc.nist.gov>
- Supports the key generation algorithm defined in the Digital Signature Standard
  - <http://www.itl.nist.gov/fipspubs/fip186.htm>
- Integrated entropy sources capable of providing the PRNG with entropy for its seed.

## 27.2 Modes of Operation

The RNG operates in the following modes.

### 27.2.1 Self Test Mode

In this mode the RNG performs a self test of the statistical counters and the PRNG algorithm to verify that the hardware is functioning properly. The self test takes ~29,000 cycles to complete. When self test completes an interrupt may be generated, if there are no outstanding commands in the command register. This mode is entered by setting the RNGCMD[ST] bit. When self test mode completes, the RNG remains idle until seed mode is requested or the RNG transitions to seed mode if automatic seeding is enabled.

### 27.2.2 Seed Generation Mode

During seed generation, the RNG adds entropy generated in the TRNG to the 256-bit XKEY register. The PRNG algorithm executes 20,000 times sampling the entropy from the TRNG to create an initial seed for random number generation. At the same time, the TRNG runs simple statistical tests on its output.

When seed generation is complete, the TRNG reports the pass/fail result of the tests through RNGESR. If the new seed passes the statistical tests, RNGSR[SDN] is set, signalling that the RNG is ready to compute secure pseudo-random data. The RNG then transitions to random number generation mode.

### 27.2.3 Random Number Generation Mode

When seed generation mode completes and the output FIFO is empty, the RNG enters this mode automatically. Random number generation mode quickly creates computationally random data that is derived by the initial seed produced in seed generation mode.

During random number generation, a new 160-bit random number is generated whenever the five word output FIFO is empty. When the output FIFO contains data, the RNG automatically enters sleep mode, waiting for the data to be read. When the data is read, the RNG generates a new 160-bit word and goes back to sleep.

After generating  $2^{20}$  words of random data, the RNG lets the user know that it requires reseeding through RNGSR and continues to generate random data until it is directed to reseed. However, if auto-seeding is selected, the RNG automatically completes seeding whenever it is needed.

## 27.3 Memory Map/Register Definition

[Table 27-1](#) shows the address map for the RNG module. Detailed register descriptions are found in the following section.

**Table 27-1. RNG Block Memory Map**

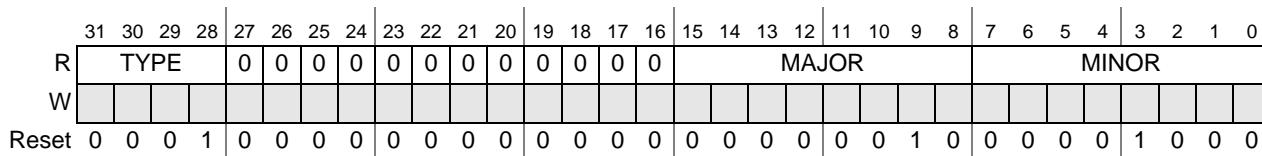
Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0C_4000	RNG Version ID Register (RNGVER)	32	R/W	0x1000_0280	<a href="#">27.3.1/27-3</a>
0xFC0C_4004	RNG Command Register (RNGCMD)	32	R/W	0x0000_0000	<a href="#">27.3.2/27-4</a>
0xFC0C_4008	RNG Control Register (RNGCR)	32	R/W	0x0000_0000	<a href="#">27.3.3/27-5</a>
0xFC0C_400C	RNG Status Register (RNGSR)	32	R	0x0000_500D	<a href="#">27.3.4/27-6</a>
0xFC0C_4010	RNG Error Status Register (RNGESR)	32	R	0x0000_0000	<a href="#">27.3.5/27-7</a>
0xFC0C_4014	RNG Output FIFO (RNGOUT)	32	R	0x0000_0000	<a href="#">27.3.6/27-8</a>
0xFC0C_4018	RNG Entropy Register (RNGER)	32	W	0x0000_0000	<a href="#">27.3.7/27-8</a>

### 27.3.1 RNG Version ID Register (RNGVER)

The read-only RNGVER register contains the current version of the RNG. It consists of the RNG type and major and minor revision numbers.

Address: 0xFC0C\_4000

Access: User read/write



**Figure 27-2. RNG Version Register (RNGVER)**

**Table 27-2. RNGVER Field Descriptions**

Field	Description
31–28 TYPE	Random number generator type. 0000 RNGA 0001 RNGB (This is the type used in this module) 0010 RNGC Else Reserved
27–16	Reserved, must be cleared.
15–8 MAJOR	Major version number. This field is always set to 0x02.
7–0 MINOR	Minor version number. Subject to change.

## 27.3.2 RNG Command Register (RNGCMD)

RNGCMD controls the RNG's operating modes and interrupt status.

Address: 0xFC0C\_4004

Access: User 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	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	GS	ST		
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		

Figure 27-3. RNG Command Register (RNGCMD)

Table 27-3. RNGCMD Field Descriptions

Field	Description
31–7	Reserved, must be cleared.
6 SR	Software reset. Performs a software reset of the RNG. This bit is self-clearing. 0 Do not perform a software reset 1 Software reset
5 CE	Clear error. Clears the errors in the RNGESR register and the RNG interrupt. This bit is self-clearing. 0 Do not clear errors and interrupt 1 Clear errors and interrupt
4 CI	Clear interrupt. Clears the RNG interrupt if an error is not present. This bit is self-clearing. 0 Do not clear interrupt 1 Clear interrupt
3–2	Reserved, must be cleared.
1 GS	Generate seed. Initiates the seed generation process described in <a href="#">Section 27.2.2, “Seed Generation Mode”</a> . Seed generation starts <ul style="list-style-type: none"> <li>• When RNGSR[BUSY] is cleared</li> <li>• If set simultaneously with ST, after self-test</li> </ul> When the seed generation process completes, this bit automatically clears and an interrupt may be generated if all requested operations are complete. 0 Not in seed generation mode 1 Generate seed mode
0 ST	Self test. Initiates a self test of the RNG's internal logic, described in <a href="#">Section 27.2.1, “Self Test Mode”</a> . The self-test starts <ul style="list-style-type: none"> <li>• When RNGSR[BUSY] is cleared, or</li> <li>• If set simultaneously with GS, self test takes precedence and is completed first.</li> </ul> When self test completes, this bit automatically clears and an interrupt may be generated if all requested operations are complete. 0 Not in self test mode 1 Self test mode

### 27.3.3 RNG Control Register (RNGCR)

Through use of this register, the RNG can be programmed to provide slightly different functionality based on its desired use.

Address: 0xFC0C\_4008

Access: User 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	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	MSK	MSK	AS	0	0	FUF	MOD
W																									ERR	DN						
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	

Figure 27-4. RNG Control Register (RNGCR)

Table 27-4. RNGCR Field Descriptions

Field	Description
31–7	Reserved, must be cleared.
7	Reserved, must be cleared.
6 MSKERR	Mask error interrupt. Masks interrupts generated by errors in the RNG. These errors can still be viewed in RNGESR. 0 No mask applied 1 Mask applied to the error interrupt
5 MSKDN	Mask done interrupt. Masks interrupts generated upon completion of seed and self test modes. The status of these jobs can be viewed by: <ul style="list-style-type: none"><li>Reading RNGSR and viewing the seed done and self test done bits (RNGSR[SDN, STDN])</li><li>Viewing RNGCMD for generate seed or self test bits (RNGCMD[GS, ST]) being set, indicating that the operation is still taking place.</li></ul> 0 No mask applied 1 Mask applied
4 AR	Auto-reseed. Setting this bit allows the RNG to automatically generate a new seed whenever one is needed. This allows software to never use the RNGCMD[GS], although it is still possible. A new seed is needed whenever the RNGSR[RS] is set. 0 Do not enable automatic reseeding 1 Enable automatic reseeding
3–2	Reserved, must be cleared.
1–0 FUFMOD	FIFO underflow response mode. Controls the RNG's response to a FIFO underflow condition. 00 Return all zeros and set RNGESR[FUF] 01 Generate bus transfer wait until data is available, then return generated word 10 Generate bus transfer error. See <a href="#">Chapter 13, “System Control Module (SCM)”,</a> for more details. 11 Generate interrupt and return all zeros. (Overrides RNGCR[MSKERR].)

### 27.3.4 RNG Status Register (RNGSR)

The RNGSR is a read-only register which reflects the internal status of the RNG.

Address: 0xFC0C\_400C

Access: User read-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	STATPF								SELFPF		0	0	0	0	0	ERR
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	FS				FL				0	NSDN	SDN	STDN	RS	SLP	BUSY	1
W																
Reset	0	1	0	1	0	0	0	0	0	0	0	0	1	1	0	1

Figure 27-5. RNG Status Register (RNGSR)

Table 27-5. RNGSR Field Descriptions

Field	Description
31–24 STATPF	Statistics test pass fail. Indicates pass or fail status of the various statistics tests on the last seed generated. <ul style="list-style-type: none"><li>• Bit 31 - Long run test (&gt;34)</li><li>• Bit 30 - Length 6+ run test</li><li>• Bit 29 - Length 5 run test</li><li>• Bit 28 - Length 4 run test</li><li>• Bit 27 - Length 3 run test</li><li>• Bit 26 - Length 2 run test</li><li>• Bit 25 - Length 1 run test</li><li>• Bit 24 - Monobit test</li></ul> 0 Pass 1 Fail
23–22 SELFPF	Self Test Pass Fail. Indicates Pass or Fail status of the TRNG and PRNG Self Tests, 1 indicates failure, 0 pass. <ul style="list-style-type: none"><li>• Bit 23 - TRNG self test pass/fail</li><li>• Bit 22 - PRNG self test pass/fail</li></ul> 0 Pass 1 Fail
21–17	Reserved, must be cleared.
16 ERR	Error. Indicates an error was detected in the RNG. Read the RNGESR register for details. 0 No error 1 Error detected
15–12 FS	FIFO size. Size of the FIFO, and maximum possible FIFO level. This value is set to five.
11–8 FL	FIFO level. Indicates the number of random words currently in the output FIFO.
6 NSDN	New seed done. Indicates that a new seed is ready for use during the next seed generation process.
5 SDN	Seed done. Indicates the RNG has generated the first seed. 0 Seed generation process not complete 1 Completed seed generation since the last reset

**Table 27-5. RNGSR Field Descriptions (continued)**

Field	Description
4 STDN	Self test done. Indicates the self test is complete. This bit is cleared by hardware reset or a new self test is initiated by setting RNGCMD[ST]. 0 Self test not complete 1 Completed a self test since the last reset
3 RS	Reseed needed. Indicates the RNG needs to be reseeded. This is done by setting RNGCMD[GS], or automatically if RNGCR[AR] is set. 0 RNG does not need to be reseeded 1 RNG needs to be reseeded
2 SLP	Sleep. Indicates if the RNG is in sleep mode. When set, the RNG is in sleep mode and all internal clocks are disabled. While in this mode, access to the FIFO is allowed. Once the FIFO is empty, the RNG fills the FIFO and then enters sleep mode again. 0 RNG is not in sleep mode 1 RNG is in sleep mode
1 BUSY	Busy. Reflects the current state of RNG. If RNG is currently seeding, generating the next seed, creating a new random number, or performing a self test, this bit is set. 0 Not busy 1 Busy
0	Reserved, must be set.

### 27.3.5 RNG Error Status Register (RNGESR)

Address: 0xFC0C\_4010

Access: User read-only

	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	FUF	SATE	STE	OSCE	LFE		
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	

**Figure 27-6. RNG Error Status Register (RNGESR)****Table 27-6. RNGESR Field Descriptions**

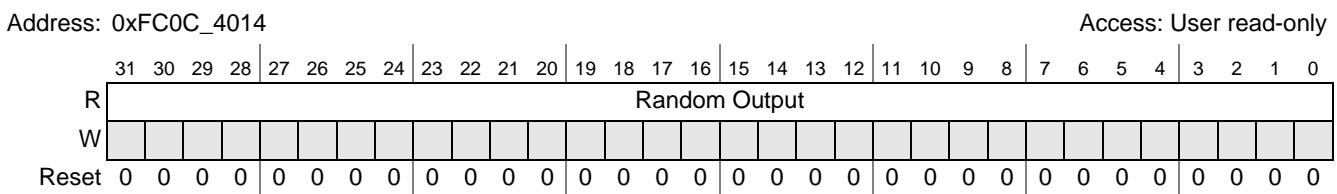
Field	Description
31–5	Reserved, must be cleared.
4 FUF	FIFO underflow error. Indicates the RNG has experienced a FIFO underflow condition resulting in the last random data read being unreliable. This bit can be masked by RNGCR[FUFMOD] and is cleared by hard or soft reset or by setting RNGCMD[CE]. 1 FIFO underflow has occurred 0 FIFO underflow has not occurred
3 SATE	Statistical test error. Indicates if RNG has failed the statistical tests for the last generated seed. This bit is sticky and is cleared by a hardware or software reset, setting RNGCMD[CE]. 1 RNG has failed the statistical tests during initialization 0 RNG has not failed the statistical tests
2 STE	Self test error. Indicates the RNG has failed the most recent self test. This bit is sticky and can only be reset by a software or hardware reset. 0 RNG has not failed self test 1 RNG has failed self test

**Table 27-6. RNGESR Field Descriptions (continued)**

Field	Description
1 OSCE	Oscillator error. Indicates the oscillator in the RNG may be broken. This bit is sticky and can only be cleared by a software or hardware reset. 0 RNG oscillator is working properly 1 Problem detected with the RNG oscillator
0 LFE	Linear feedback shift register (LFSR) error. When this bit is set, the interrupt generated was caused by a failure of one of the LFSRs in one of the RNG's three entropy sources. This bit is sticky and can only be reset via a hardware reset. 0 LFSRs are working properly 1 LFSR failure has occurred

### 27.3.6 RNG Output FIFO (RNGOUT)

The RNGOUT provides temporary storage for random data generated by the RNG. This allows the user to read multiple random longwords back-to-back. A read of this address when the FIFO is not empty, returns 32 bits of random data. If the FIFO is read when empty, a FIFO underrun response is returned according to RNGCR[FUFMOD]. For optimal system performance, poll RNGSR[FL] to ensure random values are present before reading the FIFO.

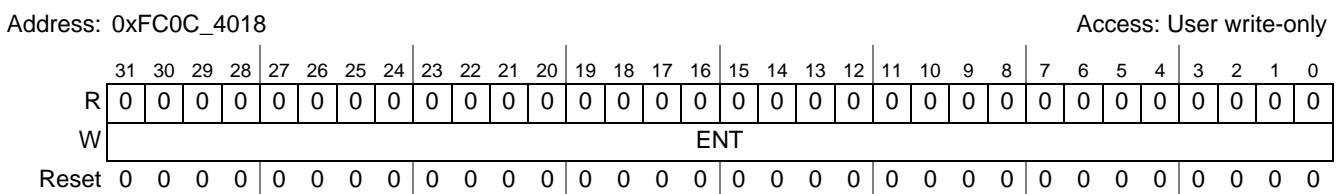
**Figure 27-7. RNG Output FIFO (RNGOUT)**

### 27.3.7 RNG Entropy Register (RNGER)

The RNGER is a write-only register which allows the user to insert entropy into the RNG. When written this register causes an addition of the write data to the XKEY within the RNG. If the system has a means to collect quality entropy (user key strokes or other random patterns), this is the way to add it directly into the accumulated entropy within the RNG. Writing the RNGER does not have a detrimental effect on the quality of random numbers as the number written is added to the state, not used to replace the existing state. Writing all zeros to the entropy register does nothing to the quality of random numbers generated.

#### NOTE

This register can only be written when the RNG is not busy (RNGSR[BUSY] = 0).

**Figure 27-8. RNG Entropy Register (RNGER)**

**Table 27-7. RNGER Field Descriptions**

Field	Description
31–0 ENT	Entropy input. This value is added directly to the internal state of the RNG thus modifying its internal state and affecting future random numbers generated by the RNG.

## 27.4 Functional Description

The RNG performs two functional operations, as described in [Section 27.2, “Modes of Operation”](#): seed generation and random number generation. These operations are performed with cooperation from the two major functional blocks in the RNG described below.

### 27.4.1 Pseudorandom Number Generator (PRNG)

The PRNG implements the NIST-approved PRNG described in the *Digital Signature Standard*. The 160-bit output of the SHA-1 block is the next five words of random data. The PRNG is designed to generate  $2^{20}$  words of random data before requiring reseeding, using the TRNG only during the seeding/initialization process. The initial seed takes approximately two million clock cycles. After this the RNG can generate five 32-bit words every 112 clock cycles. Reseeding takes place transparently through use of the simultaneous reseed LFSRs. The entropy stored in this 128-bit LFSR and 128-bit shift register is added directly into the XKEY structure via the RNG XSEED generator whenever reseeding is required.

### 27.4.2 True Random Number Generator (TRNG)

The TRNG is comprised of two entropy sources each providing a single bit of output. Concatenated together, these two output bits are expected to provide one bit of entropy every 100 clock cycles. In addition to generating entropy, the TRNG also performs several statistical tests on its output. The pass/fail status of these tests are reflected in RNGESR.

### 27.4.3 RNG Interrupts

There is a single RNG interrupt generated to the processor’s interrupt controller. The source of the interrupt is determined by reading the RNG status register. If an error is the cause of the interrupt, further information is available by reading the RNG error status register. The available sources are described in the following table.

**Table 27-8. RNG Interrupt Sources**

Sources	Status Bit Field	Mask Bit Field	Description
Error	RNGSR[ERR]	RNGCR[MERR]	Error detected. See RNGESR for details.
LFSR	RNGESR[LSFRE]	RNGCR[MERR]	Fault in one of the TRNG’s LFSRs
Oscillator	RNGESR[OSCE]	RNGCR[MERR]	TRNG ring oscillator may be malfunctioning
Self test	RNGESR[STE]	RNGCR[MERR]	Self test failed

**Table 27-8. RNG Interrupt Sources (continued)**

Sources	Status Bit Field	Mask Bit Field	Description
Statistical test	RNGESR[SATE]	RNGCR[MERR]	Statistics test for last seed generation failed
FIFO Underflow	RNGESR[FUF]	RNGCR[MERR]	FIFO read while empty
Seed generation done	RNGSR[SDN]	RNGCR[MSKDN]	First seed was generated
Self test done	RNGSR[STDN]	RNGCR[MSKDN]	Self test finished

## 27.5 Initialization/Application Information

### 27.5.1 Manual Seeding

The intended general operation of the RNG is as follows:

1. Reset/initialize.
2. Write to the RNG control register to setup the RNG for the desired functionality.
3. Write to RNGCMD to run self-test or seed generation.
4. Wait for interrupt to indicate completion of the requested operation(s).
5. Repeat steps 3–4 if seed generation is not complete.
6. Poll RNGSR for FIFO level.
7. Read available random data from output FIFO.
8. Repeat steps 6 and 7 as needed, until  $2^{20}$  words have been generated.
9. Write to RNGCMD to run seed mode.
10. Repeat steps 4–9.

### 27.5.2 Automatic Seeding

The intended general operation of the RNG with automatic seeding enabled is as follows:

1. Reset/initialize.
2. Write to the RNG control register to setup the RNG for automatic seeding and the desired functionality.
3. Wait for interrupt to indicate completion of first seed
4. Poll RNGSR for FIFO level.
5. Read available random data from output FIFO.
6. Repeat steps 4 and 5 as needed. Automatic seeding occurs when necessary and is transparent to operation.





# Chapter 28

## Subscriber Identification Module (SIM)

### 28.1 Introduction

The subscriber identification module (SIM) facilitates communication to SIM cards or Eurochip pre-paid phone cards. The SIM module has two ports that can be used to interface with the various cards.

#### 28.1.1 Block Diagram

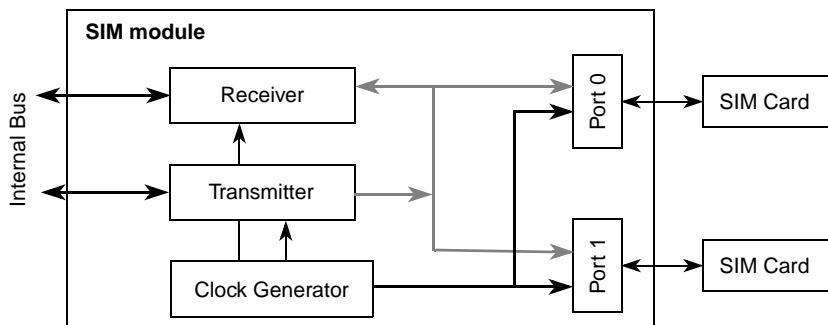


Figure 28-1. SIM Block Diagram

#### 28.1.2 Features

The SIM contains the following features:

- Programmable clock divisor for SIM card clock generation
- Transmitter block with a transmit state machine, transmit shift register, and transmit FIFO
  - 16-byte transmit FIFO
  - Automatic NACK generation on parity and overrun errors
  - Hardware data format support (inverse convention or direct convention)
  - Retransmission of data upon SIM card NACK request with programmable maximum threshold of retransmissions
  - Programmable guard time between transmitted bytes
- Receiver block with a receive state machine, receive FIFO, and control logic
  - 32-byte receive FIFO
  - Decoding of initial character mode for setting data format
  - Hardware data format support (inverse convention or direct convention)
  - NACK detection

### Subscriber Identification Module (SIM)

- 11 ETU character support
- Character wait time counter
- Supports numerous port control functions necessary for SIM card interaction
  - SIM card presence detect with interrupt capability
  - Deep sleep wake-up via SIM card presence detect interrupt
  - Manual control of all SIM card interface signals
  - Automatic power-down of port logic on SIM card presence detect

### 28.1.3 Modes of Operation

The SIM module I/O interface operates in one mode summarized below.

- Internal one wire interface. In this mode the TX pin is routed to the smart card. The RX signal is connected to the TX pin internal to the processor. The 3VOLT bit in the port control register is set and SIM\_ODCR[ODPn] is set. For this interface to work properly the TX pin must be pulled high by a resistor. The value should be selected small enough to give a fast rise time.

## 28.2 External Signal Description

See [Table 28-1](#) for a summary of the SIM module signals.

**Table 28-1. SIM Signal Descriptions**

Signal	I/O	Description
SIM_CLK[1:0]	O	Clock for the smart card. Typical frequencies are 1–5 MHz. This clock is 372 times the data rate that is on SIM_DATA. There is no required timing relationship between this clock signal and any of the other data signals. This is because of the asynchronous nature of the protocol.
SIM_RST[1:0]	O	Reset signal.
SIM_VEN[1:0]	O	Power supply enable signal.
SIM_DATA[1:0]	I/O	Bidirectional transmit/receive data signal.
SIM_PD[1:0]	I	Card insertion detect signal.

## 28.3 Memory Map/Register Definition

[Table 28-2](#) shows the SIM memory map.

**Table 28-2. SIM Memory Map**

Address	Register	Access	Reset Value	Section/Page
0xFC0A_C000	SIM port 1 control register (SIM_PCR1)	R/W	0x0000_0000	<a href="#">28.3.1/28-4</a>
0xFC0A_C004	SIM setup register (SIM_SETUP)	R/W	0x0000_0000	<a href="#">28.3.2/28-5</a>
0xFC0A_C008	SIM port 1 detect register (SIM_DETECT1)	R/W	0x0000_----	<a href="#">28.3.3/28-6</a>
0xFC0A_C00C	SIM port 1 transmit buffer register (SIM_TBUF1)	R/W	0x0000_0000	<a href="#">28.3.4/28-7</a>
0xFC0A_C010	SIM port 1 receive buffer register (SIM_RBUF1)	R	0x0000_0000	<a href="#">28.3.5/28-7</a>

**Table 28-2. SIM Memory Map (continued)**

Address	Register	Access	Reset Value	Section/Page
0xFC0A_C014	SIM port 0 control register (SIM_PCR0)	R/W	0x0000_0000	<a href="#">28.3.1/28-4</a>
0xFC0A_C018	SIM control register (SIM_CR)	R/W	0x0000_0006	<a href="#">28.3.6/28-8</a>
0xFC0A_C01C	SIM clock prescaler register (SIM_PRE)	R/W	0x0000_0002	<a href="#">28.3.7/28-10</a>
0xFC0A_C020	SIM receive threshold register (SIM_RTHR)	R/W	0x0000_0001	<a href="#">28.3.8/28-11</a>
0xFC0A_C024	SIM enable register (SIM_EN)	R/W	0x0000_0000	<a href="#">28.3.9/28-11</a>
0xFC0A_C028	SIM transmit status register (SIM_TSR)	R/W	0x0000_00B8	<a href="#">28.3.10/28-12</a>
0xFC0A_C02C	SIM receive status register (SIM_RSR)	R/W	0x0000_0040	<a href="#">28.3.11/28-13</a>
0xFC0A_C030	SIM interrupt mask register (SIM_IMR)	R/W	0x0000_1FFF	<a href="#">28.3.12/28-15</a>
0xFC0A_C034	SIM port0 transmit buffer register (SIM_TBUFO)	R/W	0x0000_0000	<a href="#">28.3.4/28-7</a>
0xFC0A_C038	SIM port0 receive buffer register (SIM_RBUFO)	R	0x0000_0000	<a href="#">28.3.4/28-7</a>
0xFC0A_C03C	SIM port0 detect register (SIM_DETECT0)	R/W	0x0000_----	<a href="#">28.3.3/28-6</a>
0xFC0A_C040	SIM data format register (SIM_FORMAT0)	R/W	0x0000_0000	<a href="#">28.3.13/28-17</a>
0xFC0A_C044	SIM transmit threshold register (SIM_TTHR)	R/W	0x0000_0000	<a href="#">28.3.14/28-17</a>
0xFC0A_C048	SIM transmit guard control register (SIM_TGCR)	R/W	0x0000_0000	<a href="#">28.3.15/28-18</a>
0xFC0A_C04C	SIM open drain configuration control register (SIM_ODCR)	R/W	0x0000_0000	<a href="#">28.3.16/28-19</a>
0xFC0A_C050	SIM reset control register (SIM_RCR)	R/W	0x0000_0000	<a href="#">28.3.17/28-19</a>
0xFC0A_C054	SIM character wait time register (SIM_CWTR)	R/W	0x0000_FFFF	<a href="#">28.3.18/28-20</a>
0xFC0A_C058	SIM general purpose counter register (SIM_GPCNT)	R/W	0x0000_FFFF	<a href="#">28.3.19/28-21</a>
0xFC0A_C05C	SIM divisor register (SIM_DIV)	R/W	0x0000_00FF	<a href="#">28.3.20/28-21</a>
0xFC0A_C060	SIM block wait time register (SIM_BWT)	R/W	0x0000_FFFF	<a href="#">28.3.21/28-22</a>
0xFC0A_C064	SIM block guard time register (SIM_BGT)	R/W	0x0000_0000	<a href="#">28.3.22/28-22</a>
0xFC0A_C068	SIM block wait time register high (SIM_BWTH)	R/W	0x0000_FFFF	<a href="#">28.3.23/28-23</a>
0xFC0A_C06C	SIM transmit FIFO status register (SIM_TFSR)	R	0x0000_0000	<a href="#">28.3.24/28-23</a>
0xFC0A_C070	SIM receive FIFO counter register (SIM_RFRCR)	R	0x0000_0000	<a href="#">28.3.25/28-24</a>
0xFC0A_C074	SIM receive FIFO write pointer register (SIM_RFWP)	R	0x0000_0000	<a href="#">28.3.26/28-24</a>
0xFC0A_C078	SIM receive FIFO read pointer register (SIM_RFRP)	R	0x0000_0000	<a href="#">28.3.27/28-24</a>

### 28.3.1 SIM Port Control Registers (SIM\_PCRn)

Address: 0xFC0A\_C000 (SIM\_PCR1)  
0xFC0A\_C014 (SIM\_PCR0)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	3VOLT	SCSP	SCEN	SRST	STEN	SVEN	SAPD
W									SFPD							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-2. SIM Port 0–1 Control Registers (SIM\_PCRn)

Table 28-3. SIM\_PCRn Field Descriptions

Field	Description
31–8	Reserved
7 SFPD	Force auto power-down. Starts the automatic power down sequence for the port. This bit will autoclear. Reading this bit returns a zero. 0 No effect 1 Start auto power-down
6 3VOLT	External one-wire interface. 0 Reserved 1 SIM_DATA $n$ pin is bidirectional <b>Note:</b> Since only one data pin is available on this device, this bit must be set.
5 SCSP	SIM card clock stop polarity. Controls the polarity of the idle SIM clock when the clock is disabled by SCEN. This bit is forced to zero by hardware during the auto power down sequence. This forces the clock to be a logic 0 when stopped by auto power down as required by ISO 7816 spec. 0 Clock is logic 0 when stopped by SCEN 1 Clock is logic 1 when stopped by SCEN
4 SCEN	SIM card clock enable. Enables the clock to the SIM card. It is forced low by hardware during the auto power down sequence. 0 SIM card clock disabled 1 SIM card clock enabled
3 SRST	SIM card reset. Controls state of reset line to the SIM card. It is forced low by hardware during the auto power down sequence. SIM card reset signals are active low. 0 SIM card reset inactive 1 SIM card reset active
2 STEN	SIM card transmit enable. Enables the transmit data to the SIM card. It can be forced low by hardware during the auto power down sequence. 0 Transmit data is forced to zero 1 Transmit data controlled by SIM module

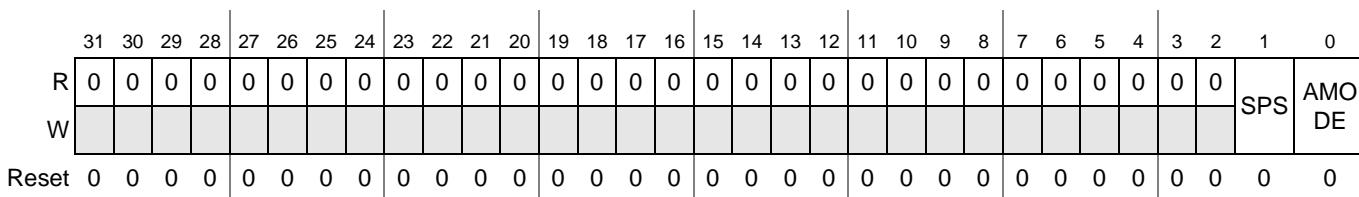
**Table 28-3. SIM\_PCRn Field Descriptions (continued)**

Field	Description
1 SVEN	SIM card Vcc enable. Controls the state of the SVEN pin on SIM card port. The SVEN pin controls the SIM card Vcc enable in the power management chip. It is forced low by hardware during the auto power down sequence. 0 SIM card voltage disabled 1 SIM card voltage enabled
0 SAPD	SIM card auto power down. Enables the auto power down function. It will be forced low at the end of the auto power down sequence. 0 Auto power down disabled 1 Auto power down enabled

### 28.3.2 SIM Port 1 Setup Register (SIM\_SETUP)

Address: 0xFC0A\_C004 (SIM\_SETUP)

Access: User read/write

**Figure 28-3. SIM Setup Register (SIM\_SETUP)****Table 28-4. SIM\_SETUP Field Descriptions**

Field	Description
31–2	Reserved, must be cleared.
1 SPS	SIM card port select. Controls which port the SIM interface uses. <b>Note:</b> The AMODE bit must be cleared when the SPS bit is set. 0 Port 0 enabled 1 Port 1 enabled
0 AMODE	Alternate SIM card mode enable. Enables an alternate SIM module to control this port. <b>Note:</b> The SPS bit must be cleared to give the alternate SIM module control. 0 Alternate port disabled 1 Alternate port enabled

### 28.3.3 SIM Port Detect Registers (SIM\_DETECTn)

Address: 0xFC0A\_C008 (SIM\_DETECT1)  
0xFC0A\_C03C (SIM\_DETECT0)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	0	0	0	SPDS	SPDP	SDI	SDIM
W															w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	—	0	1

Figure 28-4. SIM Port 1 Detect Register (SIM\_DETECTn)

Table 28-5. SIM\_DETECTn Field Descriptions

Field	Description
31–4	Reserved
3 SPDS	SIM presence detect select. Controls which edge of the SIM_PD pin is used to detect the presence of the SIM card. 0 Falling edge of SIM_PD input 1 Rising edge of SIM_PD input
2 SPDP	SIM_PD input pin status. Reflects the state of the SIM_PD pin. This bit is not a latched register bit, but instead a synchronized version of the state of the SIM_PD pin itself. 0 SIMPD pin is logic low 1 SIMPD pin is logic high
1 SDI	SIM detect interrupt flag. Indicates an insertion or removal of a SIM card has been detected. This bit may generate an interrupt if SDIM is cleared. Write a one to this bit to clear it. 0 No insertion or removal of SIM card detected 1 Insertion or removal of SIM card detected
0 SDIM	SIM detect interrupt mask. Interrupt mask for the SDI interrupt flag. 0 SDI enabled 1 SDI masked
<b>Summary:</b>	
SPDS determines which edge transition of the SIM_PD pin is used for SIM card presence detection. Presence detection determines if the card has been inserted or removed. The occurrence of the SIMPD1 edge specified by SPDS will cause the following: SDI to be set; if the SDIM mask is clear, an interrupt on SIMIRQ_N; and if SIM_PCRn[SAPD] is set, an auto power down sequence to begin. If SIM card insertion is expected, SAPD can be cleared to avoid the auto power down sequence. There is no auto power up sequence. The bit SPDP can be used to determine the current state of the SIM_PD pin.	

### 28.3.4 SIM Port Transmit Buffer Registers (SIM\_TBUF $n$ )

Address: 0xFC0A\_C00C (SIM\_TBUF1)  
0xFC0A\_C034 (SIM\_TBUF0)

Access: User 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	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	TXBUF					
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		

Figure 28-5. SIM Port Transmit Buffer Registers (SIM\_TBUF $n$ )

Table 28-6. SIM\_TBUF $n$  Field Descriptions

Field	Description																
31–8	Reserved, must be cleared.																
7–0 TXBUF	Port transmit buffer. Write to the next available location in the transmit buffer. Writes to this register are ignored depending on SIM_SETUP[SPS].																
	<table border="1"> <thead> <tr> <th>Transmit Buffer</th> <th>SIM_SETUP[SPS]</th> <th>Writes Ignored?</th> </tr> </thead> <tbody> <tr> <td>SIM_TFBUF0[TXBUF]</td> <td>0</td> <td>No</td> </tr> <tr> <td></td> <td>1</td> <td>Yes</td> </tr> <tr> <td>SIM_TFBUF1[TXBUF]</td> <td>0</td> <td>Yes</td> </tr> <tr> <td></td> <td>1</td> <td>No</td> </tr> </tbody> </table>	Transmit Buffer	SIM_SETUP[SPS]	Writes Ignored?	SIM_TFBUF0[TXBUF]	0	No		1	Yes	SIM_TFBUF1[TXBUF]	0	Yes		1	No	
Transmit Buffer	SIM_SETUP[SPS]	Writes Ignored?															
SIM_TFBUF0[TXBUF]	0	No															
	1	Yes															
SIM_TFBUF1[TXBUF]	0	Yes															
	1	No															

### 28.3.5 SIM Port Receive Buffer Registers (SIM\_RBUF $n$ )

Address: 0xFC0A\_C010 (SIM\_RBUF1)  
0xFC0A\_C038 (SIM\_RBUF0)

Access: User read-only

	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	CWT	FE	PE	RXBUF								
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			

Figure 28-6. SIM Port Receive Buffer Registers (SIM\_RBUF $n$ )

Table 28-7. SIM\_RBUF $n$  Field Descriptions

Field	Description
31–11	Reserved
10 CWT	Port CWT flag. Indicates that the current byte was late. It is not necessary to clear the byte since it is overwritten by the next byte received at that location of the FIFO. 0 Byte was on time 1 Byte was late

**Table 28-7. SIM\_RBUF<sub>n</sub> Field Descriptions (continued)**

Field	Description
9 FE	Port frame error flag. Indicates a frame error was detected during reception of the current byte read from the port's receiver. This bit cannot generate an interrupt. It is not necessary to clear the byte since it is overwritten by the next byte received at that location of the FIFO. 0 Byte contains no framing error 1 Byte contains a framing error
8 PE	Port parity error flag. Indicates a parity error was detected during reception of the current byte read from the port's receiver. This bit cannot create an interrupt. It is not necessary to clear the byte since it is overwritten by the next byte received at that location of the FIFO. A parity error can create a NACK pulse. 0 Byte contains no parity error (default) 1 Byte contains a parity error
7–0 RXBUF	Port receive buffer. Read from the next location in the receive buffer. Reads from this register return zero when SPS is cleared.

### 28.3.6 SIM Control Register (SIM\_CR)

Address: 0xFC0A\_C018 (SIM\_CR)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	BWT EN	XMT_ CRC_ LRC	CRC EN	LRCE N	CWT EN	GPCNT_ CLKSEL	BAUD_SEL				0	SAM PLE 12	ONA CK	ANAC K	ICM	0
W					0	0	0	0	0	0	0		0	1	1	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

**Figure 28-7. SIM Control Register (SIM\_CR)****Table 28-8. SIM\_CR Field Descriptions**

Field	Description
31–16	Reserved
15 BWTEN	Block wait time enable. Enables the BWT and BGT functions. These functions can then be individually selected using the interrupt mask. 0 Disable BWT and BGT 1 Enable BWT and BGT
14 XMT_CRC_L RC	Transmit CRC or LRC. Specifies whether to transmit the redundancy checking data at the end of a transmission (when the FIFO becomes empty). 0 No redundancy check info transmitted 1 Transmit LRC or CRC info when FIFO empties

**Table 28-8. SIM\_CR Field Descriptions (continued)**

Field	Description
13 CRCEN	CRC Enable. Enables the calculation of the 16-bit CRC value for both receiver and transmitter. The result of the calculation is continuously compared to the expected remainder and reflected in SIM_RSR[CRCOK] register. Clearing this bit resets the current CRC residual value in the SIM hardware. 0 16-bit CRC disabled 1 16-bit CRC enabled
12 LRCEN	LRC Enable. Enables the calculation of the 8-bit LRC value for both receiver and transmitter. The result of the calculation is continuously compared to zero and reflected in SIM_RSR[LRCOK]. Clearing this bit resets the current LRC value in the SIM hardware. 0 8-bit linear redundancy checking disabled 1 8-bit linear redundancy checking enabled
11 CWTEN	Character wait time counter enable. Enables the character wait time counter. Clearing this bit resets the counter to zero. 0 Character wait time counter off 1 Character wait time counter on
10–9 GPCNT_ CLKSEL	General purpose counter clock select. Selects which clock source is used by SIM module general purpose counter. The only way to reset the counter is to clear this field. The counter begins counting as soon as the clock input is selected and the clocks are enabled. These input clocks are enabled through other register bits of the SIM module (KILL_CLOCK, RXEN, and TXEN respectively). 00 Disabled/reset 01 Card clock 10 Receive clock 11 ETU clock (transmit clock)
8–6 BAUD_SEL	SIM baud rate select. Selects the asynchronous baud rate divisor of the clock When set to 111, the divisor is set to the value programmed in the DIVISOR register. This allows for more flexible baud rate determination. 000 31 (372/1 Fi/Di) 001 32 (512/2 Fi/Di) 010 16 (512/4 Fi/Di) 011 8 (512/8 Fi/Di) 100 4 (512/16 Fi/Di) 101 2 (512/32 Fi/Di) 110 1 (512/64 Fi/Di) 111 SIM_DIV
5	Reserved
4 SAMPLE12	Sample12. Set the third stage divider. Sets the corresponding sample rate which is the number of times a bit being received is sampled. 0 Divide by 8 1 Divide by 12
3 ONACK	Overrun NACK Enable. 0 NACK generation on overrun is disabled 1 NACK generation on overrun is enabled
2 ANACK	Automatic NACK Enable. Enables NACK generation for parity errors or invalid initial characters when ICM is set. 0 NACK generation on errors disabled 1 NACK generation on errors enabled

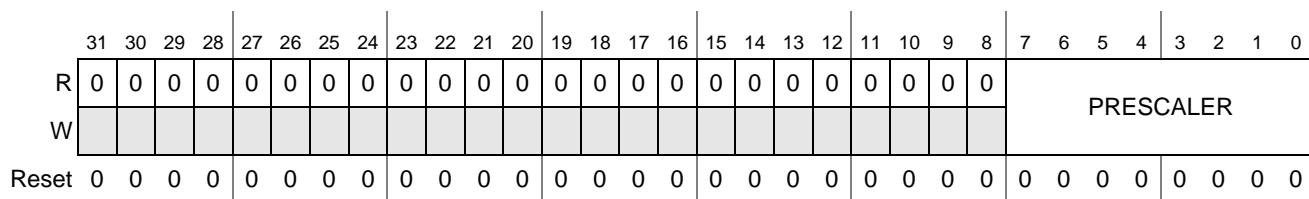
**Table 28-8. SIM\_CR Field Descriptions (continued)**

Field	Description
1 ICM	Initial character mode. Enables initial character mode. This bit is automatically cleared by hardware after a valid initial character is received. This bit is set following reset. 0 Initial character mode disabled 1 Initial character mode enabled
0	Reserved

### 28.3.7 SIM Clock Prescaler Register (SIM\_PRE)

Address: 0xFC0A\_C01C (SIM\_PRE)

Access: User read/write

**Figure 28-8. SIM Clock Prescaler Register (SIM\_PRE)****Table 28-9. SIM\_PRE Field Descriptions**

Field	Description
31–8	Reserved
7–0 PRESCALER	Clock prescaler divisor register. The value written to this register will determine the first stage divider setting. If the ipg_clk is 66Mhz, a typical setting would be 0x0e. This would set the SIM card clock to 66Mhz/14 = 4.7Mhz. The duty cycle of divided clock will be between 45% and 55% according to ISO7816 requirement. For the odd divider factor (2K+1), the duty cycle will be K/(2K+1) and (K+1)/(2k+1). So for 0x03, the duty cycle is 33%-66%. For 0x05, the duty cycle is 40%-60%. For 0x07, the duty cycle is 43%-57%, and for 0x09, the duty cycle is 44%-56%. For all other values, the clock duty cycle can meet the ISO7816 requirement. 0x00~0x02 ipg_clk / 2 0x03 ipg_clk / 3 0x04 ipg_clk / 4 0x05 ipg_clk / 5 0x06 ipg_clk / 6 ... 0xFD ipg_clk / 253 0xFE ipg_clk / 254 0xFF ipg_clk / 255

### 28.3.8 SIM Receive Threshold Register (SIM\_RTHR)

Address: 0xFC0A\_C020 (SIM\_RTHR)

Access: User 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	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																																
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		

Figure 28-9. SIM Receive Threshold Register (SIM\_RTHR)

Table 28-10. SIM\_RTHR Field Descriptions

Field	Description
31–13	Reserved
8–5 RTH	Receive Nack threshold. Specifies the number of consecutive NACK's transmitted by the SIM module, for a given character, before the receive threshold error (RTE) flag is triggered. A value of 0 indicates that RTE is never set. When a valid character is received by the SIM, the internal counter keeping track of the NACK count resets to zero for the subsequent byte being received. If SIM_CR[ANACK] is cleared, RTH has no effect.
4–0 RDT	Receive data threshold. Determines the number of unread bytes that must exist in the FIFO to trigger the receive data register full (RDRF) interrupt flag. If the number of unread bytes in the receive FIFO is greater than or equal to the value in RDT, the SIM_RSR[RDRF] flag is set. A value of zero indicates that there must be 285 unread bytes in the FIFO to trigger RDRF. The RDT value can be altered at any time, and the RDRF flag will be updated accordingly.

### 28.3.9 SIM Enable Register (SIM\_EN)

Address: 0xFC0A\_C024 (SIM\_EN)

Access: User 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	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																																
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		

Figure 28-10. SIM Enable Register (SIM\_EN)

Table 28-11. SIM\_EN Field Descriptions

Field	Description
31–2	Reserved

**Table 28-11. SIM\_EN Field Descriptions (continued)**

Field	Description
1 TXEN	SIM transmit enable. Enables the SIM transmit state machine. When the SIM is being used to receive data, TXEN should be cleared. This bit also enables the internal transmit and receive clocks to the general purpose counter. <b>Note:</b> Setting this bit (transition from 0 to 1) will reset the CRC and LRC values. 0 SIM transmitter disabled 1 SIM transmitter enabled
0 RXEN	SIM receiver enable. Enables the SIM receive state machine. RXEN must be set whenever the SIM module is in use. The SIM module has an automatic receive mode operation that disables the reception of characters when the transmitter is operational. Once the transmitter has completed sending the last character, the receiver is automatically enabled. This bit also enables the RCV_CLK input to the general purpose counter. 0 SIM receiver disabled 1 SIM receiver enabled

### 28.3.10 SIM Transmit Status Register (SIM\_TSR)

Address: 0xFC0A\_C028 (SIM\_TSR)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	GP CNT	TDTF	TFO	TC	ETC	TFE	0	0	XTE
W								w1c	w1c		w1c	w1c	w1c			w1c
Reset	0	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0

**Figure 28-11. SIM Transmit Status Register (SIM\_TSR)****Table 28-12. SIM\_TSR Field Descriptions**

Field	Description
31–9	Reserved, must be cleared.
8 GPCNT	General purpose counter flag. Indicates when the general purpose counter has reached the value in the GPCNT register. 0 GPCNT time not reached 1 General purpose counter has reached the GPCNT value
7 TDTF	Transmit data threshold flag. This flag is set when the number of bytes in the FIFO goes above the value programmed in SIM_TTHR[TDT]. If the data level never goes above the threshold, then this flag doesn't set. 0 Number of bytes in FIFO is greater than TDT 1 Number of bytes in FIFO is less than or equal to TDT

**Table 28-12. SIM\_TSR Field Descriptions (continued)**

Field	Description
6 TFO	Transmit FIFO overflow error. Indicates when the Transmit FIFO has been written with more than 16 bytes. This bit is only cleared by setting either SIM_RCR[FLUSH_XMT, SOFT_RESET]. 0 No transmit FIFO overflow error occurred 1 A Transmit FIFO overflow error occurred
5 TC	Transmit complete. Indicates the SIM transmitter is ready for a new transmission. This bit is set after the guard time has expired for the last byte in the transmit FIFO. TC generates an interrupt if SIM_IMR[TCIM] is cleared. Write one to TC to clear it. 0 Transmit pending or in progress 1 Transmit complete
4 ETC	Early transmit complete. Indicates the SIM transmitter has finished sending the current byte and the transmit FIFO is empty. This bit differs from TC in that it is set before the guard time of the last byte has elapsed. ETC generates an interrupt if SIM_IMR[ETCCIM] is cleared. Write one to ETC to clear it. 0 Transmit pending or in progress 1 Transmit complete
3 TFE	Transmit FIFO empty. Indicates the transmit FIFO is empty. This bit is set when the last byte in the transmit FIFO has been transferred to the SIM transmit shift register. TFE generates an interrupt if SIM_IMR[TFEIM] is cleared. Write one to TFE to clear it. 0 Transmit FIFO is not empty 1 Transmit FIFO is empty
2–1	Reserved, must be cleared
0 XTE	Transmit NACK threshold error. Indicates the transmit NACK threshold has been reached. When XTE is set, no further transmissions are done until XTE is cleared. Any data transmissions pending in the transmit FIFO are aborted, and TC, ETC, and TFE are set. XTE generates an interrupt if SIM_IMR[XTM] is cleared. Write one to XTE to clear it. 0 Transmit NACK threshold has not been reached 1 Transmit NACK threshold reached; transmitter frozen

### 28.3.11 SIM Receive Status Register (SIM\_RSR)

0xFC0A_C02C (SIM_RSR)																Access: User read/write			
31	30	29	28		27	26	25	24		23	22	21	20		19	18	17	16	
R	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	
15	14	13	12		11	10	9	8		7	6	5	4		3	2	1	0	
R	0	0	0	0	BGT	BWT	RTE	CWT	CRC OK	LRC OK	RDRF	RFD		0	0	0	OEF		
W					w1c	w1c	w1c	w1c										w1c	
Reset	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	

**Figure 28-12. SIM Receive Status Register (SIM\_RSR)**

Table 28-13. SIM\_RSR Field Descriptions

Field	Description
31–12	Reserved, must be cleared.
11 BGT	Block guard time error flag. Indicates the block guard time was too small. The threshold is set by the block guard time register (SIM_BGT). 0 Block guard time was sufficient 1 Block guard time was too small
10 BWT	Block wait time error flag. Indicates the block wait time has been exceeded. The threshold is set by the block wait time registers (SIM_BWT, SIM_BWTH). 0 Block wait time not exceeded 1 Block wait time was exceeded
9 RTE	Receive NACK threshold error flag. Indicates if the number of consecutive NACK's generated by the SIM module in response to receive parity errors for the byte being received, equals the value programmed in SIM_RTHR[RTH]. This bit never sets unless SIM_CR[ANACK] is set. SIM_PCRn[SAPD] must be set to enable the threshold error to trigger the auto power down sequence. Write one to RTE to clear it. Clearing this bit resets the internal counter for consecutive NACK's being transmitted for a given byte. 0 Number of NACKs generated by the receiver is less than the value programmed in SIM_RTHR[RTH] 1 Number of NACKs generated by the receiver is equal to the value programmed in SIM_RTHR[RTH]
8 CWT	Character wait time counter flag. Indicates when the time between received characters is equal to or greater than the value programmed in the SIM_CWTR register. 0 No CWT violation has occurred 1 Time between two consecutive characters exceeded the value in SIM_CWTR
7 CRCOK	Cyclic redundancy check okay flag. Indicates when the calculated 16-bit CRC value matches the expected value for the current input data stream. The value is calculated across all received characters from the point SIM_CR[CRCEN] is set. The current CRC residual is reset by three mechanisms: <ul style="list-style-type: none"><li>• Clear SIM_CR[CRCEN]</li><li>• Set SIM_EN[TXEN]</li><li>• Automatically by hardware when ETC flag is set at the end of a transmission.</li></ul> 0 Current CRC value does not match remainder 1 Current calculated CRC value matches the expected result
6 LRCOK	Linear redundancy check okay flag. Indicates when the calculated 8-bit LRC value is zero value for the current input data stream. The value is calculated across all received characters from the point SIM_CR[LRCEN] is set. The current LRC residual is reset by three mechanisms: <ul style="list-style-type: none"><li>• Clear SIM_CR[LRCEN]</li><li>• Set SIM_EN[TXEN]</li><li>• Automatically by hardware when ETC flag is set at the end of a transmission.</li></ul> 0 Current LRC value does not match remainder 1 Current calculated LRC value matches the expected result of zero
5 RDRF	Receive data register full. Indicates the SIM receive FIFO has reached the threshold level set by SIM_RTHR[RDT]. RDRF is set any time the number of unread bytes in the receive FIFO is equal to or greater than the value set by RDT. The flag is cleared by reading enough bytes out of the receive FIFO to bring the number of bytes left in the FIFO below RDT level. Another way to clear the flag is to set RDT higher than the number of unread bytes currently in the FIFO. RDRF generates an interrupt if SIM_CR[RIM] is cleared. 0 Number of unread bytes in receive buffer is less than value set by RDT 1 Number of unread bytes in receive buffer is greater than or equal to the value set by RDT

**Table 28-13. SIM\_RSR Field Descriptions (continued)**

Field	Description
4 RFD	Receive FIFO has unread data. Indicates there is at least one unread byte in the receive data FIFO. This bit is only cleared by reading all bytes out of the receive FIFO. RFD cannot be used to generate an interrupt. Normally, the SIM triggers the interrupt with RDRF and software uses RFD to read all of the bytes out of the receive FIFO. 0 There are no unread bytes in the receive FIFO 1 There is at least one unread byte in the receive FIFO
3–1	Reserved, must be cleared.
0 OEF	Overrun error flag. Indicates the SIM was unable to store received data due to already having 285 unread bytes in the FIFO. It does not necessarily indicate that data has been lost. If SIM_CR[ONACK] is set, a NACK pulse is generated on bytes that would otherwise cause a loss of data due to a full FIFO. These bytes should be retransmitted by the SIM card which implies that no data has actually been lost. In this case, the OEF flag is just an indicator that this situation has occurred which may be helpful in system debug. If ONACK is not set, a set OEF flag indicates a loss of data since all bytes received with the OEF flag set will indeed be lost (including the byte that caused the bit to set). The OEF flag generates an interrupt if SIM_IMR[OIM] is cleared. The OEF flag is a write-one-to-clear bit. 0 No overrun error has occurred 1 A byte was received when the received FIFO was already full

### 28.3.12 SIM Interrupt Mask Register (SIM\_IMR)

Address: 0xFC0A\_C030 (INT\_MASK)

Access: User read/write

R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	BGT M	BWT M	RTM	CWT M	GPC NTM	TDTF M	TFO M	XTM	TFEI M	ETCI M	OIM	TCIM	RIM
W																
Reset	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1

**Figure 28-13. SIM Interrupt Mask Register (SIM\_IMR)****Table 28-14. SIM\_IMR Field Descriptions**

Field	Description
31–13	Reserved, must be cleared.
12 BGTM	Block guard time interrupt mask. Enables the interrupt mask for SIM_RSR[BGT]. 0 Enabled 1 Masked

**Table 28-14. SIM\_IMR Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>
11 BWTM	Block wait time interrupt mask. Enables the interrupt mask for SIM_RSR[BWT]. 0 Enabled 1 Masked
10 RTM	Receive NACK threshold interrupt mask. Enables the interrupt mask for SIM_RSR[RTE]. 0 Enabled 1 Masked
9 CWTM	Character wait time interrupt mask. Enables the interrupt mask for SIM_RSR[CWT]. 0 Enabled 1 Masked
8 GPCNTM	General purpose counter interrupt mask. Enables the interrupt mask for SIM_TSR[GPCNT]. 0 Enabled 1 Masked
7 TDTFM	Transmit data threshold interrupt mask. Enables the interrupt mask for SIM_TSR[TDTF]. 0 Enabled 1 Masked
6 TFOM	Transmit FIFO overflow error interrupt mask. Enables the interrupt mask for SIM_TSR[TFO]. 0 Enabled 1 Masked
5 XTM	Transmit NACK threshold interrupt mask. Enables the interrupt mask for SIM_TSR[XTE]. 0 Enabled 1 Masked
4 TFEIM	Transmit FIFO empty interrupt mask. Enables the interrupt mask for SIM_TSR[TFE]. 0 Enabled 1 Masked
3 ETCIM	Early transmit complete interrupt mask. Enables the interrupt mask for SIM_TSR[ETC]. 0 Enabled 1 Masked
2 OIM	Overrun interrupt mask. Enables the interrupt mask for SIM_RSR[OEF]. 0 Enabled 1 Masked
1 TCIM	Transmit complete interrupt mask. Enables the interrupt mask for SIM_TSR[TC]. 0 Enabled 1 Masked
0 RIM	Receive Interrupt Mask. Enables the interrupt mask for SIM_RSR[RDRF]. 0 Enabled 1 Masked

### 28.3.13 SIM Data Format Register (SIM\_FORMAT)

Address: 0xFC0A\_C040 (SIM\_FORMAT)

Access: User 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	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	IC	
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	

Figure 28-14. SIM Data Format Register (SIM\_FORMAT)

Table 28-15. SIM\_FORMAT Field Descriptions

Field	Description	
31–1	Reserved, must be cleared	
0 IC	Inverse convention. Configures the SIM to use inverse or direct convention for its data format. This bit can be controlled by software, but it is normally set by hardware as a result of the interpretation of the initial character when in ICM mode. 0 Direction convention transfers enabled (default). 1 Inverse convention transfers enabled.	

### 28.3.14 SIM Transmit Threshold Register (SIM\_TTHR)

Address: 0xFC0A\_C044 (SIM\_TTHR)

Access: User 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	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	XTH	TDT					
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	

Figure 28-15. SIM Transmit Threshold Register (SIM\_TTHR)

Table 28-16. SIM\_TTHR Field Descriptions

Field	Description	
31–8	Reserved, must be cleared.	

**Table 28-16. SIM\_TTHR Field Descriptions (continued)**

Field	Description
7–4 XTH	Transmit NACK threshold. When this threshold is reached for the current byte being transmitted, SIM_TSR[XTE] is set. This causes the remaining transmissions queued in the transmit FIFO to be aborted and no more transmissions occur until software clears XTE. To trigger XTE, a given byte being transmitted must reach the XTH threshold itself. Transmit NACKs accumulated on one byte are not carried over to the next. 0x0 XTE is never set; retransmission after NACK reception is disabled 0x1 XTE is set after 1 NACK is received; 0 retransmissions occur 0x2 XTE is set after 2 NACKs are received; at most 1 retransmission occurs 0x3 XTE is set after 3 NACKs are received; at most 2 retransmissions occur ... 0xF XTE is set after 15 NACKs are received; at most 14 retransmissions occur
3–0 TDT	Transmit data threshold. When the number of bytes in the transmit FIFO is less than or equal to TDT, SIM_TSR[TDTF] is set.

### 28.3.15 SIM Transmit Guard Control Register (SIM\_TGCR)

Address: 0xFC0A\_C048 (SIM\_TGCR)

Access: User 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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RCVR	11	GETU						
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	

**Figure 28-16. SIM Transmit Guard Control Register (SIM\_TGCR)****Table 28-17. SIM\_TGCR Field Descriptions**

Field	Description
31–9	Reserved, must be cleared.
8 RCVR11	Receiver 11 ETUs. Configure the SIM receiver for 11 ETU operation (1 stop bit). This bit is provided for the T=1 protocol. 0 Receiver configured for 12 ETU operation 1 Receiver configured for 11 ETU operation
7–0 GETU	Transmit guard ETUs. Controls the number of additional elementary time units (ETUs) inserted between bytes transmitted by the SIM. An ETU is equivalent to one bit time at the given baud rate (for example, the length of a start bit). The guard time has no effect on the SIM receiver. 0x00 No additional ETUs inserted ... 0xFE 254 additional ETUs inserted 0xFF One ETU removed (Stop bits reduced from two to one)

### 28.3.16 SIM Open Drain Configuration Control Register (SIM\_ODCR)

Address: 0xFC0A\_C04C (SIM\_ODCR)

Access: User 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	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	OD	OD	P1	P0	
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	

Figure 28-17. SIM Open Drain Configuration Control Register (SIM\_ODCR)

Table 28-18. SIM\_ODCR Field Descriptions

Field	Description
31–2	Reserved
1 ODP1	Open drain control for port 1. Since the data pin is bi-directional, this bit must be set. 0 Reserved 1 SIM_DATA1 is open-drain <b>Note:</b> This bit resets to 0 which is a reserved setting. You must set this bit for proper operation.
0 ODP0	Open drain control for port 0. Since the data pin is bi-directional, this bit must be set. 0 Reserved 1 SIM_DATA0 is open-drain <b>Note:</b> This bit resets to 0 which is a reserved setting. You must set this bit for proper operation.

### 28.3.17 SIM Reset Control Register (SIM\_RCR)

Address: 0xFC0A\_C050 (SIM\_RCR)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	DBUG	STOP	DOZE	KILL	0	FLUSH	FLUSH
W													SOFT	XMT		RCV
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 28-18. SIM Reset Control Register (SIM\_RCR)

**Table 28-19. SIM\_RCR Field Descriptions**

Field	Description
31–7	Reserved, must be cleared.
6 DBUG	Debug. Configures the SIM when a debug event occurs. When set, a debug event causes the receive FIFO read pointer to freeze. 0 Debug event has no affect on SIM module 1 Debug event prohibits read pointer changes for receive FIFO
5 STOP	Stop. Configures the SIM when a processor stop instruction is executed. This bit provides support for SIM cards that do not allow the SIM card clock to be stopped while power is applied. 0 Stop instruction shuts down all SIM clocks 1 Stop instruction shuts down all clocks except for the BAUD_CLK (clock provided to SIM Card)
4 DOZE	Doze. Configures the SIM module when a processor DOZE instruction is executed. 0 DOZE instruction has no effect on SIM module 1 DOZE instruction will cause SIM module to gate SIM clocks when the transmit FIFO is empty
3 KILL_CLOCK	Kill SIM Clock. Disables the SIM clock input to the SIM module. This bit gates all SIM clocks including the SIM card clock regardless of the state of the STOP bit. 0 SIM input clock enabled 1 SIM input clock disabled
2 SOFT_RST	Software reset. Resets the entire SIM module. This acts the same as a hardware reset for the SIM module. This bit is self-clearing. <b>Note:</b> Software should allow a minimum of 4 reference clock cycles (CKIH) before attempting to access the SIM module after a software reset. 0 SIM normal operation 1 SIM held in reset
1 FLUSH_XMT	Flush transmitter. Resets the SIM transmitter. The receive portion of the SIM module is not affected. The software must clear this bit before the SIM transmitter can operate. 0 SIM transmitter normal operation 1 SIM transmitter held in reset
0 FLUSH_RCV	Flush Receiver. Resets the SIM receiver. The transmit portion of the SIM module is not affected. The software must clear this bit before the SIM receiver can operate. 0 SIM receiver normal operation 1 SIM receiver held in reset

### 28.3.18 SIM Character Wait Time Register (SIM\_CWTR)

Address: 0xFC0A\_C054 (SIM\_CWTR)

Access: User 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	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	CWT	
W																															
Reset	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	1	1	

**Figure 28-19. SIM Character Wait Time Register (SIM\_CWTR)**

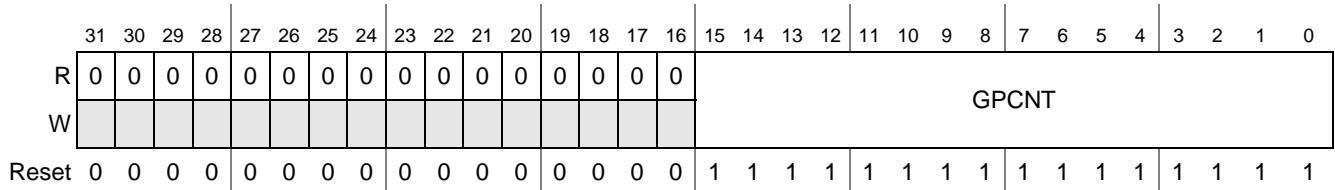
**Table 28-20. SIM\_CWTR Field Descriptions**

Field	Description
31–16	Reserved, must be cleared.
15–0 CWT	Character wait time. Specifies the number of ETU times allowed between characters. Default is 0xFFFF

### 28.3.19 SIM General Purpose Counter Register (SIM\_GPCNT)

Address: 0xFC0A\_C058 (SIM\_GPCNT)

Access: User read/write



**Figure 28-20. SIM General Purpose Counter Register (SIM\_GPCNT)**

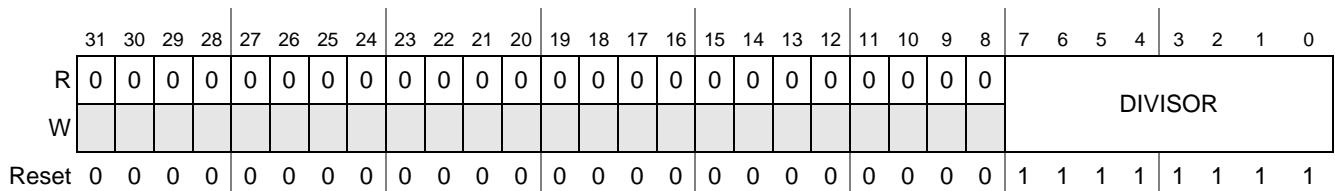
**Table 28-21. SIM\_GPCNT Field Descriptions**

Field	Description
31–16	Reserved, must be cleared.
15–0 GPCNT	General purpose counter. The value written to this field is compared to the general purpose counter in the SIM module. Once the general purpose counter reaches this value, SIM_TSR[GPCNT] is set. This counter is intended to be used for any events that must be monitored for duration based on the card clock, receiver sample rate, or ETU rate (transmit clock). Example: ATR arrival time and ATR duration. Default is 0xFFFF

### **28.3.20 SIM Divisor Register (SIM\_DIV)**

Address: 0xFC0A\_C05C (SIM\_DIV)

Access: User read/write



**Figure 28-21. SIM Divisor Register (SIM DIV)**

**Table 28-22. SIM DIV Field Descriptions**

Field	Description
31–8	Reserved
7–0 DIVISOR	Divisor register. The value written to this register generates the SIM receive clock. SIM_CR[BAUD_SEL] must be set to 111 to control the divisor value using the DIVISOR field. Default is 0xFF. <b>Note:</b> A value of 0 is reserved.

### 28.3.21 SIM Block Wait Time Low Register (SIM\_BWTL)

Address: 0xFC0A\_C060 (SIM\_BWTL)

Access: User 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	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																																
Reset	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	1	1	1		

Figure 28-22. SIM Block Wait Time Low Register (SIM\_BWTL)

Table 28-23. SIM\_BWTL Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 BWT	Lower block wait time. This value is the lower 16 bits of the block wait time. The time from the start bit of the last byte sent from the SIM to the start bit of the first byte sent from the smart card must be less than the 32-bit value formed SIM_BWTH and SIM_BWTL. If not, SIM_RSR[BWT] is set. Default is 0xFFFF

### 28.3.22 SIM Block Guard Time Register (SIM\_BGT)

Address: 0xFC0A\_C064 (SIM\_BGT)

Access: User 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	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																																
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		

Figure 28-23. SIM Block Guard Time Register (SIM\_BGT)

Table 28-24. SIM\_BGT Field Descriptions

Field	Description
31–16	Reserved, must be cleared
15–0 BGT	Block guard time. The time from the start bit of the last byte sent from the SIM to the start bit of the first byte sent from the smart card must be greater than this value. If not, SIM_RSR[BGT] is set. Default is 0x0000

### 28.3.23 SIM Block Wait Time High Register (SIM\_BWTH)

Address: 0xFC0A\_C068 (SIM\_BWTH)

Access: User 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	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																																
Reset	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	1	1	1		

Figure 28-24. SIM Block Wait Time High Register (SIM\_BWTH)

Table 28-25. SIM\_BWTH Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 BWT	Upper block wait time. This value is the upper 16 bits of the block wait time. The time from the start bit of the last byte sent from the SIM to the start bit of the first byte sent from the smart card must be less than the 32-bit value formed SIM_BWTH and SIM_BWTL. If not, SIM_RSR[BWT] is set. Default is 0xFFFF

### 28.3.24 SIM Transmit FIFO Status Register (SIM\_TFSR)

Address: 0xFC0A\_C06C (SIM\_TFSR)

Access: User read-only

	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	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		

Figure 28-25. SIM Transmit FIFO Status Register (SIM\_TFSR)

Table 28-26. SIM\_TFSR Field Descriptions

Field	Description
31–12	Reserved, must be cleared.
11–8 CNT	Indicates the number of bytes in the transmit FIFO. 0000 FIFO is empty or full 0001 One byte in the FIFO ... 1111 15 bytes in the FIFO
7–4 WPTR	Indicates the transmit FIFO write pointer.
3–0 RPTR	Indicates the transmit FIFO read pointer.

### 28.3.25 SIM Receive FIFO Counter Register (SIM\_RFCR)

Address: 0xFC0A\_C070 (SIM\_RFCR)

Access: User read-only

	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	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	

Figure 28-26. SIM Receive FIFO Counter Register (SIM\_RFCR)

Table 28-27. SIM\_RFCR Field Descriptions

Field	Description
31–9	Reserved, must be cleared.
8–0 CNT	Indicates the number of bytes of data currently in the FIFO. A value of zero indicate the receive FIFO is empty or full.

### 28.3.26 SIM Receive FIFO Write Pointer Register (SIM\_RFWP)

Address: 0xFC0A\_C074 (SIM\_RFWP)

Access: User read-only

	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	WPTR				
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	

Figure 28-27. SIM Receive FIFO Write Pointer Register (SIM\_RFWP)

Table 28-28. SIM\_RFWP Field Descriptions

Field	Description
31–9	Reserved, must be cleared.
8–0 WPTR	Indicates the receive FIFO write pointer.

### 28.3.27 SIM Receive FIFO Read Pointer Register (SIM\_RFRP)

Address: 0xFC0A\_C078 (SIM\_RFRP)

Access: User read-only

	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	RPTR				
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	

Figure 28-28. SIM Receive FIFO Read Pointer Register (SIM\_RFRP)

**Table 28-29. SIM\_RFRP Field Descriptions**

Field	Description
31–9	Reserved, must be cleared.
8–0 RPTR	Indicates the receiver FIFO read pointer.

## 28.4 Functional Description

To best describe the organization of the SIM module from a user's point of view, it is instructive to view the SIM at a number of different levels of hierarchy. The SIM module is essentially a standard UART with some special provisions made for SIM card communication. The SIM consists of seven main parts:

- Clock generator
- Transmitter
- Receiver
- Port controller
- General purpose counter
- LRC blocks
- CRC blocks

### 28.4.1 SIM Clock Generator

The clock generator is responsible for generating the baud rate clock (BAUD\_CLK), and clocks to the transmitter, receiver, and port controller sections of the SIM module.

#### 28.4.1.1 Baud Clock Generation

The baud rate clock generation performed by the clock generator results in different frequencies. The default frequency is a divide by two of the peripheral clock. The baud rate can be programmed to be another divisor using the SIM\_PRE register as shown in [Section 28.3.7, “SIM Clock Prescaler Register \(SIM\\_PRE\)”](#). The baud clock is generated in two forms in the design. The BAUD\_CLK that is used by the SIM cards (SCLK0, SCLK1) must be 45–55% duty cycle at the divide values. This is necessary to meet the requirements of the ISO 7816 specification. The BAUD\_CLK used internal to the SIM module is a gated version of the peripheral clock.

#### 28.4.1.2 Transmitter Clock Generation

The internal transmitter clock (XMT\_CLK) is generated by the clock generator based on the values of the SIM\_CR[BAUD\_SEL, SAMPLE12] bitfields. The transmit clock is gated by the transmit enable (SIM\_EN[TXEN]) register bit. When the transmitter is enabled, the clock generator counts the appropriate number of internal receive clock (RCV\_CLK) positive edges to determine when to toggle the transmitter clock output. The transmitter clock is always based upon the receive clock.

### 28.4.1.3 Receiver Clock Generation

The internal receiver clock (RCV\_CLK) is generated by the clock generator based on the value passed to it for the baud rate select (SIM\_CR[BAUD\_SEL]). The receiver clock is gated by the receiver enable (SIM\_EN[RXEN]) register bit. When the receiver is enabled, the clock generator counts the appropriate number of BAUD\_CLK positive edges to determine when to toggle the receiver clock output. The number of BAUD\_CLK positive edges is set by SIM\_CR[BAUD\_SEL] and is programmable to these divisors: 31 (slowest because used with the 372/1 Fi/Di rate), 32, 16, 8, 4, 2, 1, and SIM\_DIV[DIVISOR].

### 28.4.1.4 Port Control Clock Generation

The port controller clocks are provided by the clock generator for the SIM card ports. These clocks are equivalent in frequency to the BAUD\_CLK and are gated by the SIM clock enable (SIM\_PCRn[SCEN]) signals. The level at which the card clocks (SIM\_CLKn) are stopped when disabled is determined by the SIM clock select polarity (SIM\_PCRn[SCSP]) inputs to the clock generator. Synchronizers are implemented to ensure glitch free operation of the card clocks when enabling, disabling, or changing the clock stopped polarity.

### 28.4.1.5 Low Power Mode Clock Control

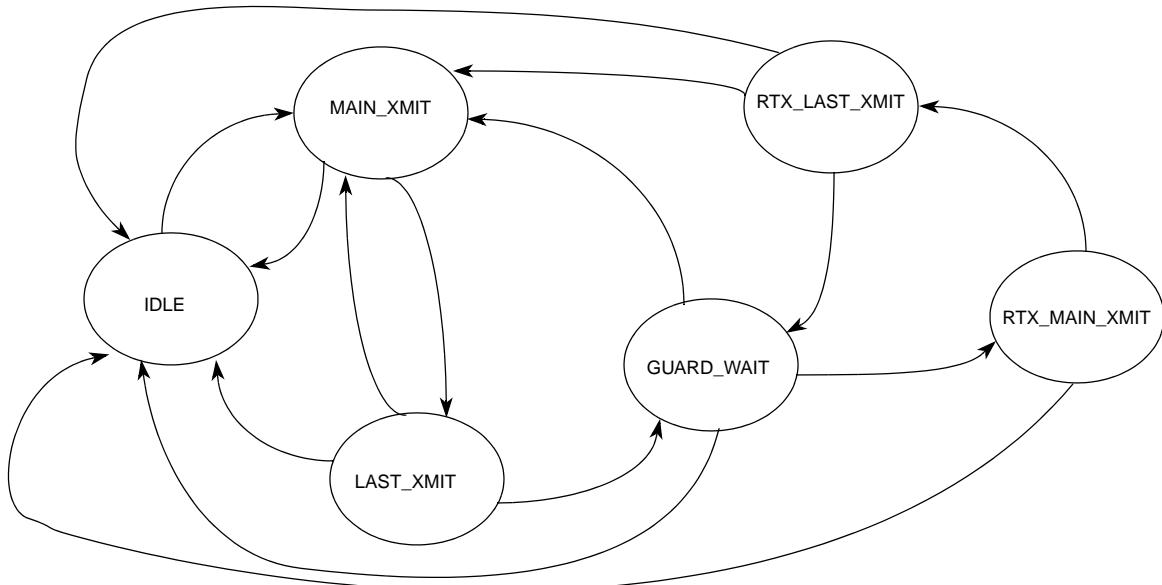
The clock generator block is responsible for gating the clocks to the SIM module appropriately whenever a low power mode instruction is decoded. There are two available low-power states of the processor: stop and doze. The response to doze mode is controlled by SIM\_RCR[DOZE]. Likewise, the response to stop mode is controlled by SIM\_RCR[STOP]. See [Section 28.3.17, “SIM Reset Control Register \(SIM\\_RCR\)”](#).

## 28.4.2 SIM Transmitter

The transmitter block comprises the following sections of logic: transmit state machine, transmit shift register, transmit FIFO, guard time generator, transmit NACK control, and transmit data convention.

### 28.4.2.1 Transmit State Machine

The Transmit state machine is the heart of the transmitter block. The state machine is responsible for sequencing through a transmit operation while reacting to inputs from the receiver, the transmit FIFO, and the guard time circuit. See [Figure 28-29](#) for flow diagram of the transmit state machine.

**Figure 28-29. Transmit State Machine**

The functions performed by each state are:

- **IDLE**
  - This is the initial state. The state machine waits here until SIM\_EN[TXEN] is set and a write to the transmit FIFO has occurred. The data pointed to by the transmit read pointer is loaded into the shift register, and the state machine transitions to the MAIN\_XMIT state. Any time SIM\_EN[TXEN] is cleared, the state machine returns to this state.
- **MAIN\_XMIT**
  - The transmitter is operating normally in this state. The data in the shift register is shifting once every transmit clock cycle. When the second to last bit of the current transmission is about to be sent, the state machine transitions to the LAST\_XMIT state.
- **LAST\_XMIT**
  - This state transmits the last bit of the current transmission and determines the next operation. One of the following will occur.
    - If SIM\_TGCR[GETU] is non-zero, jump to the GUARD\_WAIT state.
    - If a transmit NACK error occurred, with a zero in SIM\_TGCR[GETU], jump to MAIN\_XMIT state to retransmit the current byte.
    - If no transmit NACK, and SIM\_TGCR[GETU] is zero, load the shift register, jump to MAIN\_XMIT to transmit the next byte
    - If no transmit NACK, SIM\_TGCR[GETU] is zero, and the FIFO is empty, jump to IDLE state; set the transmit complete (TC) flag.
- **GUARD\_WAIT**
  - The state machine remains in this state until the guard time counter has expired.
    - If a transmit NACK error occurred on last transmission, jump to RTX\_MAIN\_XMIT and re-transmit

- If no transmit NACK and the FIFO is not empty, load the shift register, jump to MAIN\_XMIT to transmit the next byte.
- If no transmit NACK and the FIFO is empty, return to the IDLE state.
- If transmit NACK threshold is detected, stop transmitter, set XTE flag, and jump to the IDLE state.
- RTX\_MAIN\_XMIT
  - The transmitter is operating normally in this state. The data in the shift register is shifting once every transmit clock cycle. When the second to last bit of the current transmission is about to be sent, the state machine transitions to the RTX\_LAST\_XMIT state. This state is identical to the MAIN\_XMIT state except that it retransmits the previously NACKed byte.
- RTX\_LAST\_XMIT
  - This state transmits the last bit of the current re-transmission and determines the next operation. One of the following will occur.
    - If SIM\_TGCR[GETU] is non-zero, jump to the GUARD\_WAIT state.
    - If a transmit NACK error occurred, with a zero in SIM\_TGCR[GETU], jump to GUARD\_WAIT state to check transmit NACK threshold.
    - If no transmit NACK, SIM\_TGCR[GETU] is zero, and the FIFO is not empty, load the shift register, jump to MAIN\_XMIT to transmit the next byte
    - If no transmit NACK, SIM\_TGCR[GETU] is zero, and the FIFO is empty, jump to IDLE state; set the transmit complete (TC) flag.

### 28.4.2.2 Transmit Shift Register

The transmit shift register is 11 bits wide and controlled by the transmit state machine described previously. The shift register shifts out data at the transmit clock frequency.

### 28.4.2.3 Transmit FIFO

The transmit FIFO is implemented inside the transmitter block. The FIFO depth is 16 bytes. The FIFO block is shared by both SIM module ports. The transmit FIFO cannot be accessed by the alternate SIM module through the alternate port. Each write to the transmit FIFO increases the transmit FIFO write pointer. Each time the transmit shift register is loaded from the transmit FIFO, the transmit FIFO read pointer is incremented. When the read and write pointers are equal, the transmit FIFO empty flag (TFE) is set. Software has no visibility of the transmit FIFO pointers, but a transmit FIFO threshold value can be set to alert the software when the number of bytes in the FIFO has reached a specified level. A read of the transmit FIFO register (SIM\_TBWF) returns the last byte written to the FIFO. Writes to the transmit FIFO can occur at any time.

The transmit FIFO can be flushed by setting SIM\_RCR[FLUSHTX]. A transmit NACK threshold error (XTE) halts the transmitter and flushes the transmit FIFO. The flush operation resets the transmit read and write pointers to the same values. Everything in the transmitter block is reset by the transmit flush operation. This does not include the control registers associated with the transmitter. The transmit data threshold flag (TDTF) is not cleared by the flush operation.

#### 28.4.2.4 Transmit Guard Time Generator

The guard time generator is simply a counter that is clocked by the transmit clock in order to delay the beginning of the next transmission and the setting of the transmit complete interrupt flag (TC) by a programmable amount of transmit bit widths (ETUs). The duration of the count is controlled by SIM\_TGCR[GETU]. See [Figure 28-30](#) for depiction of the three transmit operations to show the effect of the guard time generator logic

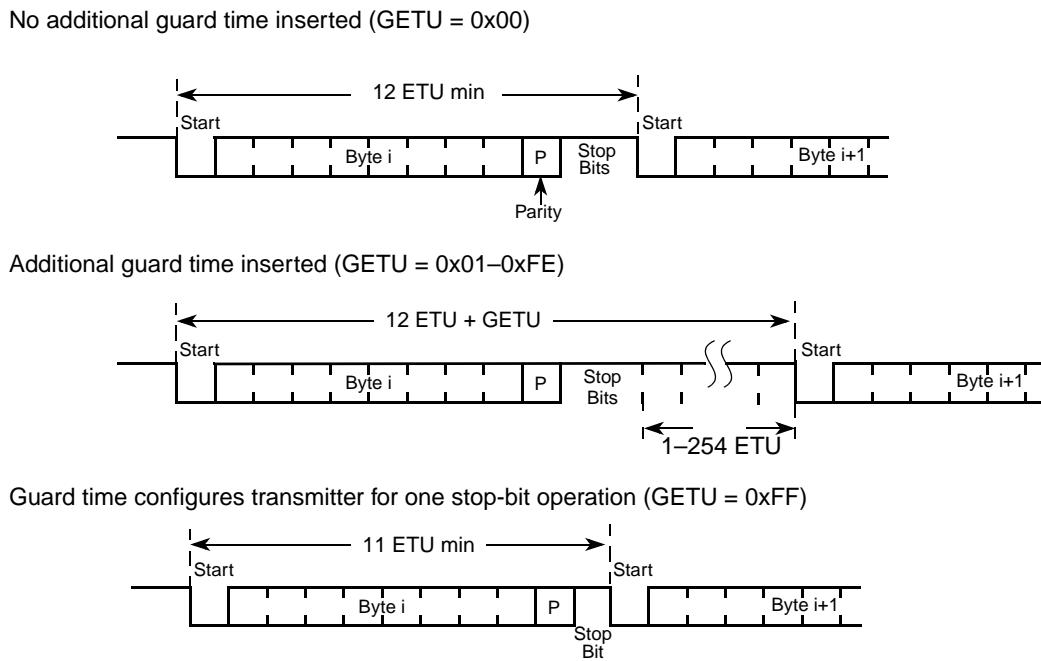


Figure 28-30. Transmit Guard Time

#### 28.4.2.5 Transmit NACK Generator

The transmit NACK generator is responsible for driving the transmitter output low during the stop bit time to signify an error was detected in the received data from the SIM card. This logic responds to a NACK request generated by the receiver block. [Figure 28-31](#) shows a typical SIM transaction with the NACK pulse inserted.

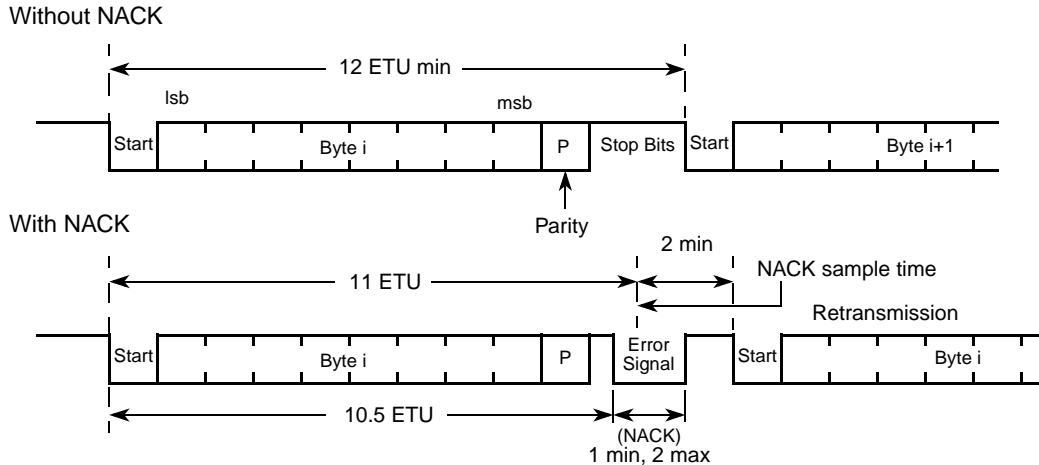
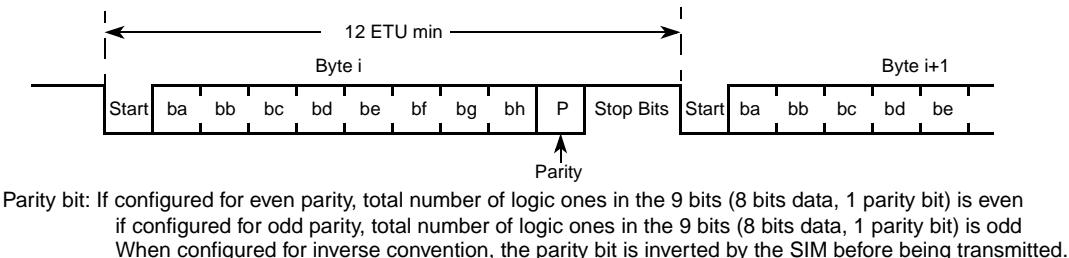


Figure 28-31. Transmit NACK Generator

The transmit NACK generator is also responsible for keeping track of the number of NACKs received during a transmit operation. The SIM receive state machine detects NACKs generated by the SIM card, and reports them to the transmit NACK logic. When the number of detected NACKs equals the programmed threshold (SIM\_TTHR[XTH]), an interrupt is generated, the transmit FIFO is flushed, and the transmitter is disabled.

#### 28.4.2.6 Transmit Data Convention Logic

The transmit data convention logic supports the two different data conventions available in SIM cards. See [Figure 28-32](#) for illustration of SIM data conventions.



Direct convention: ba is lsb of the data byte sent. bh is msb.  
Neither the data bits nor parity bit is logically inverted.

Inverse convention: ba is msb of the data byte sent. bh is lsb.  
Both the data bits and parity bit are logically inverted by hardware.

Figure 28-32. SIM Data Conventions

The direct data convention is the default. If SIM\_FORMAT[IC] is set, the transmit data convention logic converts the output of the transmit FIFO to the inverse convention before sending it to the transmit shift register.

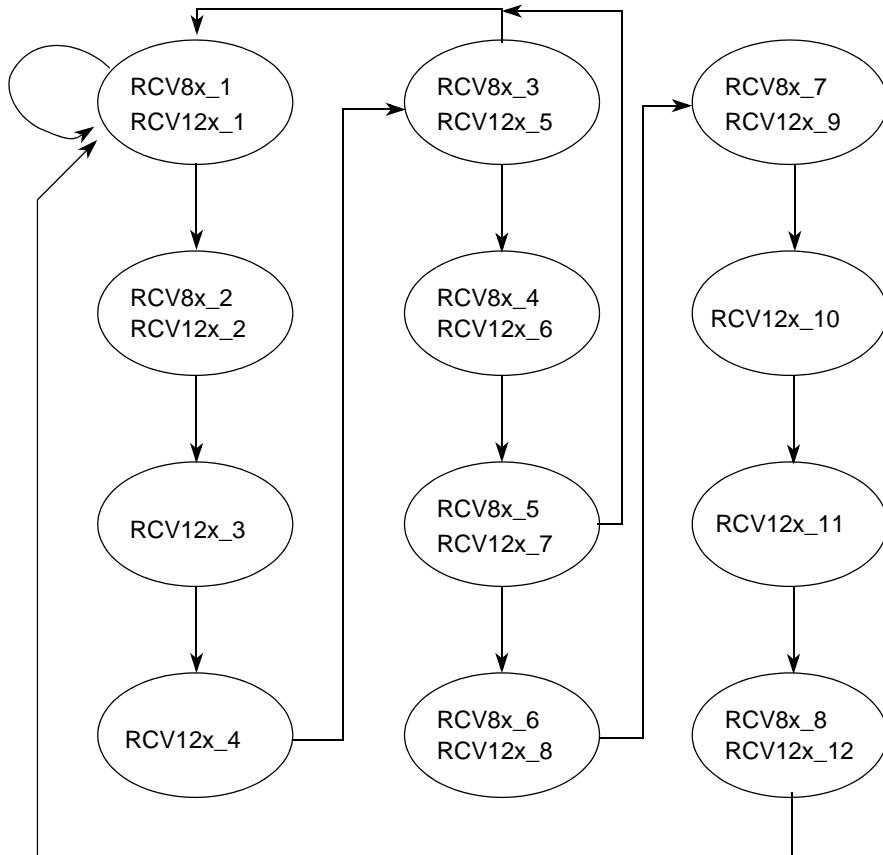
#### 28.4.3 SIM Receiver

The receiver block is comprised of the receive state machine and the receive FIFO.

### 28.4.3.1 Receive State Machine

The receive state machine samples the receive data pin and captures the bit value into the receive shift register. Additionally, the receive state machine detects the start bit, parity errors, framing errors, and the initial character when operating in initial character mode.

Once enabled by SIM\_EN[RXEN] , the receive state machine sequences through the states as shown in Figure 28-33.



**Figure 28-33. Receive State Machine**

The states identified with an 8x are used when operating in a 8x oversampling mode. The states identified with a 12x are used when operating in a 12x oversampling mode (SIM\_CR[SAMPLE12] is set). The number following the oversampling mode identifier represents the state number in the current mode. There are 12 states in 12x mode, and eight states in 8x mode. Some states simply implement a one RCV\_CLK delay. States that perform additional functions are:

- RCV8x\_1, RCV12x\_1
  - This is the initial state of the receive state machine. If the first bit has not been received, the state machine remains in this state until a valid start bit is detected. For every subsequent bit, this state is simply a one RCV\_CLK cycle delay.
- RCV8x\_2, RCV12x\_2
  - This state captures the first sample of the current receive data input.

- RCV12x\_3
  - This state captures the second sample of the current receive data input.
- RCV12x\_4
  - This state captures the third sample of the current receive data input.
- RCV8x\_3, RCV12x\_5
  - This state checks if the SIM is receiving a correct start Bit. If not, return to state 1.
- RCV8x\_4, RCV12x\_6
  - If in the 11th bit, check for parity or initial character errors and send NACK if needed.
- RCV8x\_5, RCV12x\_7
  - Store current bit value in the shift register. If this is the first bit and the value is not zero, restart the state machine.
- RCV8x\_6, RCV12x\_8
  - If the current bit is the last bit of the transfer, set flag to transfer shift register to receive buffer.
- RCV8x\_7, RCV12x\_9
  - Clear flag for transferring shift register to receive buffer.
- RCV12x\_10
  - If the current bit is the last bit of the transfer (first stop bit), this state samples the NACK window.
- RCV12x\_11
  - If the current bit is the last bit of the transfer (first stop bit), this state samples the NACK window.
- RCV8x\_8, RCV12x\_12
  - This state represents the end of the current receive input bit. Several operations occur during this state, including:
    - Increment bit counter
    - Perform a majority vote on the NACK samples and notify the transmitter if a NACK pulse was detected.

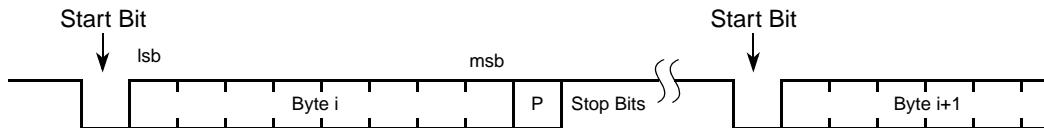
### 28.4.3.2 Data Sampling/Voting

The receive state machine runs at the receive clock rate (RCV\_CLK). This clock oversamples the received data at either an 8x or 12x sample rate. For each input bit, the receive state machine captures three samples. A majority voting algorithm is then applied to determine the value of the bit received. The value common to two or more samples is determined to be the bit value in the receive shift register.

### 28.4.3.3 Start Bit Detection

The SIM receive input is defined as high when not active. The data transmission is defined to begin with a low pulse for a bit duration. This is called the start bit. The receive state machine is responsible for detecting and validating the start bit. The receive state machine samples the start bit three times using a majority voting scheme to determine if the start bit is valid. This will effectively filter out any low receive

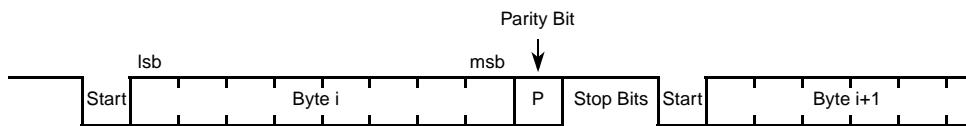
inputs shorter than one RCV\_CLK period. See [Figure 28-34](#) for illustration of a typical SIM data transaction with the start bit identified.



**Figure 28-34. Start Bit Diagram**

#### 28.4.3.4 Parity Error Detection

The receive state machine is responsible for detecting parity errors in the received data. Data is always transmitted with even parity, except when in inverse convention mode. In this mode, all data and parity bits are complemented making the data appear as odd parity. The parity bit is defined as the tenth bit of the received data. The parity of the second through tenth received bits is calculated by the receiver parity logic. This logic determines if the parity of the nine received bits is correct. See [Figure 28-35](#) for illustration of a typical SIM data transaction with the parity bit identified.



**Figure 28-35. Parity Bit Diagram**

When a parity error is detected on a given byte, SIM\_RBUFn[PE] is set for that byte. A parity error cannot generate an interrupt. However, it can signal the SIM transmitter to send a NACK pulse to the SIM card asking for a retransmission of the corrupted data. NACK generation upon a parity error is enabled by setting SIM\_CR[ANACK].

#### 28.4.3.5 Framing Error Detection

The receive state machine is responsible for detecting framing errors in the received data. A SIM data transaction is defined as 11 or 12 bits long consisting of the start bit, eight data bits, one parity bit, and one or two stop bits. A framing error occurs when the stop bit is not detected during the 11th bit time of a data transaction. The stop bit is generally defined as two bit times (ETUs) of a high pulse following the parity bit. When SIM\_TGCR[GETU] is 0xFF, the stop bit is defined as one bit time. A framing error occurs only when the parity bit of the current byte is low, and the stop bit arrives late. See [Figure 28-36](#) for illustration of a typical SIM data transaction with the stop bits identified. Also shown is a SIM data transaction with a late arriving stop bit indicating a framing error.

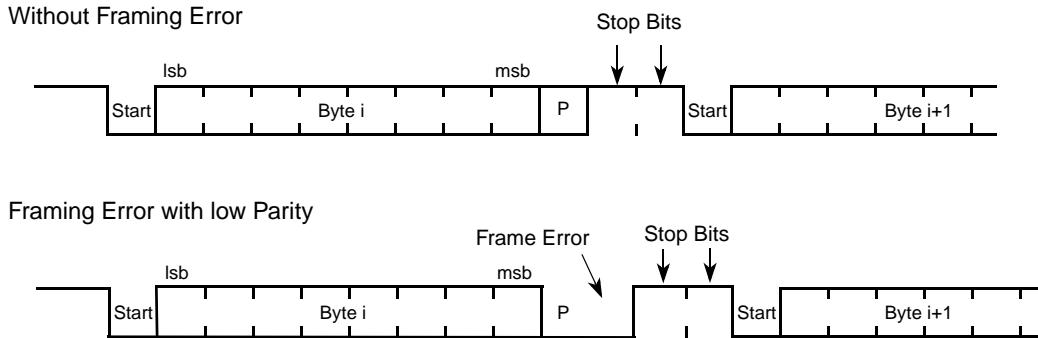


Figure 28-36. Framing Error Diagram

When a framing error is detected on a given byte, SIM\_RBUF $n$ [FE] is set for that byte. A framing error cannot generate an interrupt, nor can it create a NACK pulse to the SIM card asking for a retransmission of the corrupted data.

#### 28.4.3.6 NACK Detection

The existence of the NACK pulse is sampled by the receive state machine at 11 elementary time units (ETUs) after the falling edge of the start bit. An ETU is equivalent in time to one transmit clock period. When the receiver detects a NACK, it signals the transmitter that an error occurred. The transmitter will not initiate retransmission for at least another two ETU times as required by the ISO 7816 specification.

#### 28.4.3.7 Initial Character Detection

The SIM receive state machine supports the detection of special characters that allow it to determine what data format is being used by the connected SIM card. When placed in initial character mode, the SIM expects to receive one of two potential characters that it will use to set the data format control bit (SIM\_FORMAT[IC]).

The two possible data formats are inverse convention and direct convention. [Figure 28-37](#) and [Figure 28-38](#) illustrate the differences between the two formats. Essentially, inverse convention flips the order of the data and the data and parity bits are logically inverted. When receiving inverse convention data, the transformation of the data back to direct convention format is done in hardware, including the inversion of the data and parity bits.

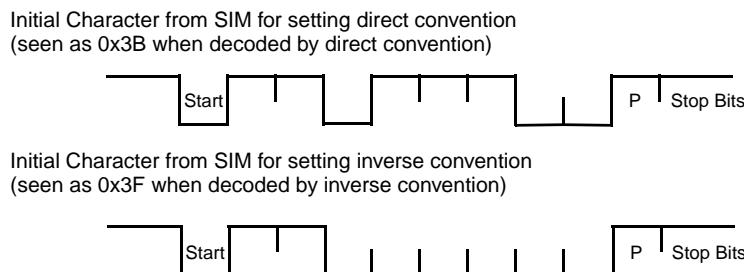
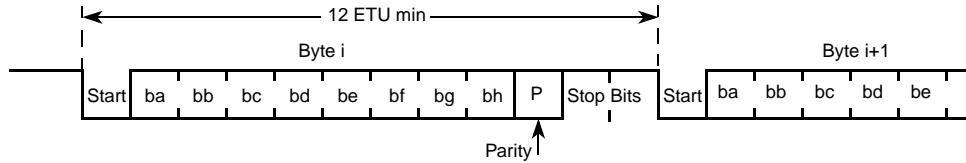


Figure 28-37. Valid Initial Characters



Parity bit: If configured for even parity, total number of logic ones in the 9 bits (8 bits data, 1 parity bit) will be even.

If configured for odd parity, total number of logic ones in the 9 bits (8 bits data, 1 parity bit) will be odd.

When configured for inverse convention, the parity bit is inverted by SIM card before being transmitted.

Direct convention: ba is lsb of data byte to be sent. bh is msb.

Neither the data bits nor parity bit is logically inverted.

Inverse convention: ba is msb of data byte to be sent. bh is lsb.

Both the data bits and parity bit are logically inverted by SIM card.

**Figure 28-38. Inverse Convention Versus Direct Convention**

#### 28.4.3.8 Receive FIFO

The receive FIFO is implemented inside a sub-block of the receiver. The FIFO depth is 285 bytes and is shared by both SIM module ports. The receive FIFO is accessed through the SIM\_RBUF $n$  registers.

The receive FIFO is loaded from the receive shift register after the final bit of the current SIM card transmission has been received. The FIFO contains ten bits per transmission. The lower eight bits contain the received data byte. Bits 8 and 9 contain the parity and framing status for the received byte.

Each read from the receive FIFO increments the receive FIFO read pointer. Each time the receive shift register is transferred to the receive FIFO, the receive FIFO write pointer increments. When the difference between the read and write pointers equals the programmed threshold value (RDT), the receive data register full flag (RDRF) is set. An interrupt is generated by RDRF if SIM\_IMR[RIM] is cleared. Software has no visibility of the receive FIFO pointers. A write to the receive FIFO register (SIM\_RBUF $n$ ) generates an invalid access exception to the processor.

The receive FIFO is flushed by setting SIM\_RCR[RCV\_FLUSH]. The flush operation resets the receive read and write pointers to equal values. All logic associated with the receiver will be reset by the flush operation except for receiver control registers.

#### 28.4.3.9 Overrun Detection

The receive FIFO logic is responsible for detecting an overrun condition. When a received byte is transferred from the receive shift register to a receive FIFO that contains 285 unread bytes, the SIM receiver flags an overrun condition (SIM\_RSR[OEF]). The received byte is discarded leaving the 285 unread bytes in the FIFO unaltered. The SIM module generates a NACK to the SIM card if SIM\_CR[ONACK] is set. The SIM module continually NACK SIM card transmissions until a read of the receive FIFO occurs.

#### 28.4.3.10 Character Wait Time Counter

The SIM receiver block includes a 16-bit counter that counts the number of bit times (ETUs) between received characters. When enabled, the character wait time counter (CWT) does not start counting until

after the start bit(s) of a valid character are received. The counter is synchronized to the receive character bit positions so that an accurate count of the number of ETUs between characters is made. The CWT has a 16-bit programmable comparator that software can write the expected number of ETUs between characters to. If the time between characters exceeds this value, an interrupt flag is set and an interrupt generated if the mask is clear.

## 28.4.4 SIM Port Control

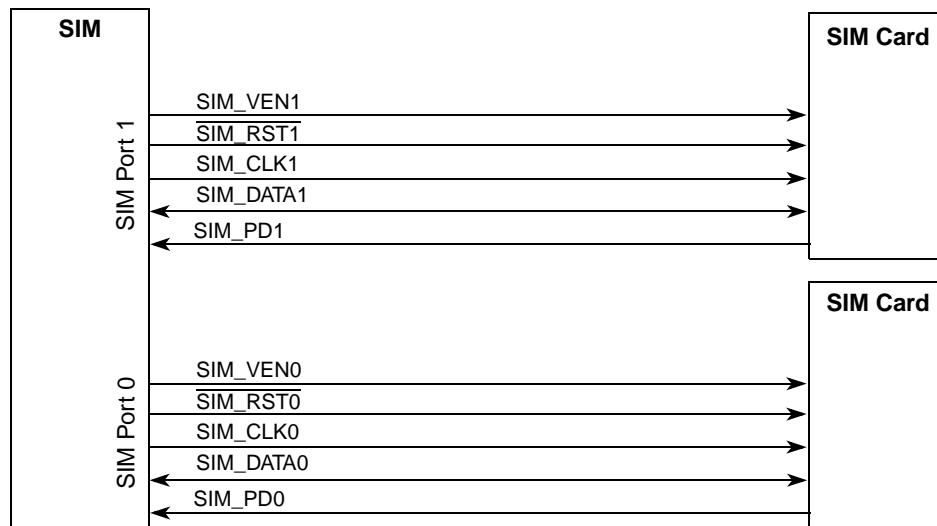
The port control block comprises the following functions: SIM card interface, SIM Card presence detect, and SIM card auto-power down.

### 28.4.4.1 SIM Card Interface

The SIM module allows for direct control of two separate SIM cards. The SIM module does not support simultaneous communication with two SIM cards.

The SIM card clock is generated in the SIM clock generator. The SIM card reset and card voltage enable are controlled by software through SIM\_PCR $n$ .

See [Figure 28-39](#) for an example SIM module connection to two SIM cards. The power management chip shown is used to provide Vcc for the SIM card, and level translators for the remaining signals when interfacing to a SIM card operating at a different voltage than SIM module. The SIM module can directly access a SIM card if it is operating at the same voltage.



**Figure 28-39. SIM Card Hookup**

### 28.4.4.2 SIM Card Presence Detect

The SIM\_PD $x$  input allows for detection of the insertion or removal of a SIM card. Software can use SIM\_DETECT $n$ [SPDS] to configure which edge of the SIM\_PD $x$  pin causes a presence detect event. A maskable interrupt can be generated when a SIM\_PD $x$  event occurs.

You can place an external pull-down resistor on the SIM\_PD<sub>x</sub> pins. Doing so allows a high-to-low transition on the SIM\_PD<sub>x</sub> pins to occur, when an external driver drives a logic high on the SIM\_PD<sub>x</sub> pins when a card is present and the SIM card is then removed.

#### 28.4.4.3 SIM Card Automatic Power Down

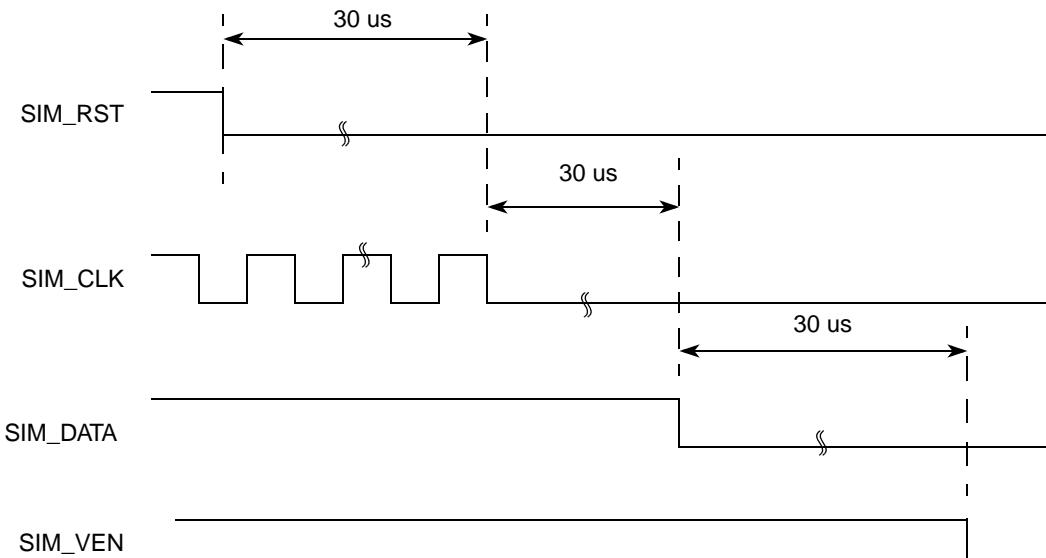
When interfacing to the SIM cards, it is necessary to follow a particular sequence when powering them up and down. The SIM port control block contains hardware that provides the correct sequence to power down a SIM card (see [Figure 28-40](#)). The power-up sequence must be done manually by the software using the pin control bits supplied in SIM\_PCR<sub>n</sub>.

##### NOTE

The on-chip 32-kHz oscillator must be enabled for the automatic power down feature to operate. Set the RTC\_CFG\_DATA[OSCEN] bit in the RTC module. See [Section 37.4.14, “RTC Configuration Data Register \(RTC\\_CFG\\_DATA\)”](#), for more details.

The power-down sequence is specified in ISO 7816 as:

1. SIM\_RST transitions from high to low.
2. SIM\_CLK is negated.
3. I/O transitions from high impedance to low.
4. SIM Vcc is turned off.



**Figure 28-40. Auto Power Down Sequence**

#### 28.4.5 SIM General Purpose Counter

The SIM module provides a 16-bit counter for timing events during SIM card communication. The clock source for the counter is selectable between three sources: BAUD\_CLK, RCV\_CLK, or XMT\_CLK (ETU clock). SIM\_CR[GPCNT\_CLKSEL] selects the clock input. The counter is enabled as soon as the input

clock is selected. The starting of the counter is immediate once the input clock is running. Software controls the three input clock sources by using SIM\_RCR[KILL\_CLOCK] and SIM\_EN[RXEN, TXEN].

**Table 28-30. General Purpose Counter Clock Source Selections**

	<b>SIM_RCR [KILL_CLK]</b>	<b>SIM_CR [GPCNT_</b> <b>CLKSEL]</b>	<b>SIM_EN [RXEN]</b>	<b>SIM_CR [CWTEN]</b>	<b>SIM_EN [TXEN]</b>
Card Clock Source	0	01	—	—	—
Receive Clock Source	0	10	1	—	—
	0	10	—	—	1
ETU (Transmit) Clock Source	0	11	1	1	—
	0	11	—	—	1

The counter is reset by clearing SIM\_CR[GPCNT\_CLKSEL]. A 16-bit comparator value allows software to select a count value to interrupt the processor if the mask is clear.

## 28.4.6 SIM LRC Block

The SIM module provides an 8-bit linear redundancy check (LRC) generator/checker for T=1 SIM cards that support LRC. This block is enabled through SIM\_CR[LRCEN] and performs an 8-bit exclusive-OR on all received or transmitted characters. At the end of the reception of a block of characters, the result is expected to be zero. If so, SIM\_RSR[LRCOK] is set. During transmission, the LRC block exclusive-ORs each character that is transmitted with the current value of the LRC. If SIM\_CR[XMT\_CRC\_LRC] is set, the LRC value is automatically sent by the SIM transmitter as the final character when the transmit FIFO empties.

The LRC value is reset by any of the following:

- Clearing SIM\_CR[LCREN]
- At the end of a transmission (either after the LRC byte is transmitted, or after the last character in the transmit FIFO is sent when XMT\_CRC\_LRC is clear)
- Setting SIM\_EN[TXEN]

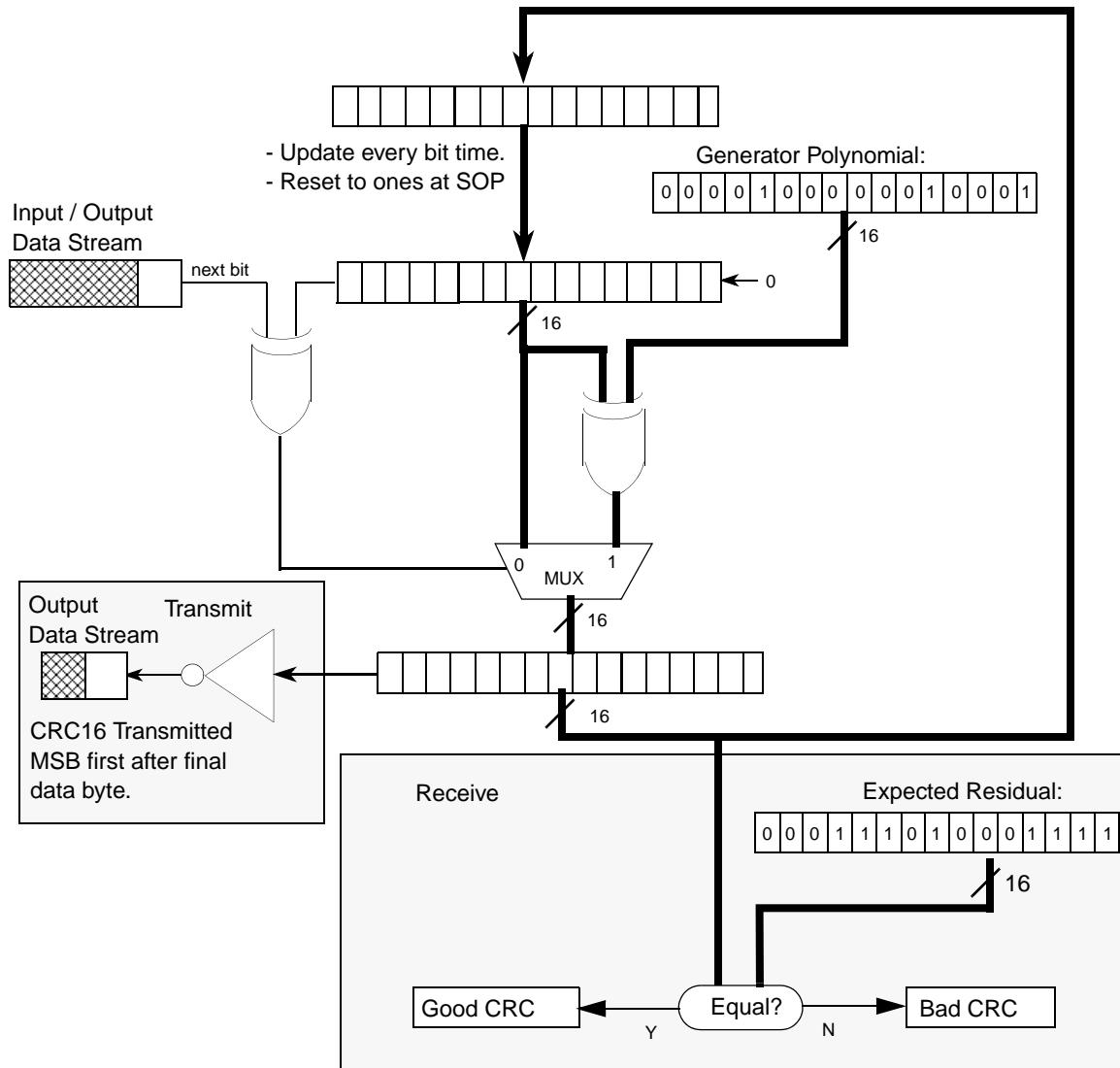
## 28.4.7 SIM CRC Block

The SIM module provides an 16-bit cyclic redundancy check (CRC) generator/checker for T=1 SIM cards that support CRC. This block is enabled through SIM\_CR[CRCEN]. This block performs a polynomial based check on all received or transmitted characters. The polynomial description is shown in Figure 4-17. The polynomial used is the standard CRC-CCITT where  $g(x) = x^{16} + x^{12} + x^5 + 1$ . The CRC register is initialized to all ones before the data is shifted in. Before transmission the resulting CRC is inverted. For example, transmitting a 0xFA results in the following:

- Data = 0x5F (bit reversal of 0xFA)
- CRC = 0x4AEA
- Invert the crc = 0xB515 (bit reversal of 0xADA8)

- Data transmitted = 0xFA, 0xAD, 0xA8

See [Figure 28-41](#) for an illustration of the SIM CRC block.



**Figure 28-41. CRC Block Diagram**

At the end of the reception of a block of characters, the residual from the CRC calculation is compared to 0x1D0F. If equal, SIM\_RSR[CRCOK] is set. During transmission, the CRC block updates the current value of the CRC residual using each character. If SIM\_CR[XMT\_CRC\_LRC] is set, the CRC value is automatically inverted and sent by the SIM transmitter as the final two characters when the transmit FIFO empties.

The CRC value is reset by any of the following:

- Clearing SIM\_CR[CRCEN]
- At the end of a transmission (either after the CRC characters are transmitted, or after the last character in the transmit FIFO is sent when XMT\_CRC\_LRC is cleared)

- Setting SIM\_EN[TXEN]

### 28.4.8 Module Interrupts

See [Table 28-31](#) for the list of all possible interrupt sources and their corresponding mask bits. The SIM interrupts are logically ORed to create two interrupts to the interrupt controller:

- SIM data interrupt — includes five transmit and receive status interrupts
- SIM general interrupt — includes ten various SIM interrupts

All mask bits disable the interrupt when set, whereas a zero implies that the interrupt is enabled (unmasked). All mask bits are set out of reset (all interrupts are masked).

**Table 28-31. SIM Module Interrupts**

Flag	Mask Register	Description
<b>Data Interrupt</b>		
SIM_TSR[TC]	SIM_IMR[TCIM]	Transmit complete
SIM_TSR[ETC]	SIM_IMR[ETCIM]	Early transmit complete
SIM_TSR[TFE]	SIM_IMR[TFEIM]	Transmit FIFO empty
SIM_TSR[TDTF]	SIM_IMR[TDTFM]	Transmit data threshold flag
SIM_RSR[RDRF]	SIM_IMR[RIM]	Receive data register full (FIFO threshold level reached)
<b>General Interrupt</b>		
SIM_TSR[GPCNT]	SIM_IMR[GPCNTM]	General purpose counter comparator flag
SIM_TSR[XTE]	SIM_IMR[XTM]	Transmit threshold error
SIM_TSR[TFO]	SIM_IMR[TFOM]	Transmit FIFO overfill error
SIM_RSR[OEF]	SIM_IMR[OIM]	Overrun error flag
SIM_RSR[CWT]	SIM_IMR[CWTM]	Character wait time counter comparator flag
SIM_RSR[BWT]	SIM_IMR[BWTM]	Block wait time counter comparator flag
SIM_RSR[BGT]	SIM_IMR[BGTM]	Block guard time counter comparator flag
SIM_RSR[RTE]	SIM_IMR[RTM]	Receive NACK threshold error
SIM_DETECT1[SDI1]	SIM_DETECT1[SDIM1]	SIM detect interrupt for port 1
SIM_DETECT0[SDI0]	SIM_DETECT0[SDIM0]	SIM detect interrupt for port 0

## 28.5 Initialization/Application Information

This section describes the intended programming model for using the SIM module. The section describes how to begin a typical mode of operation using the registers.

## 28.5.1 Configuring SIM for Operation

The following list of items must be performed to configure the SIM module for operation:

1. Port selection in the SIM\_SETUP register.
  - a) Select which SIM module port is active by writing the SPS bit.
2. Enable the selected port (port 1, SIM\_PCR1; port 0, SIM\_PCR0) following the SIM power-up procedure specified in ISO 7816.
  - a) Set SVEN to enable power to the SIM card.
  - b) Set STEN to enable SIM module transmit data output. This is required to allow the SIM receiver to create NACK pulses.
  - c) Set SCEN to enable SIM card clock.
  - d) Set SRST to release the SIM card from reset.
3. Choose the port baud rate in SIM\_CR.
  - a) Select the SIM card clock rate by using CLK\_PRE.
  - b) Select the SIM card baud rate by using BAUD\_SEL and SAMPLE12

### NOTE

Follow the ISO 7816 spec with regards to card clock frequencies to ensure the maximum frequency specification is not violated.

4. Select data format type, or place SIM receiver in initial character mode.
  - a) Select inverse convention or direct convention by using SIM\_FORMAT[IC], or
  - b) Enable initial character mode by using SIM\_CR[ICM]

### 28.5.1.1 Configuring SIM Receive

The following list of items must be performed to configure the SIM receiver for operation:

1. Enable NACK capability in SIM\_CR.
  - a) Select NACK generation on parity errors, or invalid initial character by using ANACK.
  - b) Select NACK generation on overrun conditions by using ONACK.
2. Select the desired receive NACK threshold and receive FIFO threshold in SIM\_RTHR.
  - a) Program the threshold at which the RDRF flag is set by using RDT.
  - b) Program the threshold at which the RTE flag is set by writing to RTH. If an automatic power down after RTE flag is set is desired, then enable the SIM card auto power down by setting SIM\_PCRn[SAPD].
3. Configure the character wait time counter, the block wait timer counter, and the block guard time counter.
4. Enable interrupts in SIM\_IMR.
  - a) Enable the receive data register full interrupt by using RIM.
  - b) Enable the receive threshold error interrupt by using RTM.
  - c) Enable the overrun condition interrupt by using OIM.

- d) Enable the character wait time interrupt by using CWTM.

### 28.5.1.2 Configuring SIM Transmitter

The following list of items must be performed to configure the SIM transmitter for operation:

1. Select desired re-transmission threshold for NACKed characters in SIM\_TTHR.
  - a) Program the threshold at which the XTE flag will be set by using XTH.
2. Select the guard time between transmissions in SIM\_TGCR.
  - a. Program the desired guard time between characters transmitted by the SIM module by using GETU.
3. Select the desired transmit FIFO threshold level in SIM\_TTHR.
  - a) Program the desired threshold using TDT.
4. Enable interrupts in SIM\_IMR.
  - a) Enable the transmit complete interrupt by using TCIM.
  - b) Enable the early transmit complete interrupt by using ETCIM.
  - c) Enable the transmit FIFO empty interrupt by using TFEIM.
  - d) Enable the transmit threshold error interrupt by using XTM.
  - e) Enable the transmit FIFO threshold interrupt by using TDTFM.
  - f) Enable the transmit FIFO overflow interrupt by using TFOM.

### 28.5.1.3 Configuring SIM General Purpose Counter

The following list of items must be performed to configure the SIM general purpose counter for operation:

1. Select the desired clock source for the general purpose counter using the SIM\_CR register.
  - a) Use SIM\_CR[GPCNT\_CLKSEL] to select the desired clock source for the counter
2. Program counter comparator using the SIM\_GPCNT register.
  - a) Use the GPCNT bits to select the desired count value at which the GPCNT interrupt flag is set.
3. Enable the selected clock source for the general purpose counter using SIM\_RCR or SIM\_EN.
  - a) If the GP counter is configured for the card clock, enable the clock by clearing SIM\_RCR[KILL\_CLOCK].
  - b) If the GP counter is configured for the receive oversample clock, enable this clock by setting SIM\_EN[RXEN] or SIM\_EN[TXEN].
  - c) If the GP counter is configured for the transmit oversample clock, enable this clock by setting SIM\_EN[TXEN].
4. Enable interrupts in SIM\_IMR.
  - a) Enable the general purpose counter interrupt by clearing GPCNTM.

### 28.5.1.4 Configuring SIM to Measure WWT (Work Wait Time) for Type=0 Smart Cards

Work wait time is a combination of BWT and CWT. If you want the SIM to enforce a WWT of 100 bits, then you must activate CWT and BWT, and program them both for a value of 100 bits. When measuring WWT, the BGT timer is not used so it is masked (inactive) by the BGTM bit.

Below is the sequence to program the SIM to send and receive data while checking WWT.

1. Program SIM\_CWTR, SIM\_BWTL, and SIM\_BWTH registers to the WWT (work wait time) value that needs to be enforced. Clear the BGT register.
2. Activate both the CWT and BWT functions by setting SIM\_CR[CWTEN, BWTEN].
3. Enable the CWT and BWT interrupts by clearing SIM\_IMR[CWTM, BWTM].
4. Program the data to send to the smart card by writing data to SIM\_TBUF $n$ .
5. Activate the SIM transmit and receive by setting SIM\_EN[TXEN, RXEN].
6. If the software has more data to send, then as the FIFO starts to empty, it should write more data to SIM\_TBUF $n$ . If the software does not have any more data to send then proceed to the next step.
7. When the smart card has completed its transmission to the SIM module, disable the BWT by clearing SIM\_CR[BWTEN]. This prepares the SIM to measure the next block wait time.
8. Disable the SIM transmitter by clearing SIM\_EN[TXEN].
9. Program the next data to send by writing data to SIM\_TBUF $n$ .
10. Enable the BWT function by setting SIM\_CR[BWTEN].
11. Enable the SIM transmitter by setting SIM\_EN[TXEN].
12. Steps 6 to 10 can be repeated for each transmission to the smart card.
13. If a CWT or a BWT interrupt occurs, the software should consider this a WWT violation. The software should clear the CWT or BWT interrupt by writing one to SIM\_RSR[BWT or CWT].

### 28.5.1.5 Configuring SIM to measure CWT, BWT, BGT for Type=1 Smart Cards

The following list of items must be performed to configure the SIM to measure CWT, BWT, and BGT.

1. Program SIM\_CWTR, SIM\_BWTL, SIM\_BWTH, and SIM\_BGT to the enforced value.
2. Activate the CWT and BWT functions by setting SIM\_CR[CWTEN, BWTEN].
3. Enable the CWT, BWT, and BGT interrupts by clearing SIM\_IMR[CWTM, BWTM, BGTM].
4. Program the data to send to the smart card by writing data to SIM\_TBUF $n$
5. Activate the SIM transmit and receive by setting SIM\_EN[TXEN, RXEN].
6. As the FIFO approaches empty, if the software has more data to send, it should write more data to SIM\_TBUF $n$ . If the software does not have any more data to send, proceed to the next step.
7. When the smart card has completed its transmission to the SIM module, disable the BWT by clearing SIM\_CR[BWTEN]. This prepares the SIM to measure the next block wait time.
8. Disable the SIM transmitter by clearing SIM\_EN[TXEN].
9. Program the next data to send by writing data to SIM\_TBUF $n$ .
10. Enable the BWT function by setting SIM\_CR[BWTEN].

11. Enable the SIM transmitter by setting SIM[TXEN].
12. Steps 6 to 10 can be repeated for each transmission to the smart card.
13. If a CWT or a BWT interrupt occurs, the software should read SIM\_RSR to determine if the error was caused by a CWT, BWT, or BGT violation. The software should clear the CWT, BWT, or BGT interrupt by writing a one to SIM\_RSR[CWT, BWT, or BGT]

### 28.5.1.6 Configuring SIM Linear Redundancy Check (LRC) Block

The following list must be performed to configure the SIM linear redundancy check block for operation:

1. Enable the LRC block by using SIM\_CR
  - a) Use LRCEN to enable the LRC block
  - b) Use XMT\_CRC\_LRC to enable the transmission of the LRC character after the last character in the transmit FIFO is sent. Refer to the T=1 programming model for more details.

### 28.5.1.7 Configuring SIM Cyclic Redundancy Check (CRC) Block

The following list must be performed to configure the SIM cyclic redundancy check block for operation:

1. Enable the CRC block by using SIM\_CR.
  - a) Use CRCEN to enable the CRC block
  - b) Use XMT\_CRC\_LRC to enable the transmission of the CRC characters after the last character in the transmit FIFO is sent. Refer to the T=1 programming model for more details.

## 28.5.2 Using SIM Receiver

When the SIM has been properly configured (correct baud rate and data format), SIM receptions can be enabled by setting the receive enable bit, SIM\_EN[RXEN]. As bytes are received, they are placed in the 285 byte deep receive data FIFO. Unread bytes can be accessed from this FIFO at any time. There is no need to disable the receiver to access the FIFO. The FIFO should only be read when the receive FIFO data flag, SIM\_RSR[RFD] is set. The RFD flag, which cannot generate an interrupt, is set when there is at least one unread byte in the receive FIFO. If the receive FIFO is read when RFD is cleared, it will simply produce the last byte read.

The correct address to read the data contained in the receive FIFO depends on which SIM port is selected. SIM\_SETUP[SPS] controls port selection. If cleared, read the data from the SIM\_RBUFO; if set, read the data from SIM\_RBUF1.

The receive data register full flag, SIM\_RSR[RDRF], is used to determine when the receive FIFO has reached a given threshold value. This flag generates an interrupt if SIM\_IMR[RIM] is cleared. To control at when RDRF is set, program the receive data threshold, SIM\_RTHR[RDT]. If the number of unread bytes in the receive FIFO is equal to or greater than the value set by RDT, RDRF is set.

#### NOTE

A value of 0x0 in RDT implies that there must be 285 unread bytes in the receive FIFO to trigger RDRF.

The value in RDT can be changed at any time to alter this threshold level. The comparison between the number of unread bytes in the FIFO and the value set by RDT is continuously updated so that any change in either will be immediately reflected in the state of RDRF. For instance, if RDT is set to 0x5 and there are three unread bytes in the FIFO, changing RDT to 0x2 immediately sets RDRF. Likewise, setting RDT back to 0x5 clears RDRF. Similarly, if there are five unread bytes in the receive FIFO and RDT is set to 0x3, RDRF remains set until three reads are complete (assuming RDT is constant and no new data is received).

The standard flow for receiving bytes from the SIM card is to:

1. Set RDT to the appropriate value.
2. Wait for RDRF to cause an interrupt (RIM clear).
3. Read bytes out of the receive FIFO as long as RFD is set.

In addition to checking RFD between every byte, it is also recommended to check for the existence of a set OEF flag as well.

### 28.5.2.1 Receive Parity Errors and Parity NACK Generation

The SIM receiver checks every byte received for proper parity. SIM\_FORMAT[IC] controls whether it checks for odd or even parity. When checking for odd parity, the number of logic ones contained in the nine received bits (eight data bits and one parity bit) should be odd. Likewise, when checking for even parity, the number of logic ones contained in the nine received bits should be even.

When a parity error is detected on a given byte, SIM\_RBUF $n$ [PE] for that byte is set. The PE flag for each byte is read out of the FIFO when the data itself is read. There is no need to clear the parity error flag in the FIFO. It is simply overwritten the next time a byte is received into that position of the FIFO. A parity error cannot cause an interrupt.

If SIM\_CR[ANACK] is set, the SIM automatically requests the SIM card to resend a byte found with a parity error by generating a NACK pulse on the SIM\_DATA pin. Bytes with parity errors that cause a NACK pulse are still placed into the FIFO just as bytes that do not cause a NACK pulse. Software must discard data bytes with parity errors.

To control NACK generation by the SIM receiver use SIM\_RTHR[RTH]. This set of bits specify the number of consecutive NACKs generated by the SIM module on a received byte, before setting the receive threshold interrupt flag, SIM\_RSR[RTE]. The RTE flag also forces the SIM port to power-down the card if SIM\_PCR $n$ [SAPD] is set.

#### NOTE

SIM\_CR[ANACK] must be set to enable this feature. In initial character mode, ANACK enables the retransmission of initial characters in the event that an invalid initial character is received.

When a valid character has been received by the SIM, the internal counter keeping track of the number of NACKs transmitted on the current byte resets to zero. Clearing SIM\_RSR[RTE] also clears that counter.

When generating a NACK pulse, the SIM generates the low pulse starting at 10.5 ETUs and lasting for one ETU.

### 28.5.2.2 Receive Frame Errors

The SIM receiver checks every byte received for a proper stop bit. A stop bit should exist during at least the first half of the 11th ETU time after the start of the character. If this is not true, a frame error is flagged. When a frame error is detected on a given byte, SIM\_TBUF $n$ [FE] for that byte is set in the FIFO. The FE flag for each byte is read out of the FIFO when the data itself is read. There is no need to clear the frame error flag in the FIFO. It is simply overwritten the next time a byte is received into that position of the FIFO. A frame error cannot cause an interrupt, nor can it create a NACK pulse to the receiver asking for a retransmission of the corrupted data.

### 28.5.2.3 Receive Overrun Errors and Overrun NACK Generation

When 285 unread bytes are in the FIFO, a received character will cause the SIM receiver to flag an overrun condition. This condition always sets the overrun error flag, SIM\_RSR[OEF]. The received byte is discarded, leaving the 285 unread bytes in the FIFO unaltered.

If SIM\_CR[ONACK] is set, the SIM automatically requests the SIM card to resend the byte that caused the overrun condition by generating a NACK pulse on the SIM\_DATA pin. In this case, the existence of an OEF flag does not indicate the loss of data, but rather a NACK (retransmission request) due to a full receive FIFO. As opposed to transmit NACK generation, there is no limit to the number of times an overrun condition causes a NACK other than to disable ONACK itself. When generating a NACK pulse, the SIM generates the low pulse starting at 10.5 ETUs and lasting for one ETU.

If ONACK is cleared, a set SIM\_RSR[OEF] indicates the loss of data. The OEF flag generates an interrupt if SIM\_IMR[OIM] is cleared.

To clear OEF, software must write a one to SIM\_RSR[OEF]. A high OEF flag has no effect on the operation of the SIM receiver other than to create an interrupt if OIM is clear.

### 28.5.2.4 Initial Character Mode and Resulting Receive Data Formats

The SIM receiver supports the detection of special characters that allow it to determine what data format is being used by the connected SIM card. When placed in initial character mode, the SIM expects to receive one of two potential values that it uses to set the data format control bit, SIM\_FORMAT[IC].

The two possible data formats are inverse convention and direct convention. Essentially, inverse convention differs from direct convention in that the order of the data is flipped, and the data and parity bit are logically inverted. When receiving inverse convention data, the transformation of the data back to direct convention format is done in hardware, including the inversion of the data and parity bits.

To place the SIM into initial character mode, set SIM\_CR[ICM]. When a valid initial character is received, SIM\_FORMAT[IC] is set accordingly by the hardware, and ICM is cleared. Software can read the state of IC to determine which mode the SIM is currently using.

A 0x3B (as decoded by direct convention) with parity bit set causes direct convention to be used (IC cleared); whereas, a 0x3F (as decoded by inverse convention) with parity bit set causes inverse convention to be used (IC set).

When the receiver is in initial character mode, all received bytes continue to be placed into the receive FIFO whether they are valid initial characters or not. If a valid initial character is received that causes the

data format being used to change, all subsequent bytes are decoded with that format before being placed into the FIFO (including the initial character byte itself). That is, if IC is cleared and the correct initial character for setting inverse convention is received, that character and all subsequently received characters are stored in the FIFO after having been decoded using inverse convention (for example, the initial character is stored as 0x3F).

If the receiver is in initial character mode (ICM is set) and an invalid initial character is received, the SIM can be configured to automatically request that the initial character be retransmitted by setting SIM\_CR[ANACK]. When generating a NACK pulse, the SIM generates the low pulse starting at 10.5 ETUs and lasting for one ETU. The invalid initial character is placed into the receive FIFO and marked with a parity error, signifying that this is an invalid initial character.

### **28.5.2.5 Initial Character Mode Programming Notes**

The usage of initial character mode requires close attention to the programming model. A parity error in the initial byte for direct convention (0x3B) could be decoded as what appears as a valid initial character for inverse convention (0x3F). The SIM module does not recognize this as a valid initial character for inverse convention and marks the character by setting the parity error flag. The software must look for the existence of a parity error before recognizing a character as a valid initial character.

### **28.5.2.6 Automatic Receiver Mode**

The SIM module has an automatic receive mode that inhibits the data being transmitted by the SIM module from entering the SIM receive buffer through the feedback path of the SIM data pin. The SIM module receiver should normally be enabled while the transmitter is operational. Automatic receive mode saves the software from having to actively manage the transition from transmitter to receiver. The auto-receive mode is always active when the receiver is enabled.

### **28.5.2.7 Using the SIM Receiver with T=1 SIM Cards**

T=1 SIM cards present several requirements beyond standard T=0 cards. The features provided to meet the requirements that pertain to the SIM receiver are as follows:

- 11 ETU characters
  - T=1 cards can transmit with character lengths of 11 ETUs (one stop bit). The SIM module provides SIM\_TGCR[RCVR11] register to configure the receiver state machine to accept 11 ETU characters.
- Character wait time counter
  - The character waiting time (CWT) is the time between the start bits of two consecutive characters received from the smart card. The value of CWT can range from 12 ETU to 32,779 ETU. The SIM module provides a 16-bit counter with programmable comparator clocked at the ETU bit rate to identify when the CWT has been exceeded by the SIM card.
- Block waiting time
  - The block waiting time (BWT) is the maximum time between the start bits of the last character of a transmitted block and the first character of the next received block. The BWT must not

exceed a value that is programmable. The SIM module provides a 32-bit block wait timer (divided in two registers) that identifies when the BWT has been violated.

- Block guard time
  - The block guard time (BGT) is the minimum delay between the start bits of the last character of a transmitted block and the first character of the next received block. The BGT must be greater than a value that is programmable. The SIM module provides a 16-bit block guard timer that identifies when the BGT has been violated.
- Error detection code
  - T=1 cards can specify LRC or CRC error detection codes to be used. The SIM module provides hardware support for LRC and CRC operations.

### 28.5.3 Using SIM Transmitter

Once the SIM is properly configured (such as, SIM\_PCRn[STEN] set, correct baud rate, and correct data format), enable the transmitter by setting SIM\_EN[TXEN]. If data was previously written to the transmit FIFO, the transmitter begins to send the first character. If no data is written to the transmit FIFO before enabling the transmitter, then the transmitter waits until the first character is written before beginning transmission. Clearing SIM\_EN[TXEN] while the transmitter is in operation, halts any transmission in progress, flushes the transmit FIFO, and resets the transmit state machine. Data can be written to the transmit FIFO at any time.

The transmit data threshold flag, SIM\_TSR[TDTF] is used to determine when the transmit FIFO has reached a given threshold value. This flag creates an interrupt if SIM\_IMR[TDTFM] is cleared. To control at which point TDTF is set, program the transmit data threshold, SIM\_TTHR[TDT]. If the number of bytes remaining in the transmit FIFO is equal to or less than the value set by TDT, TDTF is set.

#### NOTE

A value of zero in TDT implies that the transmit FIFO must be empty to trigger TDTF.

The value in TDT can be changed at any time to alter this threshold level. The comparison between the number of remaining bytes in the transmit FIFO and the value set by TDT is continuously updated so that any change in either will be immediately reflected in the state of TDTF. Unlike the RDRF flag for the receive FIFO, TDTF is latched and remains set until the software writes a one to SIM\_TSR[TDTF]. For instance, if TDT is set to 0x5, and there are six bytes remaining in the FIFO, changing TDT to 0x6 immediately causes TDTF to set. However, setting TDT back to 0x5 does not cause TDTF to clear.

The standard flow for transmitting bytes from the SIM card is to:

1. Set TDT[3:0] to the appropriate value
2. Write up to 16 bytes to the transmit FIFO
3. Enable the transmitter
4. Wait for TDTF to cause an interrupt (TDTFM clear)
5. Write additional bytes to the transmit FIFO

### 28.5.3.1 Transmit Data Formats

There are two possible data formats the SIM module uses when transferring data to the card: inverse or direct convention. The format used depends on the state of inverse convention bit, SIM\_FORMAT[IC]. Software can set the IC bit or it can be set automatically by hardware when using initial character mode.

### 28.5.3.2 Transmit NACK

The SIM transmitter can respond to NACKs created by the SIM card. A NACK is decoded if the SIM card creates a logic low level on the SIM receive pin during the stop bit time at the end of a transmitted byte. To prevent a situation where the SIM interface is stalled by an infinite number of NACK pulses on a given byte, the SIM module can be configured to limit the number of times it will respond to NACKs by SIM\_TTHR[XTH]. If the threshold is reached, a transmit threshold error, SIM\_TSRE[XTE], is asserted.

When XTE is set, the SIM transmitter is halted, all pending transfers are aborted, and the TC, ETC, and TFE flags are set. All bytes remaining in the transmit FIFO are lost. There is no way to restart the transmission on the next byte in the FIFO. The transmitter remains frozen until XTE is cleared by software by writing a one to SIM\_TTHR[XTE]. The XTE flag can generate an interrupt if SIM\_TTHR[XTM] is cleared.

You can disable the detection of NACKs from the SIM card by setting XTH to 0x0. Setting XTH to 0x1, disables all retransmissions while still setting XTE on the first NACK received. In general, XTE is set on the NACK that causes the threshold to be reached. This final NACK does not cause a retransmission, whereas all previous NACKs do.

### 28.5.3.3 Transmit Guard Time

The time between data bytes sent from the SIM transmitter can be altered using the transmit guard time control. By default, the minimum time between start bits of successive transmitted bytes is 12 ETUs (a start bit, eight data bits, a parity bit, and two stop bits). The number of stop bits (idle bits) can be extended by an integer number of ETUs. The number of additional ETUs can be programmed directly into SIM\_TGCR[GETU]. Setting GETU bits to 0xFF configures the SIM transmitter to use only one stop bit for each character transmission.

### 28.5.3.4 Using the SIM Transmitter with T=1 SIM Cards

T=1 SIM cards present several requirements beyond standard T=0 cards. The features provided to meet the requirements that pertain to the SIM transmitter are as follows:

- 11 ETU characters
  - The SIM module transmitter has a programmable guard time register that allows the programmer to specify the number of ETUs between character transmissions. Programming a value of 255 (0xFF) in SIM\_TGCR[GETU] sets the number of ETUs per character transmitted to 11.
- Character waiting time
  - The character waiting time (CWT) is the time between the start bits of two consecutive characters. The value of CWT can range from 12–32,779 ETU. The time between transmitted

characters is controlled by the programmable guard time in SIM\_TGCR. However, the time between the last byte in the transmit FIFO, and the next transmitted byte can be largely affected by software response time to the transmit interrupts. The SIM transmitter provides a transmit FIFO threshold (TDTF) interrupt to signal the system when the expected number of characters have been transmitted from the transmit FIFO. The minimum CWT is achieved only if the software can respond to the TDTF interrupt and write new data to the transmit FIFO before the last character in the transmit FIFO has been sent.

- Block waiting time
  - The block waiting time (BWT) is the maximum time between the start bits of the last character of a transmitted block and the first character of the next received block. The value of BWT is always greater than 1800 ETU. The SIM transmitter provides a general purpose counter that can be used to track the BWT. The BWT is purely determined by software response time to the transmit interrupts.
- Block guard time
  - The block guard time (BGT) is the minimum delay between the start bits of the last character of a transmitted block and the first character of the next received block. The value of BGT is 22 ETU. The SIM module supports the BGT by providing an interrupt when the last byte is received, and transmitting within 2 ETU after SIM\_EN[TXEN] is set. The BGT is determined by the speed at which the software can react to an interrupt and enable the transmitter.
- Error detection code
  - T=1 cards can specify LRC or CRC error detection codes to be used. The SIM module provides hardware support for both LRC and CRC operation.

## 28.5.4 Suggested Programming Model

This section describes the suggested programming model for supporting T=1, T=0, and known special cards using the SIM module. This should be used as a rough guide for configuring the SIM module for SIM cards specified by ISO 7816-3 and EMV. Some details are not addressed. Other uses for some of the SIM features are not included (for example, GP counter for some ISO timing requirements).

### 28.5.4.1 Detecting Answer To Reset (ATR)

The first step to communicating with a SIM card is providing power and a clock signal to the card. Once the card is detected as present (using the presence detect features or some other method), the SIM card should be powered up according to the power-up sequence specified in the ISO 7816-3 specification.

1. Apply voltage to the SIM card by setting SIM\_PCRn[SVEN].
2. Set the SIM\_ODCR[ODPn] bit to enable the SIM\_DATA pin.
3. Set the SIM\_PCRn[3VOLT] bit to enable the data to the SIM as bidirectional through the SIM\_DATA pin.
4. Select the appropriate clock frequency for the SIM card by programming SIM\_PRE.
5. Enable the clock to the SIM card by setting SIM\_PCRn[SCEN].
6. Remove the card from reset by setting SIM\_PCRn[SRST].

The first communication between the SIM card and the SIM module is a block of data sent from the SIM card to the SIM module after the card is powered and the card reset is removed. This block is called the answer to reset (ATR). To receive the ATR, the SIM module should be configured for 12 ETU character reception. According to the ISO 7816-3 spec, both T=0 and T=1 cards communicate initially using 12 ETU character durations.

### **NOTE**

We are aware of some card manufacturers that communicate at 11.5 ETU character durations (Geldkarte). This complicates the initial card detection sequence shown below.

7. Clear SIM\_TGCR[RCVR11].
8. Set SIM\_CR[ANACK] to enable NACK generation.

The ISO 7816-3 spec allows the SIM module to NACK any communication errors that occur during the initial communication at 12 ETU.

### **NOTE**

The Europay Mastercard and VISA (EMV) cards are similar to T=1, but do not allow the SIM module to NACK during the initial communication. This again complicates the initial card detection sequence shown below.

9. Enable RDRF and OEF interrupts by setting SIM\_IMR[RIM, OIM] to notify when characters are received.
10. Set desired threshold for received characters before generating an interrupt by writing SIM\_RTHR[RDT].
11. Set initial character mode by setting SIM\_CR[ICM].

This causes the hardware to identify the first valid character sent during the ATR as an initial character. This character automatically configures the hardware for the data convention used by the SIM card.

The ISO 7816-3 spec requires that SIM cards meet certain timing restrictions. One of these is the time from the deassertion of the card reset to the beginning of the ATR sequence. The SIM module general purpose counter can verify that the SIM card begins its ATR within the 400 to 40,000 clock cycle range.

12. Set general purpose counter comparator to 0x9C40 using SIM\_GPCNT.
13. Enable the general purpose counter interrupt by clearing SIM\_IMR[GPCNTM].
14. Enable the general purpose counter by programming SIM\_CR[GPCNT\_CLKSEL] to 01 so the card clock is used for counting.

The ISO7816-3 spec states that the maximum allowed time between two characters during the ATR is 9600 ETUs (initial waiting time). The character wait time (CWT) counter should be setup to detect any errors for this condition.

15. Set CWT counter comparator to 9600 using SIM\_CWTR.
16. Enable the CWT counter interrupt by clearing SIM\_IMR[CWTM].
17. Enable the CWT counter by setting SIM\_CR[CWTEN].

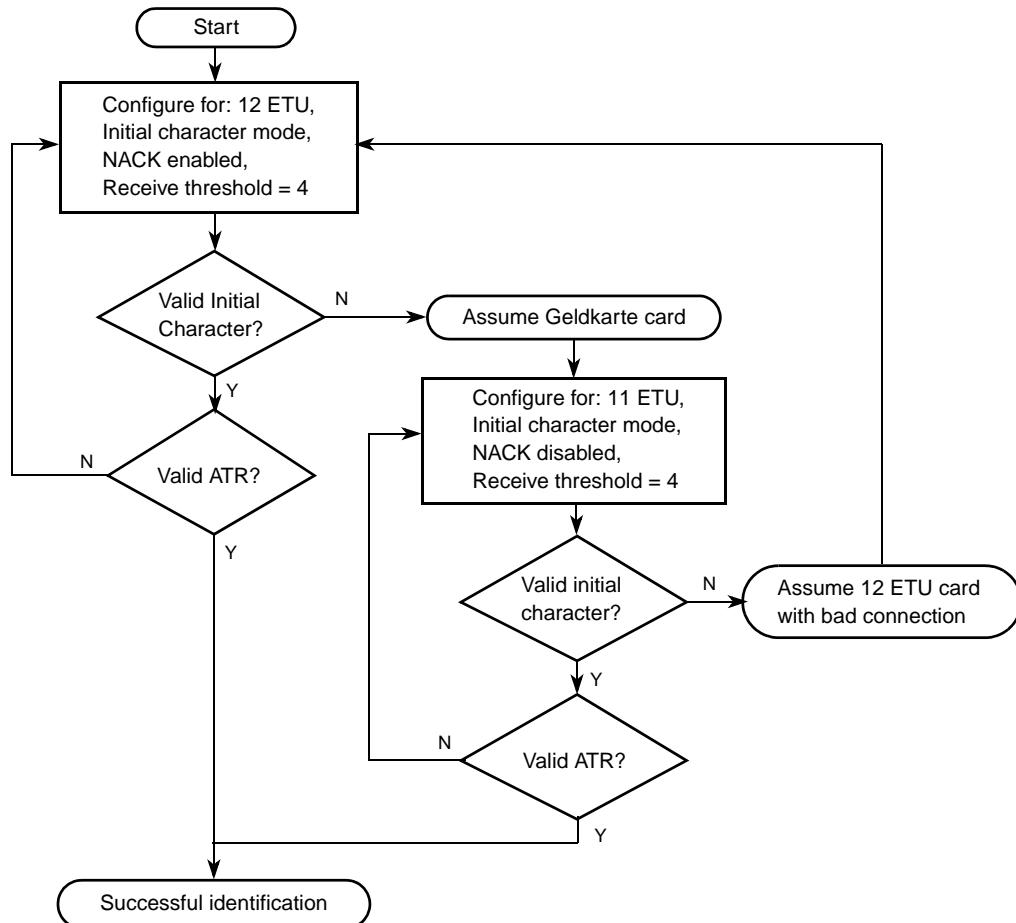
The last step in preparing for ATR reception is to enable the receiver.

18. Set SIM\_EN[RXEN].

The SIM module generates interrupts once a threshold number of characters is received. The software should react to these interrupts and read the characters from the receive FIFO (SIM\_RBUF $n$ ) until the complete ATR is received. If a general purpose counter interrupt occurs before the final ATR character is received, then the card should be deactivated according to ISO 7816-3. Otherwise, once a valid ATR is received, the software knows from the ATR information the specific characteristics for this card (refer to the ISO 7816-3 spec for details).

### 28.5.4.2 Programming Considerations for Geldkarte Cards

Geldkarte SIM cards do not send the ATR in 12 ETU mode or support NACKs. This creates an issue with detecting a valid ATR. As a result, the software and hardware do not know how to begin communication. Basically, if the card fails to send a valid ATR on the first try, disable NACKs, configure the SIM module for 11 ETU, and try again. The software should toggle between 12 and 11 ETU modes with and without NACKs enabled until a valid ATR is received, or the number of attempts to communicate passes a predetermined error threshold. [Figure 28-42](#) shows the flow chart for the suggested Geldkarte-compliant SIM initialization.



**Figure 28-42. Suggested T=1, EMV, Geldkarte Compliant SIM Initialization**

### 28.5.4.3 Programming Considerations for T=0 SIM Cards

If using a T=0 card, software should adjust the following parameters according to the information in the ATR:

1. Adjust the baud rate by changing the values of SIM\_CR[BAUD\_SEL, SAMPLE12].
2. Adjust the guard time between characters by changing the value of SIM\_TGCR[GETU].
3. Adjust NACK capability by modifying the values of SIM\_CR[ONACK, ANACK].
4. Adjust the stop clock polarity by modifying the values of SIM\_PCRn[SCSP].
5. Adjust the level of transmit NACK re-transmissions allowed by modifying SIM\_TTHR[XTH].
6. Adjust the level for the receive NACK threshold by modifying SIM\_RTHR[RTH].

If a negotiation with the SIM card is desired, the software sends a PPS response to the SIM card. To send the response, the following steps must be performed:

1. Set the desired transmit FIFO threshold level by writing SIM\_TTHR[TDT].
2. Write the characters to be sent as response (max 16) to the transmit FIFO using SIM\_TBUF $n$ .
3. Clear all transmit interrupt flags in SIM\_TS by writing a one to them.
4. Enable the desired transmit interrupts by clearing the mask bits in SIM\_IMR. If more than 16 characters are sent, use the TDTF interrupt to signify when to write more characters to the transmit FIFO. This results in the most efficient transfer times to the SIM card.
5. Enable the transmitter by setting SIM\_EN[TXEN].

At this point, the SIM module transmits the characters in the transmit FIFO. If more than 16 characters are sent, the transmit threshold interrupt is set when the threshold number of characters are remaining in the FIFO. The software can then write an additional number of characters without interrupting transmission to the SIM card.

Once the transmission is complete, the SIM module must be completely configured for standard operation with the T=0 SIM card. The software can continue to service RDRF interrupts for received characters, and TDTF interrupts for transmitted characters.

### 28.5.4.4 Programming Considerations for T=1 SIM Cards

If using a T=1 card, software should adjust the following parameters according to the information in the ATR:

1. Adjust the baud rate by changing the values of SIM\_CR[BAUD\_SEL, SAMPLE12].
2. Adjust the guard time between characters by changing the value of SIM\_TGCR[GETU]. Setting GETU to 0xFF configures the SIM transmitter for 11 ETU transmissions.
3. Disable NACK capability by clearing SIM\_CR[ONACK, ANACK]. T=1 cards do not allow NACKs.
4. Adjust the stop clock polarity by modifying the values of SIM\_PCRn[SCSP].
5. Set character wait time counter comparator to value specified in the ATR by using SIM\_CWTR.
6. Enable the character wait time counter interrupt by clearing SIM\_IMR[CWTM].
7. Enable the character wait time counter by setting SIM\_CR[CWTEN].

8. Enable CRC or LRC error checking according to the ATR information by setting either SIM\_CR[CRCEN or LRCEN]. Never set these bits at the same time.

For T=1 cards, the ATR is sent using a T=0 type of structure (12 ETU, no LRC or CRC). If a negotiation with the SIM card is desired, software must send a PPS response to the SIM card. Otherwise, the protocol is initiated with a block transfer from the SIM module. To send the response or the first block, the following steps must be performed:

1. Set the desired transmit FIFO threshold level by writing to SIM\_TTHR[TDT].
2. Write the characters to be sent as response (max 16) to the transmit FIFO using SIM\_TBUF $n$ .
3. Clear all transmit interrupt flags in SIM\_TSР by writing a one to them.
4. Enable the transmit interrupts desired by clearing the mask bits in SIM\_IMR. If more than 16 character are sent, use the TDTF interrupt to signify when to write more characters to the transmit FIFO. This results in the most efficient transfer times to the SIM card.
5. Enable transmission of the error checking characters (LRC or CRC) by setting SIM\_CR[XMT\_CRC\_LRC].

#### **NOTE**

If the card supports PPS, the software may not be allowed to send the LRC/CRC information until the PPS exchange is completed. If so, do not set the XMT\_CRC\_LRC bit during the PPS exchange.

6. Enable the transmitter by setting SIM\_EN[TXEN].

At this point, the SIM module transmits the characters in the transmit FIFO. If more than 16 characters are sent, the transmit threshold interrupt is set when the threshold number of characters are remaining in the FIFO. The software can then write an additional number of characters without interrupting transmission to the SIM card.

Once the transmission is complete, the SIM module must be completely configured for standard operation with the T=1 SIM card. The software can continue to service RDRF interrupts for received characters, and TDTF interrupts for transmitted characters.

# Chapter 29

## Analog Digital Converter (ADC)

### 29.1 Introduction

The analog-to-digital (ADC) converter block consists of two separate analog to digital converters, each with four analog inputs and their own sample and hold (S/H) circuit. A common digital control module configures and controls the converters. The module is instantiated as a dual 12-bit ADC with both converters sharing a common voltage reference and control block. This is illustrated in Figure 29-1.

#### 29.1.1 Features

ADC characteristics include:

- 12-bit resolution
- Maximum ADC clock frequency is 10 MHz with 100 ns period
- Sampling rate up to 4 million samples per second<sup>1</sup>
- Single conversion time of 8.5 ADC clock cycles ( $8.5 \times 100 \text{ ns} = 708.30 \text{ ns}$ )
- Additional conversion time of 6 ADC clock cycles ( $6 \times 100 \text{ ns} = 499.98 \text{ ns}$ )
- Eight conversions in 26.5 ADC clock cycles ( $26.5 \times 100 \text{ ns} = 2.21 \mu\text{s}$ ) using parallel mode
- Can be synchronized to the PWM via the PWM\_SYNC0/1 input signal
- Sequentially scan and store up to 8 measurements
- While operating simultaneously and in parallel, scan and store up to four measurements on each ADC converter
- While operating asynchronously in parallel, scan and store up to four measurements on each ADC converter
- Optional interrupts at end of scan if an out-of-range limit is exceeded or at zero crossing
- Optional sample correction by subtracting a pre-programmed offset value
- Signed or unsigned result
- Single-ended or differential inputs
- PWM outputs with hysteresis for three of the analog inputs

---

1. While in loop mode, the time between each conversion is six ADC clock cycles (499.98 ns). Using simultaneous conversion, two samples can be obtained in 499.98 ns. Samples per second is calculated according to 499.98 ns per two samples or 4,000,160 samples per second.

## 29.2 Block Diagram

Figure 29-1 illustrates the dual ADC configuration.

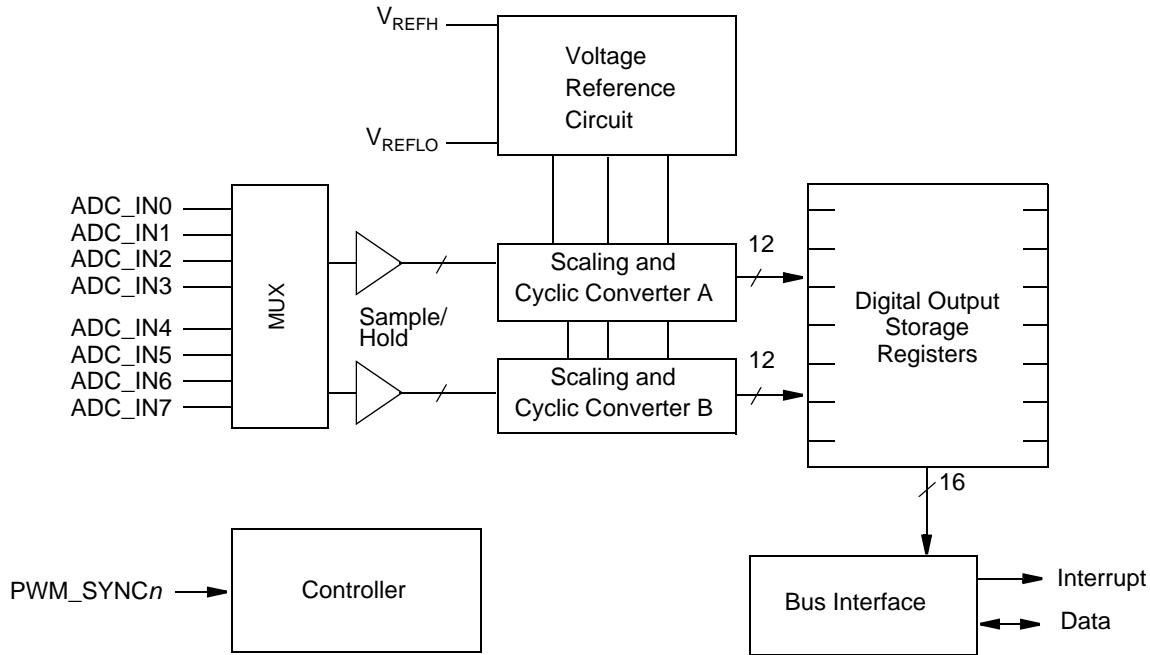


Figure 29-1. Dual ADC Block Diagram

## 29.3 External Signal Description

Table 29-1 shows the ADC signal interface.

Table 29-1. ADC Signal Description

Signal	I/O	Function
ADC_IN[7:0]	I	Analog input to be converted.
ADC_VDD	—	Dedicated power supply pins to reduce noise coupling and to improve accuracy. The power provided to these pins is suggested to come from a low noise filtered source. Connect uncoupling capacitors between ADC_VDD and ADC_VSS. ADC_VSS is shared among the analog and digital circuitry.
ADC_VSS	—	<b>Note:</b> On this device, the ADC reference voltages are internally connected to this supply.

## 29.4 Memory Map/Register Definition

Do not reconfigure the ADC during scan operations, as this can lead to unpredictable results. The following accesses are allowed during scan operations:

- Reading status
- Reading conversion results
- Clearing interrupts

- Clearing zero crossing and limit status flags
- Starting/stopping scans using START and STOP bits.

**Table 29-2. ADC Memory Map**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC09_4000	Control register 1 (ADC_CR1)	16	R/W	0x5005	<a href="#">29.4.1/29-3</a>
0xFC09_4002	Control register 2 (ADC_CR2)	16	R/W	0x5080	<a href="#">29.4.2/29-6</a>
0xFC09_4004	Zero crossing control register (ADC_ZCCR)	16	R/W	0x0000	<a href="#">29.4.3/29-7</a>
0xFC09_4006	Channel list register 1 (ADC_LST1)	16	R/W	0x3210	<a href="#">29.4.4/29-7</a>
0xFC09_4008	Channel list register 2 (ADC_LST2)	16	R/W	0x7654	<a href="#">29.4.4/29-7</a>
0xFC09_400A	Sample disable register (ADC_SDIS)	16	R/W	0x0000	<a href="#">29.4.5/29-9</a>
0xFC09_400C	Status register (ADC_SR)	16	R/W	0x0000	<a href="#">29.4.6/29-9</a>
0xFC09_400E	Limit status register (ADC_LSR)	16	R/W	0x0000	<a href="#">29.4.7/29-11</a>
0xFC09_4010	Zero crossing status register (ADC_ZCSR)	16	R/W	0x0000	<a href="#">29.4.8/29-12</a>
0xFC09_4012 + 2×n	Result registers (ADC_RSLT $n$ ) $n = 0\text{--}7$	16	R/W	0x0000	<a href="#">29.4.9/29-13</a>
0xFC09_4022 + 2×n	Low limit 0–7 registers (ADC_LLMT $n$ ) $n = 0\text{--}7$	16	R/W	0x0000	<a href="#">29.4.10/29-13</a>
0xFC09_4032 + 2×n	High limit 0–7 registers (ADC_HLMT $n$ ) $n = 0\text{--}7$	16	R/W	0x7FF8	<a href="#">29.4.10/29-13</a>
0xFC09_4042 + 2×n	Offset 0–7 registers (ADC_OFSt $n$ ) $n = 0\text{--}7$	16	R/W	0x0000	<a href="#">29.4.11/29-14</a>
0xFC09_4052	Power control register (ADC_PWR)	16	R/W	0x3D8F	<a href="#">29.4.12/29-15</a>
0xFC09_4054	Calibration register (ADC_CAL)	16	R/W	0x0000	<a href="#">29.4.13/29-18</a>
0xFC09_4056	Power control register 2 (ADC_PWR2)	16	R/W	0x0005	<a href="#">29.4.14/29-19</a>
0xFC09_4058	Conversion divisor register (ADC_DIV)	16	R/W	0x0505	<a href="#">29.4.15/29-19</a>
0xFC09_405A	Auto-standby divisor register (ADC_ASDIV)	16	R/W	0x0137	<a href="#">29.4.16/29-20</a>

### 29.4.1 ADC Control Register 1 (ADC\_CR1)

This register controls all types of scans except parallel scans in the B converter when ADC\_CR2[SIMULT] is cleared. Non-simultaneous parallel scan modes allow independent parallel scanning in the A and B converter. Bits 14, 13, 12, and 11 in ADC\_CR2 register control converter B scans in non-simultaneous parallel scan modes.

## Analog Digital Converter (ADC)

Address: 0xFC09\_4000

Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	DMA EN	STOP 0	0 START 0	SYNC 0	EOS IE0	ZC IE	LLMT IE	HLMT IE	CHNCFG				0	SMODE		
W	0	1	0	1	0	0	0	0	0	0	0	0	0	1	0	1
Reset																

Figure 29-2. Control Register 1 (ADC\_CR1)

Table 29-3. ADC\_CR1 Field Descriptions

Field	Description
15 DMAEN	DMA enable. Enables DMA use, instead of interrupts. Also when set, all bits and signals related to the interrupt interface are redundant. This bit is used for both the converters 0 DMA disabled 1 DMA enabled
14 STOP0	ADC stop enable. When set, the current scan is stopped and no further scans can begin. Any further SYNC0 input pulses or writes to the START0 bit are ignored until this bit is cleared. After the ADC is in stop mode, you can change the result registers, and these changes are treated as if the analog core supplied the data. Therefore, limit checking, zero crossing, and associated interrupts can occur when authorized. 0 Normal operation 1 Stop mode <b>Note:</b> This is not the same as the processor's low-power stop mode.
13 START0	Start conversion. A scan is started by setting the START0 bit. This is a write-only bit. Writing one to the START0 bit again while the scan remains in process is ignored. 0 No action 1 Start command is issued <b>Note:</b> The ADC must be in a stable power configuration prior to writing this bit. Refer to the functional description of power modes for further details.
12 SYNC0	SYNC0 enable. If set, a conversion may be initiated by asserting a positive edge on the SYNC0 input. Any subsequent SYNC0 input pulses while the scan remains in process are ignored. 0 Scan is initiated by setting the START0 bit only 1 Use a SYNC0 input pulse or START0 bit to initiate a scan <b>Note:</b> The ADC must be in a stable power mode prior to SYNC0 input assertion. Refer to the functional description of power modes for further details. <b>Note:</b> In once-scan modes, only the first SYNC0 input pulse is honored. Subsequent SYNC0 input pulses are ignored until the SYNC0 input is re-armed by writing to ADC_CR1. This can be done at any time, including while the scan remains in process.
11 EOSIE0	End of scan interrupt 0 enable. Enables an EOSI0 interrupt to be generated upon completion of the scan. For loop scan modes, the interrupt triggers after the completion of each iteration of the loop. 0 Interrupt disabled 1 Interrupt enabled
10 ZCIE	Zero crossing interrupt enable. Enables the zero crossing interrupt if the current result value has a sign change from the previous result as configured by the ADC_ZCCR register. 0 Interrupt disabled 1 Interrupt enabled

**Table 29-3. ADC\_CR1 Field Descriptions (continued)**

Field	Description																													
9 LLMTIE	<p>Low limit interrupt enable. Enables the low limit exceeded interrupt when the current result value is less than the low limit register value. The raw result value is compared to ADC_LLMT, before the offset register value is subtracted.</p> <p>0 Interrupt disabled 1 Interrupt enabled</p>																													
8 HLMTIE	<p>High limit interrupt enable. Enables the high limit exceeded interrupt if the current result value is greater than the high limit register value. The raw result value is compared to ADC_HLMT, before the offset register value is subtracted.</p> <p>0 Interrupt disabled 1 Interrupt enabled</p>																													
7–4 CHNCFG	<p>Channel configure. Configures the analog inputs for single-ended or differential conversions.</p> <table border="1"> <thead> <tr> <th>CHNCFG</th> <th>Inputs</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>xxx1</td> <td>ADC_IN0 – ADC_IN1</td> <td>Differential pair (ADC_IN0 is + and ADC_IN1 is –)</td> </tr> <tr> <td>xxx0</td> <td>ADC_IN0 – ADC_IN1</td> <td>Single-ended inputs</td> </tr> <tr> <td>xx1x</td> <td>ADC_IN2 – ADC_IN3</td> <td>Differential pair (ADC_IN2 is + and ADC_IN3 is –)</td> </tr> <tr> <td>xx0x</td> <td>ADC_IN2 – ADC_IN3</td> <td>Single-ended inputs</td> </tr> <tr> <td>x1xx</td> <td>ADC_IN4 – ADC_IN5</td> <td>Differential pair (ADC_IN4 is + and ADC_IN5 is –)</td> </tr> <tr> <td>x0xx</td> <td>ADC_IN4 – ADC_IN5</td> <td>Single-ended inputs</td> </tr> <tr> <td>1xxx</td> <td>ADC_IN6 – ADC_IN7</td> <td>Differential pair (ADC_IN6 is + and ADC_IN7 is –)</td> </tr> <tr> <td>0xxx</td> <td>ADC_IN6 – ADC_IN7</td> <td>Single-ended inputs</td> </tr> </tbody> </table>			CHNCFG	Inputs	Description	xxx1	ADC_IN0 – ADC_IN1	Differential pair (ADC_IN0 is + and ADC_IN1 is –)	xxx0	ADC_IN0 – ADC_IN1	Single-ended inputs	xx1x	ADC_IN2 – ADC_IN3	Differential pair (ADC_IN2 is + and ADC_IN3 is –)	xx0x	ADC_IN2 – ADC_IN3	Single-ended inputs	x1xx	ADC_IN4 – ADC_IN5	Differential pair (ADC_IN4 is + and ADC_IN5 is –)	x0xx	ADC_IN4 – ADC_IN5	Single-ended inputs	1xxx	ADC_IN6 – ADC_IN7	Differential pair (ADC_IN6 is + and ADC_IN7 is –)	0xxx	ADC_IN6 – ADC_IN7	Single-ended inputs
CHNCFG	Inputs	Description																												
xxx1	ADC_IN0 – ADC_IN1	Differential pair (ADC_IN0 is + and ADC_IN1 is –)																												
xxx0	ADC_IN0 – ADC_IN1	Single-ended inputs																												
xx1x	ADC_IN2 – ADC_IN3	Differential pair (ADC_IN2 is + and ADC_IN3 is –)																												
xx0x	ADC_IN2 – ADC_IN3	Single-ended inputs																												
x1xx	ADC_IN4 – ADC_IN5	Differential pair (ADC_IN4 is + and ADC_IN5 is –)																												
x0xx	ADC_IN4 – ADC_IN5	Single-ended inputs																												
1xxx	ADC_IN6 – ADC_IN7	Differential pair (ADC_IN6 is + and ADC_IN7 is –)																												
0xxx	ADC_IN6 – ADC_IN7	Single-ended inputs																												
	<p>Differential measurements:</p> <ul style="list-style-type: none"> <li>return the max value (<math>2^{12} - 1</math>) when the positive input is <math>V_{REFH}</math> and the negative input is <math>V_{REFLO}</math></li> <li>return zero when the + input is at <math>V_{REFLO}</math> and the negative input is at <math>V_{REFH}</math></li> <li>scale linearly between based on the voltage difference between the two signals</li> </ul> <p>Single-ended measurements:</p> <ul style="list-style-type: none"> <li>return the max value when the input is at <math>V_{REFH}</math></li> <li>return zero when the input is at <math>V_{REFLO}</math></li> <li>scale linearly between based on the amount by which the input exceeds <math>V_{REFLO}</math></li> </ul>																													
3	Reserved, must be cleared.																													
2–0 SMODE	<p>Scan mode control. This bit:</p> <ul style="list-style-type: none"> <li>Determines whether the slots perform one long sequential scan or two shorter parallel scans, each performed by one of the two converters</li> <li>Controls how these scans are initiated and terminated</li> <li>Controls if the scans are performed once or repeatedly</li> </ul> <p>See <a href="#">Section 29.5.1, “Scan Modes”</a>, for detailed descriptions of these modes.</p> <p>000 Once sequential 001 Once parallel 010 Loop sequential 011 Loop parallel 100 Triggered sequential 101 Triggered parallel 110 Reserved 111 Reserved</p>																													

## 29.4.2 ADC Control Register 2 (ADC\_CR2)

ADC\_CR2[14:11] and the SYNC1 module input only control converter B during parallel scan modes when SIMULT = 0 (non-simultaneous parallel scan modes).

Address: 0xFC09\_4002

Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	STOP	0	SYNC	EOS	0	0	0	SIM	0	0	0	0	0	0	0
W		1	STAR	1	IE1				ULT							
Reset	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0

Figure 29-3. Control Register 2 (ADC\_CR2)

Table 29-4. ADC\_CR2 Field Descriptions

Field	Description
15	Reserved, must be cleared.
14 STOP1	During parallel scan modes when SIMULT = 0, setting STOP1 stops parallel scans in the B converter and prevents new ones from starting. Any further SYNC1 input pulses or writes to the START1 bit are ignored until the STOP1 bit is cleared. After the ADC is in stop mode, you can change the B converter ADC_RSLT $n$ , and these changes are treated as if the analog core supplied the data. Therefore, limit checking, zero crossing, and associated interrupts can occur when authorized. 0 Normal operation 1 Stop command issued <b>Note:</b> This is not the same as the processor's low power stop mode.
13 START1	Start conversion 1. During parallel scan modes when SIMULT = 0, a B converter parallel scan is started by writing one to the START1 bit. This is a write-only bit. Writing one to the START1 bit again while the scan remains in process, is ignored. 0 No action 1 Start a B converter parallel scan <b>Note:</b> The ADC must be in a stable power configuration prior to writing the START bit. Refer to the functional description of power modes for further details.
12 SYNC1	SYNC1 enable. During parallel scan modes when SIMULT = 0, setting SYNC1 permits a B converter parallel scan to be initiated by asserting the SYNC1 input for at least one ADC clock cycle. Any SYNC1 input pulses while the scan remains in process are ignored. 0 B converter parallel scan is initiated by setting the START1 bit only 1 Use a SYNC1 input pulse or START1 bit to initiate a B converter parallel scan <b>Note:</b> The ADC must be in a stable power mode prior to SYNC1 input assertion. Refer to the functional description of power modes for further details. <b>Note:</b> In once-scan modes, only the first SYNC1 input pulse is honored. Subsequent SYNC1 input pulses are ignored until the SYNC1 input is re-armed by writing to ADC_CR2. This can be done at any time, including while the scan remains in process.
11 EOSIE1	End of scan interrupt 1 enable. During parallel scan modes when SIMULT = 0, this bit enables an EOS1 interrupt to be generated upon completion of a B converter parallel scan. For loop scan mode, the interrupt triggers upon the completion of each iteration of the loop. 0 Interrupt disabled 1 Interrupt enabled
10–8	Reserved, must be cleared.

**Table 29-4. ADC\_CR2 Field Descriptions (continued)**

Field	Description
7 SIMULT	Simultaneous mode. This bit only affects parallel scan modes. 0 Parallel scans in the A and B converters operate independently. Each converter's scan continues until its sample list is exhausted (four samples) or a disabled sample in its list is encountered. For loop parallel scan mode, each converter starts its next iteration when the previous iteration in that converter is complete and continues until the STOP bit for that converter is asserted. 1 Parallel scans in the A and B converters operate simultaneously and always result in pairs of simultaneous conversions. START0, STOP0, START1, and STOP1 control bits and the SYNC0 input are used to start and stop scans in both converters simultaneously. A scan ends in both converters when either converter encounters a disabled sample slot. When the parallel scan completes, the EOSI0 triggers if EOSIEN0 is set. The CIP0 status bit indicates a parallel scan is in process.
6–0	Reserved, must be cleared.

### 29.4.3 ADC Zero Crossing Control Register (ADC\_ZCCR)

ADC\_ZCCR monitors the selected channels and determines the direction of zero crossing triggering the optional interrupt. Zero-crossing logic monitors only the sign change between the current and previous sample. ZCE0 monitors the sample stored in ADC\_RSLT0 and bit ZCE7 monitors ADC\_RSLT7. When zero crossing is disabled for a selected ADC\_RSLT $n$ , sign changes are not monitored or updated in ADC\_ZCSR.

Zero crossing functionality is only available on the first eight conversions in sequential mode and only available on the first four conversions associated with each converter in parallel modes.

Address: 0xFC09\_4004

Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	ZCE7	ZCE6	ZCE5	ZCE4	ZCE3	ZCE2	ZCE1	ZCE0								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

**Figure 29-4. Zero Crossing Control Register (ADC\_ZCCR)****Table 29-5. ADC\_ZCCR Field Descriptions**

Field	Description
15–0 ZCE $n$	Zero crossing enable. For each channel $n$ setting the ZCE $n$ field allows detection of the indicated zero-crossing condition. 00 Disabled 01 Enabled for positive to negative sign change 10 Enabled for negative to positive sign change 11 Enabled for any sign change

### 29.4.4 Channel List $n$ Registers (ADC\_LST1–2)

The ADC\_LST $n$  registers contain an ordered list of the channels to be converted when the next scan is initiated. If all samples are enabled in ADC\_SDIS, a sequential scan of inputs proceeds in order of: SAMPLE0–7. If one of the parallel sampling modes is selected instead, the converter A sampling order is SAMPLE0–3 and the converter B sampling order is SAMPLE4–7.

## Analog Digital Converter (ADC)

In sequential conversion mode full functionality (offset subtraction and high/low limit checks) is only available on the first eight conversion slots, SAMPLE0–7. In parallel conversion mode full functionality is only available on the first four conversion slots of each channel, SAMPLE0–3 for converter A and SAMPLE4–7 for converter B.

Address: 0xFC09\_4006

Access: User read/write

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

Figure 29-5. Channel List Register 1 (ADC\_LST1)

Address: 0xFC09\_4008

Access: User read/write

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

Figure 29-6. Channel List Register 2 (ADC\_LST2)

Table 29-6. ADC\_LSTn Field Descriptions

Field	Description		
SAMPLEn	Selects the input channel to be sampled.		
	<b>SAMPLEn</b>	<b>Single Ended</b>	<b>Differential</b>
	000	ADC_IN0	ADC_IN0+, ADC_IN1-
	001	ADC_IN1	ADC_IN0+, ADC_IN1-
	010	ADC_IN2	ADC_IN2+, ADC_IN3-
	011	ADC_IN3	ADC_IN2+, ADC_IN3-
	100	ADC_IN4	ADC_IN4+, ADC_IN5-
	101	ADC_IN5	ADC_IN4+, ADC_IN5-
	110	ADC_IN6	ADC_IN6+, ADC_IN7-
	111	ADC_IN7	ADC_IN6+, ADC_IN7-
In sequential modes, the sample slots are converted in order from SAMPLE0 to SAMPLE7. Any sample slot may reference any analog input (may contain a binary value between 000–111).			
In parallel modes, converter A processes sample slots SAMPLE0–3 while converter B processes sample slots SAMPLE4–7. Because converter A only has access to analog inputs ADC_IN0–3, sample slots SAMPLE0–3 should only contain binary values between 000 and 011. Likewise, because converter B only has access to analog inputs ADC_IN4–7, sample slots SAMPLE4–7 should only contain binary values between 100 and 111. No damage occurs if this constraint is violated but results are undefined.			
When inputs are configured as differential pairs, a reference to either analog input in a differential pair by a sample slot implies a differential measurement on the pair. The details of single-ended and differential measurement are described in the CHNCFG bit field.			
Disable sample slots using the ADC_SDIS register.			

## 29.4.5 Sample Disable Register (ADC\_SDIS)

This register is an extension to ADC\_LST $n$ . It allows you to enable only the desired samples programmed in the SAMPLE0–7 fields. At reset all samples are enabled.

Address: 0xFC09\_400A

Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	DS7	DS6	DS5	DS4	DS3	DS2	DS1	DS0
W									0	0	0	0	0	0	0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 29-7. Sample Disable Register (ADC\_SDIS)

Table 29-7. ADC\_SDIS $n$  Field Descriptions

Field	Description
15–8	Reserved, must be cleared.
7–0 DS $n$	Disable sample. 0 Enable SAMPLE $n$ 1 Disable SAMPLE $n$ and all subsequent samples. Which samples are actually disabled depends on the conversion mode (sequential/parallel) and the value of SIMULT.

## 29.4.6 Status Register (ADC\_SR)

This register provides the current status of the ADC module. RDY $n$  bits are cleared by reading their corresponding result (ADC\_RSLT $n$ ) registers. HLMTI and LLMTI bits are cleared by writing one to all asserted bits in the limit status register (ADC\_LSR). Likewise, ZCI is cleared by writing one to all asserted bits in ADC\_ZCSR. The EOSIn bits are cleared by writing one to them.

Except for CIP0 and CIP1, the register bits are sticky. When set, they require some specific action to clear them. They are not cleared automatically on the next scan sequence.

Address: 0xFC09\_400C

Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	CIP0	CIP1	0	EOSI1	EOSI0	ZCI	LLMTI	HLMT	RDY							
W				w1c	w1c				0	0	0	0	0	0	0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

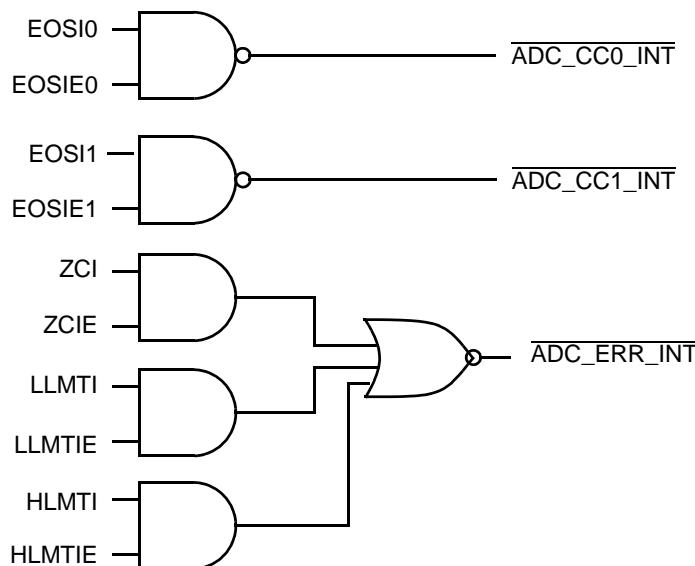
Figure 29-8. Status Register (ADC\_SR)

**Table 29-8. ADC\_SR Field Descriptions**

Field	Description
15 CIP0	Conversion in progress 0. 0 Idle state 1 A scan cycle is in progress. The ADC ignores all sync pulses or start commands. <b>Note:</b> This refers to any scan except a B converter scan in non-simultaneous parallel scan modes.
14 CIP1	Conversion in progress 1. 0 Idle state 1 A scan cycle is in progress. The ADC ignores all sync pulses or start commands. <b>Note:</b> This refers only to a B converter scan in non-simultaneous parallel scan modes.
13	Reserved, must be cleared.
12 EOSI1	End of scan interrupt 1. Indicates if a scan of analog inputs was completed since the last read of the status register or since a reset. If DMA is enabled (ADC_CR1[DMAEN] = 1), this bit is cleared by the DMA engine. Otherwise, EOSI1 is cleared by writing one to it. This bit cannot be set by software. 0 A scan cycle was not completed; no end of scan interrupt/DMA pending 1 A scan cycle was completed; end of scan interrupt/DMA pending In looping scan modes, this interrupt is triggered at the completion of each iteration of the loop. This interrupt is triggered only by the completion of a B converter scan in non-simultaneous parallel scan modes.
11 EOSI0	End of scan interrupt 0. Indicates if a scan of analog inputs was completed since the last read of the status register, or since a reset. If DMA is enabled (ADC_CR1[DMAEN] = 1), this bit is cleared by the DMA engine. Otherwise, EOSI1 is cleared by writing one to it. This bit cannot be set by software. EOSI0 is the preferred bit to poll for scan completion if interrupts are not enabled. 0 A scan cycle was not completed; no end of scan interrupt/DMA pending 1 A scan cycle was completed; end of scan interrupt/DMA pending In loop scan modes, this interrupt is triggered at the completion of each iteration of the loop mode. This interrupt is triggered upon the completion of any scan except for the completion of a B converter scan in non-simultaneous parallel scan modes.
10 ZCI	Zero crossing interrupt. If the respective offset register is configured by having a value greater than 0x0000, zero crossing checking is enabled. If ADC_OFSn register is 0x7FF8, the result always is less than or equal to zero. On the other hand, if ADC_OFSn is 0x0000, the result is always greater than or equal to zero and no zero crossing can occur because the sign of the result does not change. This interrupt asserts at the completion of an individual conversion which may or may not be the end of a scan. The ZCI bit is cleared by writing one to all active ADC_ZCSR[ZCSn] bits. 0 No ZCI interrupt request 1 Zero crossing encountered; interrupt pending if ZCIE is set
9 LLMTI	Low limit interrupt. If the respective low limit register is enabled by having a value other than 0x0000, low limit checking is enabled. This interrupt asserts at the completion of an individual conversion which may or may not be the end of a scan. The LLMTI bit is cleared by writing one to all active ADC_LSR[LLSn] bits. 0 No low limit interrupt request 1 Low limit exceeded; interrupt pending if LLMTIE is set

**Table 29-8. ADC\_SR Field Descriptions (continued)**

Field	Description
8 HLMTI	High limit interrupt. If the respective high limit register is enabled by having a value other than 0x7FF8, high limit checking is enabled. This interrupt asserts at the completion of an individual conversion which may or may not be the end of a scan.  The HLMTI bit is cleared by writing one to all active ADC_LSR[HLS $n$ ] bits. 0 No high limit interrupt request 1 High limit exceeded; interrupt pending if HLMTIE is set
7–0 RDY $n$	Ready sample 7–0. Indicate samples 7–0 are ready to be read. These bits are cleared after a read from the respective result register. The RDY $n$ bits are set as the individual channel conversions are completed. If polling the RDY $n$ bits to determine if a particular sample is completed, do not start a new scan until all enabled samples are done. 0 Sample not ready or was read 1 Sample ready to read

**Figure 29-9. ADC Interrupt**

### 29.4.7 Limit Status (ADC\_LSR) Register

ADC\_LSR latches in the result of the comparison between the result of the sample and the respective limit register (ADC\_HLMT $n$ , ADC\_LLMT $n$ ). For example, if the result for the channel programmed in SAMPLE0 is greater than the value programmed in ADC\_HLMT0, then HLS0 is set. An interrupt is generated if ADC\_CR1[HLMTIE] is set. An ADC\_LSR bit is cleared by writing a value of one to that specific bit. These bits are sticky. When set, they require a specific modification to clear them and are not cleared automatically by subsequent conversions.

## Analog Digital Converter (ADC)

Address: 0xFC09\_400E

Access: User read/write

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

Figure 29-10. Limit Status Register (ADC\_LSR)

Table 29-9. ADC\_LSRn Field Descriptions

Field	Description
15–8 HLS	High limit status 7–0. Indicates if the result for the channel is greater than the value programmed in the corresponding ADC_HLMTn register.
7–0 LLS	Low limit status 7–0. Indicates if the result for the channel is less than the value programmed in the corresponding ADC_HLMTn register.

### 29.4.8 Zero Crossing Status Register (ADC\_ZCSR)

ADC\_ZCSR latches the result of the comparison between the current result of the sample and the previous result of the same sample register. For example, if the result for the channel programmed in SAMPLE0 changes sign from the previous conversion and the respective ADC\_ZSR[ZCEn] field is 11 (any edge change) then the ZCS0 bit is set. An interrupt is generated if ADC\_CR1[ZCIE] is set. A bit can only be cleared by writing one to that specific bit. These bits are sticky. When set, they require a write to clear them. They are not cleared automatically by subsequent conversions.

Address: 0xFC09\_4010

Access: User read/write

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

Figure 29-11. Zero Crossing Status Register (ADC\_ZCSR)

Table 29-10. ADC\_ZCSRn Field Descriptions

Field	Description
15–8	Reserved, must be cleared.
7–0 ZCS	Zero crossing status. The zero crossing condition is determined by examining the ADC value after it is adjusted by the offset for the ADC_RSLTn register. Please see Figure 29-26. Each bit of the register is cleared by writing one to that bit. 0 A sign change did not occur in a comparison between the current channel n result and the previous channel result, or zero crossing control is disabled for channel n in ADC_ZCCR. 1 In a comparison between the current channel n result and the previous channel n result, a sign change condition occurred as defined in ADC_ZCCR.

#### 29.4.9 ADC Result Registers (ADC\_RSLT0–7)

These eight result registers contain the converted results from a scan. The SAMPLE0 result is loaded into ADC\_RSLT0, SAMPLE1 result in ADC\_RSLT1, etc. In a parallel scan mode, the first channel pair designated by SAMPLE0 and SAMPLE4 in register ADC\_LST1–2 are stored in ADC\_RSLT0 and ADC\_RSLT4 respectively.

## **NOTE**

When writing to this register, only the RESULT portion of the value written is used. This value is modified, illustrated in Figure 29-26 and the result of the subtraction is stored. The SEXT bit is only set as a result of this subtraction and is not directly determined by the value written.

Address: 0xFC09_4012 (ADC_RSLT0)	0xFC09_401A (ADC_RSLT4)	Access: User read/write
0xFC09_4014 (ADC_RSLT1)	0xFC09_401C (ADC_RSLT5)	
0xFC09_4016 (ADC_RSLT2)	0xFC09_401E (ADC_RSLT6)	
0xFC09_4018 (ADC_RSLT3)	0xFC09_4020 (ADC_RSLT7)	
R	15 14 13 12   11 10 9 8   7 6 5 4   3 2 1 0 SEXT   RESULT   TEST_DATA	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

**Figure 29-12. Result Registers (ADC\_RSLT $n$ )**

**Table 29-11. ADC\_RSLTn Field Descriptions**

Field	Description
15 SEXT	<p>Sign extend bit of the result. If all positive results are required, set the respective ADC_OFSn register to zero.</p> <p>0 Positive result 1 Negative result</p>
14–3 RESULT	<p>Digital result of the conversion. The ADC_RSLTn register can be interpreted as:</p> <ul style="list-style-type: none"> <li>Signed fractional number — the RSLTn can be used directly.</li> <li>Signed integer — you may right shift with sign extend, arithmetic shift right (ASR) three places and interpret the number, or accept the number as presented, knowing there are missing codes. The lower three bits of this register are always zero.</li> </ul> <p>Negative results (SEXT = 1) are always presented in two's compliment format. If the ADC_RSLTn registers must always be positive, clear ADC_OFSn.</p> <p>The interpretation of the numbers programmed into the ADC_LLMTn, ADC_HLMTn, and ADC_OFSn registers should match the interpretation of ADC_RSLTn.</p> <p>See <a href="#">Section 29.5.4, “ADC Data Processing</a> for a description of reading this field and how/when it can be used.</p>
2–0	Reserved, must be cleared.

#### 29.4.10 Low and High Limit Registers (ADC\_LLMT0–7 & ADCHLMT0–7)

Each ADC sample is compared against its corresponding limit registers. The comparison is based upon the raw conversion value with no offset correction applied. Please refer to [Figure 29-26](#).

## Analog Digital Converter (ADC)

The ADC tests if the result is greater than the high limit or less than the low limit. Disable limit checking by programming the respective high limit register with 0x7FF8 and the low limit register with 0x0000. At reset, limit checking is disabled.

Address: 0xFC09\_4022 (ADC\_LLMT0)

0xFC09\_402A (ADC\_LLMT4)

Access: User read/write

0xFC09\_4024 (ADC\_LLMT1)

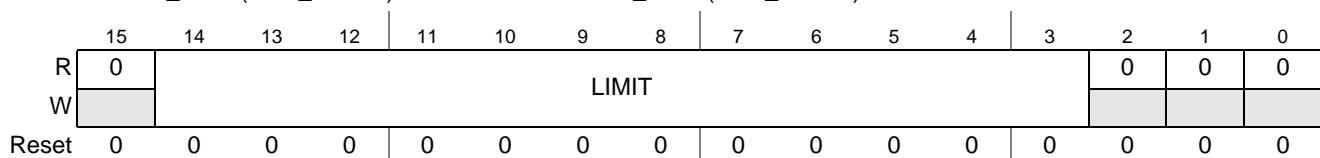
0xFC09\_402C (ADC\_LLMT5)

0xFC09\_4026 (ADC\_LLMT2)

0xFC09\_402E (ADC\_LLMT6)

0xFC09\_4028 (ADC\_LLMT3)

0xFC09\_4030 (ADC\_LLMT7)



**Figure 29-13. Low Limit Registers (ADC\_LLMTn)**

Address: 0xFC09\_4032 (ADC\_HLMT0)

0xFC09\_403A (ADC\_HLMT4)

Access: User read/write

0xFC09\_4034 (ADC\_HLMT1)

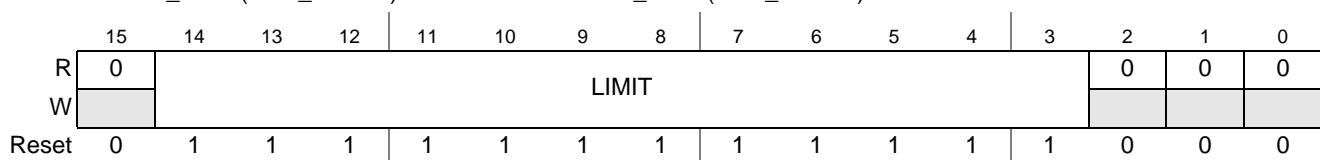
0xFC09\_403C (ADC\_HLMT5)

0xFC09\_4036 (ADC\_HLMT2)

0xFC09\_403E (ADC\_HLMT6)

0xFC09\_4038 (ADC\_HLMT3)

0xFC09\_4040 (ADC\_HLMT7)



**Figure 29-14. High Limit Registers (ADC\_HLMTn)**

**Table 29-12. ADC\_LLMTn & ADC\_HLMTn Field Descriptions**

Field	Description
15	Reserved, must be cleared.
14–3 LIMIT	High or low limit to compare the ADC sample result.
2–0	Reserved, must be cleared.

### 29.4.11 Offset Registers (ADC\_OFS0–7)

The ADC\_OFSn registers correct the ADC result before it is stored in the ADC\_RSLTn registers.

Address: 0xFC09\_4042 (ADC\_OFS0)

0xFC09\_404A (ADC\_OFS4)

Access: User read/write

0xFC09\_4044 (ADC\_OFS1)

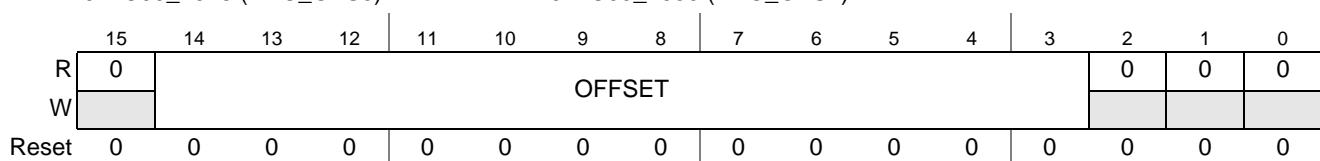
0xFC09\_404C (ADC\_OFS5)

0xFC09\_4046 (ADC\_OFS2)

0xFC09\_404E (ADC\_OFS6)

0xFC09\_4048 (ADC\_OFS3)

0xFC09\_4050 (ADC\_OFS7)



**Figure 29-15. Offset Registers (ADC\_OFSn)**

**Table 29-13. ADC\_OFSt Field Descriptions**

Field	Description
15	Reserved, must be cleared.
14–3 OFFSET	The offset value is subtracted from the ADC result. To obtain unsigned results, program the respective ADC_OFSt register with a value of 0x0000. This results in a range of 0x0000 to 0x7FF8.
qa	Reserved, must be cleared.

## 29.4.12 Power Control Register (ADC\_PWR)

This register controls the power management features of the ADC module. There are individual manual power down controls for the two ADC converters and the voltage reference generator. There are also five distinct power modes. The following terms are used to describe power modes and their related controls.

- Power down state — Each converter and the voltage reference generator can individually be put into a power down state. When powered down, the unit consumes no power. Results of scans referencing a powered down converter are undefined. The voltage reference generator and at least one converter must be powered up to use the ADC module.
- Manual power down controls — Each converter and the voltage reference generator have a manual power control bit capable of putting that component into the power down state. Converters have other mechanisms with the capacity to automatically put them into the power down state.
- Idle state — The ADC module is idle when neither of the two converters has a scan in process.
- Active state — The ADC module is active when at least one of the two converters has a scan in process.
- Current mode — Both converters share a common current mode. Normal current mode is used to power the converters at clock rates above 600 kHz.
- Start-up delay — Auto-power down modes cause a start-up delay when the ADC module goes between the idle and active states to allow time to switch clocks or power configurations.

See [Section 29.5.7, “Power Management”](#) for details of the five power modes and how to configure them. See [Section 29.5.12, “Interrupt Operation”](#) for a more detailed description of the clocking system and the control of current mode.

Address: 0xFC09_4052 (ADC_PWR)												Access: User read/write				
R	ASB	APD	PSTS 3	PSTS 2	PSTS 1	PSTS 0	PUDELAY						PD 3	PD 2	PD 1	PD 0
W	0	0	1	1	1	1	0	1	1	0	0	0	1	1	1	1
Reset																

**Figure 29-16. Power Control Register (ADC\_PWR)**

**Table 29-14. ADC\_PWR Field Descriptions**

<b>Field</b>	<b>Description</b>
15 ASB	<p>Auto standby mode enable. This bit is ignored if APD is set. When the ADC is idle, ASB mode selects the standby clock as the ADC clock source, putting the converters into standby current mode. At the start of any scan, the conversion clock is selected as the ADC clock and then a delay of PUDELAY ADC clock cycles is imposed for current levels to stabilize. After this delay, the ADC initiates the scan. When the ADC returns to the idle state, the standby clock is again selected and the converters revert to the standby current state.</p> <p>0 ASB mode disabled 1 ASB mode enabled</p> <p><b>Note:</b> This mode is not recommended for conversion clock rates at or below 200 kHz. Instead, clear ASB and APD, and use standby power mode (normal mode with a sufficiently slow conversion clock so standby current mode automatically engages). This provides the advantages of standby current mode while avoiding the clock switching and PUDELAY.</p> <p><b>Note:</b> Ideally, you should set this bit before clearing PD0 and PD1 to get the benefit of this power-saving mode right after power-up.</p>
14 APD	<p>Auto power down. Powers down the converters when not in use for a scan. When a scan is started in APD mode, a delay of PUDELAY ADC clock cycles is imposed during which the needed converters, if idle, are powered up. The ADC then initiates a scan equivalent to that when APD is not active. When the scan completes, the converters are powered down again.</p> <p>0 Auto power down mode is not active 1 Auto power down mode is active</p> <p><b>Note:</b> If ASB or APD is set while a scan is in progress, that scan is unaffected and the ADC waits to enter its low power state until after all conversions are complete and both ADCs are idle.</p> <p><b>Note:</b> ASB and APD are not useful in loop modes. The continuous nature of scanning means the low power state can never be entered.</p>
13 PSTS3	<p>Voltage reference converter B power status.</p> <p>0 Voltage reference circuit is currently powered up 1 Voltage reference circuit is currently powered down</p>
12 PSTS2	<p>Voltage reference converter A power status.</p> <p>0 Voltage reference circuit is currently powered up 1 Voltage reference circuit is currently powered down</p>
11 PSTS1	<p>Converter B Power status. PSTS1 is set immediately following a write of one to PD1. It is de-asserted PUDELAY ADC clock cycles after a write of 0 to PD1 if APD is cleared. This bit can be read as a status bit to determine when the ADC is ready for operation. During auto-powerdown mode, this bit indicates the current powered state of converter B.</p> <p>0 Converter B is currently powered up 1 Converter B is currently powered down</p>
10 PSTS0	<p>Converter A power status. PSTS0 is set immediately following a write of one to PD0. It is de-asserted PUDELAY ADC clock cycles after a write of 0 to PD0 if APD is cleared. This bit can be read as a status bit to determine when the ADC is ready for operation. During auto-powerdown mode, this bit indicates the current powered state of converter A.</p> <p>0 Converter A is currently powered up 1 Converter A is currently powered down</p>

**Table 29-14. ADC\_PWR Field Descriptions (continued)**

Field	Description
9–4 PUDELAY	<p>Power-up delay. Determines the number of ADC clocks provided to power up an ADC converter (after clearing PD0 or PD1) before allowing a scan to start. It also determines the number of ADC clocks of delay provided in APD and ASB modes between when the ADC goes from the idle to active state and when the scan is allowed to start. The default value is 24 ADC clocks for a 10 MHz conversion clock as a 2 <math>\mu</math>s delay is required after PD2/PD3 is cleared. Accuracy of the initial conversions in a scan is degraded if PUDELAY is set to too small of a value. After power-up, you can reprogram PUDELAY to 13 for minimal delay in APD and lesser in ASB.</p> <p>The value required for PUDELAY depends on the power down state chosen:</p> <ul style="list-style-type: none"> <li>• Power down state 1, PD0/1 = 1, PD2/3 = 1 The entire ADC is in power down mode. Upon clearing PDn, there is a delay of 2 <math>\mu</math>s before the ADC can start making conversions. Value of 24 is required for a 10 MHz conversion clock.</li> <li>• Power down state 2, PD0/1 = 1, PD2/3 = 0 In this partial power down state only the amplifiers in the recursive sub-ranging sections (RSD) are powered off, which significantly reduces the power consumption of the ADC. Upon clearing PD0/1, there is a delay of 13 clock cycles before the ADC can start making conversions.</li> <li>• Power down state 3, PD0/1 = 0, PD2/3 = 1 In this partial power down state only the bias reference generator, switch reference generator and the overrant reference generator are turned off. Upon releasing PD2/3, there is a delay of 2 <math>\mu</math>s before the ADC can start making conversions.</li> </ul> <p><b>Note:</b> PUDELAY defaults to a value typically sufficient for any power mode. The latency of a scan can be reduced by decreasing PUDELAY to the lowest value that accuracy is not degraded. Refer to the processor's data sheet for further details.</p>
3 PD3	<p>Voltage reference converter B power down enable. Forces the voltage reference circuit to power down.</p> <p>0 Manually power up</p> <p>1 Power down controlled by PD1. The voltage reference is activated when PD1 is cleared.</p> <p><b>Note:</b> After clearing this bit, wait at least 2<math>\mu</math>s (use PUDELAY to enforce this delay) before initiating a scan to stabilize the current levels within the converter.</p>
2 PD2	<p>Voltage reference converter A power down enable. Forces the voltage reference circuit to power down.</p> <p>0 Manually power up</p> <p>1 Power down controlled by PD0. The voltage reference is activated when PD0 is cleared.</p> <p><b>Note:</b> After clearing this bit, wait at least 2<math>\mu</math>s (use PUDELAY to enforce this delay) before initiating a scan to stabilize the current levels within the converter.</p>
1 PD1	<p>Manual converter B power down.</p> <p>0 Manually power up. This converter is powered up continuously (APD = 0) or automatically when needed (APD = 1). Do not clear this bit unless PD3 is cleared.</p> <p>1 Immediately power down converter B. The results of a scan using this converter are invalid.</p> <p><b>Note:</b> In any power mode except auto-powerdown (APD=1), when clearing PD1, wait PUDELAY ADC clock cycles before initiating a scan to stabilize power levels within the converter. Poll PSTS1 to determine when the PUDELAY time has elapsed. Failure to do this results in loss of accuracy of the first two samples.</p>
0 PD0	<p>Manual converter A power down.</p> <p>0 Manually power up. This converter is powered up continuously (APD = 0) or automatically when needed (APD = 1). Do not clear this bit unless PD2 is cleared.</p> <p>1 Immediately power down converter A. The results of a scan using this converter are invalid.</p> <p><b>Note:</b> In any power mode except auto-powerdown (APD=1), when clearing PD0, wait PUDELAY ADC clock cycles before initiating a scan to stabilize power levels within the converter. Poll PSTS0 to determine when the PUDELAY time has elapsed. Failure to do this results in loss of accuracy of the first two samples.</p>

### 29.4.13 Calibration Register (ADC\_CAL)

The ADC provides for off-chip references used for ADC conversions.

Address: 0xFC09\_4054 (ADC\_CAL)

Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	VREF H1	VREF L1	VREF H0	VREF L0	0	0	0	0	0	0	0	0	TEST 1	TEST 0	DAC 1	DAC 0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 29-17. Calibration Register (ADC\_CAL)

Table 29-15. ADC\_CAL Field Descriptions

Field	Description
15 VREFH1	V <sub>REFH</sub> source 1. Selects the source of the V <sub>REFH</sub> reference for all conversions in converter 1. 0 Internal V <sub>DDA</sub> 1 ADC_IN4
14 VREFL1	V <sub>REFLO</sub> source 1. Selects the source of the V <sub>REFLO</sub> reference for all conversions in converter 1. 0 Internal V <sub>SSA</sub> 1 ADC_IN5
13 VREFH0	V <sub>REFH</sub> source 0. Selects the source of the V <sub>REFH</sub> reference for all conversions in converter 0. 0 Internal V <sub>DDA</sub> 1 ADC_IN0
12 VREFL0	V <sub>REFLO</sub> source 0. Selects the source of the V <sub>REFLO</sub> reference for all conversions in converter 0. 0 Internal V <sub>SSA</sub> 1 ADC_IN1
11–4	Reserved, must be cleared.
3 TEST1	Analog BIST mode converter 1. Forces the analog input select mux on ADC_IN7 to always be in pass-through mode. In this mode additional capacitance from the internal node or added as an external capacitor can filter the DAC1 output. DAC1 must be set before TEST1 can be set. Clearing DAC1 also clears TEST1. 0 Normal operation 1 ADC_IN7 input mux is always enabled
2 TEST0	Analog BIST mode converter 0. Forces the analog input select mux on ADC_IN3 to always be in pass-through mode. In this mode additional capacitance from the internal node or added as an external capacitor can filter the DAC0 output. DAC0 must be set before TEST0 can be set. Clearing DAC0 also clears TEST0. 0 Normal operation 1 ADC_IN3 input mux is always enabled
1 DAC1	DAC1 alternate source 1. Selects the source of the ADCB3 input as DAC1 output. 0 Normal operation 1 ADC_IN7 input is replaced with DAC1 output
0 DAC0	DAC0 alternate source 0. Selects the source of the ADCA3 input as DAC0 output. 0 Normal operation 1 ADC_IN3 input is replaced with DAC0 output

### 29.4.14 Power Control Register 2 (ADC\_PWR2)

Address: 0xFC09\_4056 (ADC\_PWR2)

Access: User read/write

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

Figure 29-18. Power Control Register 2 (ADC\_PWR2)

Table 29-16. ADC\_PWR2 Descriptions

Field	Description
15–5	Reserved, must be cleared.
4 STNBY	Standby mode enable. 0 Not in standby mode 1 Enable standby mode. The ADC converters are placed into low power mode. Set ADC_DIV such that the conversion clock is in the 200 – 600 kHz range. Only set this bit when the bus clock is running at a lower frequency as the dividers are not large enough to generate a 200-kHz conversion clock from a 125-MHz bus clock.
3–2 SPEEDB	Converter B speed control. Configures the clock speed the ADCB operates. Faster conversion speeds require greater current consumption. Default value is set to 01 for conversion up to 10 MHz. <b>Note:</b> If the conversion clock frequency remains in a range for a specific SPEEDB setting, you can change the frequency of the conversion clock without any wait period. However, if the conversion clock frequency change is significant enough that it falls into a different SPEEDB setting, then you must wait 2 $\mu$ s after changing the SPEEDB setting before using the ADC. <b>Note:</b> Before changing SPEEDB, set ADC_PWR[PD3] or ADC_PWR[PD1]. After changing SPEEDB, clear ADC_PWR[PD3,PD1]. This sequence is followed by PUDELAY(which should be set to 24) which forces a 2 $\mu$ s delay.
1–0 SPEEDA	Converter A speed control. Configures the clock speed that the ADCA operates. Faster conversion speeds require greater current consumption. Default value is set to 01 for conversion up to 10 MHz. <b>Note:</b> If the conversion clock frequency remains in a range for a specific SPEEDA setting, you can change the frequency of the conversion clock without any wait period. However, if the conversion clock frequency change is significant enough that it falls into a different SPEEDA setting, then you must wait 2 $\mu$ s after changing the SPEEDA setting before using the ADC. <b>Note:</b> Before changing SPEEDA, set ADC_PWR[PD2] or ADC_PWR[PD0]. After changing SPEEDA, clear ADC_PWR[PD2,PD0]. This sequence is followed by PUDELAY(which should be set to 24) which forces a 2 $\mu$ s delay.

### 29.4.15 Conversion Divisor Register (ADC\_DIV)

Address: 0xFC09\_4058 (ADC\_DIV)

Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0				DIV1				0			DIV0				
W																
Reset	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	1

Figure 29-19. Conversion Divisor Register (ADC\_DIV)

**Table 29-17. ADC\_DIV Descriptions**

Field	Description																																								
15	Reserved, must be cleared.																																								
14–8 DIV1	Same as the DIV0 description, but DIV1 is used to generate the clock for converter B during parallel non-simultaneous scan modes. See the DIV0 description for details.																																								
7	Reserved, must be cleared.																																								
6–0 DIV0	Clock divisor select. The divider circuit generates the ADC clock by dividing the system clock by $2 \times (\text{DIV0} + 1)$ . Select a DIV0 value so the ADC clock does not exceed 10 MHz or go below 600 KHz. The following table shows ADC clock frequency based on the value of DIV0 for various configurations. Default value is set for an internal clock of 125 Mhz with maximum valid conversion clock below 10 MHz.																																								
	<table border="1"> <thead> <tr> <th rowspan="2">DIV<math>n</math></th> <th rowspan="2">Divisor</th> <th colspan="2">ADC Conversion Clock (MHz)</th> </tr> <tr> <th>125MHz</th> <th>60MHz (Limp Mode)</th> </tr> </thead> <tbody> <tr> <td>0x00</td> <td>2</td> <td>62.5</td> <td>30</td> </tr> <tr> <td>0x01</td> <td>4</td> <td>31.25</td> <td>15</td> </tr> <tr> <td>0x02</td> <td>6</td> <td>20.83</td> <td>10</td> </tr> <tr> <td>0x03</td> <td>8</td> <td>15.62</td> <td>7.5</td> </tr> <tr> <td>0x04</td> <td>10</td> <td>12.5</td> <td>6</td> </tr> <tr> <td>0x05</td> <td>12</td> <td>10</td> <td>5</td> </tr> <tr> <td>...</td> <td>...</td> <td>...</td> <td>...</td> </tr> <tr> <td>0x67</td> <td>208</td> <td>0.6</td> <td>—</td> </tr> </tbody> </table>			DIV $n$	Divisor	ADC Conversion Clock (MHz)		125MHz	60MHz (Limp Mode)	0x00	2	62.5	30	0x01	4	31.25	15	0x02	6	20.83	10	0x03	8	15.62	7.5	0x04	10	12.5	6	0x05	12	10	5	...	...	...	...	0x67	208	0.6	—
DIV $n$	Divisor	ADC Conversion Clock (MHz)																																							
		125MHz	60MHz (Limp Mode)																																						
0x00	2	62.5	30																																						
0x01	4	31.25	15																																						
0x02	6	20.83	10																																						
0x03	8	15.62	7.5																																						
0x04	10	12.5	6																																						
0x05	12	10	5																																						
...	...	...	...																																						
0x67	208	0.6	—																																						
	<b>Note:</b> When ADC_PWR2[STNBY] is set, program values in both dividers so that a 200 – 600 kHz conversion clock is generated.																																								

#### 29.4.16 Auto-Standby Divisor Register (ADC\_ASDIV)

Address: 0xFC09\_405A (ADC\_ASDIV)

Access: User read/write

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

**Figure 29-20. Auto-Standby Divisor Register (ADC\_ASDIV)**

**Table 29-18. ADC\_AS DIV Descriptions**

Field	Description																																							
15–7	Reserved, must be cleared.																																							
6–0 AS DIV	Clock divisor select. The divider circuit generates the ADC auto-standby clock by dividing the system clock by $2 \times (\text{AS DIV} + 1)$ . Select a AS DIV value so the ADC auto-standby clock is in the 200–600 kHz range. The following table shows the ADC auto-standby clock frequency based on the value of AS DIV for various configurations. Default value is set for an internal clock of 125MHz.																																							
	<table border="1"> <thead> <tr> <th>AS DIV</th> <th>ADC Clock In (MHz)</th> <th>Divisor</th> <th>Auto-Standby Clock (kHz)</th> </tr> </thead> <tbody> <tr> <td>0x137</td> <td>125</td> <td>624</td> <td>200.32</td> </tr> <tr> <td>0x12B</td> <td>120</td> <td>600</td> <td>200</td> </tr> <tr> <td>0x0F9</td> <td>100</td> <td>500</td> <td>200</td> </tr> <tr> <td>0x0EF</td> <td>96</td> <td>480</td> <td>200</td> </tr> <tr> <td>...</td> <td>...</td> <td>...</td> <td>...</td> </tr> <tr> <td>0x03B</td> <td>24</td> <td>120</td> <td>200</td> </tr> <tr> <td>0x027</td> <td>16</td> <td>80</td> <td>200</td> </tr> <tr> <td>0x013</td> <td>8</td> <td>60</td> <td>200</td> </tr> </tbody> </table>	AS DIV	ADC Clock In (MHz)	Divisor	Auto-Standby Clock (kHz)	0x137	125	624	200.32	0x12B	120	600	200	0x0F9	100	500	200	0x0EF	96	480	200	...	...	...	...	0x03B	24	120	200	0x027	16	80	200	0x013	8	60	200			
AS DIV	ADC Clock In (MHz)	Divisor	Auto-Standby Clock (kHz)																																					
0x137	125	624	200.32																																					
0x12B	120	600	200																																					
0x0F9	100	500	200																																					
0x0EF	96	480	200																																					
...	...	...	...																																					
0x03B	24	120	200																																					
0x027	16	80	200																																					
0x013	8	60	200																																					

## 29.5 Functional Description

The ADC block consists of two 4-channel input select function, two independent sample and hold (S/H) circuits feeding two separate 12-bit ADCs. The two separate converters store their results in an accessible buffer, awaiting further processing.

The conversion process is initiated by a SYNC signal from one of the on-chip timer channels (see Chapter 10, “Chip Configuration Module (CCM)”) or by writing one to a ADC\_CRn[START] bit.

Starting a single conversion actually begins a sequence of conversions, or a scan. A conversion takes up to eight-single ended or differential samples, one at a time in sequential scan mode. In parallel scan mode, the eight samples are allocated, four to converter A and four to converter B. In parallel scan modes, converter A can only sample analog inputs ADC\_IN0–3 while converter B can only sample analog inputs ADC\_IN4–7. Each converter can take up to four samples.

The scan sequence is determined by defining eight sample slots, processed in order SAMPLE0–7 during sequential scan mode. In parallel scan mode, the SAMPLE0–3 are processed in order by converter A, and SAMPLE4–7 are processed in order by converter B. Sample slots may be disabled using the ADC\_SDIS register to terminate a scan early.

The following pairs of analog inputs can be configured as a differential pair: ADC\_IN0–1, ADC\_IN2–3, and ADC\_IN4–5, ADC\_IN6–7. When configured as a differential pair, a reference to either member of the pair by a sample slot results in a differential measurement using that differential pair.

The ADC can perform a single scan and halt, perform a scan when triggered, or perform the scan sequence repeatedly until manually stopped. These modes are described in the following section.

## 29.5.1 Scan Modes

The various scan modes defined by ADC\_CR1[SMODE] are described in [Table 29-19](#).

**Table 29-19. Scan Modes**

SMODE	Scan	Description
000	Once sequential	Upon start or an enabled sync signal, samples are taken one at a time starting with SAMPLE0, until the first disabled sample is encountered. If no disabled sample is encountered, conversion concludes after SAMPLE7. If the scan is initiated by a SYNC signal only one scan is completed until the converter is rearmed by writing to ADC_CR1.
001	Once parallel	Upon start or an enabled sync signal, converter A converts SAMPLE0–3 and converter B converts SAMPLE4–7 in parallel. When SIMULT is set, scanning stops when either converter encounters a disabled sample or both converters complete their 4 samples. When SIMULT is cleared, scanning stops in a converter when that converter encounters a disabled sample or that converter completes its four samples. If the scan is initiated by a SYNC signal, only one scan is completed until the converter is rearmed by writing to ADC_CR1. If SIMULT is cleared, then the B converter must be rearmed by writing to ADC_CR2.
010	Loop sequential	Upon an initial start or enabled sync pulse, up to eight samples in order SAMPLE0–7 are taken one at a time until a disabled sample is encountered. The process repeats until the STOP0 bit is set. While a loop mode is running, do not give any additional start commands or sync pulses. If ASB or APD is the selected power mode control, PUDELAY is only applied on the first conversion.
011	Loop parallel	Upon an initial start or enabled sync pulse, converter A converts SAMPLE0–3 and converter B converts SAMPLE4–7. Each time a converter completes its current scan, it immediately restarts its scan sequence. This continues until the STOP bit is set. While a loop mode is running, do not give any additional start commands or sync pulses. When SIMULT is set, scanning restarts when either converter encounters a disabled sample. When SIMULT is cleared, scanning restarts in a converter when that converter encounters a disabled sample. If auto-standby (ASB) or auto-power down (APD) is the selected power mode control, PUDELAY is only applied on the first conversion.
100	Triggered sequential	Upon start or an enabled sync signal, samples are taken one at a time starting with SAMPLE0, until a disabled sample is encountered. If no disabled sample is encountered, conversion concludes after SAMPLE7. If external sync is enabled, new scans are started for each SYNC pulse that is non-overlapping with a current scan in progress.
101	Triggered parallel	Upon start or an enabled sync signal, converter A converts SAMPLE0–3 and converter B converts SAMPLE4–7 in parallel. When SIMULT is set, scanning stops when either converter encounters a disabled sample. When SIMULT is cleared, scanning stops in a converter when that converter encounters a disabled sample. If external sync is enabled new scans are started for each non-overlapping SYNC pulse with a current scan in progress.

The parallel scan modes can be simultaneous or non-simultaneous, as defined by ADC\_CR2[SIMULT]:

- Simultaneous scan mode — the parallel scans in the two converters are executed simultaneously, always resulting in simultaneous pairs of conversions: one by converter A and one by converter B. The two converters share the same start, stop, sync, end-of-scan interrupt enable control, and interrupt. Scanning in both converters is terminated when either converter encounters a disabled sample.
- Non-simultaneous scan mode — the parallel scans in the two converters are done independently. The two converters have their own start, stop, sync, end-of-scan interrupt enable controls and interrupts. Scanning in either converter terminates only when that converter encounters a disabled sample.

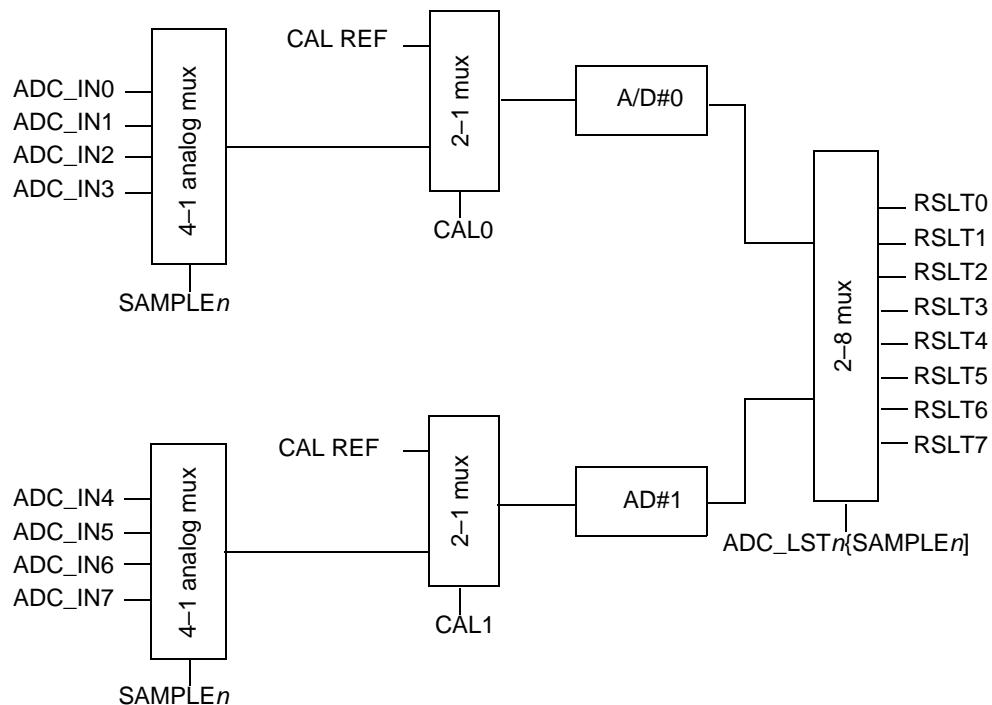


Figure 29-21. ADC Sequential Operation Mode

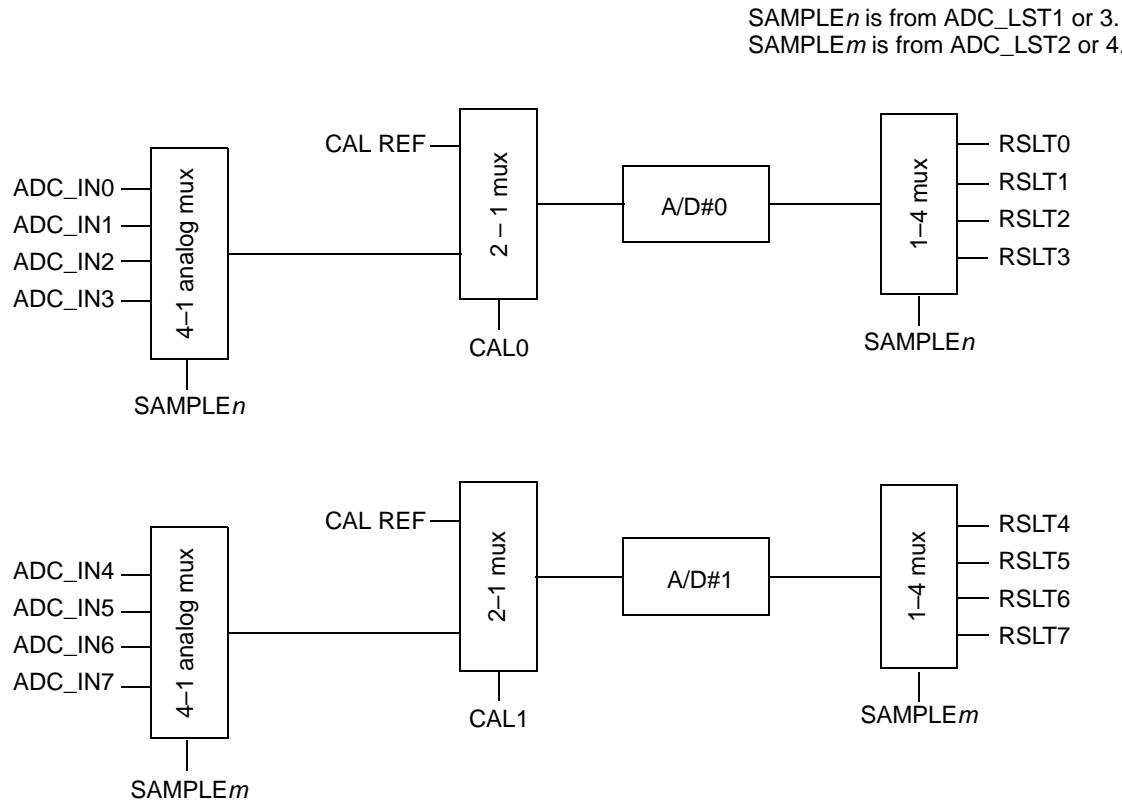


Figure 29-22. ADC Parallel Operation Mode

### 29.5.2 Input Mux Function

The input mux function is illustrated in [Figure 29-23](#). The channel select and single-ended differential switches are indirectly controlled based on settings within the ADC\_LST $n$ , ADC\_SDIS, and ADC\_CR[CHNCFG] fields.

- MUXing for sequential, single-ended mode conversions – During each conversion cycle (sample), any one input of the two eight input groups can be directed to its corresponding output.
- MUXing for sequential, differential mode conversions – During any conversion cycle (sample), either member of a differential pair may be referenced resulting in a differential measurement on that pair.
- MUXing for parallel, single ended mode conversions – During any conversion cycle (sample), any of ADC\_IN0–3 can be directed to the converter A input and any of ADC\_IN4–7 can be directed to the converter B input.
- MUXing for parallel, differential mode conversions – During any conversion cycle (sample), either member of differential pair ADC\_IN0/1, ADC\_IN2/3 can be referenced resulting in a differential measurement of that pair at converter A input. Likewise, either member of differential pair ADC\_IN4/5, ADC\_IN6/7 can be referenced resulting in a differential measurement of that pair at the converter B input.

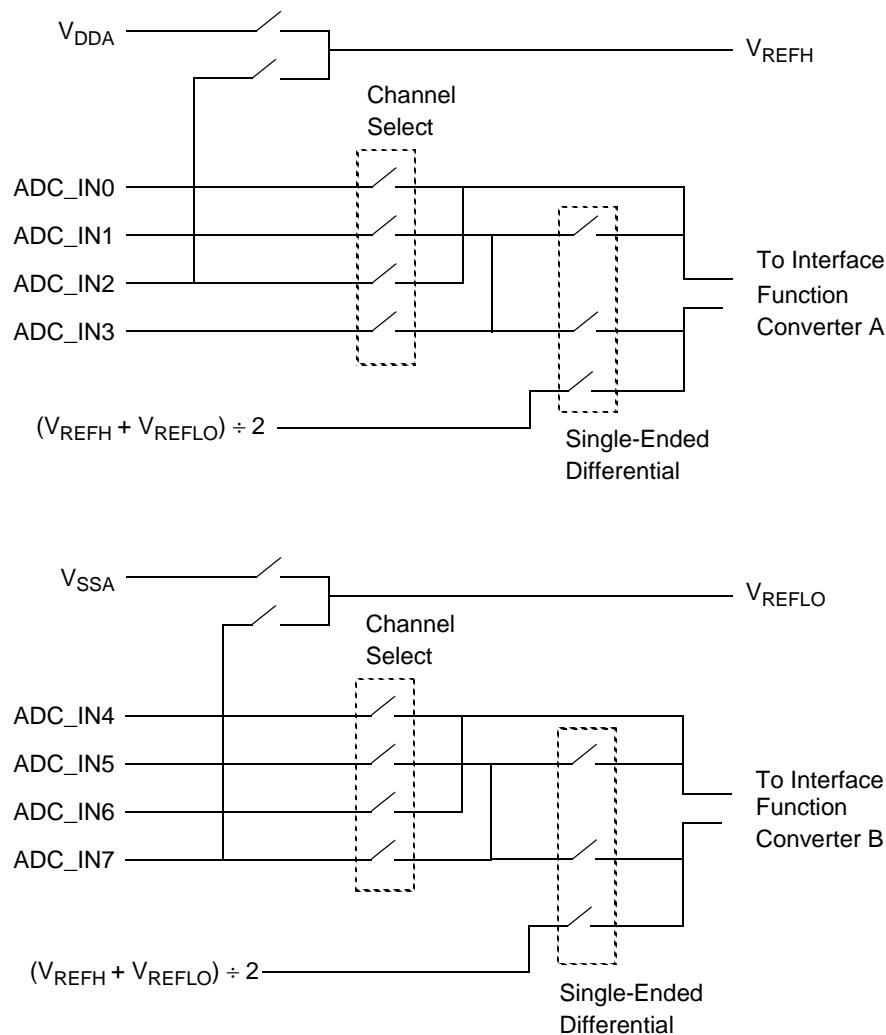


Figure 29-23. Input Select MUX

Details of single-ended and differential measurement from user perspective, are described under the CHNCFG bit fields. Internally, all measurements are performed differentially. During single-ended measurements,  $(V_{REFH} + V_{REFLO})/2$  is used as the negative input voltage while the selected analog input is used as the positive input.

Table 29-20. Analog MUX Controls for Each Conversion Mode

Conversion Mode	Channel Select Switches	Single-Ended Differential Switches
General Comments	—	The two lower switches within the dashed box are controlled such that one switch is always closed and the other open.
Sequential, Single-Ended	The two 1-of-4 select muxes can be set for the appropriate input line.	The lower switch is closed, providing $(V_{REFH}+V_{REFLO})/2$ to the differential input of the A/D. In this mode, the upper switch is always closed so that any of the four inputs can get to the A/D input.

**Table 29-20. Analog MUX Controls for Each Conversion Mode (continued)**

<b>Conversion Mode</b>	<b>Channel Select Switches</b>	<b>Single-Ended Differential Switches</b>
Sequential, Differential	The channel select switches are turned on in pairs, providing a dual 1-of-2 select function, such that either of the two differential channels can be routed to the A/D input.	The upper and lower switches are open and the middle switch is closed, providing the differential channel to the differential input of the A/D.
Parallel, Single Ended	The two 1-of-4 select muxes can be set for the appropriate input line.	The lower switch is closed, providing $V_{REF}/2$ to the differential input of the A/D. In this mode, the upper switch is always closed so that any of the four inputs can get to the A/D input.
Parallel, Differential	The channel select switches are turned on in pairs, providing a dual 1-of-4 select function, such that either of the two differential channels can be routed to the A/D input.	The upper and lower switches are open and the middle switch is closed, providing the differential channel to the differential input of the A/D.

### 29.5.3 ADC Sample Conversion Modes of Operation

The ADC consists of a cyclic, algorithmic architecture using two recursive sub-ranging sections (RSD#1 and RSD#2), shown in [Figure 29-24](#). Each sub-ranging section resolves a single bit for each conversion clock, resulting in an overall conversion rate of two bits per clock cycle. Each sub-ranging section is designed to run at a maximum clock speed of 10 MHz so a complete 12-bit conversion can be completed in six conversion clocks, not including sample or post-processing time.

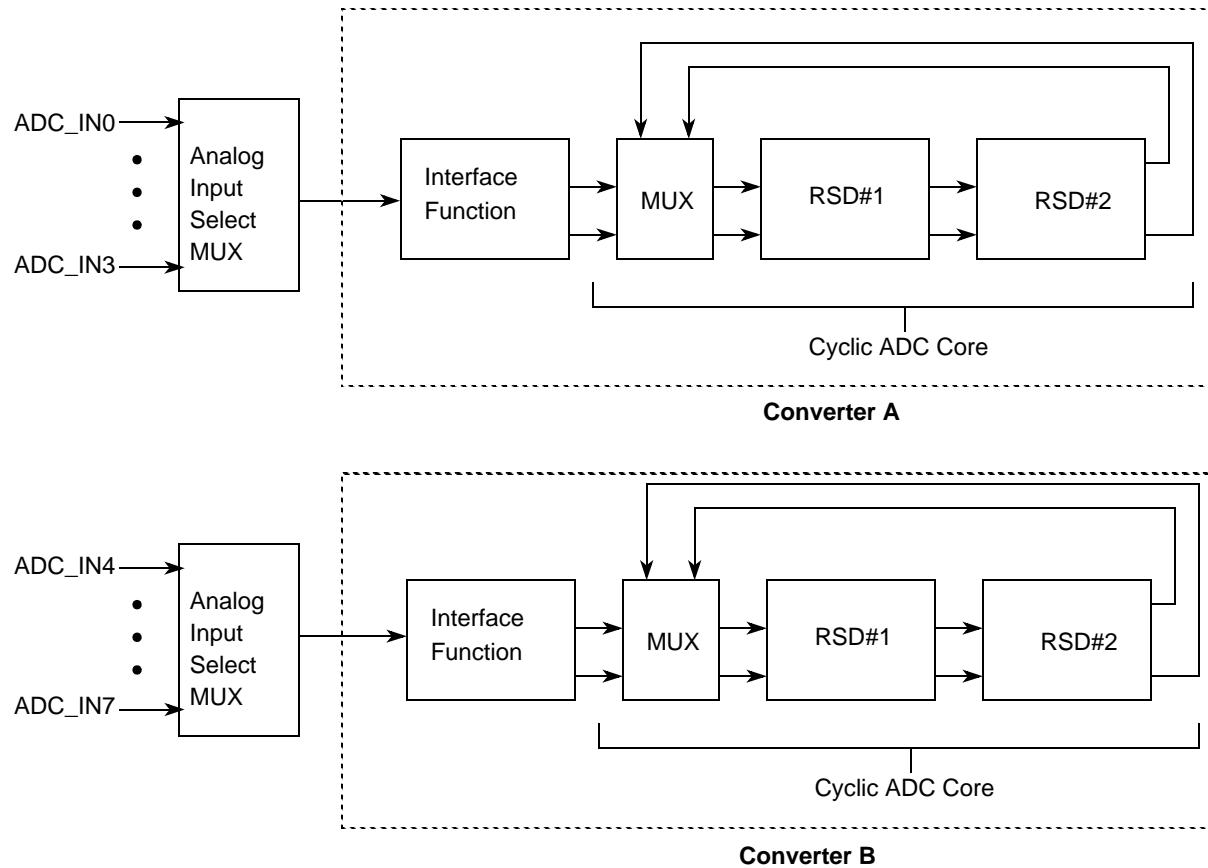


Figure 29-24. Cyclic ADC – Top Level Block Diagram

### 29.5.3.1 Normal Mode Operation

The mode of operation for a given sample is determined by ADC\_CR1[CHNCFG]. The two normal modes of operation are:

- Single-ended mode (CHNCFG = 0) — The input mux of the ADC selects one of the four analog inputs and directs it to the plus terminal of the A/D core. The minus terminal of the A/D core is connected to the V<sub>REFLO</sub> reference during this mode. The ADC measures the voltage of the selected analog input and compares it against the (V<sub>REFH</sub> – V<sub>REFLO</sub>) reference voltage range.
- Differential mode (CHNCFG = 1) — The ADC measures the voltage difference between two analog inputs and compares that against the (V<sub>REFH</sub> – V<sub>REFLO</sub>) voltage range. The input is selected as an input pair: ADC\_IN0/1, ADC\_IN2/3, ADC\_IN4/5, or ADC\_IN6/7. In this mode, the plus terminal of the A/D core is connected to the even analog input while the minus terminal is connected to the odd analog input.

A mix and match combination of single-ended and differential configurations may exist. For example:

- ADC\_IN0 and ADC\_IN1 differential, ADC\_IN2 and ADC\_IN3 single-ended
- ADC\_IN4 and ADC\_IN5 differential, and ADC\_IN6 and ADC\_IN7 single-ended

### 29.5.3.1.1 Single-Ended Samples

The ADC module performs a ratio-metric conversion. For single-ended measurements, the digital result is proportional to the ratio of the analog input to the reference voltage in the following diagram.

$$\text{Single-ended value} = \text{round}\left(\frac{V_{IN} - V_{REFLO}}{V_{REFH} - V_{REFLO}} \times 4096\right) \times 8 \quad \text{Eqn. 29-1}$$

$V_{IN}$  = Applied voltage at the input pin

$V_{REFH}$  and  $V_{REFLO}$  = Voltage at the external reference pins on the device (typically  $V_{REFH} = V_{SS}$  and  $V_{REFLO} = V_{DD}$ )

#### NOTE

The 12-bit result is rounded to the nearest LSB.

The ADC is a 12-bit function with 4096 possible states. However, the 12 bits have been left shifted three bits on the 16-bit data bus so its magnitude, as read from the data bus, is now 32,760.

### 29.5.3.1.2 Differential Samples

For differential measurements, the digital result is proportional to the ratio of the difference in the inputs to the difference in the reference voltages ( $V_{REFH}$  and  $V_{REFLO}$ ).

When converting differential measurements, the following formula is useful:

$$\text{Differential value} = \text{round}\left(\frac{V_{IN+} - V_{IN-}}{V_{REFH} - V_{REFLO}} \times 4096\right) \times 8 \quad \text{Eqn. 29-2}$$

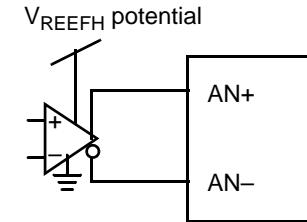
$V_{IN}$  = Applied voltage at the input pin

$V_{REFH}$  and  $V_{REFLO}$  = Voltage at the external reference pins on the device (typically  $V_{REFH} = V_{SS}$  and  $V_{REFLO} = V_{DD}$ )

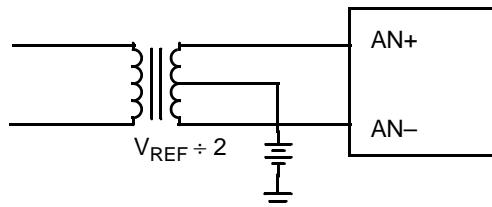
#### NOTE

The 12-bit result is rounded to the nearest LSB.

The ADC is a 12-bit function with 4096 possible states. However, the 12 bits have been left shifted three bits on the 16-bit data bus so its magnitude, as read from the data bus, is now 32,760.



Differential buffer centers about mid-point.



Center tap held at  $(V_{REFFH} + V_{REFLO}) / 2$

**Note:** Normally  $V_{REFLO}$  is set to  $VSS = 0V$ .

**Figure 29-25. Typical Connections for Differential Measurements**

#### 29.5.4 ADC Data Processing

As shown in [Figure 29-26](#), the result of the ADC conversion process is normally sent to an adder for offset correction. The adder subtracts the  $ADC\_OFSn$  register value from each sample and the resultant value is then stored in the  $ADC\_RSLTn$  registers. At the same time, the raw ADC value and the  $ADC\_RSLTn$  values are checked for limit violations and zero-crossing. Appropriate interrupts are asserted, if enabled.

The result value sign is determined from the ADC unsigned result minus the respective  $ADC\_OFSn$  register value. If the  $ADC\_OFSn$  register is zero, the  $ADC\_RSLTn$  register value is unsigned and equals the cyclic converter unsigned result. The range of the  $ADC\_RSLTn$  register is 0x0000–0x7FF8 if the entire  $ADC\_OFSn$  register is zero. This is equal to the raw value of the ADC core.

The processor may write to the  $ADC\_RSLTn$  registers when the STOP bit for that scan is set. This write operation is treated as if it came from the ADC analog core. Therefore, the limit-checking, zero-crossing, and the  $ADC\_OFSn$  registers function as if in normal mode. For example, if the STOP bit is set and the processor writes to  $ADC\_RSLT5$ , the data written to the  $ADC\_RSLT5$  is muxed to the ADC digital logic inputs, processed, and stored into  $ADC\_RSLT5$  as if the analog core had provided the data. This test data must be justified, as illustrated by the  $ADC\_RSLTn$  definition and does not include the sign bit.

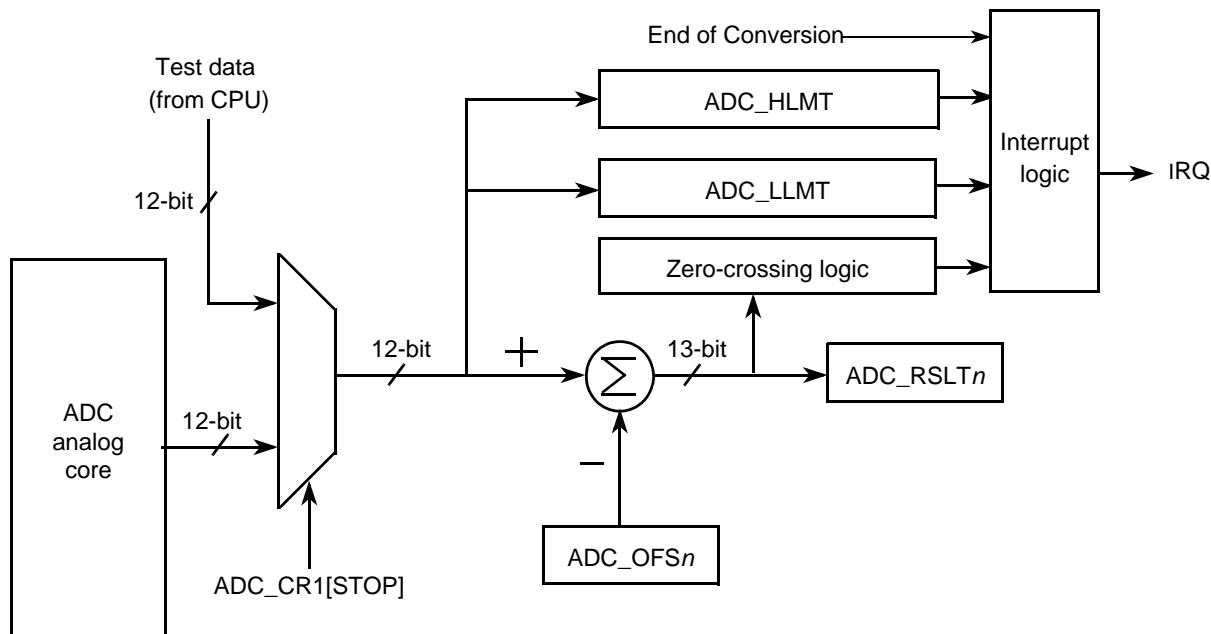


Figure 29-26. Result Register Data Manipulation

### 29.5.5 Sequential vs. Parallel Sampling

All scan modes use the eight sample slots in the ADC\_LST $n$  registers. Slots define which input or differential pair to measure at each step in a scan sequence. The ADC\_SDIS register defines which of these sample slots are enabled. Input pairs ADC\_IN0/1, ADC\_IN2/3, ADC\_IN4/5, and ADC\_IN6/7 can be measured differentially using the CHNCFG field. If a sample refers to an input not configured as a member of a differential pair, a single-ended measurement is made. If a sample refers to either member of a differential pair, a differential measurement is made. Refer to CHNCFG field description for details of differential and single-ended measurement.

Scan modes are sequential or parallel. In sequential scans, up to eight sample slots are sampled one at a time in order SAMPLE 0–7. Each sample may refer to any of the eight analog inputs ADC\_IN0–7, thus the same input may be referenced by more than one sample slot. Scanning is initiated when the START0 bit is set or, if the SYNC0 bit is set, when the SYNC0 input is asserted. A scan ends when the first disabled sample slot is encountered per the SDIS register. Completion of the scan triggers the EOSI0 interrupt if the EOSIEN0 bit is set. The START0 bit and SYNC0 input are ignored while a scan is in process. When STOP0 is set, scanning stops and cannot restart.

Parallel scans differ in that converter A performs up to four samples (SAMPLE 0–3) in parallel to converter B (SAMPLE 4–7). Samples 0–3 may only reference inputs ADC\_IN0–3, while samples 4–7 may only reference inputs ADC\_IN4–7. Within that constraint, any sample may reference any pin and the same input may be referenced by more than one sample slot. If SIMULT is set, the scans in both converters are initiated when the START0 bit is set or, if the SYNC0 bit is set, when the SYNC0 input is asserted. The scan in both converters terminates when either converter encounters a disabled sample slot. Completion of a scan triggers the EOSI0 interrupt if the EOSIEN0 bit is set. Samples are always taken simultaneously in both converters. When STOP0 is set, scanning stops and cannot restart.

Setting non-simultaneous mode (`SIMULT` = 0) causes parallel scanning to operate independently in each converter. Each converter has its own set of `STARTn`, `STOPn`, `SYNCn`, and `EOSIENn` control bits, `SYNCn` input, `EOSIn` interrupt, and conversion in progress (`CIPn`) status indicators ( $n = 0$  for converter A,  $n = 1$  for converter B). Though still operating in parallel, the scans in the A and B converter start and stop independently according to their own controls and may be simultaneous, phase-shifted, or asynchronous depending on when scans are initiated on the respective converters. The A and B converter may be of different length (still up to a maximum of four) and each converter's scan completes when a disabled sample is encountered in that converter's sample list only. `STOP0` stops converter A only, while `STOP1` stops converter B only. Loop scan modes iterate independently. Converter A processes input selections from SAMPLE 0–3 and converter B processes input selections from SAMPLE 4–7. Each converter independently restarts its scan after completing its list or encountering a disabled sample slot.

## 29.5.6 Scan Sequencing

Scan modes break down into three types based on how they repeat. See [Section 29.5.5, “Sequential vs. Parallel Sampling”](#) to understand the operation of sequential and parallel scan modes and their related controls. These types are:

1. Once scan modes — Execute a sequential or parallel scan only once each time it is started. All scan modes ignore sync pulses occurring while a scan is in process. However, once scan modes continue to ignore sync pulses until the sync input is re-armed. Re-arming, however, can occur any time after a scan has started by writing to the `ADC_CCrn` register.
2. Triggered scan modes — Identical to the once scan modes except re-arming of sync inputs is not necessary.
3. Loop scan modes — Automatically restart a scan, parallel or sequential, when the previous scan completes. In parallel loop scan modes, the converter A scan restarts as soon as the converter A scan completes and the converter B scan restarts when the converter B scan completes. All subsequent start and sync pulses are ignored after the scan begins. Scanning is only terminated by setting the appropriate `STOPn` bit.

## 29.5.7 Power Management

The five supported power modes are described below. They are presented in order from highest to lowest power utilization at the expense of increased conversion latency and/or startup delay.

### 29.5.7.1 Manual Power Down of Unused Converters

If the channel list registers (see [Section 29.4.4, “Channel List n Registers \(ADC\\_LST1–2\)”](#)) do not use input pins `ADC_IN4`–`7`, then converter B is never used. Similarly, if `ADC_IN0`–`3` are not used, then converter A is never used. In such cases the unused ADC can be manually powered down using the `ADC_PWR[PDn]` bits. Please see [Section 29.4.12, “Power Control Register \(ADC\\_PWR\)”](#).

Since the default value of the `PD0` and `PD1` bits is powered down, this technique can be used in most applications by simply not powering up the unused ADC.

### 29.5.7.2 Normal Power Mode

This mode operates when:

1. At least one ADC converter is powered up ( $\text{ADC\_PWR[PD0 or PD1]} = 0$ )
2. Both auto power down and auto-standby modes are disabled ( $\text{ADC\_PWR[APD, ASB]} = 0$ )
3. The ADC's clock is enabled ( $\text{PPMRH0[CD37]} = 1$ . See [Chapter 9, “Power Management”](#).)

In this mode the ADC uses the conversion clock as the ADC clock source when active or idle. To minimize conversion latency configure the conversion clock to near 10 MHz. No startup delay ( $\text{ADC\_PWR[PUDELAY]}$ ) is imposed.

### 29.5.7.3 Auto Standby Mode

This mode operates when:

1. At least one ADC converter is powered up ( $\text{ADC\_PWR[PD0 or PD1]} = 0$ )
2. Auto power down is disabled ( $\text{ADC\_PWR[APD]} = 0$ )
3. Auto standby is enabled ( $\text{ADC\_PWR[ASB]} = 1$ )
4. The ADC's clock is enabled ( $\text{PPMRH0[CD37]} = 1$ . See [Chapter 9, “Power Management”](#).)

In auto standby mode, the ADC uses the conversion clock when active and the 200–600 kHz standby clock when idle. The standby (low current) state automatically engages when the ADC is idle. Configure the conversion clock to near 10 MHz to minimize conversion latency. The ADC executes a startup delay of PUDELAY ADC clocks at the start of all scans, allowing the ADC to switch to the conversion clock and to revert from standby to normal current mode.

Auto-standby is a compromise between normal and auto-power down modes offering moderate power savings at the cost of moderate latency when leaving the idle state to start a new scan.

### 29.5.7.4 Auto Power Down Mode

This mode operates when:

1. At least one ADC converter is powered up ( $\text{ADC\_PWR[PD0 or PD1]} = 0$ )
2. Auto power down mode enabled ( $\text{ADC\_PWR[APD]} = 1$ )
3. The ADC's clock is enabled ( $\text{PPMRH0[CD37]} = 1$ . See [Chapter 9, “Power Management”](#).)

In this mode, the ADC uses the conversion clock when active and gates off the conversion clock and powers down the converters when idle. Configure the conversion clock to near 10 MHz to minimize conversion latency when active. A startup delay of PUDELAY ADC clocks is executed at the start of all scans, allowing the ADC to stabilize when switching to normal current mode from a completely powered-off condition. This mode uses less power than auto standby. However, it requires more startup latency when leaving the idle state to start a scan (higher PUDELAY value).

### 29.5.7.5 Standby Mode

This mode operates when:

1. At least one ADC converter is powered up ( $\text{ADC\_PWR}[\text{PD}0 \text{ or } \text{PD}1] = 0$ )
1. Standby mode is enabled ( $\text{ADC\_PWR2}[\text{STNBY}] = 1$ )
2. The ADC's clock is enabled ( $\text{PPMRH0}[\text{CD}37] = 1$ . See [Chapter 9, “Power Management”](#).)

In this mode, the conversion clock must be in the range of 200–600 kHz. Program  $\text{ADC\_DIV}$  accordingly.

### 29.5.7.6 Power Down Mode

This mode operates when:

1. Both ADC converters are powered down ( $\text{ADC\_PWR}[\text{PD}0, \text{PD}1] = 1$ )
2. Both voltage references are powered down ( $\text{ADC\_PWR}[\text{PD}2, \text{PD}3] = 1$ )
3. The ADC's clock is disabled ( $\text{PPMRH0}[\text{CD}37] = 1$ . See [Chapter 9, “Power Management”](#).)

In this configuration, the clock trees to the ADC and its analog components are shut down and the ADC uses no power.

## 29.5.8 Power Management Details

The ADC voltage reference and converters are powered down ( $\text{ADC\_PWR}[\text{PD}n] = 1$ ) on reset. Individual converters can be manually powered down when not in use and the voltage references can be manually powered down when no converter is in use. When the ADC reference is powered down the output reference voltages are set to low ( $V_{SSA}$ ) and the ADC data output is driven low.

A delay of PUDELAY ADC clock cycles is imposed when PD0 or PD1 are cleared to power up a regulator and also whenever going from an idle (neither converter has a scan in process) to an active state (at least one converter has a scan in process). The ADC recommends using two PUDELAY values: a large value for full power up and a moderate value for going from standby current levels to full power up. The following is an explanation of how to use PUDELAY when starting the ADC or changing modes.

When starting up in normal mode:

1. Set PUDELAY to the large power up value.
2. Clear auto standby (ASB) and auto power down (APD) bits.
3. Clear PD0 and PD1 to power up the required converters.
4. Poll  $\text{ADC\_PWR}[\text{PSTS}n]$  until all required converters are powered up.

This provides a full power-up delay before scans begin. Scan operations can now be started. Normal mode does not use PUDELAY at the start of scan; thus, no further delay is imposed.

When starting up in ASB mode:

1. Follow the normal mode steps.
2. Before starting scan operations, set PUDELAY to the moderate standby recovery value.
3. Set ASB bit.

Auto-standby mode automatically reduces current levels until active, and then imposes the PUDELAY to allow current levels to rise from standby to full power levels.

When starting up in APD mode:

1. Set PUDELAY to the large power up value.
2. Clear ASB and set APD.
3. Clear PD0 and PD1 bits for the required converters.

The converters remain powered off until scanning is active. Then, the large PUDELAY is imposed to move from powered down to fully powered, before starting scan operations.

#### NOTE

Power off both regulators ( $\text{PD0} = \text{PD1} = 1$ ) when re-configuring clocking or power controls to avoid ambiguity and ensure the proper delays are applied when powering up or starting scans.

Attempts to start a scan during the PUDELAY is ignored until the appropriate ADC\_PWR[PSTS $n$ ] are cleared.

Any attempt to use a converter when powered down or with the voltage reference disabled results in invalid results. You may read the result registers after a converter powers down for results calculated before the power down. A new scan sequence must be started with a SYNC pulse or a write to the START bit before new valid results are available.

In APD mode, when the ADC goes from idle to active, a converter is only powered up if it is required for the scan as determined by the ADC\_LST $n$  and ADC\_SDIS registers.

#### 29.5.8.1 Stop Mode Operation

Any conversion sequence in progress can be stopped by setting the relevant STOP $n$  bit. Any further sync pulses or writes to the START $n$  bit are ignored until the STOP $n$  bit is cleared. While in stop mode, the ADC\_RSLT $n$  registers can be modified by writes from the processor. Any write to ADC\_RSLT $n$  in the ADC stop mode is treated as if the analog core supplied the data. Therefore, limit checking, zero crossing, and associated interrupts can occur if enabled.

#### 29.5.9 Clock Operation

The ADC has one external clock input to drive two clock domains within the ADC module.

**Table 29-21. ADC Clock Summary**

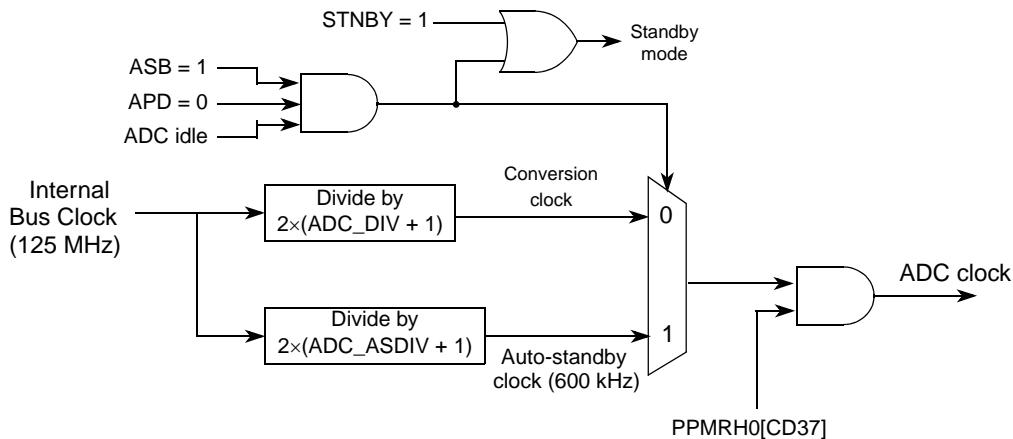
Clock	Direction	Characteristics
Peripheral clock	Input	Max rate is 125 MHz. Generates the reference and conversion clocks
Conversion clocks	Output	A 600 kHz – 10 MHz conversion clock for each converter.

As shown in [Figure 29-27](#), the conversion clock is the primary source for the ADC clock and is always selected as the ADC clock when conversions are in process. Configure ADC\_DIV so that conversion clock frequency is between 600 kHz and 10 MHz. Operating the ADC at out-of-spec conversion clock

frequencies or reconfiguring clock rates or power modes while the regulators are powered (ADC\_PWR[PD0 or PD1] = 0) degrades conversion accuracy.

The conversion clock that the ADC uses for sampling is calculated using the internal bus clock and the clock divisor bits in ADC\_DIV. The ADC clock is active while in looping modes or if power management is set to normal mode. It is also active during all ADC power-up sequences for a period of time determined by ADC\_PWR[PUDELAY]. If a conversion is being initiated in power savings mode, then the ADC clock continues until the conversion sequence completes.

The following diagram shows the structure of the clocking system.



**Figure 29-27. ADC Clock Generation**

The internal bus clock feeds a divider to generate the auto-standby clock. This clock is selected only during auto-standby power mode and when both converters are idle. The logic which selects the standby clock also asserts standby current mode. Standby current mode is only available at an ADC clock rate of 200–600 kHz. It provides substantial power savings, yet requires less latency when switching back to the conversion clock at the start of a scan than auto-powerdown mode. APD mode only uses the conversion clock as the ADC clock source, but fully powers down the converters when idle.

The standby current mode is also engaged when ADC\_PWR2[STNBY] is set and the internal bus clock frequency is low enough to generate less than 600 kHz from DIV. This insures a 200–600kHz ADC conversion clock rate while a conversion is in process. This configuration, referred to as standby power mode, provides accurate conversion without startup latency at the reduced power levels of standby current mode.

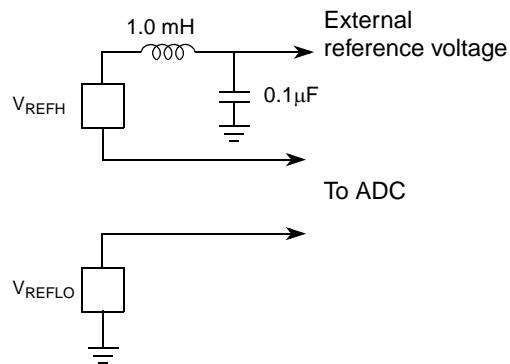
The ADC clock is an output of the gasket used to operate the two converters during scan operations. It is derived by muxing the conversion clock (divided version of the conversion clock source) and the standby clock.

### 29.5.10 Voltage Reference Pins $V_{REFH}$ & $V_{REFLO}$

The voltage difference between  $V_{REFH}$  and  $V_{REFLO}$  provides the reference voltage all analog inputs are measured against.  $V_{REFH}$  is nominally set to  $V_{DDA}$ .  $V_{REFLO}$  is nominally set to 0 V. An external reference voltage should be provided from a low-noise, filtered source capable of providing up to 1 mA of reference current.

When tying  $V_{REFH}$  to the same potential as  $V_{DDA}$  relative measurements are made with respect to the amplitude of  $V_{DDA}$ . Ensure the voltage applied to  $V_{REFH}$  is as noise free as possible. Any noise residing on  $V_{REFH}$  is directly transferred to the digital result.

**Figure 29-28** illustrates a minimum configuration for a noise-filtered  $V_{REFH}$ .



**Figure 29-28. ADC Voltage Reference Circuit**

### 29.5.11 Supply Pins $V_{DDA}$ and $V_{SSA}$

Dedicated power supply pins are provided for reducing noise coupling and to improve accuracy. The power provided to these pins should come from a low noise filtered source. Uncoupling capacitors should be connected between  $V_{DDA}$  and  $V_{SSA}$ .

### 29.5.12 Interrupt Operation

The ADC generates an interrupt under three conditions as shown in [Table 29-22](#). These interrupts are optional and enabled through `ADC_CR1`.

#### NOTE

Additional bits need to be configured in the interrupt control module to enable the CPU to acknowledge the ADC interrupts. See [Chapter 17, “Interrupt Controller Modules”](#), for details.

**Table 29-22. Interrupt Summary**

<b>Interrupt</b>	<b>Description</b>
Threshold	<ul style="list-style-type: none"> <li>• Zero-crossing — The current result value has a sign change from the previous result as configured by the ADC_ZCCR register.</li> <li>• Low limit — The current result value is less than the low limit register value. The raw result value is compared to ADC_LLMT before ADC_OFS<math>n</math> value is subtracted.</li> <li>• High limit — The current result value is greater than the high limit register value. The raw result value is compared to ADC_HLMT before ADC_OFS<math>n</math> value is subtracted.</li> </ul>
Converter A conversion complete	Generated upon completion of any scan and convert sequence when EOSIE0 is set. See ADC_SR[EOSI0] bit descriptions
Converter B conversion complete	Generated upon completion of any scan and convert sequence when EOSIE1 is set. See ADC_SR[EOSI1] bit descriptions



# Chapter 30

## Digital-to-Analog Converter (DAC)

### 30.1 Introduction

The DAC accepts a 12-bit digital signal and creates a monotonic 12-bit analog output varying from ~DAC\_VREFL to ~DAC\_VREFH. The DAC module consists of a conversion unit, an output amplifier, and the associated digital control blocks.

#### NOTE

On this device, the DAC\_VREFH and DAC\_VREFL signals are internally connected to the DAC/ADC supply voltages, VDDA\_DAC\_ADC and VSSA\_DAC\_ADC, respectively.

#### 30.1.1 Features

DAC features include:

- 12-bit resolution
- Guaranteed 6-sigma monotonicity over input word 497–3599
- High- and low-speed conversions
  - 1  $\mu$ s conversion rate for high speed, 2  $\mu$ s for low speed
- Power-down mode
- DAC can drive 3-k $\Omega$ , 400-pF load
- Choice of asynchronous or synchronous updates
  - Sync input can be connected to the on-chip PWM, timers, and edge port modules
- Automatic mode allows the DAC to generate its own output waveforms including square, triangle, and sawtooth
- Automatic mode allows programmable period, update rate, and range
- DMA support with configurable watermark level

### 30.1.2 Block Diagram

The digital-to-analog converter (DAC) configuration is provided in the following block diagram.

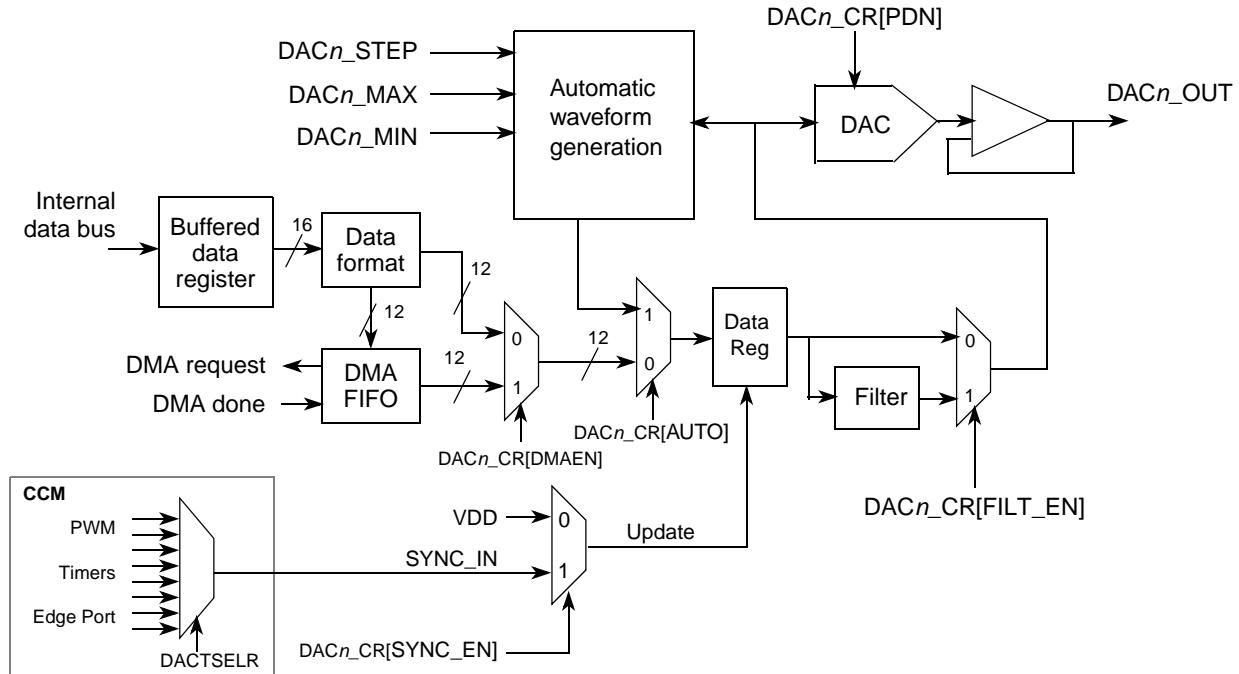


Figure 30-1. DAC Block Diagram

## 30.2 External Signal Descriptions

Table 30-1. External Signal Properties

Name	I/O	Function
<b>DACn_OUT</b>	O	Analog output. Each DAC has a single current mode analog output pin. The digital words to be converted are 12 bits long with an lsb representing 0.806 mV. Analog output ranges from $\sim V_{SSA} + 40$ mV to $\sim V_{DDA} - 40$ mV (actual output range can be found in the device's data sheet) and can drive a 3-k $\Omega$ load.
<b>DAC_VREFH</b>	—	DAC reference voltages
<b>DAC_VREFL</b>	—	<b>Note:</b> On this device, the DAC_VREFH and DAC_VREFL signals are internally connected to the DAC/ADC supply voltages, VDDA_DAC_ADC and VSSA_DAC_ADC, respectively.

### 30.3 Memory Map/Register Definition

Table 30-2 lists the DAC registers.

Table 30-2. DAC Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
DAC0 DAC1					
0xFC09_8000 0xFC09_C000	Control register (DACn_CR)	16	R/W	0x1101	<a href="#">30.3.1/30-3</a>
0xFC09_8002 0xFC09_C002	Buffered data register (DACn_DATA)	16	R/W	0x0000	<a href="#">30.3.2/30-5</a>
0xFC09_8004 0xFC09_C004	Step size register (DACn_STEP)	16	R/W	0x0000	<a href="#">30.3.3/30-5</a>
0xFC09_8006 0xFC09_C006	Minimum value register (DACn_MIN)	16	R/W	0x0000	<a href="#">30.3.4/30-6</a>
0xFC09_8008 0xFC09_C008	Maximum value register (DACn_MAX)	16	R/W	0xFFFF	<a href="#">30.3.5/30-6</a>
0xFC09_800A 0xFC09_C00A	Status register (DACn_SR)	16	R	0x0001	<a href="#">30.3.6/30-7</a>
0xFC09_800C 0xFC09_C00C	Filter count register (DACn_FILTCNT)	16	R/W	0x001D	<a href="#">30.3.7/30-7</a>

#### 30.3.1 DAC Control Register (DACn\_CR)

Address: 0xFC09\_8000 (DAC0\_CR)  
0xFC09\_C000 (DAC1\_CR)

Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	FILT_EN	0	0	WMLVL		DMA EN	HSLS	UP	DOWN	AUTO	SYNC_EN	FORMAT	PDN
W							0	0	0	0	0	0	0	0	0	1
Reset	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1

Figure 30-2. Control Register (DACn\_CR)

Table 30-3. DACn\_CR Field Descriptions

Field	Description
15–13	Reserved, must be cleared.
12 FILT_EN	Glitch filter enable. Enables the glitch suppression filter, which introduces a latency equivalent to DACn_FILTCNT internal bus clock cycles for DAC updates. 0 Filter disabled (not a use case) 1 Filter enabled <b>Note:</b> Always set this bit.
11–10	Reserved, must be cleared.

**Table 30-3. DACn\_CR Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>
9–8 WMLVL	Watermark level. Represents the FIFO level when a DMA request is sent. 00 0 01 2 10 4 11 6
7 DMAEN	DMA enable. Enables DMA support. When set, the analog DAC input fetches data from the FIFO. <ul style="list-style-type: none"> <li>• If SYNC_EN is cleared, data is read from the FIFO and presented to the DAC input every clock cycle.</li> <li>• If SYNC_EN is set, the data is read from FIFO on the SYNC_IN trigger and causes a delay of the data being presenting to the DAC input.</li> </ul> The data is written to the FIFO each clock cycle. Therefore, if DMAEN is set, always set SYNC_EN. 0 Disabled 1 Enabled
6 HSLS	High speed/low speed. Allows you to choose between DAC speed and power consumption. When cleared, high-speed mode is selected and the settling time of the DAC is 1 $\mu$ s, but consumes more power. When set, low-speed mode is selected, which saves power but the settling time increases to 2 $\mu$ s. 0 High speed mode 1 Low speed mode
5 UP	Enable up counting. Enables counting up in automatic mode. See <a href="#">Section 30.4.2, “Automatic Mode”</a> for details. 0 Disabled 1 Enabled
4 DOWN	Enable down counting. enables counting down in automatic mode. See <a href="#">Section 30.4.2, “Automatic Mode”</a> for details. 0 Disabled 1 Enabled
3 AUTO	Automatic waveform generation mode. When set, an external source driving SYNC_IN determines the data update rate, while the DACn_STEP, DACn_MIN, and DACn_MAX registers and the UP and DOWN bits shape the waveform. See <a href="#">Section 30.4.2, “Automatic Mode”</a> for a full description. If SYNC_EN is cleared in this mode, the data input to the DAC is updated every clock cycle. However, the DAC output is not able to keep up with this update rate. Therefore, never set this bit when SYNC_EN is cleared. 0 Normal mode, automatic waveform generation disabled 1 Automatic waveform generation enabled
2 SYNC_EN	Synchronization enable. Enables the SYNC_IN input to trigger an update of the buffered data being presented to the DAC input. If cleared, then asynchronous mode is selected and data written to DACn_DATA is presented to the DAC input in the following internal bus clock cycle. 0 Asynchronous mode — Data written to DACn_DATA is presented to DAC inputs on next clock cycle 1 Synchronous mode — Rising edge of SYNC_IN updates data presented to DAC input <b>Note:</b> If DMAEN is set, always set SYNC_EN.
1 FORMAT	Data format. Selects right or left justification of the 12-bit data words. 0 Right justified 1 Left justified
0 PDN	Power down. Removes power from the analog portion of the DAC when it is not used, resulting in its output being pulled low. This bit does not reset the registers and upon clearing of PDN the DAC outputs the value currently presented to its inputs (if DACOUT is set). The analog block requires 12 $\mu$ s to recover from the power down state before proper operation is guaranteed. 0 DAC is operational 1 DAC is powered down

### 30.3.2 Buffered Data Register (DACn\_DATA)

Address: 0xFC09\_8002 (DAC0\_DATA)  
0xFC09\_C002 (DAC1\_DATA)

Access: User read/write

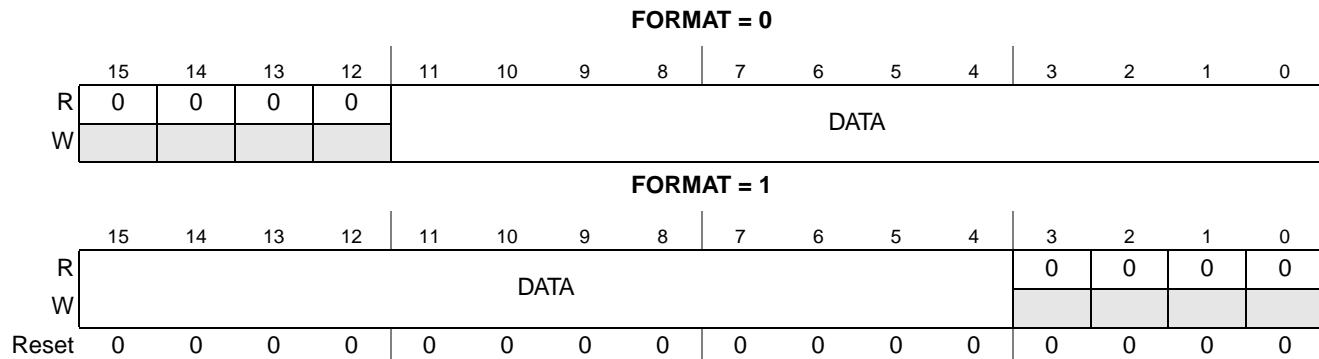


Figure 30-3. Buffered Data Register (DACn\_DATA)

Table 30-4. DACn\_DATA Field Descriptions

Field	Description
DATA	<p>When the DAC is in operational mode (PDN = 0), the digital data contained in this register is presented to the DAC upon the rising edge of the SYNC_IN signal (or at the next clock cycle if SYNC_EN is clear) and converted to analog and output by the DAC. The data in this register can be updated at any rate, but the DAC is only guaranteed to operate at a maximum update rate of 0.5 MHz for low speed and 1 MHz for high speed.</p> <p><b>Note:</b> Reading this register returns the value of the data presented to the analog DAC at that time, not the value in this register.</p>

### 30.3.3 Step Size Register (DACn\_STEP)

Address: 0xFC09\_8004 (DAC0\_STEP)  
0xFC09\_C004 (DAC1\_STEP)

Access: User read/write

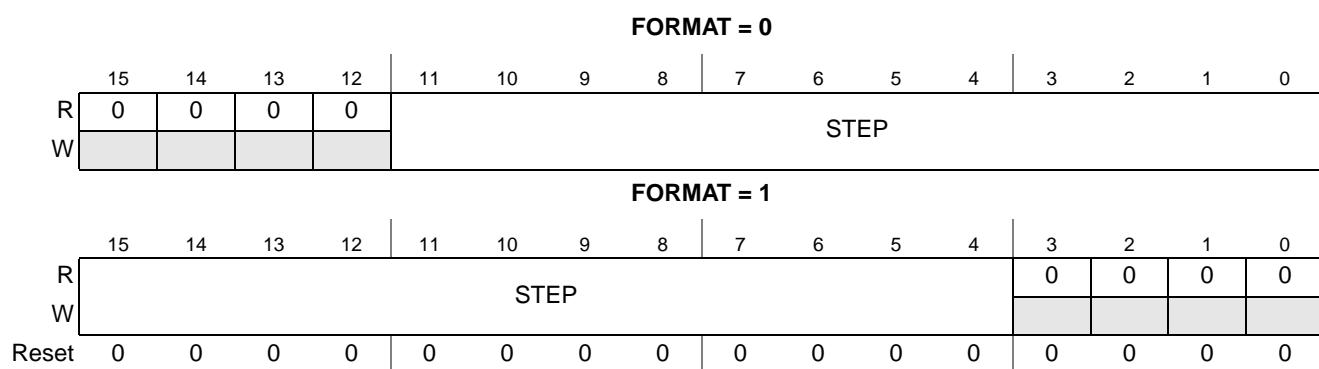


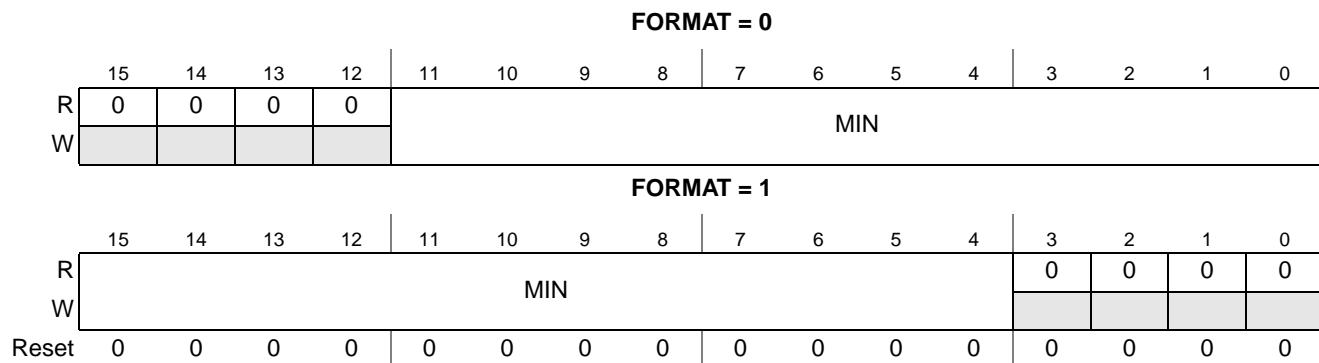
Figure 30-4. Step Size Register (DACn\_STEP)

**Table 30-5. DACn\_STEP Field Descriptions**

Field	Description
STEP	Step size. When the DAC is in automatic mode (DACn_CR[AUTO] = 1), the step size contained in this register is added to, or subtracted from, the current value creating the next value presented to the DAC inputs. This register is not used in normal mode operation.

**30.3.4 Minimum Value Register (DACn\_MIN)**Address: 0xFC09\_8006 (DAC0\_MIN)  
0xFC09\_C006 (DAC1\_MIN)

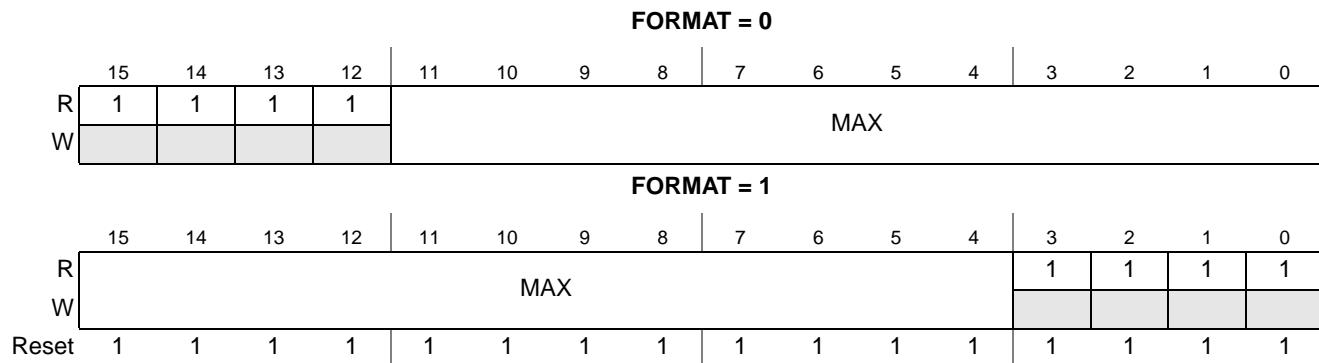
Access: User read/write

**Figure 30-5. Minimum Value Register (DACn\_MIN)****Table 30-6. DACn\_MIN Field Descriptions**

Field	Description
MIN	Minimum value. When the DAC is in automatic mode (DACn_CR[AUTO] = 1), the minimum value contained in this register acts as the lower range limit during automatic waveform generation. See <a href="#">Section 30.4.2, “Automatic Mode”</a> for more details. This register is not used in normal mode operation. Refer to the device’s data sheet for limitations on the low end voltage output of the DAC.

**30.3.5 Maximum Value Register (DACn\_MAX)**Address: 0xFC09\_8008 (DAC0\_MAX)  
0xFC09\_C008 (DAC1\_MAX)

Access: User read/write

**Figure 30-6. Maximum Value Register (DACn\_MAX)**

**Table 30-7. DACn\_MAX Field Descriptions**

Field	Description
MAX	Maximum value. When the DAC is in automatic mode (DACn_CR[AUTO] = 1), the maximum value contained in this register acts as the upper range limit during automatic waveform generation. See <a href="#">Section 30.4.2, “Automatic Mode”</a> for more details. This register is not used in normal mode operation. Refer to the device’s data sheet for limitations on the high end voltage output of the DAC.

### 30.3.6 Status Register (DACn\_SR)

Address: 0xFC09\_800A (DAC0\_SR)

Access: User read/write

0xFC09\_C00A (DAC1\_SR)

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

**Figure 30-7. Status Register (DACn\_SR)****Table 30-8. DACn\_SR Field Descriptions**

Field	Description
15–2	Reserved, must be cleared.
1 FULL	Indicates full status of DMA FIFO. 0 FIFO is not full 1 FIFO is full
0 EMPTY	Indicates empty status of DMA FIFO. 0 FIFO is not empty 1 FIFO is empty

### 30.3.7 Filter Count Register (DACn\_FILTCNT)

Address: 0xFC09\_800C (DAC0\_FILTCNT)

Access: User read/write

0xFC09\_C00C (DAC1\_FILTCNT)

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

**Figure 30-8. Filter Count Register (DACn\_FILTCNT)**

**Table 30-9. DACn\_FILTCNT Field Descriptions**

Field	Description
15–6	Reserved, must be cleared.
5–0 FILTCNT	Glitch filter count. Represents the number of internal bus clock cycles that the DAC output is held unchanged after new data is presented to the DAC's inputs. The number of clock cycles for which DAC output is held unchanged is FILT_CNT + 1. Approximately 240 ns is needed for worst case settling of the DAC output. Therefore, use a value of 29 for 125 MHz operation and a value of 33 for 140 MHz operation. The reset value is 29. <b>Note:</b> When using the glitch filter ensure that the filter count is less than the update count otherwise the DAC output is never updated.

## 30.4 Functional Description

### 30.4.1 Normal Mode

The DAC receives data words via the DACn\_DATA register. This digital word is applied to the DAC inputs based on DACn\_CR[SYNC\_EN]. The analog DAC generates an analog representation of the digital word in less than two microseconds.

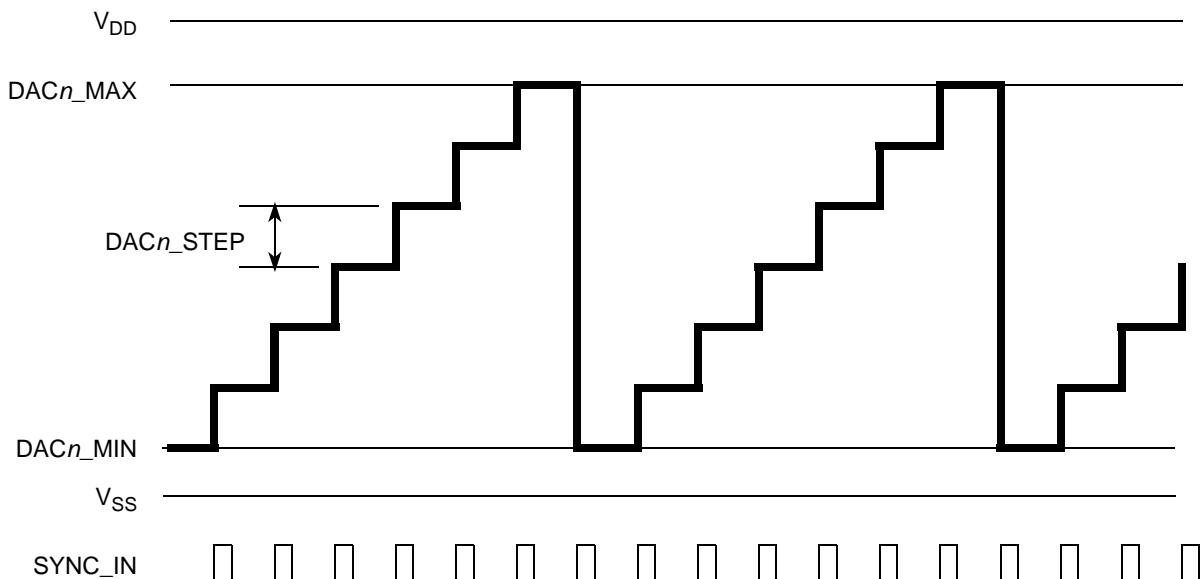
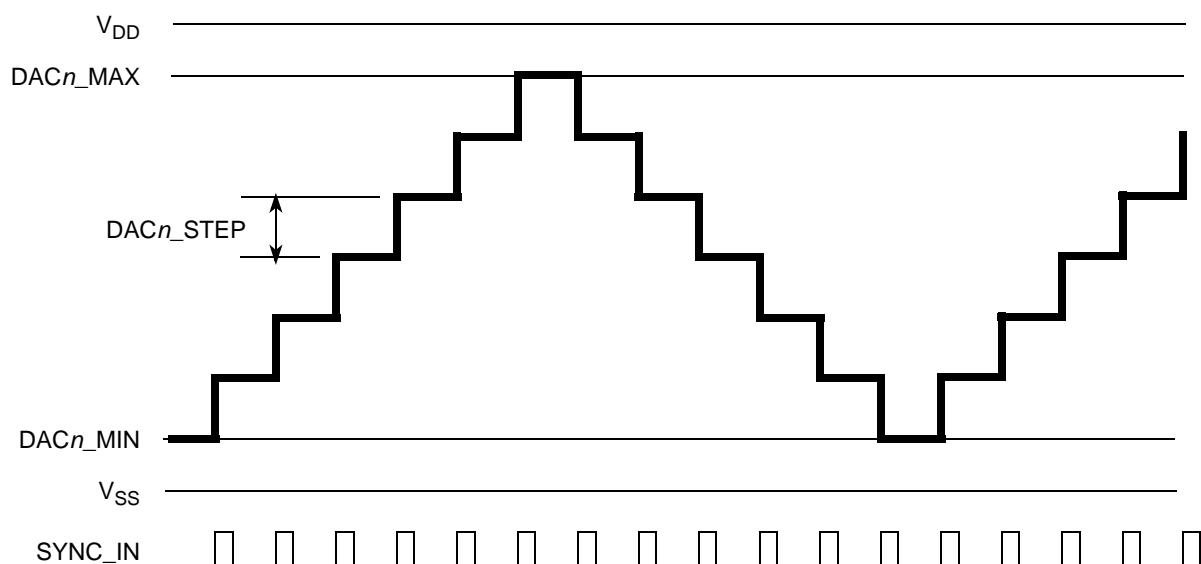
### 30.4.2 Automatic Mode

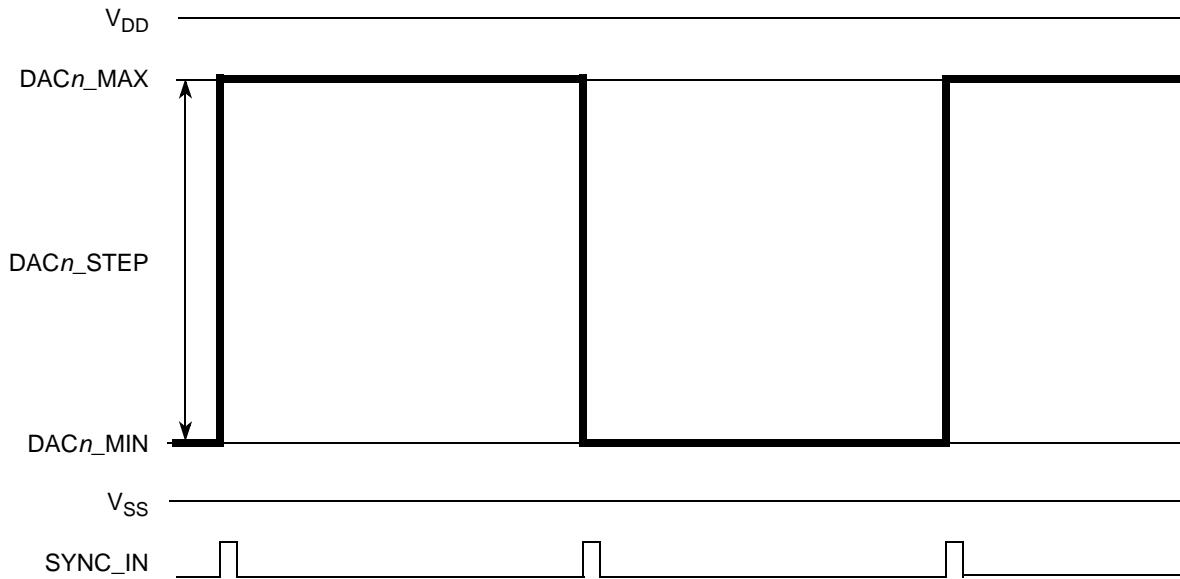
This mode allows the automatic generation of triangle, sawtooth, and square waveforms without CPU intervention. The update rate, incremental step size, and minimum and maximum values are programmable.

1. The value in the DACn\_DATA register is used as a starting point.
2. When the SYNC\_IN input triggers an update of the data presented to the DAC, the DACn\_STEP value is added to or subtracted from the current DACn\_DATA value.
3. If DACn\_CR[UP] is set, then DACn\_STEP is added to DACn\_DATA each update until maximum value (DACn\_MAX) is reached.
4. If DACn\_CR[DOWN] is set, the generator begins subtracting STEP from DATA. If DACn\_CR[DOWN] is cleared, the generator reloads the minimum value (DACn\_MIN).
5. Similarly, upon the data reaching DACn\_MIN while counting down, the generator resumes up counting if DACn\_CR[UP] is set, or reloads MAXVAL if DACn\_CR[UP] is cleared.
6. The initial direction of the count depends on which bit (UP or DOWN) is set last. The following figures illustrate several examples of automatically-generated waveforms.

#### NOTE

The waveforms shown are ideal. Actual waveforms are limited by the slew rate of the DAC output.

**Figure 30-9. Sawtooth Waveform Example with UP = 1 and DOWN = 0****Figure 30-10. Triangle Waveform Example with UP = 1 and DOWN = 1**

**Figure 30-11. Square Wave Waveform Example with UP = 1 and DOWN = 0**

The above examples show the waveform period is a function of the difference between  $DACn\_MAX$  and  $DACn\_MIN$ ,  $DACn\_STEP$ , and the update rate as shown below.

$$\text{Period} = 2 \times \left[ \frac{DACn\_MAX - DACn\_MIN}{DACn\_STEP} \times \text{UpdatePeriod} \right] \quad \text{Eqn. 30-1}$$

- Increasing  $DACn\_STEP$  decreases the resolution of the output steps.
- Increasing the update rate decreases the waveform period.
- Varying  $DACn\_MIN$  and  $DACn\_MAX$  changes the DC offset and the amplitude of the waveform.

### 30.4.3 Sources of Waveform Distortion

#### 30.4.3.1 Switching Glitches

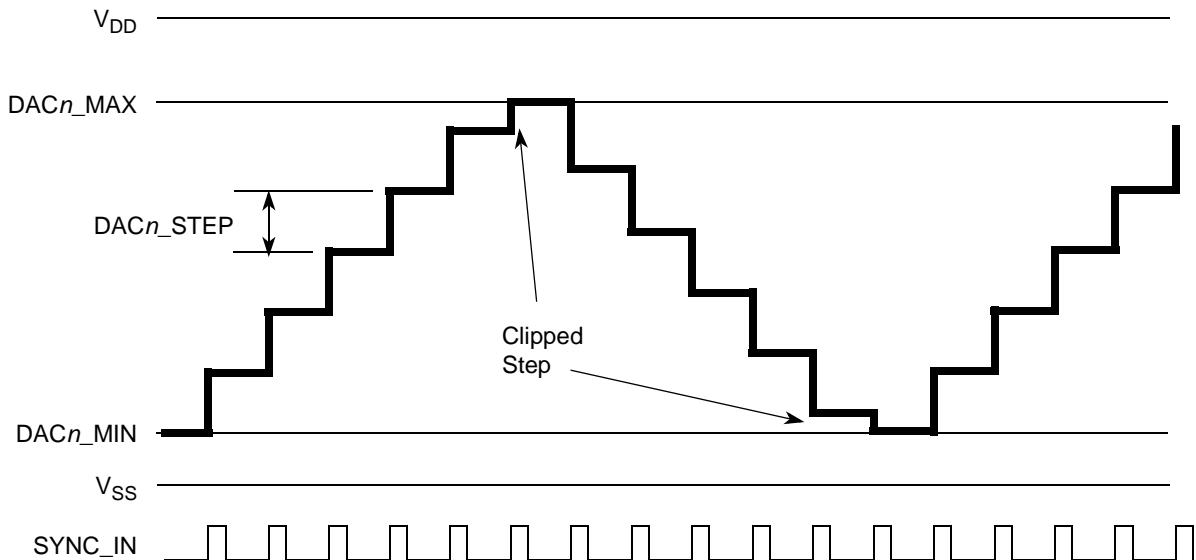
When a new digital value is presented to the DAC input, some glitches may appear on the output as the new values propagate through the circuitry. Eventually, these glitches settle out and the output slews to its new value. These glitches can be avoided by setting  $DACn\_CR[FILT\_EN]$  and a suitable value for  $DACn\_FILTCNT$ . This causes the DAC to hold its current output for a number of clock cycles equal to  $DACn\_FILTCNT$  while the switching glitches settle out. After the filter time is satisfied, the output smoothly slews to the new value.

#### 30.4.3.2 Slew Effects

The example waveforms are ideal waveforms and show transitions as step functions. In reality the DAC output has a finite slew rate designed to round off the steps. Whether this rounding off is noticeable depends on the output step size (larger output changes require longer settling times) and on the update period (longer dwell times make the settling times less noticeable).

### 30.4.3.3 Clipping Effects (Auto Mode Only)

One form of clipping can occur during automatic waveform generation when the difference between  $DACn\_MAX$  and  $DACn\_MIN$  is not a near even multiple of the  $DACn\_STEP$  value. This results in a less than whole step as the waveform approaches  $DACn\_MAX$  and  $DACn\_MIN$ . An example is drawn below.



**Figure 30-12. Triangle Waveform Example with Clipping**

Another form of clipping occurs when  $DACn\_MAX$  or  $DACn\_MIN$  is beyond the output range of the DAC. The maximum and minimum voltages potentially driven out are defined in the device's data sheet.

## 30.5 Initialization/Application Information

Assume the desire to create a waveform to go down from 3.0 to 1.5 V in 1 millisecond.

1. Calculate  $DACn\_MAX$  and  $DACn\_MIN$ . Based on each DAC least significant bit (lsb) representing 0.806 mV the results are:

$$DACn\_MAX = 1.5 / 0.000806 = 0xE8A$$

$$DACn\_MIN = 3.0 / 0.000806 = 0x745$$

2. This represents a difference of:

$$0xE8A - 0x745 = 1861 \text{ lsbs to be accomplished in one millisecond}$$

3. A one-millisecond time period means the DAC can safely update 500 times since the DAC has a two-microsecond conversion time.

4. To go from  $DACn\_MAX$  to  $DACn\_MIN$  in 500 steps requires:

$$DACn\_STEP = \text{round}\{1861 / 500\} = 0x004$$

5. To go from  $DACn\_MAX$  to  $DACn\_MIN$  with a  $DACn\_STEP$  size of 0x004 requires:

$$1861 / 4 = 465 \text{ steps}$$

6. To keep the waveform period of one millisecond using 465 updates calls for an update period of:

$$465 / 1 \text{ ms} = 465 \text{ kHz}$$

7. Assuming operation is using a 32 MHz system clock, program a timer module to pulse the SYNC\_IN input every:

$$32,000,000 / 465,000 = 69 \text{ counts}$$

When the timer module is programmed, along with the  $\text{DAC}_n\text{\_MAX}$ ,  $\text{DAC}_n\text{\_MIN}$ , and  $\text{DAC}_n\text{\_STEP}$  registers:

1. Write  $\text{DAC}_n\text{\_DATA}$  with a value equal to  $\text{DAC}_n\text{\_MAX}$  (because only a count down is occurring) as a starting point.

#### NOTE

When writing to the DAC registers, be certain  $\text{DAC}_n\text{\_CR[FORMAT]}$  is the desired value and the data values are properly justified to match this bit.

2. Set  $\text{DAC}_n\text{\_CR[SYNC\_EN, DOWN, AUTO]}$ .
3. Optionally, set  $\text{DAC}_n\text{\_FILTCNT}$  and  $\text{DAC}_n\text{\_CR[FILT\_EN]}$  to suppress glitches on the output.
4. Set  $\text{DAC}_n\text{\_CR[DACOUT]}$  and clear  $\text{DAC}_n\text{\_CR[UP, PDN]}$ .
5. The desired waveform begins within 12 microseconds from the clearing of PDN and continues until PDN is set or the timer is stopped.

# Chapter 31

## 10/100Mbps Ethernet MAC-NET Core

### 31.1 Introduction

The MAC-NET core, in conjunction with a 10/100 MAC, implements layer 3 network acceleration functions. These functions are designed to accelerate the processing of various common networking protocols, such as IP, TCP, UDP and ICMP, providing wire speed services to client applications.

#### NOTE

~~This device contains two MAC-NET modules. However, MAC-NET1 is only available when both MAC-NETs operate in RMII mode.~~

#### 31.1.1 Overview

The core implements a dual speed 10/100 Mbps Ethernet MAC compliant with the IEEE802.3-2002 standard. The MAC layer provides compatibility with half- or full-duplex 10/100Mbps Ethernet and Fast Ethernet LANs.

The MAC operation is fully programmable and can be used in NIC (Network Interface Card), bridging, or switching applications. The core implements the remote network monitoring (RMON) counters according to IETF RFC 2819.

The core also implements a hardware acceleration block to optimize the performance of network controllers providing IP and TCP, UDP, ICMP protocol services. The acceleration block performs critical functions in hardware, which are typically implemented with large software overhead.

The core implements programmable embedded FIFOs that can provide buffering on the receive path for loss-less flow control

Advanced power management features are available with magic packet detection and programmable power-down modes.

For industrial automation application, the IEEE 1588 standard is becoming the main technology for precise time synchronization on Ethernet networks. This provides accurate clock synchronization for distributed control nodes to overcome one of the drawbacks of Ethernet.

The programmable 10/100 Ethernet MAC with IEEE 1588 support integrates a standard IEEE 802.3 Ethernet MAC with a time-stamping module.

The core provides a flexible and evolutive solution for a large number of applications, such as SAN (storage area network), NAS (network attached storage), enterprise file servers, firewalls, gateways, or routers.

### 31.1.2 Features

The MAC-NET core includes the following features.

#### 31.1.2.1 Ethernet MAC Features

- Implements the full 802.3 specification with preamble/SFD generation, frame padding generation, CRC generation and checking
- Dynamically configurable to support 10/100 Mbps operation
- Supports full duplex and configurable half duplex operation
- Supports AMD magic packet detection with interrupt for node remote power management
- Seamless interface to commercial Fast Ethernet PHY device via a 4-bit medium independent interface (MII) operating at 25 MHz
- Simple 64-Bit FIFO interface to user application
- CRC-32 checking at full speed with optional forwarding of the frame check sequence (FCS) field to the client
- CRC-32 generation and append on transmit or forwarding of user application provided FCS selectable on a per-frame basis
- When operating in full duplex mode
  - Implements automated pause frame (802.3 x31A) generation and termination providing flow control without user application intervention
  - Pause quanta used to form pause frames, dynamically programmable
  - Pause frame generation additionally controllable by user application offering flexible traffic flow control
  - Optional forwarding of received pause frames to the user application
  - Implements standard flow-control mechanism
- In half-duplex mode, provides full collision support, including jamming, backoff, and automatic retransmission
- Support for VLAN-tagged frames according to IEEE 802.1Q
- Programmable MAC address: Insertion on transmit; discards frames with mismatching destination address on receive (except broadcast and pause frames)
- Programmable group of four supplemental MAC addresses to filter unicast traffic
- Programmable promiscuous mode support to omit MAC destination address checking on receive
- Multicast and unicast address filtering on receive based on 64 entries hash table reducing higher layer processing load
- Programmable frame maximum length providing support for any standard or proprietary frame length
- Statistics indicators for frame traffic and errors (alignment, CRC, length) and pause frames providing for IEEE 802.3 basic and mandatory management information database (MIB) package and remote network monitoring (RFC 2819)

- Simple handshake user application FIFO interface with fully programmable depth and threshold levels ensuring data rates of 1Gbps
- 64-bit client FIFO interface
- Separate status word available for each received frame on the user interface providing information such as frame length, frame type, VLAN tag, and error information
- Multiple internal loopback options
- MDIO master interface for PHY device configuration and management with two programmable MDIO base addresses
- Supports legacy FEC buffer descriptors

### 31.1.2.2 IP Protocol Performance Optimization Features

- Operates on TCP/IP and UDP/IP and ICMP/IP protocol data or IP header only
- Enables wire-speed processing
- IPv4 and IPv6 support
- Transparent passing of frames of other types and protocols
- Support for VLAN tagged frames according to IEEE 802.1q with transparent forwarding of VLAN tag and control field
- Automatic IP-header and payload (protocol specific) checksum calculation and verification on receive
- Automatic IP-header and payload (protocol specific) checksum generation and automatic insertion on transmit configurable on a per-frame basis
- Support for IP and TCP, UDP, ICMP data for checksum generation and checking
- Full header options support for IPv4 and TCP protocol headers
- IPv6 support limited to datagrams with base header only. Datagrams with extension headers are passed transparently unmodified/unchecked.
- Statistics information for received IP and protocol errors
- Configurable automatic discard of erroneous frames
- Configurable automatic host-to-network (RX) and network-to-host (TX) byte order conversion for IP and TCP/UDP/ICMP headers within the frame
- Configurable padding remove for short IP datagrams on receive
- Configurable Ethernet payload alignment to allow for 32-bit word aligned header and payload processing
- Programmable store-and-forward operation with clock and rate decoupling FIFOs

### 31.1.2.3 IEEE 1588 Functions

- Support for all IEEE 1588 frames
- Reference clock can be chosen independently of the network speed
- Software-programmable precise time-stamping of ingress and egress frames
- Timer monitoring capabilities for system calibration and timing accuracy management

- Precise time-stamping of external events with programmable interrupt generation
- Programmable event and interrupt generation for external system control
- Hardware- and software-controllable timer synchronization

### 31.1.3 Block Diagram

Figure 31-1. 10/100 Ethernet MAC-NET Core Block Diagram

## 31.2 External Signal Description

Table 31-2. ENET Signal Descriptions

Signal		Description
MII	RMII	
MII_COL	—	Asserted upon detection of a collision and remains asserted while the collision persists. This signal is not defined for full-duplex mode.
MII_CRS	—	Carrier sense. When asserted, indicates transmit or receive medium is not idle. In RMII mode, this signal is present on the RMII_CRS_DV pin.
MII_MDC	RMII_MDC	Output clock provides a timing reference to the PHY for data transfers on the MDIO signal.
MII_MDIO	RMII_MDIO	Transfers control information between the external PHY and the media-access controller. Data is synchronous to MDC. This signal is an input after reset.
MII_RXCLK	—	Provides a timing reference for RXDV, RXD[3:0], and RXER.
MII_RXDV	RMII_CRS_DV	Asserting this input indicates the PHY has valid nibbles present on the MII. RXDV must remain asserted from the first recovered nibble of the frame through to the last nibble. Asserting RXDV must start no later than the SFD and exclude any EOF. In RMII mode, this pin also generates the CRS signal.
MII_RXD0	RMII_RXD0	Contains the Ethernet input data transferred from PHY to the media-access controller when RXDV is asserted.
MII_RXD1	RMII_RXD1	
MII_RXD[3:2]	—	
MII_RXER	RMII_RXER	When asserted with RXDV, indicates the PHY detects an error in the current frame. When RXDV is negated, RXER has no effect.
MII_TXCLK	—	Input clock which provides a timing reference for TXEN, TXD[3:0], and TXER.
MII_TXD0	RMII_TXD0	The serial output Ethernet data and only valid during the assertion of TXEN.
MII_TXD1	RMII_TXD1	
MII_TXD[3:2]	—	
MII_TXEN	RMII_TXEN	Indicates when valid nibbles are present on the MII. This signal is asserted with the first nibble of a preamble and is negated before the first TXCLK following the final nibble of the frame.
MII_TXER	—	When asserted for one or more clock cycles while TXEN is also asserted, PHY sends one or more illegal symbols. TXER has no effect at 10 Mbps or when TXEN is negated.
—	RMII_REF_CLK	In RMII mode, this signal is the reference clock for receive, transmit, and the control interface.

### 31.3 Memory Map/Register Definition

The MAC with its corresponding management counters implements a register space of 512 32-bit registers. All writes must provide 32-bit of data and all reads return 32-bits of data.

Reserved bits should be written with 0 and ignored on read to allow future extension.

Unused registers read back 0 and a write has no effect.

**Table 31-3. Register Map Summary**

Address	Section	Description
<b>MACNET0</b>		
0xFC0D_4000	Configuration	Core control and status registers
0xFC0D_4200	Statistics counters	MIB block counters.
0xFC0D_4400	1588 control	1588 adjustable timer (TSM) and 1588 frame control.
0xFC0D_4500	MAC addresses	Supplemental unicast MAC addresses
<b>MACNET1</b>		
0xFC0D_8000	Configuration	Core control and status registers
0xFC0D_8200	Statistics counters	MIB block counters.
0xFC0D_8400	1588 control	1588 adjustable timer (TSM) and 1588 frame control.
0xFC0D_8500	MAC addresses	Supplemental unicast MAC addresses

**Table 31-4. MACNET Memory Map**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
<b>Core and Configuration Registers</b>					
0xFC0D_4004 0xFC0D_8004	Interrupt event register (ENETn_EIR)	32	R/W	0x00000000	<a href="#">31.3.1/31-12</a>
0xFC0D_4008 0xFC0D_8008	Interrupt mask register (ENETn_EIMR)	32	R/W	0x00000000	<a href="#">31.3.2/31-13</a>
0xFC0D_4010 0xFC0D_8010	Receive descriptor active register (ENETn_RDAR)	32	R/W	0x00000000	<a href="#">31.3.3/31-14</a>
0xFC0D_4014 0xFC0D_8014	Transmit descriptor active register (ENETn_TDAR)	32	R/W	0x00000000	<a href="#">31.3.4/31-15</a>
0xFC0D_4024 0xFC0D_8024	Ethernet control register (ENETn_ECR)	32	R/W	0xF000_0000	<a href="#">31.3.5/31-16</a>
0xFC0D_4040 0xFC0D_8040	MII management frame register (ENETn_MMFR)	32	R/W	0x0000_0000	<a href="#">31.3.6/31-17</a>

Table 31-4. MACNET Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0D_4044 0xFC0D_8044	MII speed control register (ENET $n$ _MSCR)	32	R/W	0x0000_0000	<a href="#">31.3.7/31-18</a>
0xFC0D_4064 0xFC0D_8064	MIB control/status register (ENET $n$ _MIBC)	32	R/W	0xC000_0000	<a href="#">31.3.8/31-19</a>
0xFC0D_4084 0xFC0D_8084	Receive control register (ENET $n$ _RCR)	32	R/W	0x05EE_0001	<a href="#">31.3.9/31-20</a>
0xFC0D_40C4 0xFC0D_80C4	Transmit control register (ENET $n$ _TCR)	32	R/W	0x0000_0000	<a href="#">31.3.10/31-21</a>
0xFC0D_40E4 0xFC0D_80E4	Physical address low register (ENET $n$ _PALR)	32	R/W	0x0000_0000	<a href="#">31.3.11/31-22</a>
0xFC0D_40E8 0xFC0D_80E8	Physical address high register (ENET $n$ _PAUR)	32	R/W	0x0000_8808	<a href="#">31.3.12/31-23</a>
0xFC0D_40EC 0xFC0D_80EC	Opcode/pause duration (ENET $n$ _OPD)	32	R/W	0x0001_0000	<a href="#">31.3.13/31-23</a>
0xFC0D_4118 0xFC0D_8118	Descriptor individual upper address register (ENET $n$ _IAUR)	32	R/W	0x0000_0000	<a href="#">31.3.14/31-24</a>
0xFC0D_411C 0xFC0D_811C	Descriptor individual lower address register (ENET $n$ _IALR)	32	R/W	0x0000_0000	<a href="#">31.3.15/31-24</a>
0xFC0D_4120 0xFC0D_8120	Descriptor group upper address register (ENET $n$ _GAUR)	32	R/W	0x0000_0000	<a href="#">31.3.16/31-25</a>
0xFC0D_4124 0xFC0D_8124	Descriptor group upper address register (ENET $n$ _GALR)	32	R/W	0x0000_0000	<a href="#">31.3.17/31-25</a>
0xFC0D_4144 0xFC0D_8144	Transmit FIFO watermark and store/forward control (ENET $n$ _TFWR)	32	R/W	0x0000_0000	<a href="#">31.3.18/31-26</a>
0xFC0D_414C 0xFC0D_814C	FIFO receive bound register (ENET $n$ _FRBR) <b>Note:</b> Not implemented. Available for software-compatibility.	32	R	0x0000_0600	—
0xFC0D_4150 0xFC0D_8150	FIFO receive start register (ENET $n$ _FRSR) <b>Note:</b> Not implemented. Available for software-compatibility.	32	RW	0x0000_0500	—
0xFC0D_4180 0xFC0D_8180	Receive descriptor ring start register (ENET $n$ _RDSR)	32	R/W	0x0000_0000	<a href="#">31.3.19/31-26</a>
0xFC0D_4184 0xFC0D_8184	Transmit descriptor ring start register (ENET $n$ _TDSR)	32	R/W	0x0000_0000	<a href="#">31.3.20/31-27</a>
0xFC0D_4188 0xFC0D_8188	Maximum receive buffer size (ENET $n$ _MRBR)	32	R/W	0x0000_0000	<a href="#">31.3.21/31-27</a>
0xFC0D_4190 0xFC0D_8190	Receive FIFO section full threshold (ENET $n$ _RSFL)	32	R/W	0x0000_0000	<a href="#">31.3.22/31-28</a>
0xFC0D_4194 0xFC0D_8194	Receive FIFO section empty threshold (ENET $n$ _RSEM)	32	R/W	0x0000_0000	<a href="#">31.3.23/31-29</a>

**Table 31-4. MACNET Memory Map (continued)**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
<b>MACNET0 MACNET1</b>					
0xFC0D_4198 0xFC0D_8198	Receive FIFO almost empty threshold (ENETn_RAEM)	32	R/W	0x0000_0004	<a href="#">31.3.24/31-29</a>
0xFC0D_419C 0xFC0D_819C	Receive FIFO almost full threshold (ENETn_RAFL)	32	R/W	0x0000_0004	<a href="#">31.3.25/31-29</a>
0xFC0D_41A0 0xFC0D_81A0	Transmit FIFO section empty threshold (ENETn_TSEM)	32	R/W	0x0000_0000	<a href="#">31.3.26/31-30</a>
0xFC0D_41A4 0xFC0D_81A4	Transmit FIFO almost empty threshold (ENETn_TAEM)	32	R/W	0x0000_0004	<a href="#">31.3.27/31-30</a>
0xFC0D_41A8 0xFC0D_81A8	Transmit FIFO almost full threshold (ENETn_TAFL)	32	R/W	0x0000_0008	<a href="#">31.3.28/31-31</a>
0xFC0D_41AC 0xFC0D_81AC	Transmit inter-packet gap (ENETn_TIPG)	32	R/W	0x0000_000C	<a href="#">31.3.29/31-31</a>
0xFC0D_41B0 0xFC0D_81B0	Frame truncation length (ENETn_FTRL)	32	R/W	0x0000_07FF	<a href="#">31.3.30/31-32</a>
0xFC0D_41C0 0xFC0D_81C0	Transmit accelerator function configuration (ENETn_TACC)	32	RW	0x0000_0000	<a href="#">31.3.31/31-32</a>
0xFC0D_41C4 0xFC0D_81C4	Receive accelerator function configuration (ENETn_RACC)	32	RW	0x0000_0000	<a href="#">31.3.32/31-33</a>
<b>Statistics Counters</b> (see <a href="#">Table 31-5</a> )					
<b>IEEE 1588 Control and Adjustable Timer Module (TSM)</b>					
0xFC0D_4400 0xFC0D_8400	Timer control register (ENETn_ATCR)	32	RW	0x0000_0000	<a href="#">31.3.33/31-34</a>
0xFC0D_4404 0xFC0D_8404	Timer value register (ENETn_ATVR)	32	RW	0x0000_0000	<a href="#">31.3.34/31-35</a>
0xFC0D_4408 0xFC0D_8408	Offset value for one-shot event generation (ENETn_ATOFF)	32	RW	0x0000_0000	<a href="#">31.3.35/31-36</a>
0xFC0D_440C 0xFC0D_840C	Timer period (ENETn_ATPER)	32	RW	0x3B9A_CA00	<a href="#">31.3.36/31-36</a>
0xFC0D_4410 0xFC0D_8410	Correction counter wrap-around value (ENETn_ATCOR)	32	RW	0x0000_0000	<a href="#">31.3.37/31-36</a>
0xFC0D_4414 0xFC0D_8414	Timestamp clock period and correction increment (ENETn_ATINC)	32	RW	0x0000_0000	<a href="#">31.3.38/31-37</a>
0xFC0D_4418 0xFC0D_8418	Timestamp of last transmitted frame (ENETn_ATSTMP)	32	R	0x0000_0000	<a href="#">31.3.39/31-37</a>

**Table 31-4. MACNET Memory Map (continued)**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
MACNET0 MACNET1					
<b>Optional Supplemental MAC Addresses</b>					
0xFC0D_4500 0xFC0D_8500	Supplemental MAC address lower 0 (ENET $n$ _SMACLO)	32	RW	Undefined	<a href="#">31.3.40/31-38</a>
0xFC0D_4504 0xFC0D_8504	Supplemental MAC address upper 0 (ENET $n$ _SMACU0)	32	RW	Undefined	<a href="#">31.3.41/31-38</a>
0xFC0D_4508 0xFC0D_8508	Supplemental MAC address lower 1 (ENET $n$ _SMACL1)	32	RW	Undefined	<a href="#">31.3.40/31-38</a>
0xFC0D_450C 0xFC0D_850C	Supplemental MAC address upper 1 (ENET $n$ _SMACU1)	32	RW	Undefined	<a href="#">31.3.41/31-38</a>
0xFC0D_4510 0xFC0D_8510	Supplemental MAC address lower 2 (ENET $n$ _SMACL2)	32	RW	Undefined	<a href="#">31.3.40/31-38</a>
0xFC0D_4514 0xFC0D_8514	Supplemental MAC address upper 2 (ENET $n$ _SMACU2)	32	RW	Undefined	<a href="#">31.3.41/31-38</a>
0xFC0D_4518 0xFC0D_8518	Supplemental MAC address lower 3 (ENET $n$ _SMACL3)	32	RW	Undefined	<a href="#">31.3.40/31-38</a>
0xFC0D_451C 0xFC0D_851C	Supplemental MAC Address Upper 3 (ENET $n$ _SMACU3)	32	RW	Undefined	<a href="#">31.3.41/31-38</a>

**Table 31-5. Statistic Event Counters**

Address	Register
MACNET0 MACNET1	
0xFC0D_4200 0xFC0D_8200	Count of frames not counted correctly (RMON_T_DROP) <b>Note:</b> Counter not implemented (read 0 always) as not applicable
0xFC0D_4204 0xFC0D_8204	RMON Tx packet count (RMON_T_PACKETS)
0xFC0D_4208 0xFC0D_8208	RMON Tx Broadcast Packets (RMON_T_BC_PKT)
0xFC0D_420C 0xFC0D_820C	RMON Tx Multicast Packets (RMON_T_MC_PKT)
0xFC0D_4210 0xFC0D_8210	RMON Tx Packets w CRC/Align error (RMON_T_CRC_ALIGN)
0xFC0D_4214 0xFC0D_8214	RMON Tx Packets < 64 bytes, good CRC (RMON_T_UNDERSIZE)
0xFC0D_4218 0xFC0D_8218	RMON Tx Packets > MAX_FL bytes, good CRC (RMON_T_OVERSIZE)

**Table 31-5. Statistic Event Counters (continued)**

Address	Register
MACNET0 MACNET1	
0xFC0D_421C 0xFC0D_821C	RMON Tx Packets < 64 bytes, bad CRC (RMON_T_FRAG)
0xFC0D_4220 0xFC0D_8220	RMON Tx Packets > MAX_FL bytes, bad CRC (RMON_T_JAB)
0xFC0D_4224 0xFC0D_8224	RMON Tx collision count (RMON_T_COL)
0xFC0D_4228 0xFC0D_8228	RMON Tx 64 byte packets (RMON_T_P64)
0xFC0D_422C 0xFC0D_822C	RMON Tx 65 to 127 byte packets (RMON_T_P65TO127n)
0xFC0D_4230 0xFC0D_8230	RMON Tx 128 to 255 byte packets (RMON_T_P128TO255n)
0xFC0D_4234 0xFC0D_8234	RMON Tx 256 to 511 byte packets (RMON_T_P256TO511)
0xFC0D_4238 0xFC0D_8238	RMON Tx 512 to 1023 byte packets (RMON_T_P512TO1023)
0xFC0D_423C 0xFC0D_823C	RMON Tx 1024 to 2047 byte packets (RMON_T_P1024TO2047)
0xFC0D_4240 0xFC0D_8240	RMON Tx packets w > 2048 bytes (RMON_T_P_GTE2048)
0xFC0D_4244 0xFC0D_8244	RMON Tx Octets (RMON_T_OCTETS)
0xFC0D_4248 0xFC0D_8248	Count of frames not counted correctly (IEEE_T_DROP) <b>Note:</b> Counter not implemented (read 0 always) as not applicable
0xFC0D_424C 0xFC0D_824C	Frames Transmitted OK (IEEE_T_FRAME_OK)
0xFC0D_4250 0xFC0D_8250	Frames Transmitted with Single Collision (IEEE_T_1COL)
0xFC0D_4254 0xFC0D_8254	Frames Transmitted with Multiple Collisions (IEEE_T_MCOL)
0xFC0D_4258 0xFC0D_8258	Frames Transmitted after Deferral Delay (IEEE_T_DEF)
0xFC0D_425C 0xFC0D_825C	Frames Transmitted with Late Collision (IEEE_T_LCOL)
0xFC0D_4260 0xFC0D_8260	Frames Transmitted with Excessive Collisions (IEEE_T_EXCOL)
0xFC0D_4264 0xFC0D_8264	Frames Transmitted with Tx FIFO Underrun (IEEE_T_MACERR)

**Table 31-5. Statistic Event Counters (continued)**

Address	Register
MACNET0 MACNET1	
0xFC0D_4268 0xFC0D_8268	Frames Transmitted with Carrier Sense Error (IEEE_T_CSERR)
0xFC0D_426C 0xFC0D_826C	Frames Transmitted with SQE Error (IEEE_T_SQE) <b>Note:</b> Counter not implemented (read 0 always) as no SQE information is available
0xFC0D_4270 0xFC0D_8270	Flow Control Pause frames transmitted (IEEE_T_FDXFC)
0xFC0D_4274 0xFC0D_8274	Octet count for Frames Transmitted w/o Error (IEEE_T_OCTETS_OK) <b>Note:</b> Counts total octets (includes header and FCS fields)
0xFC0D_4284 0xFC0D_8284	RMON Rx packet count (RMON_R_PACKETS)
0xFC0D_4288 0xFC0D_8288	RMON Rx Broadcast Packets (RMON_R_BC_PKT)
0xFC0D_428C 0xFC0D_828C	RMON Rx Multicast Packets (RMON_R_MC_PKT)
0xFC0D_4290 0xFC0D_8290	RMON Rx Packets w CRC/Align error (RMON_R_CRC_ALIGN)
0xFC0D_4294 0xFC0D_8294	RMON Rx Packets < 64 bytes, good CRC (RMON_R_UNDERSIZE)
0xFC0D_4298 0xFC0D_8298	RMON Rx Packets > MAX_FL, good CRC (RMON_R_OVERSIZE)
0xFC0D_429C 0xFC0D_829C	RMON Rx Packets < 64 bytes, bad CRC (RMON_R_FRAG)
0xFC0D_42A0 0xFC0D_82A0	RMON Rx Packets > MAX_FL bytes, bad CRC (RMON_R_JAB)
0xFC0D_42A4 0xFC0D_82A4	Reserved (RMON_R_RESVD_0)
0xFC0D_42A8 0xFC0D_82A8	RMON Rx 64 byte packets (RMON_R_P64)
0xFC0D_42AC 0xFC0D_82AC	RMON Rx 65 to 127 byte packets (RMON_R_P65TO127)
0xFC0D_42B0 0xFC0D_82B0	RMON Rx 128 to 255 byte packets (RMON_R_P128TO255)
0xFC0D_42B4 0xFC0D_82B4	RMON Rx 256 to 511 byte packets (RMON_R_P256TO511)
0xFC0D_42B8 0xFC0D_82B8	RMON Rx 512 to 1023 byte packets (RMON_R_P512TO1023)
0xFC0D_42BC 0xFC0D_82BC	RMON Rx 1024 to 2047 byte packets (RMON_R_P1024TO2047)

**Table 31-5. Statistic Event Counters (continued)**

Address	Register
MACNET0 MACNET1	
0xFC0D_42C0 0xFC0D_82C0	RMON Rx packets w > 2048 bytes (RMON_R_P_GTE2048)
0xFC0D_42C4 0xFC0D_82C4	RMON Rx Octets (RMON_R_OCTETS)
0xFC0D_42C8 0xFC0D_82C8	Count of frames not counted correctly (IEEE_R_DROP) <b>Note:</b> Counter increments if a frame with invalid/missing SFD character is detected and has been dropped. None of the other counters increments if this counter increments.
0xFC0D_42CC 0xFC0D_82CC	Frames Received OK (IEEE_R_FRAME_OK)
0xFC0D_42D0 0xFC0D_82D0	Frames Received with CRC Error (IEEE_R_CRC)
0xFC0D_42D4 0xFC0D_82D4	Frames Received with Alignment Error (IEEE_R_ALIGN)
0xFC0D_42D8 0xFC0D_82D7	Receive Fifo Overflow count (IEEE_R_MACERR)
0xFC0D_42DC 0xFC0D_82DC	Flow Control Pause frames received (IEEE_R_FDXFC)
0xFC0D_42E0 0xFC0D_82E0	Octet count for Frames Rcvd w/o Error (IEEE_R_OCTETS_OK) <b>Note:</b> Counts total octets (includes header and FCS fields)

### 31.3.1 Interrupt Event Register (ENETn\_EIR)

When an event occurs that sets a bit in ENETn\_EIR, an interrupt occurs if the corresponding bit in the interrupt mask register (ENETn\_EIMR) is also set. Writing a 1 to an ENETn\_EIR bit clears it; writing 0 has no effect. This register is cleared upon hardware reset.

Address: 0xFC0D_4004 (ENET0_EIR) 0xFC0D_8004 (ENET1_EIR)																Access: User read/write			
31	30	29	28		27	26	25	24		23	22	21	20		19	18	17	16	
R	0	BABR	BABT	GRA	TXF	TXB	RXF	RXB	MII	EB ERR	LC	RL	UN	PLR	WAK EUP	TS AVAIL			
W	w1c	w1c	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	0	0	
R	15	14	13	12	11	10	9	8		7	6	5	4		3	2	1	0	
W	TS TIMER	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

**Figure 31-2. Interrupt Event Register (ENETn\_EIR)**

**Table 31-6. ENET $n$ \_EIR Field Descriptions**

<b>Field</b>	<b>Description</b>
31	Reserved, must be cleared.
30 BABR	Babbling receive error. Indicates a frame was received with length in excess of ENET $n$ _RCR[MAX_FL] bytes.
29 BABT	Babbling transmit error. Indicates the transmitted frame length exceeds ENET $n$ _RCR[MAX_FL] bytes. Usually this condition is caused by a frame that is too long is placed into the transmit data buffer(s). Truncation does not occur.
28 GRA	Graceful stop complete. This interrupt is asserted after the transmitter is put into a pause state after completion of the frame currently being transmitted. See <a href="#">Section 31.4.9.4.1, “Graceful Transmit Stop (GTS)”</a> for conditions that lead to graceful stop. <b>Note:</b> The GRA interrupt is asserted only when the TX transitions into the stopped state. If this bit is cleared (by writing 1) and the TX is still stopped, the bit is not set again.
27 TXF	Transmit frame interrupt. Indicates a frame has been transmitted and the last corresponding buffer descriptor has been updated.
26 TXB	Transmit buffer interrupt. Indicates a transmit buffer descriptor has been updated.
25 RXF	Receive frame interrupt. Indicates a frame has been received and the last corresponding buffer descriptor has been updated.
24 RXB	Receive buffer interrupt. Indicates a receive buffer descriptor not the last in the frame has been updated.
23 MII	MII interrupt. Indicates the MII has completed the data transfer requested.
22 EBERR	Ethernet bus error. Indicates a system bus error occurred when a DMA transaction is underway. When this bit is set, ENET $n$ _ECR[ETHER_EN] is cleared, halting frame processing by the MAC. When this occurs, software must ensure proper actions (possibly resetting the system) to resume normal operation.
21 LC	Late collision. Indicates a collision occurred beyond the collision window (slot time) in half-duplex mode. The frame truncates with a bad CRC and the remainder of the frame is discarded.
20 RL	Collision retry limit. Indicates a collision occurred on each of 16 successive attempts to transmit the frame. The frame is discarded without being transmitted and transmission of the next frame commences. This error can only occur in half duplex mode.
19 UN	Transmit FIFO underrun. Indicates the transmit FIFO became empty before the complete frame was transmitted. A bad CRC is appended to the frame fragment and the remainder of the frame is discarded.
18 PLR	Payload receive error. This bit indicates a frame was received with a payload length error.
17 WAKEUP	Node wake-up request indication. Read only status bit to indicate that a magic packet has been detected. Acts only if ENET $n$ _ECR[MAGICEN] is set.
16 TS_AVAIL	Transmit timestamp available. Indicates that the timestamp of the last transmitted timing frame is available in the ENET $n$ _ATSTMP register.
15 TS_TIMER	The adjustable timer reached the period event. See <a href="#">Section 31.4.10.1, “Adjustable Timer Module”</a> .
14–0	Reserved, must be cleared.

### 31.3.2 Interrupt Mask Register (ENETn\_EIMR)

The ENET $n$ \_EIMR register controls which interrupt events are allowed to generate actual interrupts. A hardware reset clears this register. If the corresponding bits in the ENET $n$ \_EIR and ENET $n$ \_EIMR registers are set, an interrupt is generated. The interrupt signal remains asserted until a 1 is written to the ENET $n$ \_EIR bit (write 1 to clear) or a 0 is written to the ENET $n$ \_EIMR bit.

Address: 0xFC0D\_4008 (ENET0\_EIMR)  
0xFC0D\_8008 (ENET1\_EIMR)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	BABR	BABT	GRA	TXF	TXB	RXF	RXB	MII	EB ERR	LC	RL	UN	PLR	WAK EUP	TS AVAIL
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	TS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-3. Interrupt Mask Register (ENET $n$ \_EIMR)

Table 31-7. ENET $n$ \_EIMR Field Descriptions

Field	Description
31	Reserved, must be cleared.
30–15 See <a href="#">Figure 31-3</a> and <a href="#">Table 31-6</a>	Interrupt mask. Each bit corresponds to an interrupt source defined by the ENET $n$ _EIR register. The corresponding ENET $n$ _EIMR bit determines whether an interrupt condition can generate an interrupt. At every processor clock, the ENET $n$ _EIR samples the signal generated by the interrupting source. The corresponding ENET $n$ _EIR bit reflects the state of the interrupt signal even if the corresponding ENET $n$ _EIMR bit is cleared. 0 The corresponding interrupt source is masked. 1 The corresponding interrupt source is not masked.
14–0	Reserved, must be cleared.

### 31.3.3 Receive Descriptor Active Registers (ENET $n$ \_RDAR)

ENET $n$ \_RDAR is a command register, written by the user, indicating the receive descriptor ring is updated (the driver produced empty receive buffers with the empty bit set).

When the register is written, the RDAR bit is set. This is independent of the data actually written by the user. When set, the MAC polls the receive descriptor ring and processes receive frames (provided ENET $n$ \_ECR[ETHER\_EN] is also set). After the MAC polls a receive descriptor whose empty bit is not set, MAC clears RDAR and ceases receive descriptor ring polling until the user sets the bit again, signifying that additional descriptors are placed into the receive descriptor ring.

The ENET $n$ \_RDAR registers are cleared at reset and when ENET $n$ \_ECR[ETHER\_EN] transitions from set to cleared or when ENET $n$ \_ECR[RESET] is set.

Address: 0xFC0D_4010 (ENET0_RDAR) 0xFC0D_8010 (ENET1_RDAR)																Access: User read/write											
R 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 RDAR																W 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0											
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 31-4. Receive Descriptor Active Register (ENETn\_RDAR)

Table 31-8. ENETn\_RDAR Field Descriptions

Field	Description
31–25	Reserved, must be cleared.
24 RDAR	Set to 1 when this register is written, regardless of the value written. Cleared by the MAC device when no additional empty descriptors remain in the receive ring. Also cleared when ENETn_ECR[ETHER_EN] transitions from set to cleared or when ENETn_ECR[RESET] is set.
23–0	Reserved, must be cleared.

### 31.3.4 Transmit Descriptor Active Registers (ENETn\_TDAR)

The ENETn\_TDAR are command registers which the user writes to indicate the transmit descriptor ring is updated (transmit buffers have been produced by the driver with the ready bit set in the buffer descriptor).

When the register is written, the TDAR bit is set. This value is independent of the data actually written by the user. When set, the MAC polls the transmit descriptor ring and processes transmit frames (provided ENETn\_ECR[ETHER\_EN] is also set). After the MAC polls a transmit descriptor that contains a ready bit that is not set, MAC clears TDAR and ceases transmit descriptor ring polling until the user sets the bit again, signifying additional descriptors are placed into the transmit descriptor ring.

The ENETn\_TDAR registers are cleared at reset, when ENETn\_ECR[ETHER\_EN] transitions from set to cleared, or when ENETn\_ECR[RESET] is set.

Address: 0xFC0D_4014 (ENET0_TDAR) 0xFC0D_8014 (ENET1_TDAR)																Access: User read/write											
R 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 TDAR																W 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0											
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 31-5. Transmit Descriptor Active Register (ENETn\_TDAR)

**Table 31-9. ENET $n$ \_TDAR Field Descriptions**

Field	Description
31–25	Reserved, must be cleared.
24 TDAR	Set to 1 when this register is written, regardless of the value written. Cleared by the MAC device when no additional ready descriptors remain in the transmit ring. Also cleared when ENET $n$ _ECR[ETHER_EN] transitions from set to cleared or when ENET $n$ _ECR[RESET] is set.
23–0	Reserved, must be cleared.

### 31.3.5 Ethernet Control Register (ENET $n$ \_ECR)

ENET $n$ \_ECR is a read/write user register, though hardware may alter fields in this register as well. It controls many of the high level features of the Ethernet MAC, including legacy FEC support through the 1588EN bit.

Address: 0xFC0D\_4024 (ENET0\_ECR)  
0xFC0D\_8024 (ENET1\_ECR) Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	1	1	1	1	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	0	0	0	0	0	0	0	STOP EN	DBG EN	0	1588 EN	SLEEP	MAGIC EN	ETHER EN	RESET
W													0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 31-6. Ethernet Control Register (ENET $n$ \_ECR)****Table 31-10. ENET $n$ \_ECR Field Descriptions**

Field	Description
31–28	Reserved, must be set.
27–8	Reserved, must be cleared.
7 STOPEN	Controls device behavior in doze mode. In doze mode, if this bit is set then all the clocks of the ENET assembly are disabled (except the RMII/MII clock). Doze mode is like a conditional stop mode entry the ENET assembly depending on ECR[STOPEN]. <b>Note:</b> If module clocks are gated in this mode, the module can still wake the system after receiving a magic packet in stop mode. MAGICEN must be set prior to entering sleep/stop mode.
6 DBGEN	Enables the MAC to enter hardware freeze mode when the device enters debug mode. See <a href="#">Section 31.4.9.3, "Hardware Freeze"</a> . 0 MAC continues operation in debug mode 1 MAC enters hardware freeze mode when the processor is in debug mode
5	Reserved, must be cleared.
4 1588EN	IEEE1588 enable. Enables enhanced functionality of the MAC. 0 Legacy FEC buffer descriptors and functions enabled 1 Enhanced frame time-stamping functions enabled

**Table 31-10. ENET<sub>n</sub>\_ECR Field Descriptions (continued)**

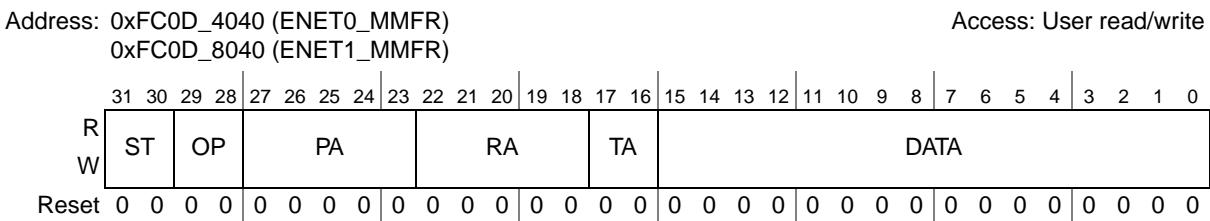
Field	Description
3 SLEEP	Sleep mode enable. 0 Normal operating mode 1 Sleep mode <b>Note:</b> MAGICEN must be set to allow the MAC to interrupt the processor from sleep mode.
2 MAGICEN	Magic packet detection enable. 0 Magic detection logic disabled 1 The MAC core detects magic packets and asserts ENET <sub>n</sub> _EIR[WAKEUP] when a frame is detected <b>Note:</b> MAGICEN is relevant only if the SLEEP bit is set. If MAGICEN is set, changing the SLEEP bit enables/disables sleep mode and magic packet detection.
1 ETHEREN	0 Reception immediately stops and transmission stops after a bad CRC is appended to any currently transmitted frame. The buffer descriptors for an aborted transmit frame are not updated. The DMA, buffer descriptor, and FIFO control logic are reset, including the buffer descriptor and FIFO pointers (see <a href="#">Section 31.4.9.2, “Soft Reset”</a> ). 1 MAC is enabled, and reception and transmission are possible.  Hardware clears this bit under the following conditions: <ul style="list-style-type: none"><li>• RESET is set by software</li><li>• An error condition causes the EBERR bit to set</li></ul>
0 RESET	When the bit is set, it has no other effect on the MAC except that it clears the ETHER_EN bit.

### 31.3.6 MII Management Frame Register (ENET<sub>n</sub>\_MMFR)

Performing a write to ENET<sub>n</sub>\_MMFR triggers a management frame transaction to the PHY device unless ENET<sub>n</sub>\_MSCR is programmed to zero.

If, while writing to ENET<sub>n</sub>\_MMFR, ENET<sub>n</sub>\_MSCR is changed from zero to non-zero, an MII frame is generated with the data previously written to the ENET<sub>n</sub>\_MMFR. This allows ENET<sub>n</sub>\_MMFR and ENET<sub>n</sub>\_MSCR to be programmed in either order if ENET<sub>n</sub>\_MSCR is currently zero.

If the ENET<sub>n</sub>\_MMFR register is written while frame generation is in progress, the frame contents are altered. Software must use the ENET<sub>n</sub>\_EIR[MII] interrupt indication to avoid writing to the ENET<sub>n</sub>\_MMFR register while frame generation is in progress.

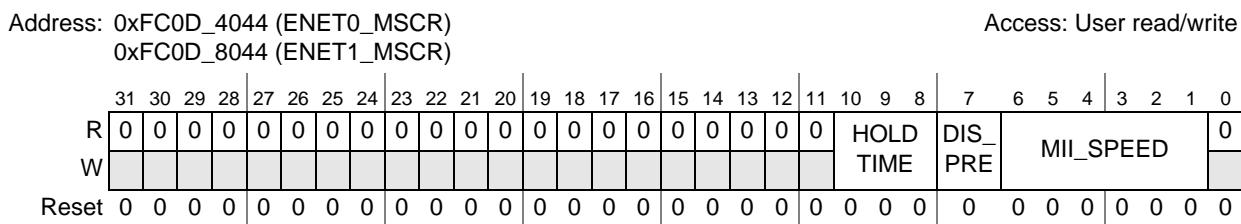
**Figure 31-7. MII Management Frame Register (ENET<sub>n</sub>\_MMFR)**

**Table 31-11. ENET<sub>n</sub>\_MMFR Field Descriptions**

Field	Description
31–30 ST	Start of frame delimiter. These bits must be programmed to 01 for a valid MII management frame.
29–28 OP	Operation code. 00 Write frame operation, but not MII compliant 01 Write frame operation for a valid MII management frame 10 Read frame operation for a valid MII management frame 11 Read frame operation, but not MII compliant
27–23 PA	PHY address. Specifies one of up to 32 attached PHY devices.
22–18 RA	Register address. Specifies one of up to 32 registers within the specified PHY device.
17–16 TA	Turn around. This field must be programmed to 10 to generate a valid MII management frame.
15–0 DATA	Management frame data. This is the field for data to be written to or read from the PHY register.

### 31.3.7 MII Speed Control Register (ENET<sub>n</sub>\_MSCR)

The ENET<sub>n</sub>\_MSCR provides control of the MII clock (MDC pin) frequency and allows a preamble drop on the MII management frame.

**Figure 31-8. MII Speed Control Register (ENET<sub>n</sub>\_MSCR)****Table 31-12. ENET<sub>n</sub>\_MSCR Field Descriptions**

Field	Description
31–11	Reserved, must be cleared.
10–8 HOLDTIME	IEEE802.3 clause 22 defines a minimum of 10 ns for the holdtime on the MDIO output. Depending on the host bus frequency the setting may need to be increased. 000 1 internal bus cycle 001 2 internal bus cycles 010 3 internal bus cycles ... 111 8 internal bus cycles
7 DIS_PRE	Disable preamble. 0 Preamble enabled 1 Preamble (32 ones) is not prepended to the MII management frame. The MII standard allows the preamble to be dropped if the attached PHY devices do not require it.

**Table 31-12. ENETn\_MSCR Field Descriptions (continued)**

Field	Description
6-1 MII_SPEED	MII speed. Controls the frequency of the MII management interface clock (MDC) relative to the internal bus clock. A value of 0 in this field turns off MDC and leaves it in low voltage state. Any non-zero value results in the MDC frequency of $1/((\text{MII\_SPEED} + 1) \times 2)$ of the internal bus frequency.
0	Reserved, must be cleared.

The MII\_SPEED field must be programmed with a value to provide an MDC frequency of less than or equal to 2.5 MHz to be compliant with the IEEE 802.3 MII specification. The MII\_SPEED must be set to a non-zero value to source a read or write management frame. After the management frame is complete, the ENETn\_MSCR register may optionally be cleared to turn off MDC. The MDC signal generated has a 50% duty cycle except when MII\_SPEED changes during operation (change takes effect following a rising or falling edge of MDC).

If the internal bus clock is 25 MHz, programming this register to 0x0000\_0004 results in an MDC as stated the equation below.

$$25 \text{ MHz} \times \frac{1}{(4 + 1) \times 2} = 2.5 \text{ MHz} \quad \text{Eqn. 31-1}$$

Table 31-13 shows the optimum values for MII\_SPEED as a function of internal bus clock frequency.

**Table 31-13. Programming Examples for ENETn\_MSCR**

Internal MAC clock frequency	ENETn_MSCR [MII_SPEED]	MDC frequency
25 MHz	0x4	2.50 MHz
33 MHz	0x6	2.36 MHz
40 MHz	0x7	2.50 MHz
50 MHz	0x9	2.50 MHz
66 MHz	0xD	2.36 MHz

### 31.3.8 MIB Control Register (ENETn\_MIBC)

ENETn\_MIBC is a read/write register controlling and observing the state of the MIB block. Access this register to disable the MIB block operation or clear the MIB counters.

The MIB\_DIS bit resets to 1. See Table 31-5 for the locations of the MIB counters.

Address: 0xFC0D_4064 (ENET0_MIBC) 0xFC0D_8064 (ENET1_MIBC)																Access: User read/write								
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8
W	MIB_DIS	MIB_IDLE	MIB_CLEAR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset	1	1	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. MIB Control Register (ENETn\_MIBC)**

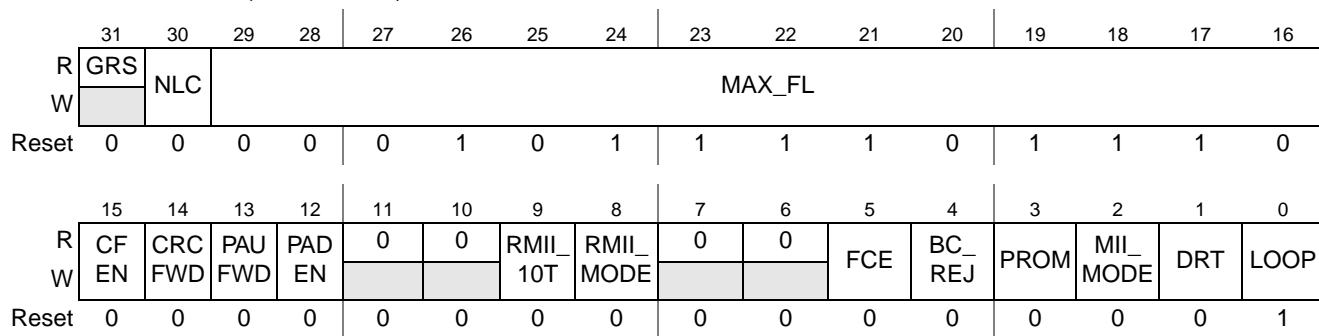
**Table 31-14. ENETn\_MIBC Field Descriptions**

Field	Description
31 MIB_DIS	A read/write control bit. If set, the MIB logic halts and not update any MIB counters.
30 MIB_IDLE	A read-only status bit. If set the MIB block is not currently updating any MIB counters.
29 MIB_CLEAR	A read/write control bit. If set all statistics counters are reset to 0.
28–0	Reserved, must be cleared.

### 31.3.9 Receive Control Register (ENETn\_RCR)

Address: 0xFC0D\_4084 (ENET0\_RCR)  
0xFC0D\_8084 (ENET1\_RCR)

Access: User read/write

**Figure 31-10. Receive Control Register (ENETn\_RCR)****Table 31-15. ENETn\_RCR Field Descriptions**

Field	Description
31 GRS	Graceful receive stopped. Read-only status indicating that the MAC receive datapath is stopped (see <a href="#">Section 31.4.9.4.2, “Graceful Receive Stop (GRS)”</a> ).
30 NLC	Payload length check disable. 0 The payload length check is disabled 1 The core checks the frame's payload length with the frame length/type field
29–16 MAX_FL	Maximum frame length. Resets to decimal 1518. Length is measured starting at DA and includes the CRC at the end of the frame. Transmit frames longer than MAX_FL causes the BABT interrupt to occur. Receive frames longer than MAX_FL causes the BABR interrupt to occur and sets the LG bit in the end of frame receive buffer descriptor. The recommended default value to be programmed is 1518 or 1522 if VLAN tags are supported.
15 CFEN	MAC control frame enable. 0 MAC control frames with any opcode other than 0x0001 are accepted and forwarded to the client interface 1 MAC control frames with any opcode other than 0x0001 (pause frame) are silently discarded
14 CRCFWD	Terminate/forward received CRC. 0 The CRC field of received frames is transmitted to the user application 1 The CRC field is stripped from the frame <b>Note:</b> If padding function is enabled (PADEN = 1), CRCFWD is ignored and the CRC field is checked and always terminated and removed.

**Table 31-15. ENET<sub>n</sub>\_RCR Field Descriptions (continued)**

Field	Description
13 PAUFWD	Terminate/forward pause frames. 0 Pause frames are terminated and discarded in the MAC 1 Pause frames are forwarded to the user application
12 PADEN	Enable frame padding remove on receive. 1 Padding is removed from received frames 0 No padding is removed on receive by the MAC
11–10	Reserved, must be cleared.
9 RMII_10T	RMII 10-Base T. Enables 10-Mbps mode of the RMII. 0 100 Mbps operation. The 50 MHz RMII reference clock (RMII_REF_CLK) is sent to the RMII, while a divided-by-2 version (25 MHz) is sent to the MAC. 1 10 Mbps operation. The 50 MHz RMII reference clock (RMII_REF_CLK) is divided by 10 (5 MHz) and used in the RMII, while a divided-by-20 version (2.5 MHz) is sent to the MAC.
8 RMII_MODE	RMII mode enable. 0 MAC configured for MII mode. 1 MAC configured for RMII operation
7–6	Reserved, must be cleared.
5 FCE	Flow control enable. If set, the receiver detects PAUSE frames. Upon PAUSE frame detection, the transmitter stops transmitting data frames for a given duration.
4 BC_REJ	Broadcast frame reject. If set, frames with DA (destination address) equal to FFFF_FFFF_FFFF are rejected unless the PROM bit is set. If BC_REJ and PROM are set, frames with broadcast DA are accepted and the M (MISS) is set in the receive buffer descriptor.
3 PROM	Promiscuous mode. All frames are accepted regardless of address matching.
2 MII_MODE	Media independent interface mode. This bit must always be set. 0 Reserved 1 MII or RMII mode, as indicated by the RMII_MODE bit
1 DRT	Disable receive on transmit. 0 Receive path operates independently of transmit (use for full duplex or to monitor transmit activity in half duplex mode). 1 Disable reception of frames while transmitting (normally used for half duplex mode).
0 LOOP	Internal loopback. 0 Loopback disabled 1 Transmitted frames are looped back internal to the device and transmit MII output signals are not asserted. The internal bus clock substitutes for the TXCLK. DRT must be cleared.

### 31.3.10 Transmit Control Register (ENET<sub>n</sub>\_TCR)

ENET<sub>n</sub>\_TCR is read/write and configures the transmit block. This register is cleared at system reset. FDEN can only be modified when ENET<sub>n</sub>\_ECR[ETHER\_EN] is cleared.

Address: 0xFC0D\_40C4 (ENET0\_TCR)  
0xFC0D\_80C4 (ENET1\_TCR)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	CRC FWD	ADD INS	ADDSEL			RFC_PAUSE	TFC_PAUSE	FDEN	0	GTS
W															0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 31-11. Transmit Control Register (ENETn\_TCR)

Table 31-16. ENETn\_TCR Field Descriptions

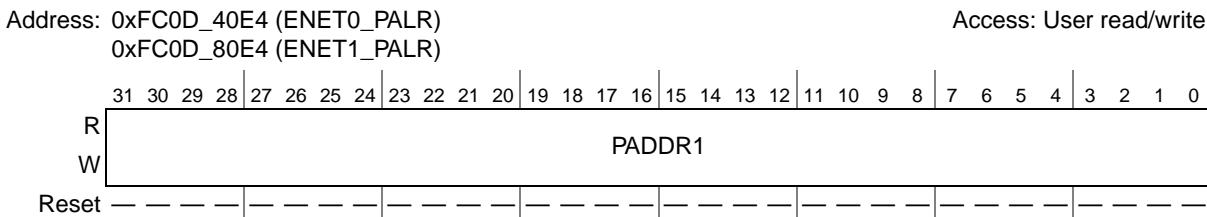
Field	Description
31–10	Reserved, must be cleared.
9 CRCFWD	Forward frame from application with CRC. 0 TxBD[TC] controls if the frame has a CRC from the application 1 The transmitter does not append any CRC to transmitted frames as it is expecting a frame with CRC from the application
8 ADDINS	Set MAC address on transmit. 0 The source MAC address is not modified by the MAC 1 The MAC overwrites the source MAC address with the programmed MAC address according to ADDSEL
7–5 ADDSEL	Source MAC address select on transmit. If ADDINS is set, indicates the MAC address that overwrites the source MAC address: 000 Node MAC address programmed on ENETn_PADDR1/2 registers 100 Supplemental MAC address 0 (ENETn_SMACx0) 101 Supplemental MAC address 1 (ENETn_SMACx1) 110 Supplemental MAC address 2 (ENETn_SMACx2) Else Supplemental MAC address 3 (ENETn_SMACx3)
4 RFC_PAUSE	Receive frame control pause. This read-only status bit is set when a full duplex flow control pause frame is received and the transmitter pauses for the duration defined in this pause frame. This bit automatically clears when the pause duration is complete.
3 TFC_PAUSE	Transmit frame control pause. 0 No PAUSE frame transmitted. 1 The MAC stops transmission of data frames after the current transmission is complete. At this time, ENETn_EIR[GRA] is set. With transmission of data frames stopped, the MAC transmits a MAC control PAUSE frame. Next, the MAC clears TFC_PAUSE and resumes transmitting data frames. If the transmitter pauses due to user assertion of GTS or reception of a PAUSE frame, the MAC may continue transmitting a MAC control PAUSE frame.
2 FDEN	Full duplex enable. If set, frames transmit independent of carrier sense and collision inputs. This bit should only be modified when ENETn_ECR[ETHER_EN] is cleared.

**Table 31-16. ENETn\_TCR Field Descriptions (continued)**

Field	Description
1	Reserved, must be cleared.
0 GTS	Graceful transmit stop. When this bit is set, MAC stops transmission after any frame currently transmitted is complete and ENETn_EIR[GRA] is set. If frame transmission is not currently underway, the GRA interrupt is asserted immediately. After transmission finishes, clear GTS to restart. The next frame in the transmit FIFO is then transmitted. If an early collision occurs during transmission when GTS is set, transmission stops after the collision. The frame is transmitted again after GTS is cleared. There may be old frames in the transmit FIFO that transmit when GTS is reasserted. To avoid this, clear ENETn_ECR[ETHER_EN] following the GRA interrupt.

### 31.3.11 Physical Address Lower Register (ENETn\_PALR)

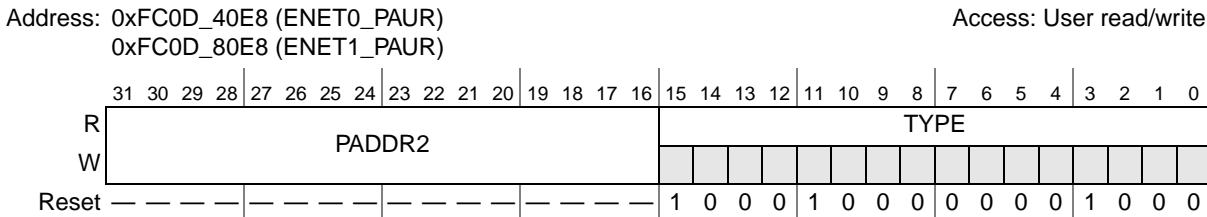
ENETn\_PALR contains the lower 32 bits (bytes 0,1,2,3) of the 48-bit address used in the address recognition process to compare with the DA (destination address) field of receive frames with an individual DA. In addition, this register is used in bytes 0 through 3 of the six-byte source address field when transmitting PAUSE frames. This register is not reset and you must initialize it.

**Figure 31-12. Physical Address Low Register (ENETn\_PALR)****Table 31-17. ENETn\_PALR Field Descriptions**

Field	Description
31-0 PADDR1	Bytes 0 (bits 31:24), 1 (bits 23:16), 2 (bits 15:8), and 3 (bits 7:0) of the 6-byte individual address are used for exact match and the source address field in PAUSE frames.

### 31.3.12 Physical Address Upper Register (ENETn\_PAUR)

ENETn\_PAUR contains the upper 16 bits (bytes 4 and 5) of the 48-bit address used in the address recognition process to compare with the DA (destination address) field of receive frames with an individual DA. In addition, this register is used in bytes 4 and 5 of the six-byte source address field when transmitting PAUSE frames. Bits 15:0 of ENETn\_PAUR contain a constant type field (0x8808) for transmission of PAUSE frames. The upper 16 bits of this register are not reset and you must initialize it.

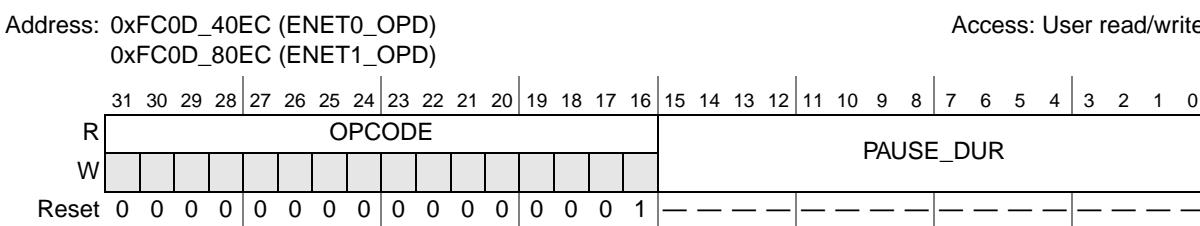
**Figure 31-13. Physical Address Upper Register (ENETn\_PAUR)**

**Table 31-18. ENETn\_PAUR Field Descriptions**

Field	Description
31–16 PADDR2	Bytes 4 (bits 31:24) and 5 (bits 23:16) of the 6-byte individual address used for exact match, and the source address field in PAUSE frames.
15–0 TYPE	Type field in PAUSE frames. These 16 read-only bits are a constant value of 0x8808.

### 31.3.13 Opcode/Pause Duration Register (ENETn\_OPD)

The ENET $n$ \_OPD is read/write accessible. This register contains the 16-bit opcode and 16-bit pause duration fields used in transmission of a PAUSE frame. The opcode field is a constant value, 0x0001. When another node detects a PAUSE frame, that node pauses transmission for the duration specified in the pause duration field. The lower 16 bits of this register are not reset and you must initialize them.



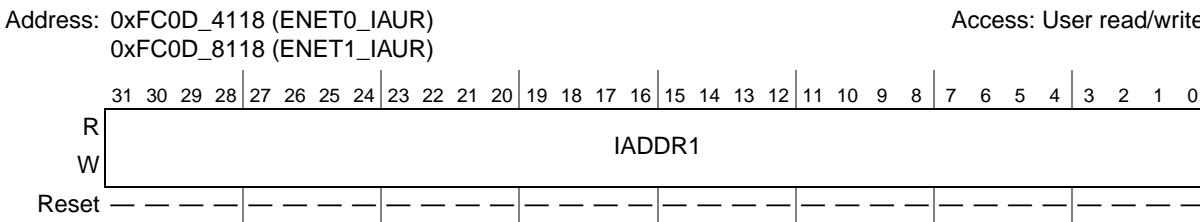
**Figure 31-14. Opcode/Pause Duration Register (ENETn\_OPD)**

**Table 31-19. ENETn\_OPD Field Descriptions**

Field	Description
31–16 OPCODE	Opcode field used in PAUSE frames. These read-only bits are a constant, 0x0001.
15–0 PAUSE_DUR	Pause duration field used in PAUSE frames.

### 31.3.14 Descriptor Individual Upper Address Register (ENETn\_IAUR)

ENETn\_IAUR contains the upper 32 bits of the 64-bit individual address hash table. The address recognition process uses this table to check for a possible match with the destination address (DA) field of receive frames with an individual DA. This register is not reset and you must initialize it.



**Figure 31-15. Descriptor Individual Upper Address Register (ENETn\_IAUR)**

**Table 31-20. ENETn\_IAUR Field Descriptions**

Field	Description
31–0 IADDR1	The upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a unicast address. Bit 31 of IADDR1 contains hash index bit 63. Bit 0 of IADDR1 contains hash index bit 32.

### 31.3.15 Descriptor Individual Lower Address Register (ENETn\_IALR)

ENETn\_IALR contains the lower 32 bits of the 64-bit individual address hash table. The address recognition process uses this table to check for a possible match with the DA field of receive frames with an individual DA. This register is not reset and you must initialize it.

**Figure 31-16. Descriptor Individual Lower Address Register (ENETn IALR)**

**Table 31-21. ENETn IALR Field Descriptions**

Field	Description
31–0 IADDR2	The lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a unicast address. Bit 31 of IADDR2 contains hash index bit 31. Bit 0 of IADDR2 contains hash index bit 0.

### 31.3.16 Descriptor Group Upper Address Register (ENETn\_GAUR)

**ENETn\_GAUR** contains the upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. You must initialize this register.

**Figure 31-17. Descriptor Group Upper Address Register (ENETn\_GAUR)**

**Table 31-22. ENETn\_GAUR Field Descriptions**

Field	Description
31–0 GADDR1	Contains the upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. Bit 31 of GADDR1 contains hash index bit 63. Bit 0 of GADDR1 contains hash index bit 32.

### 31.3.17 Descriptor Group Lower Address Register (ENETn\_GALR)

ENETn\_GALR contains the lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. You must initialize this register.

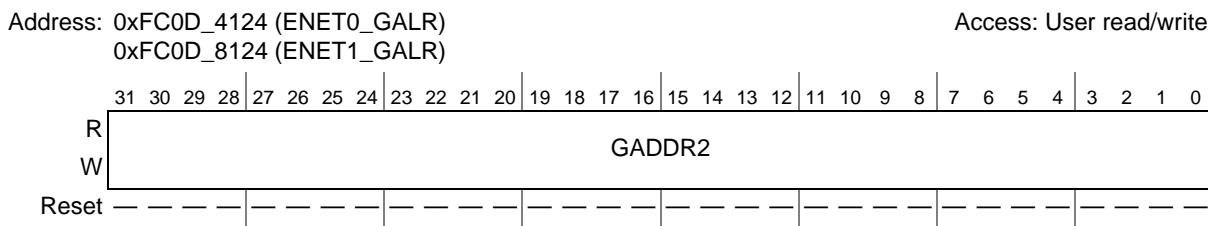


Figure 31-18. Descriptor Group Lower Address Register (ENETn\_GALR)

Table 31-23. ENETn\_GALR Field Descriptions

Field	Description
31–0 GADDR2	Contains the lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. Bit 31 of GADDR2 contains hash index bit 31. Bit 0 of GADDR2 contains hash index bit 0.

### 31.3.18 Transmit FIFO Watermark Register (ENETn\_TFWR)

If ENETn\_TFR[STRFWD] is cleared, ENETn\_TFWR[TFWR] controls the amount of data required in the transmit FIFO before transmission of a frame can begin. This allows you to minimize transmit latency (TFWR = 00 or 01) or allow for larger bus access latency (TFWR = 11) due to contention for the system bus. Setting the watermark to a high value minimizes the risk of transmit FIFO underrun due to contention for the system bus. The byte counts associated with the TFWR field may need to be modified to match a given system requirement (worst case bus access latency by the transmit data DMA channel).

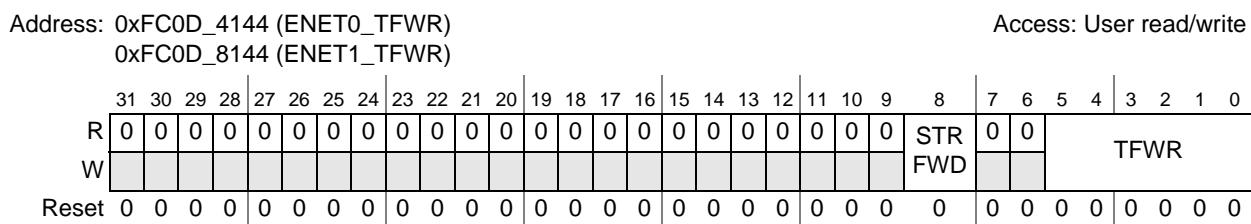


Figure 31-19. Transmit FIFO Watermark Register (ENETn\_TFWR)

Table 31-24. ENETn\_TFWR Field Descriptions

Field	Description
31–9	Reserved, must be cleared.
8 STRFWD	Store and forward enable. 0 Disabled, the transmission start threshold is programmed in TFWR 1 Enabled

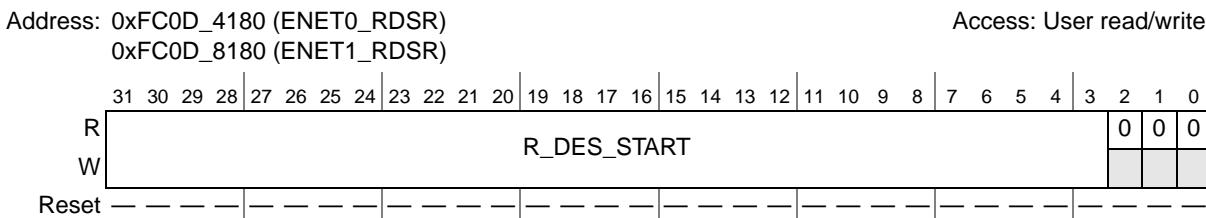
**Table 31-24. ENETn\_TFWR Field Descriptions (continued)**

Field	Description
7–6	Reserved, must be cleared.
5–0 TFWR	If STRFWD is cleared, indicates number of bytes written to transmit FIFO before transmission of a frame begins: 0x00 64 bytes written 0x01 64 bytes written 0x02 128 bytes written 0x03 192 bytes written ... 0x3F 4032 bytes written <b>Note:</b> If a frame with less than the threshold is written it is still sent, independently of this threshold setting. The threshold is only relevant if the frame is larger than the threshold given.

### 31.3.19 Receive Descriptor Ring Start Register (ENETn\_RDSR)

ENETn\_RDSR points to the start of the circular receive buffer descriptor queue in external memory. This pointer must be 64-bit aligned (bits 2–0 must be zero); however, it is recommended to be 128-bit aligned (evenly divisible by 16).

This register is not reset and must be initialized prior to operation.

**Table 31-25. Ethernet Receive Descriptor Ring Start Register (ENETn\_RDSR)****Table 31-26. ENETn\_RDSR Field Descriptions**

Field	Description
31–3 R_des_start	Pointer to start of receive buffer descriptor queue.
2–0	Reserved, must be cleared.

### 31.3.20 Transmit Buffer Descriptor Ring Start Register (ENETn\_TDSR)

ENETn\_TDSR provides a pointer to the start of the circular transmit buffer descriptor queue in external memory. This pointer must be 64-bit aligned (bits 2–0 must be zero); however, it is recommended to be 128-bit aligned (evenly divisible by 16).

This register is undefined at reset and must be initialized prior to operation.

Address: 0xFC0D_4184 (ENET0_TDSR) 0xFC0D_8184 (ENET1_TDSR)	Access: User read/write
<p>Detailed description: This is a bit map of the ENETn_TDSR register. It shows the bit range for the X_DES_START field (bits 31-20), followed by bits 19 through 0. Bit 19 is labeled 'R' (Read) and bit 18 is labeled 'W' (Write). Bit 17 is labeled 'Reset'. The bit map is divided into four sections by vertical lines: 31-29, 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. The 'R' and 'W' bits are located between the 28 and 27-26 sections.</p>	<p>Detailed description: This is a bit map of the ENETn_TDSR register. It shows the bit range for the X_DES_START field (bits 31-20), followed by bits 19 through 0. Bit 19 is labeled 'R' (Read) and bit 18 is labeled 'W' (Write). Bit 17 is labeled 'Reset'. The bit map is divided into four sections by vertical lines: 31-29, 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. The 'R' and 'W' bits are located between the 28 and 27-26 sections.</p>

**Table 31-27. Transmit Buffer Descriptor Ring Start Register (ENETn\_TDSR)****Table 31-28. ENETn\_TDSR Field Descriptions**

Field	Description
31–3 X_DES_ START	Pointer to start of transmit buffer descriptor queue.
2–0	Reserved, must be cleared.

### 31.3.21 Maximum Receive Buffer Size Register (ENETn\_MRBR)

The ENETn\_MRBR is a user-programmable register that dictates the maximum size of all receive buffers. This value should take into consideration that the receive CRC is always written into the last receive buffer. To allow one maximum size frame per buffer, ENETn\_MRBR must be set to ENETn\_RCR[MAX\_FL] or larger. To properly align the buffer, ENETn\_MRBR must be evenly divisible by 16. To ensure this, bits 3–0 are forced low.

To minimize bus utilization (descriptor fetches), set ENETn\_MRBR greater than or equal to 256 bytes.

The ENETn\_MRBR register is undefined at reset and must be initialized by the user.

Address: 0xFC0D_4188 (ENET0_MRBR) 0xFC0D_8188 (ENET1_MRBR)	Access: User read/write
<p>Detailed description: This is a bit map of the ENETn_MRBR register. It shows the bit range for the R_BUF_SIZE field (bits 31-14), followed by bits 13 through 0. Bit 13 is labeled 'R' (Read) and bit 12 is labeled 'W' (Write). Bit 11 is labeled 'Reset'. The bit map is divided into four sections by vertical lines: 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. The 'R' and 'W' bits are located between the 31-30 and 29-28 sections.</p>	<p>Detailed description: This is a bit map of the ENETn_MRBR register. It shows the bit range for the R_BUF_SIZE field (bits 31-14), followed by bits 13 through 0. Bit 13 is labeled 'R' (Read) and bit 12 is labeled 'W' (Write). Bit 11 is labeled 'Reset'. The bit map is divided into four sections by vertical lines: 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. The 'R' and 'W' bits are located between the 31-30 and 29-28 sections.</p>

**Table 31-29. Maximum Receive Buffer Size Register (ENETn\_MRBR)****Table 31-30. ENETn\_MRBR Field Descriptions**

Field	Description
31–14	Reserved, must be cleared.
13–4 R_BUF_ SIZE	Receive buffer size in bytes.
3–0	Reserved, must be cleared.

### 31.3.22 Receive FIFO Section Full Threshold (ENET $n$ \_RSFL)

Address: 0xFC0D\_4190 (ENET0\_RSFL)  
0xFC0D\_8190 (ENET1\_RSFL)

Access: User 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	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																																
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	

Table 31-31. Receive FIFO Section Full Threshold (ENET $n$ \_RSFL)Table 31-32. ENET $n$ \_RSFL Field Descriptions

Field	Description
31–8	Reserved, must be cleared.
7–0 RX_SECTION_FULL	Value, in 64-bit words, of the receive FIFO section full threshold. Clear this field to enable store and forward on the RX FIFO. When programming a value greater than 0 (cut-through operation), it must be greater than RX_ALMOST_EMPTY. See <a href="#">Table 31-88</a> for more details on the receive FIFO thresholds.

### 31.3.23 Receive FIFO Section Empty Threshold (ENET $n$ \_RSEM)

Address: 0xFC0D\_4194 (ENET0\_RSEM)  
0xFC0D\_8194 (ENET1\_RSEM)

Access: User 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	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																																
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		

Table 31-33. Receive FIFO Section Empty Threshold (ENET $n$ \_RSEM)Table 31-34. ENET $n$ \_RSEM Field Descriptions

Field	Description
31–8	Reserved, must be cleared.
7–0 RX_SECTION_EMPTY	Value, in 64-bit words, of the receive FIFO section empty threshold. See <a href="#">Table 31-88</a> for more details on the receive FIFO thresholds.

### 31.3.24 Receive FIFO Almost Empty Threshold (ENET $n$ \_RAEM)

Address: 0xFC0D\_4198 (ENET0\_RAEM)  
0xFC0D\_8198 (ENET1\_RAEM)

Access: User 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	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																																
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		

Table 31-35. Receive FIFO Almost Empty Threshold (ENET $n$ \_RAEM)

**Table 31-36. ENETn\_RAEM Field Descriptions**

Field	Description
31–8	Reserved, must be cleared.
7–0 RX_ALMOST_EMPTY	Value, in 64-bit words, of the receive FIFO almost empty threshold. See <a href="#">Table 31-88</a> for more details on the receive FIFO thresholds.

### 31.3.25 Receive FIFO Almost Full Threshold (ENETn\_RAFL)

Address: 0xFC0D\_419C (ENET0\_RAFL)  
0xFC0D\_819C (ENET1\_RAFL) Access: User read/write

**Table 31-37. Receive FIFO Almost Full Threshold (ENETn\_RAFL)**

**Table 31-38. ENETn\_RAFL Field Descriptions**

Field	Description
31–8	Reserved, must be cleared.
7–0 RX_ALMOST_FULL	Value, in 64-bit words, of the receive FIFO almost full threshold. See <a href="#">Table 31-88</a> for more details on the receive FIFO thresholds.

### 31.3.26 Transmit FIFO Section Empty Threshold (ENETn\_TSEM)

Address: 0xFC0D\_41A0 (ENET0\_TSEM)  
0xFC0D\_81A0 (ENET1\_TSEM) Access: User read/write

**Table 31-39. Transmit FIFO Section Empty Threshold (ENETn\_TSEM)**

**Table 31-40. ENETn\_TSEM Field Descriptions**

Field	Description
31–8	Reserved, must be cleared.
7–0 TX_SECTION_EMPTY	Value, in 64-bit words, of the transmit FIFO section empty threshold. See <a href="#">Table 31-89</a> for more details on the transmit FIFO thresholds.

### 31.3.27 Transmit FIFO Almost Empty Threshold (ENET $n$ \_TAEM)

Address: 0xFC0D\_41A4 (ENET0\_TAEM)  
0xFC0D\_81A4 (ENET1\_TAEM)

Access: User 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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TX_ALMOST_EMPTY							
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		

Table 31-41. Transmit FIFO Almost Empty Threshold (ENET $n$ \_TAEM)Table 31-42. ENET $n$ \_TAEM Field Descriptions

Field	Description
31–8	Reserved, must be cleared.
7–0 TX_ALMOST_EMPTY	Value, in 64-bit words, of the transmit FIFO almost empty threshold. See Table 31-89 for more details on the transmit FIFO thresholds.

### 31.3.28 Transmit FIFO Almost Full Threshold (ENET $n$ \_TAFL)

Address: 0xFC0D\_41A8 (ENET0\_TAFL)  
0xFC0D\_81A8 (ENET1\_TAFL)

Access: User 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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TX_ALMOST_FULL							
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		

Table 31-43. Transmit FIFO Almost Full Threshold (ENET $n$ \_TAFL)Table 31-44. ENET $n$ \_TAFL Field Descriptions

Field	Description
31–8	Reserved, must be cleared.
7–0 TX_ALMOST_FULL	Value, in 64-bit words, of the transmit FIFO almost full threshold. A minimum value of six is required. A recommended value of at least 8 should be set allowing a latency of two clock cycles to the application. If more latency is required the value can be increased as necessary (latency = ENET $n$ _TAFL – 5). See Table 31-89 for more details on the transmit FIFO thresholds. <b>Note:</b> A FIFO overflow is a fatal error and requires a global reset on the transmit datapath or at least deassertion of ETHER_EN.

### 31.3.29 Transmit Inter-Packet Gap (ENETn\_TIPG)

**Table 31-45. Transmit Inter-Packet Gap (ENETn\_TIPG)**

**Table 31-46. ENETn\_TIPG Field Descriptions**

Field	Description
31–5	Reserved, must be cleared.
4–0 IPG	Transmit inter-packet gap. Indicates the IPG, in bytes, between transmitted frames. Can be set between 8 and 27. If set to less than 8, the IPG is 8. If set to greater than 27, the IPG is 27.

### 31.3.30 Frame Truncation Length (ENETn\_FTRL)

**Table 31-47. Frame Truncation Length (ENETn\_FTRL)**

**Table 31-48. ENET $n$  FTRL Field Descriptions**

Field	Description
31–14	Reserved, must be cleared.
13–0 TRUNC_FL	<p>Frame truncation length. Indicates the value a receive frame is truncated, if it is greater than this value. Should be greater than or equal to ENETn_RCR[MAX_FL].</p> <p><b>Note:</b> Truncation happens at TRUNC_FL. However, when truncation occurs, the application (FIFO) may receive less data, guaranteeing that it never receives more than the set limit.</p>

### 31.3.31 Transmit Accelerator Function Configuration (ENET*n*\_TACC)

**ENETn\_TACC** controls accelerator actions when sending frames. The register can be changed before or after each frame, but must stay unmodified during frame writes into the transmit FIFO.

Address: 0xFC0D\_41C0 (ENET0\_TACC)  
0xFC0D\_81C0 (ENET1\_TACC)

Access: User 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	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	PRO	IP	0	0	SHIFT		
W																									CHK	CHK	0	0	16			
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	

**Table 31-49. Transmit Accelerator Function Configuration (ENETn\_TACC)****Table 31-50. ENETn\_TACC Field Descriptions**

Field	Description
31–5	Reserved, must be cleared.
4 PROCHK	Enables insertion of protocol checksum. 0 Checksum not inserted 1 If an IP frame with a known protocol is transmitted, the checksum is inserted automatically into the frame. The checksum field should be cleared. The other frames are not modified.
3 IPCHK	Enables insertion of IP header checksum. 0 Checksum not inserted 1 If an IP frame is transmitted, the checksum is inserted automatically. The IP header checksum field should be cleared. If a non-IP frame is transmitted the frame is not modified.
2–1	Reserved, must be cleared.
0 SHIFT16	TX FIFO shift-16. 0 Disabled 1 Indicates to the transmit data FIFO, that the written frames contain two additional octets before the frame data. This means the actual frame starts at bit 16 of the first word written into the FIFO. This function allows putting the frame payload on a 32-bit boundary in memory, as the 14-byte Ethernet header is extended to a 16-byte header. See <a href="#">Section 31.4.8.3, “32-bit Ethernet Payload Alignment”</a> .

### 31.3.32 Receive Accelerator Function Configuration (ENETn\_RACC)

Address: 0xFC0D\_41C4 (ENET0\_RACC)  
0xFC0D\_81C4 (ENET1\_RACC)

Access: User 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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SHIFT	LINE	0	0	PRO	IP	PAD	
W																								16	DIS	0	0	DIS	0	REM		
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	

**Table 31-51. Receive Accelerator Function Configuration (ENETn\_RACC)**

**Table 31-52. ENETn\_RACC Field Descriptions**

Field	Description
31–5	Reserved, must be cleared.
7 SHIFT16	<p>RX FIFO shift-16.</p> <p>0 Disabled</p> <p>1 Instructs the MAC to write two additional bytes in front of each frame received into the RX FIFO. The actual frame data then starts at bit 16 of the first word read from the RX FIFO aligning the Ethernet payload on a 32-bit boundary. See <a href="#">Section 31.4.8.3, “32-bit Ethernet Payload Alignment”</a>.</p> <p><b>Note:</b> This function only affects the FIFO storage and has no influence on the statistics, which use the actual length of the frame received.</p>
6 LINEDIS	<p>Enable discard of frames with MAC layer errors.</p> <p>0 Frames with errors are not discarded</p> <p>1 Any frame received with a CRC, length, or PHY error is automatically discarded and not forwarded to the user application interface. See <a href="#">Section 31.4.8.4, “Received Frame Discard”</a>.</p>
5–3	Reserved, must be cleared.
2 PRODIS	<p>Enable discard of frames with wrong protocol checksum.</p> <p>0 Frames with wrong checksum are not discarded</p> <p>1 If a TCP/IP, UDP/IP, or ICMP/IP frame is received that has a wrong TCP, UDP, or ICMP checksum, the frame is discarded. Discarding is only available when the RX FIFO operates in store and forward mode (ENETn_RSFL cleared). See <a href="#">Section 31.4.8.4, “Received Frame Discard”</a>.</p>
1 IPDIS	<p>Enable discard of frames with wrong IPv4 header checksum.</p> <p>0 Frames with wrong IPv4 header checksum are not discarded</p> <p>1 If an IPv4 frame is received with a mismatching header checksum, the frame is discarded. IPv6 has no header checksum and is not affected by this setting. Discarding is only available when the RX FIFO operates in store and forward mode (ENETn_RSFL cleared). See <a href="#">Section 31.4.8.4, “Received Frame Discard”</a>.</p>
0 PADREM	<p>Enable padding removal for short IP frames.</p> <p>0 Padding not removed</p> <p>1 Any bytes following the IP payload section of the frame are removed from the frame</p>

### 31.3.33 Timer Control Register (ENETn\_ATCR)

The command bits can trigger the corresponding events directly. It is not necessary to preserve any of the configuration bits when a command bit is set in the register (i.e. no read-modify-write is required). The bits are automatically cleared after the command completes.

Address: 0xFC0D\_4400 (ENET0\_ATCR)  
0xFC0D\_8400 (ENET1\_ATCR)

Access: User read/write

**Figure 31-20. Timer Control Register (ENET $n$  ATCR)**

Table 31-53. ENETn\_ATCR Field Descriptions

Field	Description
13 SLAVE	Enable timer slave mode. 0 The timer is active and all configuration bits in this register are relevant 1 The internal timer is disabled and the externally provided timer value is used. All other bits, except CAPTURE, in this register have no effect. CAPTURE can still be used to capture the current timer value.
12	Reserved, must be cleared.
11 CAPTURE	Capture timer value. 0 No effect 1 The current time is captured and can be read from the ATIMER register
10	Reserved, must be cleared.
9 RESTART	Resets the timer to zero. This has no effect on the counter enable. If the counter is enabled when this bit is set, the timer is reset to zero and starts counting from there. When set, all other bits are ignored during a write.
8	Reserved, must be cleared.
7 PINPER	Enable MAC output assertion on period event. 0 Disable 1 Enable
6	Reserved, must be cleared.
5	Reserved, must be set.
4 PEREN	Enable periodical event. 0 Disable 1 A period event interrupt can be generated (ENETn_EIR[TS_TIMER]) and the MAC output is asserted when the timer wraps around according to the periodic setting ENETn_ATPER. Set the timer period value before setting this bit.
3 OFFRST	Reset timer on offset event. 0 The timer is not affected and no action occurs (besides clearing OFFEN) when the offset is reached 1 If OFFEN is set, the timer resets to zero when the offset setting is reached <b>Note:</b> The offset event does not cause a timer interrupt.
2 OFFEN	Enable one-shot offset event. 0 Disable 1 The timer can be reset to zero when the given offset time is reached (offset event). The bit is cleared when the offset event is reached, so no further event occurs until the bit is set again. Set the timer offset value before setting this bit.
1	Reserved, must be cleared.
0 EN	Enable timer. 0 The timer stops at the current value 1 The timer starts incrementing

### 31.3.34 Timer Value Register (ENETn\_ATVR)

Address: 0xFC0D_4404 (ENET0_ATVR) 0xFC0D_8404 (ENET1_ATVR)	Access: User read/write

Table 31-54. Timer Value Register (ENETn\_ATVR)

Table 31-55. ENETn\_ATVR Field Descriptions

Field	Description
31–0 ATIME	A write sets the timer. A read returns the last captured value. To read the current value, issue a capture command (set ENETn_ATCR[CAPTURE]) prior to reading this register.

### 31.3.35 Timer Offset Register (ENETn\_ATOFF)

Address: 0xFC0D_4408 (ENET0_ATOFF) 0xFC0D_8408 (ENET1_ATOFF)	Access: User read/write

Table 31-56. Timer Offset Register (ENETn\_ATOFF)

Table 31-57. ENETn\_ATOFF Field Descriptions

Field	Description
31–0 OFFSET	Offset value for one-shot event generation. When the timer reaches the value an event can be generated to reset the counter. If the increment value in ENETn_ATINC is given in true nanoseconds, this value is also given in true nanoseconds.

### 31.3.36 Timer Period Register (ENETn\_ATPER)

Address: 0xFC0D_440C (ENET0_ATPER) 0xFC0D_840C (ENET1_ATPER)	Access: User read/write

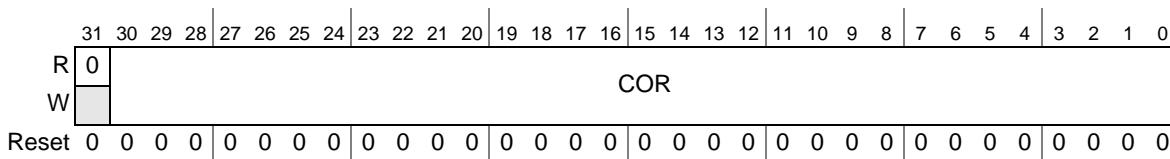
Table 31-58. Timer Period Register (ENETn\_ATPER)

**Table 31-59. ENET $n$ \_ATPER Field Descriptions**

Field	Description
31–0 PERIOD	<p>Value for generating periodic events. Each instance the timer reaches this value, the period event occurs and the timer restarts.</p> <p>If the increment value in ENET<math>n</math>_ATINC is given in true nanoseconds, this value is also given in true nanoseconds. The value should be initialized to 1,000,000,000 (<math>1 \times 10^9</math>) to represent a timer wrap around of one second. The increment value set in ENET<math>n</math>_ATINC should be set to the true nanoseconds of the period of clock ts_clk, hence implementing a true 1 second counter.</p> <p>See <a href="#">Section 31.4.10.1, “Adjustable Timer Module”</a> for a description of the timer implementation.</p>

### 31.3.37 Timer Correction Register (ENET $n$ \_ATCOR)

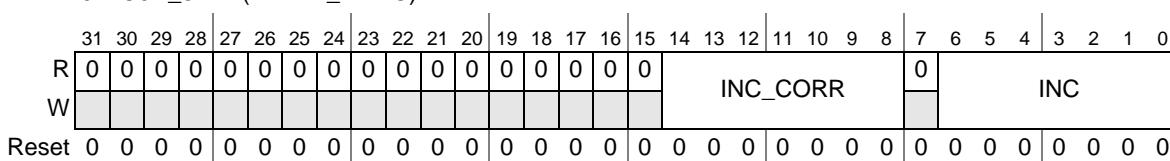
Address: 0xFC0D\_4410 (ENET0\_ATCOR)  
 Access: User read/write  
 0xFC0D\_8410 (ENET1\_ATCOR)

**Table 31-60. Timer Correction Register (ENET $n$ \_ATCOR)****Table 31-61. ENET $n$ \_ATCOR Field Descriptions**

Field	Description
31	Reserved, must be cleared.
30–0 COR	<p>Correction counter wrap-around value. Defines after how many timer clock cycles the correction counter should be reset and trigger a correction increment on the timer. The amount of correction is defined in ENET<math>n</math>_ATINC[INC_CORR].</p> <p>A value of 0 disables the correction counter and no corrections occur.</p> <p><b>Note:</b> This value is given in clock cycles, not in nanoseconds as all other values.</p> <p>See <a href="#">Section 31.4.10.1, “Adjustable Timer Module”</a> for a description of the timer implementation.</p>

### 31.3.38 Time-Stamping Clock Period Register (ENET $n$ \_ATINC)

Address: 0xFC0D\_4414 (ENET0\_ATINC)  
 Access: User read/write  
 0xFC0D\_8414 (ENET1\_ATINC)

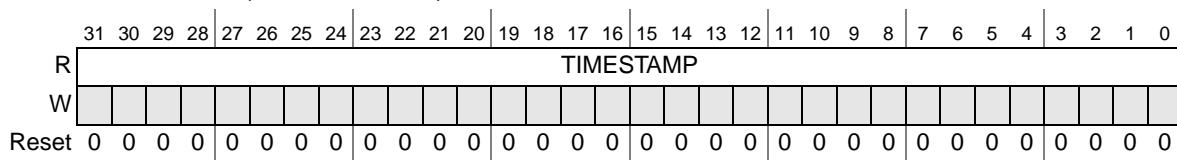
**Table 31-62. Clock Period for the Time Stamping Clock (ENET $n$ \_ATINC)**

**Table 31-63. ENETn\_ATINC Field Descriptions**

Field	Description
31–15	Reserved, must be cleared.
14–8 INC_CORR	Correction increment value. This value is added every time the correction timer expires (every clock cycle given in ENETn_ATCOR). A value smaller than INC slows the timer, while a value larger than INC speeds the timer.
7	Reserved, must be cleared.
6–0 INC	Clock period of the timestamping clock in nanoseconds. The timer increments by this amount each clock cycle. For example, set to 10 for 100 MHz, 8 for 125 MHz, 5 for 200 MHz. <b>Note:</b> For highest precision, use a value that is an integer fraction of the period set in ENETn_ATPER.

### 31.3.39 Timestamp of Last Transmitted Frame (ENETn\_ATSTMP)

Address: 0xFC0D\_4418 (ENET0\_ATSTMP) Access: User read-only  
0xFC0D\_8418 (ENET1\_ATSTMP)



**Table 31-64. Timestamp of Last Transmitted Frame (ENETn\_ATSTMP)**

**Table 31-65. ENETn\_ATSTMP Field Descriptions**

Field	Description
31–0 TIMESTAMP	Timestamp of the last frame transmitted by the core that had TxBD[TS] set. This register is only valid when ENETn_EIR[TS_AVAIL] is set.

### 31.3.40 Supplemental MAC Address Lower Registers (ENETn\_SMACLx)

`ENETn_SMACLx` contains the lower 32 bits (bytes 0, 1, 2, 3) of the 48-bit individual address used for exact match. These registers are not reset and you must initialize them. If the supplemental MAC addresses are not used, set them to the same value as the `ENETn_PALR/PAUR` registers.

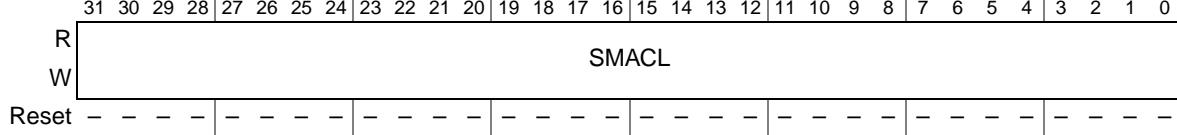
Address: 0xFC0D\_4500 (ENET0\_SMACLO)      0xFC0D\_8500 (ENET1\_SMACLO)      Access: User read/write

0xFC0D\_4508 (ENET0\_SMACL1) 0xFC0D\_8508 (ENET1\_SMACL1)

0xFC0D\_4510 (ENET0\_SMACL2)      0xFC0D\_8510 (ENET1\_SMACL2)

0xFC0D\_4518 (ENET0\_SMACL3)      0xFC0D\_8518 (ENET1\_SMACL3)

$=$   $\langle$   $\rangle$   $-$   $\langle$   $\rangle$   $-$   $\langle$   $\rangle$   $-$   $\langle$   $\rangle$   $-$   $\langle$   $\rangle$



**Table 31-66. Supplemental MAC Address Lower Registers (ENETn\_SMACLx)**

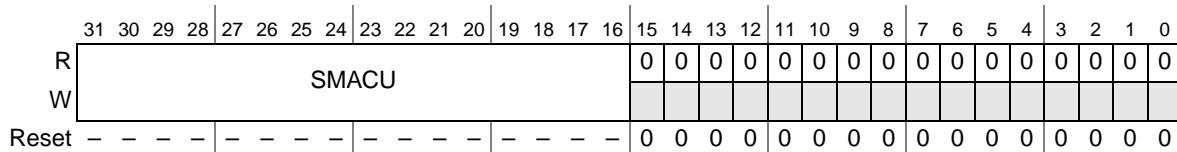
**Table 31-67. ENETn\_SMACLx Field Descriptions**

Field	Description
31–0 SMACL	Lower 32 bits (bytes 0,1,2,3) of the 48-bit individual address used for exact match

### 31.3.41 Supplemental MAC Address Upper Registers (ENETn\_SMACUx)

`ENETn_SMACUx` contains the upper 16 bits (bytes 4 and 5) of the 48-bit individual address used for exact match. The upper 16 bits of this register are not reset and you must initialize them. If the supplemental MAC addresses are not used, set them to the same value as the `ENETn_PALR/PAUR` registers.

Address: 0xFC0D_4504 (ENET0_SMACU0)	0xFC0D_8504 (ENET1_SMACU0)	Access: User read/write
0xFC0D_450C (ENET0_SMACU1)	0xFC0D_850C (ENET1_SMACU1)	
0xFC0D_4514 (ENET0_SMACU2)	0xFC0D_8514 (ENET1_SMACU2)	
0xFC0D_451C (ENET0_SMACU3)	0xFC0D_851C (ENET1_SMACU3)	



**Table 31-68. Supplemental MAC Address Upper Registers (ENETn\_SMACUx)**

**Table 31-69. ENETn\_SMACUx Field Descriptions**

Field	Description
31–16 SMACH	Upper 16 bits (bytes 4 and 5) of the 48-bit individual address used for exact match
15–0	Reserved, must be cleared.

## 31.4 Functional Description

The following sections describe functional details of the MAC-NET core.

### 31.4.1 Ethernet MAC Frame Formats

The IEEE 802.3 standard defines the Ethernet frame format as follows:

- Minimum length of 64 bytes
  - Maximum length of 1518 bytes, excluding the preamble and the SFD bytes

An Ethernet frame consists of the following fields:

- Seven bytes preamble
  - Start frame delimiter (SFD)
  - Two address fields
  - Length or type field
  - Data field

- Frame check sequence (CRC value)

**Figure 31-21. MAC Frame Format Overview**

Optionally, MAC frames can be VLAN-tagged with an additional four-byte field inserted between the MAC source address and the type/length field. VLAN tagging is defined by the IEEE P802.1q specification. VLAN-tagged frames have a maximum length of 1522 bytes, excluding the preamble and the SFD bytes.

**Figure 31-22. VLAN-Tagged MAC Frame Format Overview****Table 31-70. MAC Frame definition**

Term	Description
Frame length	Defines the length, in octets, of the complete frame without preamble and SFD. A frame has a valid length if it contains at least 64 octets and does not exceed the programmed maximum length (typical 1518).
Payload length	The length/type field indicates the length of the frame's payload section. The most significant byte is sent/received first. <ul style="list-style-type: none"> <li>If the length/type field is set to a value less than 46, the payload is padded so that the minimum frame length requirement (64 bytes) is met. For VLAN-tagged frames, a value less than 42 indicates a padded frame.</li> <li>If the length/type field is set to a value larger than the programmed frame maximum length (e.g. 1518) it is interpreted as a type field.</li> </ul>
Destination and source address	48-bit MAC addresses. The least significant byte is sent/received first and the first two least significant bits of the MAC address distinguish MAC frames as detailed in <a href="#">Section 31.4.4.3, "MAC Address Check"</a> .

### NOTE

Although the IEEE specification defines a maximum frame length, the MAC core provides the flexibility to program any value for the frame maximum length.

#### 31.4.1.1 Pause Frames

The receiving device generates a pause frame to indicate a congestion to the emitting device, which should stop sending data.

Pause frames are indicated by the length/type set to 0x8808. The two first bytes of a pause frame following the type, defines a 16-bit opcode field set to 0x0001 always. A 16-bit pause quanta is defined in the frame payload bytes 2 (P1) and 3 (P2) as defined in the following table. The P1 pause quanta byte is the most significant.

**Table 31-71. Pause Frame Format (Values in Hex)**

1	2	3	4	5	6	7	8	9	10	11	12	13	14
55	55	55	55	55	55	55	D5	01	80	C2	00	00	01
Preamble					SFD	Multicast Destination Address							
15	16	17	18	19	20	21	22	23	24	25	26	27 – 68	

**Table 31-71. Pause Frame Format (Values in Hex) (continued)**

00	00	00	00	00	00	88	08	00	01	hi	lo	00					
Source Address				Type		Opcode		P1	P2	pad (42)							
69      70      71      72				26    6B    AE    0A				CRC-32									

There is no payload length field found within a pause frame and a pause frame is always padded with 42 bytes (0x00).

If a pause frame with a pause value greater zero (XOFF condition) is received, the MAC stops transmitting data as soon the current frame transfer is completed. The MAC stops transmitting data for the value defined in pause quanta. One pause quanta fraction refers to 512 bit times.

If a pause frame with a pause value of zero (XON condition) is received, the transmitter is allowed to send data immediately (see [Section 31.4.6, “Full Duplex Flow Control Operation”](#) for details).

### 31.4.1.2 Magic Packets

A magic packet is a unicast, multicast, or broadcast packet, which carries a defined sequence in the payload section. Magic packets are received and inspected only under specific conditions as described in [Section 31.4.7, “Magic Packet Detection”](#).

The defined sequence to decode a magic packet is formed with a synchronization stream (six consecutive 0xFF bytes) followed by sequence of six consecutive unicast MAC addresses of the node to be awakened.

The sequence can be located anywhere in the magic packet payload and the magic packet is formed with standard Ethernet header and optional padding and CRC.

### 31.4.2 IP and Higher Layers Frame Format

The following sections use the term datagram to describe the protocol specific data unit that is found within the payload section of its container entity.

For example, an IP datagram specifies the payload section of an Ethernet frame. A TCP datagram specifies the payload section within an IP datagram.

### 31.4.2.1 Ethernet Types

IP datagrams are carried in the payload section of an Ethernet frame. The Ethernet frame type/length field discriminates several datagram types. The following table lists the types of interest:

**Table 31-72. Ethernet Type Value Examples**

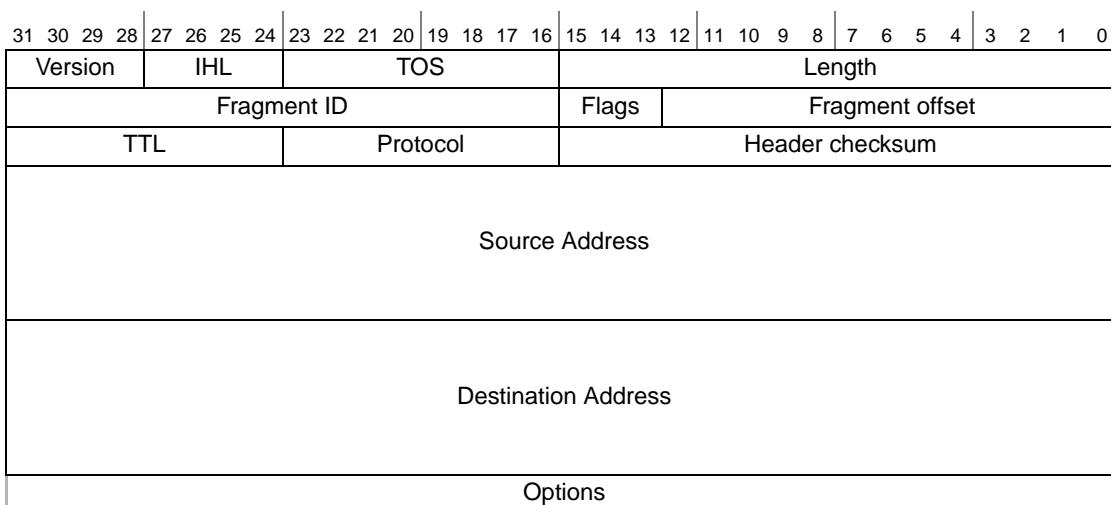
Type	Description
0x8100	VLAN-tagged frame. The actual type is found 4 octets later in the frame
0x0800	IP
0x0806	ARP
0x86DD	IPv6

### 31.4.2.2 IPv4 Datagram Format

The following figure shows the IP Version 4 (IPv4) header, which is located at the beginning of an IP datagram. It is organized in 32-bit words. The first byte sent/received is the leftmost byte of the first word (i.e. version/IHL field).

The IP header can contain further options, which are always padded if necessary to guarantee the payload following the header is aligned to a 32-bit boundary.

The IP header is followed by the payload immediately, which can contain further protocol headers (e.g., TCP or UDP as indicated by the protocol field value). The complete IP datagram is transported in the payload section of an Ethernet frame.



**Figure 31-23. IPv4 Header Format**

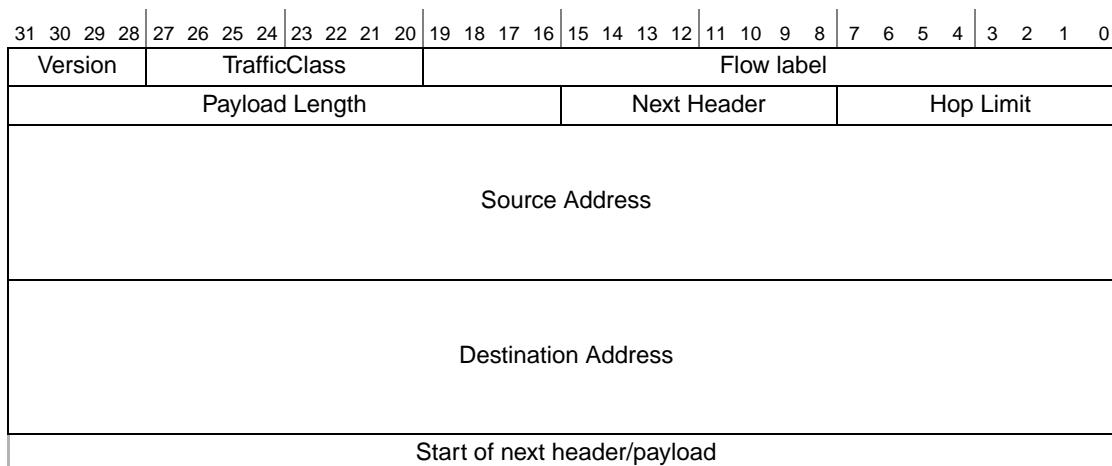
**Table 31-73. IPv4 Header Fields**

Field Name	Description
Version	4-bit IP version information. 0x4 for IPv4 frames.
IHL	4-bit internet header length information. Determines number of 32-bit words found within the IP header. If no options are present, the default value is 0x5.
TOS	Type of service/DiffServ field
Length	Total length of the datagram in bytes, including all octets of header and payload
Fragment ID, flags, fragment offset	Fields used for IP fragmentation
TTL	Time-to-live. If zero, datagram must be discarded
Protocol	Protocol identifier of protocol that follows in the datagram
Header checksum	Checksum over all IP header fields
Source address	Source IP address
Destination address	Destination IP address

### 31.4.2.3 IPv6 Datagram Format

The following figure shows the IP version 6 (IPv6) header, which is located at the beginning of an IP datagram. It is organized in 32-bit words and has a fixed length of ten words (40 bytes). The next header field identifies the type of the header to follow the IPv6 header. It is defined identical to the protocol identifier within IPv4 with new definitions for identifying extension headers, which can be inserted between the IPv6 header and the protocol header, shifting the protocol header accordingly. The accelerator currently only supports IPv6 without extension headers (i.e. next header identifies TCP, UDP, or ICMP protocol).

The first byte sent/received is the leftmost byte of the first word (i.e. version/traffic class fields).

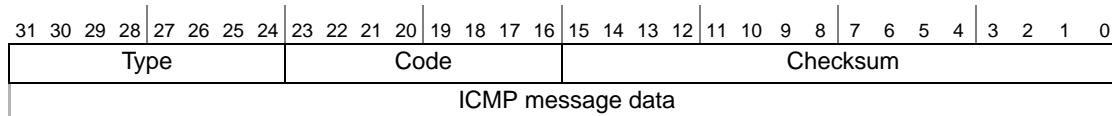
**Figure 31-24. IPv6 Header Format**

**Table 31-74. IPv6 Header Fields**

<b>Field Name</b>	<b>Description</b>
Version	4-bit IP version information. 0x6 for all IPv6 frames
Traffic class	8-bit field defining the traffic class
Flow label	20-bit flow label identifying frames of the same flow
Payload length	16-bit length of the datagram payload in bytes. It includes all octets following the IPv6 header.
Next header	Identifies the header that follows the IPv6 header. This can be the protocol header or any IPv6 defined extension header.
Hop limit	Hop counter, decremented by one by each station that forwards the frame. If hop limit is 0 the frame must be discarded.
Source address	128-bit IPv6 source address
Destination address	128-bit IPv6 destination address

#### 31.4.2.4 Internet Control Message Protocol (ICMP) Datagram Format

Following the IP header, an internet control message protocol (ICMP) datagram is found when the protocol identifier is 1. The ICMP datagram has a four octet header followed by additional message data.

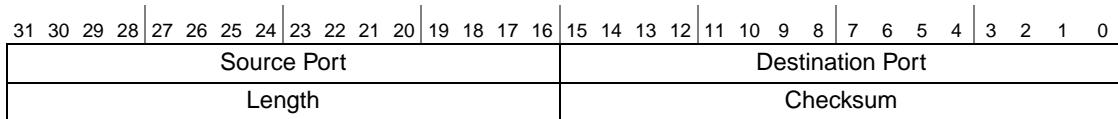
**Figure 31-25. ICMP Header Format****Table 31-75. IP Header Fields**

<b>Field Name</b>	<b>Description</b>
Type	8-bit type information
Code	8-bit code that is related to the message type
Checksum	16-bit one's complement checksum over the complete ICMP datagram

#### 31.4.2.5 User Datagram Protocol (UDP) Datagram Format

Following the IP header, a user datagram protocol header is found when the protocol identifier is 17.

Following the UDP header is the payload of the datagram. The header byte order follows the conventions given for the IP header above.

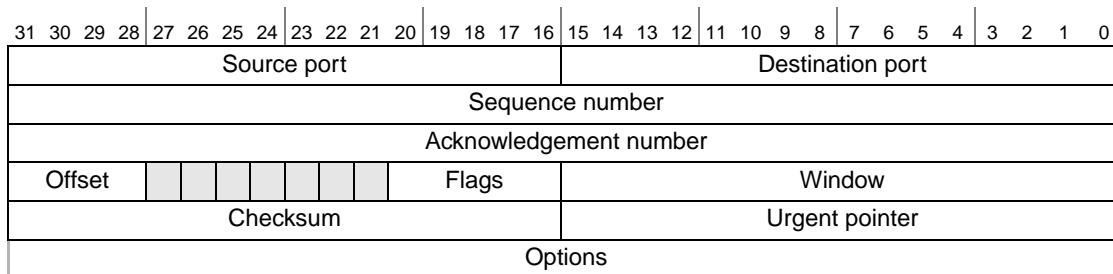
**Figure 31-26. UDP Header Format****Table 31-76. UDP Header Fields**

Field Name	Description
Source port	Source application port
Destination port	Destination application port
Length	Length of user data which follows immediately the header including the UDP header. That is, the minimum value is 8.
Checksum	Checksum over the complete datagram and some IP header information

### 31.4.2.6 TCP Datagram Format

Following the IP Header, a TCP header is found when the protocol identifier has a value of 6.

The TCP payload immediately follows the TCP header.

**Figure 31-27. TCP Header Format****Table 31-77. TCP Header Fields**

Field Name	Description
Source port	Source application port
Destination port	Destination application port
Sequence number	Transmit sequence number
Ack. number	Receive sequence number
Offset	Data offset. Number of 32-bit words within the TCP header. If no options, a value of 5.
Flags	URG, ACK, PSH, RST, SYN, FIN flags
Window	TCP receive window size information

**Table 31-77. TCP Header Fields (continued)**

Field Name	Description
Checksum	Checksum over the complete datagram (TCP header and data) and IP header information
Options	Additional 32-bit words for protocol options

### 31.4.3 IEEE 1588 Message Formats

The following sections describe the IEEE 1588 message formats.

#### 31.4.3.1 Transport Encapsulation

The precision time protocol (PTP) datagrams are encapsulated in Ethernet frames using the UDP/IP transport mechanism, or optionally, with the newer 1588v2 directly in Ethernet frames (layer 2). Typically, multicast addresses are used to allow efficient distribution of the synchronization messages.

##### 31.4.3.1.1 UDP/IP

The 1588 messages (v1 and v2) can be transported using UDP/IP multicast messages. The following IP multicast groups are defined for PTP. The table also shows their respective MAC layer multicast address mapping according to RFC 1112 (last three octets of IP follow the fixed value of 01-00-5E).

**Table 31-78. UDP/IP Multicast Domains**

Name	IP Address	MAC Address mapping
DefaultPTPdomain	224.0.1.129	01-00-5E-00-01-81
AlternatePTPdomain1	224.0.1.130	01-00-5E-00-01-82
AlternatePTPdomain2	224.0.1.131	01-00-5E-00-01-83
AlternatePTPdomain3	224.0.1.132	01-00-5E-00-01-84

**Table 31-79. UDP Port Numbers**

Message Type	UDP Port	Note
Event	319	Used for SYNC and DELAY_REQUEST messages
General	320	All other messages (e.g., follow-up, delay-response)

##### 31.4.3.1.2 Native Ethernet (PTPv2)

In addition to using UDP/IP frames, IEEE 1588v2 defines a native Ethernet frame format that uses ethertype = 0x88F7. The payload of the Ethernet frame immediately contains the PTP datagram, starting with the PTPv2 header.

Besides others, version 2 adds a peer delay mechanism to allow delay measurements between individual point-to-point links along a path over multiple nodes. The following multicast domains are additionally defined in PTPv2.

**Table 31-80. PTPv2 Multicast Domains**

Name	MAC Address
Normal messages	01-1B-19-00-00-00
Peer delay messages	01-80-C2-00-00-0E

### 31.4.3.2 PTP Header

All PTP frames contain a common header, which determines the protocol version and the type of message, which defines the further content of the message. All multi-octet fields are transmitted in big-endian order (the most significant byte is transmitted/received first).

The version field's (versionPTP) last four bits are at the same position (i.e. second byte) for PTPv1 and PTPv2 headers, allowing a correct identification by inspecting the first two bytes of the message.

#### 31.4.3.2.1 PTPv1 Header

**Table 31-81. Common PTPv1 Message Header**

Offset	Octets	Bits							
		7	6	5	4	3	2	1	0
0	2	versionPTP = 0x0001							
2	2	versionNetwork							
4	16	subdomain							
20	1	messageType							
21	1	sourceCommunicationTechnology							
22	6	sourceUuid							
28	2	sourcePortId							
30	2	sequenceId							
32	1	control							
33	1	0x00							
34	2	flags							
36	4	reserved							

The type of message is encoded in the messageType and control fields as follows:

**Table 31-82. PTPv1 Message Type Identification**

messageType	control	Message Name	Message
0x01	0x0	SYNC	Event message
0x01	0x1	DELAY_REQ	Event message

**Table 31-82. PTPv1 Message Type Identification (continued)**

<b>messageType</b>	<b>control</b>	<b>Message Name</b>	<b>Message</b>
0x02	0x2	FOLLOW_UP	General message
0x02	0x3	DELAY_RESP	General message
0x02	0x4	MANAGEMENT	General message
other	other	—	Reserved

The field sequenceId is used to non-ambiguously identify a message.

### 31.4.3.2.2 PTPv2 Header

**Table 31-83. Common PTPv2 message Header**

<b>Offset</b>	<b>Octets</b>	<b>Bits</b>										
		7	6	5	4	3	2	1	0			
0	1	transportSpecific					messageId					
1	1	reserved					versionPTP = 0x2					
2	2	messageLength										
4	1	domainNumber										
5	1	reserved										
6	2	flags										
8	8	correctionField										
16	4	reserved										
20	10	sourcePortIdentity										
30	2	sequenceId										
32	1	control										
33	1	logMeanMessageInterval										

The type of message is encoded in the field messageId as follows:

**Table 31-84. PTPv2 Message Type Identification**

<b>messageId</b>	<b>Message Name</b>	<b>Message</b>
0x0	SYNC	Event message
0x1	DELAY_REQ	Event message
0x2	PATH_DELAY_REQ	Event message
0x3	PATH_DELAY_RESP	Event message
0x4–0x7	—	reserved

**Table 31-84. PTPv2 Message Type Identification (continued)**

messageId	Message Name	Message
0x8	FOLLOW_UP	General message
0x9	DELAY_RESP	General message
0xa	PATH_DELAY_FOLLOW_UP	General message
0xb	ANNOUNCE	General message
0xc	SIGNALING	General message
0xd	MANAGEMENT	General message

The PTPv2 flags field contains further details on the type of message, especially if one-step or two-step implementations are used. The flags field consists of two octets with the following meanings for the bits. Reserved bits are cleared (false).

**Table 31-85. PTPv2 Message Flags Field Definitions**

Bit	Name	Description
0	ALTERNATE_MASTER	See IEEE 1588 Clause 17.4
1	TWO_STEP	1 Two-step clock 0 One-step clock
2	UNICAST	1 Transport layer address uses a unicast destination address 0 Multicast is used
3	—	Reserved
4	—	Reserved
5	Profile specific	
6	Profile specific	
7	—	Reserved

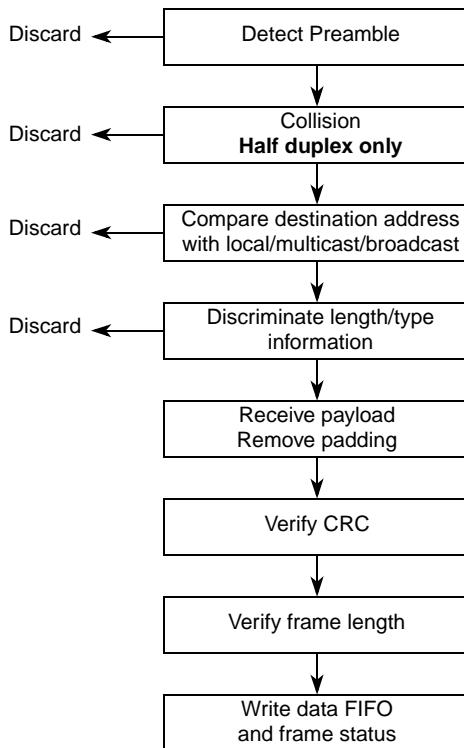
### 31.4.4 MAC Receive

The MAC receive engine performs the following tasks:

- Check frame framing
- Remove frame preamble and frame SFD field
- Frame discarding based on frame destination address field
- Terminate pause frames
- Check frame length
- Remove payload padding if it exists
- Calculate and verify CRC-32
- Write received frames in the core receive FIFO

If the MAC is programmed to operate in half duplex mode, the MAC performs the following additional action:

- Check if the frame is received with a collision



**Figure 31-28. MAC Receive Flow**

#### 31.4.4.1 Collision Detection in Half Duplex Mode

If the packet is received with a collision detected during reception of the first 64 bytes, the packet is discarded (if frame size was less than ~14 octets) or transmitted to the user application with an error and RxBD[CE] set.

#### 31.4.4.2 Preamble Processing

The IEEE 802.3 standard allows a maximum size of 56 bits (seven bytes) for the preamble, while the MAC core allows any arbitrary preamble length. The MAC core checks for the start frame delimiter (SFD) byte. If the next byte of the preamble, which is different from 0x55, is not 0xD5, the frame is discarded.

Although the IEEE specification specifies that frames should be separated by at least 96 bits (inter-packet gap), the MAC core is designed to accept frames only separated by 64 MII (10/100 Mbps operation) bits.

The MAC core removes the preamble and SFD bytes.

#### 31.4.4.3 MAC Address Check

The destination address bit 0 differentiates between multicast and unicast addresses:

- If bit 0 is 0, the MAC address is an individual (unicast) address
- If bit 0 is 1, the MAC address defines a group (multicast) address
- If all 48 bits of the MAC address are set, it indicates a broadcast address

### 31.4.4.3.1 Unicast Address Check

If a unicast address is received, the destination MAC address is compared to the node MAC address programmed by the host in the ENET $n$ \_PADDR1/2 registers. In addition, it is compared to the supplemental MAC addresses programmed in the ENET $n$ \_SMAC $n$  registers. If the destination address matches any of the programmed MAC addresses, the frame is accepted.

If only one MAC address is required, the supplemental MAC addresses should be programmed with the node MAC address.

If promiscuous mode is enabled (ENET $n$ \_RCR[PROM] = 1) no address checking is performed and all unicast frames are accepted.

### 31.4.4.3.2 Multicast and Unicast Address Resolution

The hash table algorithm used in the group and individual hash filtering operates as follows. The 48-bit destination address is mapped into one of 64 bits, represented by 64 bits in ENET $n$ \_GAUR/GALR (group address hash match) or ENET $n$ \_IAUR/IALR (individual address hash match). This mapping is performed by passing the 48-bit address through the on-chip 32-bit CRC generator and selecting the six most significant bits of the CRC-encoded result to generate a number between 0 and 63. The msb of the CRC result selects ENET $n$ \_GAUR (msb = 1) or ENET $n$ \_GALR (msb = 0). The five lsbs of the hash result select the bit within the selected register. If the CRC generator selects a bit set in the hash table, the frame is accepted; else, it is rejected.

For example, if eight group addresses are stored in the hash table and random group addresses are received, the hash table prevents roughly 56/64 (or 87.5%) of the group address frames from reaching memory. Those that do reach memory must be further filtered by the processor to determine if they truly contain one of the eight desired addresses.

The effectiveness of the hash table declines as the number of addresses increases.

The user must initialize the hash table registers. Use this CRC32 polynomial to compute the hash:

$$\bullet \quad FCS(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$$

If promiscuous mode is enabled (ENET $n$ \_RCR[PROM] = 1) all unicast and multicast frames are accepted regardless of ENET $n$ \_GAUR/GALR and ENET $n$ \_IAUR/IALR settings.

### 31.4.4.3.3 Broadcast Address Reject

All broadcast frames are accepted if BC\_REJ is cleared or ENET $n$ \_RCR[PROM] is set. If PROM is cleared when ENET $n$ \_RCR[BC\_REJ] is set, all broadcast frames are rejected.

**Table 31-86. Broadcast Address Reject Programming**

PROM	BC_REJ	Broadcast Frames
0	0	Accepted
0	1	Rejected
1	0	Accepted
1	1	Accepted

#### 31.4.4.3.4 Miss-Bit Implementation

For higher layer filtering purposes, RxBD[M] indicates an address miss when the MAC operates in promiscuous mode and accepted a frame that would otherwise be rejected.

If a group/individual hash or exact match does not occur and promiscuous mode is enabled (ENET $n$ \_RCR[PROM] = 1), the frame is accepted and the M bit is set in the buffer descriptor; otherwise, the frame is rejected.

This means the status bit is set in any of the following conditions during promiscuous mode:

- A broadcast frame is received when BC\_REJ is set.
- A unicast is received that does not match either of:
  - Node address (ENET $n$ \_PALR[PADDR1] and ENET $n$ \_PAUR[PADDR2])
  - Supplemental unicast addresses (ENET $n$ \_SMACU $x$  and ENET $n$ \_SMACL $x$ )
  - Hash table for unicast (ENET $n$ \_IAUR[IADDR1] and ENET $n$ \_IALR[IADDR2])
- A multicast is received that does not match the ENET $n$ \_GAUR[GADDR1] and ENET $n$ \_GALR[GADDR2] hash table entries

#### 31.4.4.4 Frame Length/Type Verification: Payload Length Check

If the length/type is less than 0x600 and NLC is set, the MAC checks the payload length and reports any error in the frame status word and interrupt bit PLR.

If the length/type is greater than or equal to 0x600, the MAC interprets the field as a type and no payload length check is performed.

The length check is performed on VLAN and stacked VLAN frames. If a padded frame is received, no length check can be performed due to the extended frame payload (i.e. padded frames can never have a payload length error).

#### 31.4.4.5 Frame Length/Type Verification: Frame Length Check

When the receive frame length exceeds MAX\_FL bytes, the BABR interrupt is generated and the RxBD[LG] bit is set.

The frame is not truncated unless the frame length exceeds the value programmed in ENET $n$ \_FTRL[TRUNC\_FL]. If the frame is truncated, RxBD[TR] is set. In addition, a truncated frame always has the CRC error indication set (RxBD[CR]).

### 31.4.4.6 VLAN Frames Processing

VLAN frames have a length/type field set to 0x8100 immediately followed by a 16-Bit VLAN control information field. VLAN-tagged frames are received as normal frames (the VLAN tag is not interpreted by the MAC function) and are completely (including the VLAN tag) pushed to the user application. If the length/type field of the VLAN-tagged frame, which is found four octets later in the frame, is less than 42, the padding is removed. In addition, the frame status word (RxBD[NO]) indicates that the current frame is VLAN tagged.

### 31.4.4.7 Pause Frame Termination

The receive engine terminates pause frames and they are not transferred to the receive FIFO. The quanta is extracted and sent to the MAC transmit path via a small internal clock rate decoupling asynchronous FIFO.

The quanta is written only if a correct CRC and frame length are detected by the control state machine. If not, the quanta is discarded and the MAC transmit path is not paused.

Good pause frames are ignored if ENETn\_RCR[FCE] is cleared and are forwarded to the client interface when ENETn\_RCR[PAUFWD] is set.

### 31.4.4.8 CRC Check

The CRC-32 field is checked and forwarded to the core FIFO interface if ENETn\_RCR[CRCFWD] is cleared and ENETn\_RCR[PADEN] is set. When CRCFWD is set (regardless of PADEN), the CRC-32 field is checked and terminated (not transmitted to the FIFO).

The CRC polynomial, as specified in the 802.3 standard, is:

- $FCS(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$

The 32 bits of the CRC value are placed in the frame check sequence (FCS) field with the  $x^{31}$  term as right-most bit of the first octet. The CRC bits are thus received in the following order:  $x^{31}, x^{30}, \dots, x^1, x^0$ .

If a CRC error is detected, the frame is marked invalid and RxBD[CR] is set.

### 31.4.4.9 Frame Padding Removal

When a frame is received with a payload length field set to less than 46 (42 for VLAN-tagged frames and 38 for frames with stacked VLANs), the zero padding can be removed before the frame is written into the data FIFO depending on the setting of ENETn\_RCR[PADEN].

#### NOTE

If a frame is received with excess padding (i.e. the length field is set as mentioned above, but the frame has more than 64 octets) and padding removal is enabled, the padding is removed as normal and no error is reported if the frame is otherwise correct (e.g. good CRC, less than maximum length, and no other error).

### 31.4.5 MAC Transmit

Frame transmission starts when the transmit FIFO holds enough data. Once a transfer starts, the MAC transmit function performs the following tasks:

- Generates preamble and SFD field before frame transmission
- Generates XOFF pause frames if the receive FIFO reports a congestion or if ENET $n$ \_TCR[TFC\_PAUSE] is set with ENET $n$ \_OPD[PAUSE\_DUR] set to a non-zero value
- Generates XON pause frames if the receive FIFO congestion condition is cleared or if TFC\_PAUSE is set with PAUSE\_DUR cleared
- Suspends Ethernet frame transfer (XOFF) if a non-zero pause quanta is received from the MAC receive path
- Adds padding to the frame if required
- Calculates and appends CRC-32 to the transmitted frame
- Send frame with correct inter-packet gap (IPG) (deferring)

When the MAC is configured to operate in half duplex mode, the following additional tasks are performed:

- Collision detection
- Frame retransmit after back-off timer expires

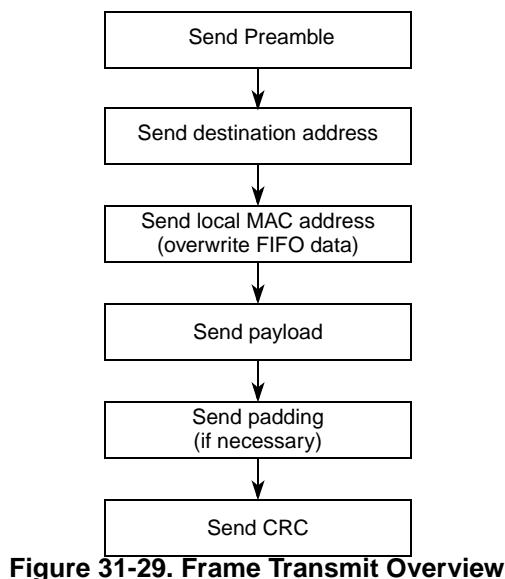


Figure 31-29. Frame Transmit Overview

#### 31.4.5.1 Frame Payload Padding

The IEEE specification defines a minimum frame length of 64 bytes. If the frame sent to the MAC from the user application has a size smaller than 60 bytes, the MAC automatically adds padding bytes (0x00) to comply with the Ethernet minimum frame length specification. Transmit padding is always performed and cannot be disabled.

If the MAC is not allowed to append a CRC (TxBD[TC] = 1), the user application is responsible for providing frames with a minimum length of 64 octets.

### 31.4.5.2 MAC Address Insertion

On each frame received from the core transmit FIFO interface, the source MAC address is either:

- Replaced by the address programmed in the PADDR1/2 fields (ENETn\_TCR[ADDINS] = 1)
- Transparently forwarded to the Ethernet line (ENETn\_TCR[ADDINS] = 0)

### 31.4.5.3 CRC-32 generation

The CRC-32 field is optionally generated and appended at the end of a frame. The CRC polynomial, as specified in the 802.3 standard, is:

- $FCS(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$

The 32 bits of the CRC value are placed in the FCS field so that the  $x^{31}$  term is the right-most bit of the first octet. The CRC bits are thus transmitted in the following order:  $x^{31}, x^{30}, \dots, x^1, x^0$ .

### 31.4.5.4 Inter-Packet Gap

In full duplex mode, after frame transmission and before transmission of a new frame, an inter-packet gap (programmed in ENETn\_TIPG) is maintained. The minimum IPG can be programmed between 8 and 27 byte-times (64 and 216 bit-times).

In half duplex mode, the core constantly monitors the line. Actual transmission of the data onto the network occurs only if it has been idle for a 96-bit time period and any back-off time requirements have been satisfied. In accordance with the standard, the core begins to measure the IPG from MII\_CRS de-assertion.

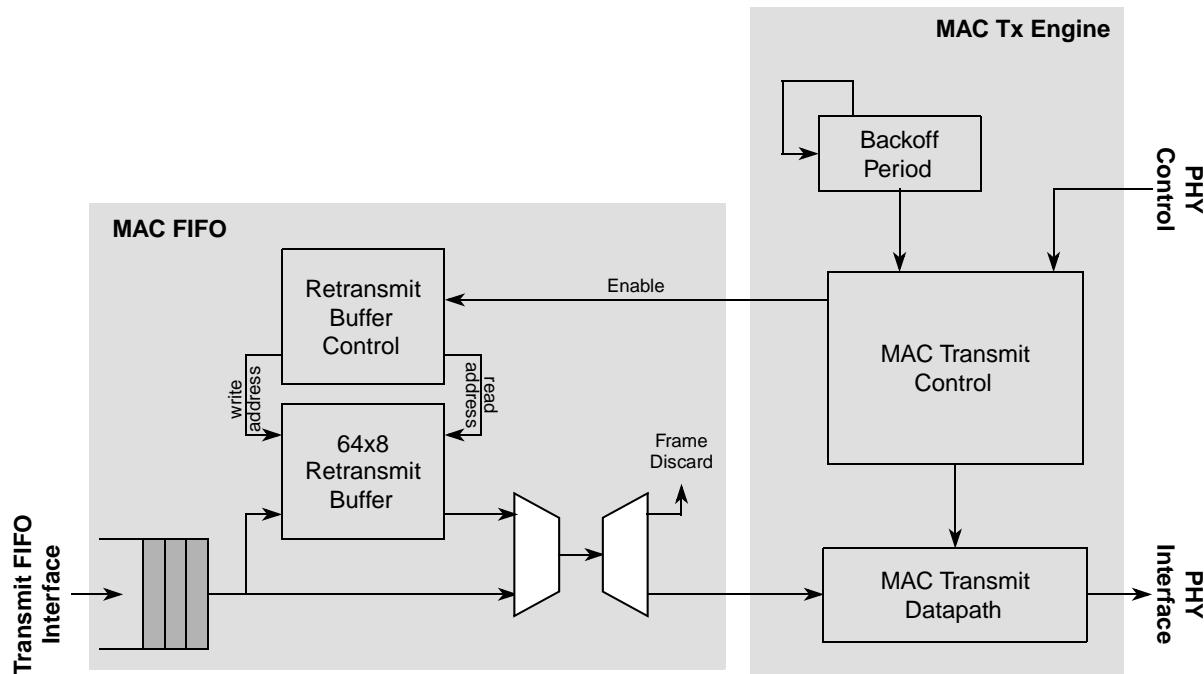
### 31.4.5.5 Collision Detection and Handling—Half Duplex Operation Only

A collision occurs on a half-duplex network when concurrent transmissions from two or more nodes take place. During transmission, the core monitors the line condition and detects a collision when the PHY device asserts MII\_COL.

When the core detects a collision while transmitting, it stops transmission of the data and transmits a 32-bit jam pattern. If the collision is detected during the preamble or the SFD transmission, the jam pattern is transmitted after completing the SFD, which results in a minimum 96-bit fragment. The jam pattern is a fixed pattern that is not compared to the actual frame CRC and has a very low probability (0.532) of having a jam pattern identical to the CRC.

If a collision occurs before transmission of 64 bytes (including preamble and SFD), the MAC core waits for the back-off period and retransmits the packet data (stored in a 64-byte re-transmit buffer) already sent on the line. The backoff period is generated from a pseudo-random process (truncated binary exponential backoff).

If a collision occurs after transmission of 64 bytes (including preamble and SFD), the MAC discards the remainder of the frame, optionally sets the LC interrupt bit, and sets TxBD[LCE].

**Figure 31-30. Packet Re-Transmit Overview**

The back-off time is represented by an integer multiple of slot times (one slot is equal to a 512-bit time period). The number of the delay slot times, before the  $n^{\text{th}}$  re-transmission attempt, is chosen as a uniformly-distributed random integer in the range:

- $0 < r < 2^k$
- $k = \min(n, N)$ ; where  $n$  is the number of retransmissions and  $N = 10$

For example, after the first collision, the backoff period is 0 or 1 slot time. If a collision occurs on the first retransmission, the backoff period is 0, 1, 2, or 3 and so on.

The maximum backoff time (in 512-bit time slots) is limited by  $N = 10$  as specified in the IEEE 802.3 standard.

If a collision occurs after 16 consecutive retransmissions, the core reports an excessive collision condition (ENETn\_EIR[RL] interrupt bit and TxBD[EE]) and discards the current packet from the FIFO.

In networks violating the standard requirements, a collision may occur after transmission of the first 64 bytes. In this case, the core stops the current packet transmission and discards the rest of the packet from the transmit FIFO. The core resumes transmission with the next packet available in the core transmit FIFO.

### 31.4.6 Full Duplex Flow Control Operation

Three conditions are handled by the core's flow control engine:

- Remote device congestion — The remote device connected to the same Ethernet segment as the core reports a congestion requesting the core to stop sending data

- Core FIFO congestion — When the core's receive FIFO reaches a user-programmable threshold (RX section empty), the core sends a pause frame back to the remote device requesting the data transfer to stop
- Local device congestion — Any device connected to the core can request (typically, via the host processor) the remote device to stop transmitting data

### 31.4.6.1 Remote Device Congestion

When the MAC transmit control gets a valid pause quanta from the receive path and if ENET $n$ \_RCR[FCE] is set, the MAC transmit logic:

- Completes the transfer of the current frame
- Stops sending data for the amount of time specified by the pause quanta in 512 bit time increments
- Sets ENET $n$ \_TCR[RFC\_PAUSE]

Frame transfer resumes when the time specified by the quanta expires and if no new quanta value is received or if a new pause frame with a quanta value set to 0x0000 is received. The MAC also resets RFC\_PAUSE to zero.

If ENET $n$ \_RCR[FCE] cleared, the MAC ignores received pause frames.

Optionally and independent of ENET $n$ \_RCR[FCE], pause frames are forwarded to the client interface if PAUFWD is set.

### 31.4.6.2 Local Device/FIFO Congestion

The MAC transmit engine generates pause frames when the local receive FIFO is not able to receive more than a pre-defined number of words (FIFO programmable threshold) or when pause frame generation is requested by the local host processor:

- To generate a pause frame, the host processor sets ENET $n$ \_TCR[TFC\_PAUSE]. A single pause frame is generated when the current frame transfer is completed and TFC\_PAUSE is automatically cleared. Optionally, an interrupt is generated.
- A XOFF pause frame is generated when the receive FIFO asserts its section empty flag (internal). A XOFF pause frame is generated automatically, when the current frame transfer completes.
- A XON pause frame is generated when the receive FIFO deasserts its section empty flag (internal). A XON pause frame is generated automatically, when the current frame transfer completes.

When a XOFF pause frame is generated, the pause quanta (payload byte P1 and P2) is filled with the value programmed in ENET $n$ \_OPD[PAUSE\_DUR].

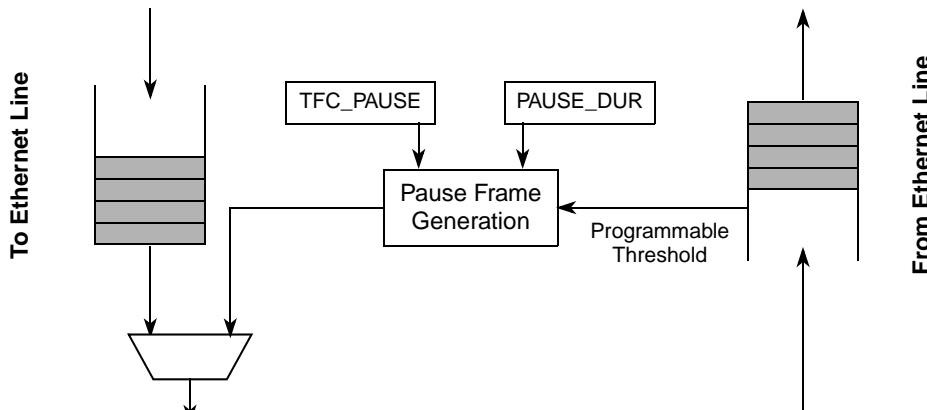


Figure 31-31. Pause Frame Generation Overview

**NOTE**

Although the flow control mechanism should prevent any FIFO overflow on the MAC core receive path, the core receive FIFO is protected. When an overflow is detected on the receive FIFO, the current frame is truncated with an error indication set in the frame status word. The frame should subsequently be discarded by the user application.

### 31.4.7 Magic Packet Detection

Magic packet detection wakes a node that is put in power-down mode by the node management agent. Magic packet detection is supported only if the MAC is configured in sleep mode.

#### 31.4.7.1 Sleep Mode

To put the MAC in sleep mode, set ENETn\_ECR[SLEEP] and ensure that ENETn\_ECR[MAGICEN] is set to enable magic packet detection.

In addition, when the processor is in stop mode, sleep mode is entered, without affecting the ENETn\_ECR register bits.

When the core is in sleep mode:

- The MAC transmit logic is disabled
- The core FIFO receive/transmit functions are disabled
- The MAC receive logic is kept in normal mode, but it ignores all traffic from the line except magic packets. They are detected so that a remote agent can wake the node.

#### 31.4.7.2 Magic Packet Detection

The core is designed to detect magic packets (see [Section 31.4.1.2, “Magic Packets”](#)) with the destination address set to:

- Any multicast address

- The broadcast address
- The unicast address programmed in PADDR1/2
- If enabled, to any of the unicast addresses programmed in the core supplemental MAC address registers (ENET $n$ \_SMACL $x$  and ENET $n$ \_SMACU $x$ )

When a magic packet is detected, ENET $n$ \_EIR[WAKEUP] is set and none of the statistic registers are incremented.

### 31.4.7.3 Wake-up

When a magic packet is detected, indicated by ENET $n$ \_EIR[WAKEUP], ENET $n$ \_ECR[SLEEP] should be cleared to resume normal operation of the MAC. Clearing the SLEEP bit automatically masks ENET $n$ \_ECR[MAGICEN], disabling magic packet detection.

## 31.4.8 IP Accelerator Functions

The following sections describe the IP accelerator functions.

### 31.4.8.1 Checksum Calculation

The IP and ICMP, TCP, UDP checksums are calculated with one's complement arithmetic summing up 16-bit values.

- For ICMP the checksum is calculated over the complete ICMP datagram (i.e. without IP header).
- For TCP and UDP the checksums contain the header and data sections and values from the IP header, which can be seen as a pseudo header that is not actually present in the datastream.

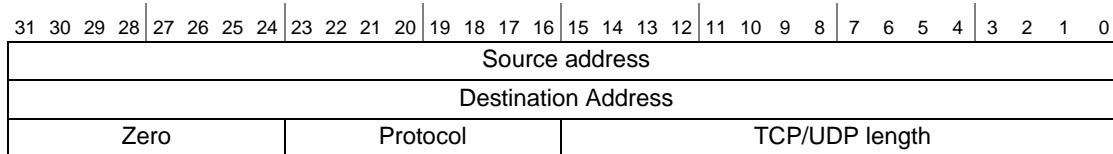


Figure 31-32. IPv4 Pseudo Header for Checksum calculation

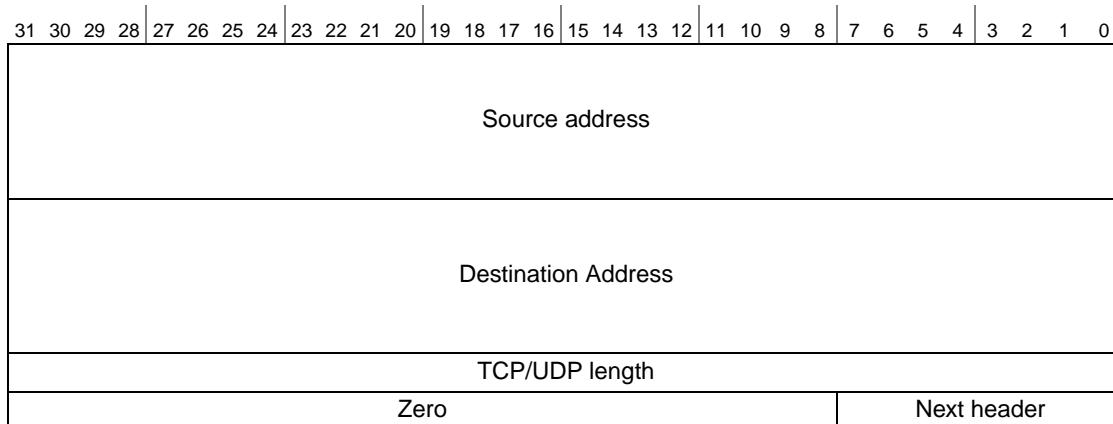


Figure 31-33. IPv6 Pseudo Header for Checksum Calculation

The TCP/UDP length value is the length of the TCP or UDP datagram, which is equal to the payload of an IP datagram. It is derived by subtracting the IP header length from the complete IP datagram length that is given in the IP header (IPv4) or directly taken from the IP header (IPv6). The protocol field is the corresponding value from the IP header and Zero is filled with zeroes.

For IPv6 the complete 128-bit addresses are considered. The next header value identifies the upper layer protocol (TCP or UDP) and may differ from the IPv6 header's actual next header value if extension headers are inserted before the protocol header.

The checksum calculation uses 16-bit words in network byte order: The first byte sent/received is the MSB, and the second byte sent/received is the LSB of the 16-bit value to add to the checksum. If the frame ends on an odd number of bytes, a zero byte is appended for checksum calculation only (not actually transmitted).

### 31.4.8.2 Additional Padding Processing

According to IEEE 802.3, any Ethernet frame must have a minimum length of 64 octets. The MAC usually removes padding on receive when a frame with length information is received. As IP frames have a type value instead of length, the MAC does not remove padding for short IP frames, as it is not aware of the frame contents.

The IP accelerator function can be configured to remove the Ethernet padding bytes that might follow the IP datagram.

On transmit, the MAC automatically adds padding as necessary to fill any frame to a 64-byte length.

### 31.4.8.3 32-bit Ethernet Payload Alignment

The data FIFOs allow inserting two additional arbitrary bytes in front of a frame. This extends the 14-byte Ethernet header to a 16-byte header, which leads to alignment of the Ethernet payload, following the Ethernet header, on a 32-bit boundary.

This function can be enabled for transmit and receive independently with the corresponding SHIFT16 bits in the ENETn\_TACC and ENETn\_RACC registers.

When enabled, the valid frame data is arranged as shown in this table.

**Table 31-87. 64-Bit Interface Data Structure with SHIFT16 Enabled**

63	56	55	48	47	40	39	32	31	24	23	16	15	8	7	0
Byte 5		Byte 4		Byte 3		Byte 2		Byte 1		Byte 0		Any value		Any value	
Byte 13		Byte 12		Byte 11		Byte 10		Byte 9		Byte 8		Byte 7		Byte 6	
...															

#### 31.4.8.3.1 Receive Processing

When ENETn\_RACC[SHIFT16] is set, each frame is received with two additional bytes in front of the frame. The user application must ignore these first two bytes and find the first byte of the frame in bits 23–16 of the first word from the RX FIFO.

**NOTE**

SHIFT16 must be set during initialization and kept set during the complete operation, as it influences the FIFO write behavior.

**31.4.8.3.2 Transmit Processing**

When ENET $n$ \_TACC[SHIFT16] is set, the first two bytes of the first word written (bits 15–0) are discarded immediately by the FIFO write logic. The SHIFT16 bit can be enabled/disabled for each frame individually if required, but can be changed only between frames.

**31.4.8.4 Received Frame Discard**

As the receive FIFO must be operated in store and forward mode (ENET $n$ \_RSFL cleared), received frames can be discarded based on the following errors:

- The MAC function receives the frame with an error:
  - The frame has an invalid payload length
  - Frame length is greater than MAX\_FL
  - Frame received with a CRC-32 error
  - Frame truncated due to receive FIFO overflow
  - Frame is corrupted as PHY signaled an error (MII\_RX\_ERR asserted during reception)
- An IP frame is detected and the IP header checksum is wrong
- An IP frame with a valid IP header and a valid IP header checksum is detected, the protocol is known but the protocol specific checksum is wrong

If one of the errors occurs and the IP accelerator function is configured to discard frames (ENET $n$ \_RACC), the frame is automatically discarded. Statistics are maintained normally and are not affected by this discard function.

**31.4.8.5 IPv4 Fragments**

When an IP (IPv4) fragment frame is received only the IP header is inspected and its checksum verified. 32-bit alignment operates on fragments as on normal IP frames, as specified above.

The IP fragment frame payload is not inspected for any protocol headers. As such, a protocol header would only exist in the very first fragment. To assist in protocol-specific checksum verification, the one's-complement sum is calculated on the IP payload (all bytes following the IP header) and provided with the frame status word.

The frame fragment status bit, RxBD[FRAG], is set to indicate a fragment reception and the one's-complement sum of the IP payload is available in RxBD[Payload checksum].

**NOTE**

The application software can take advantage of the payload checksum delivered with the frame's status word to calculate the protocol-specific checksum of the datagram after all fragments have been received and reassembled.

For example, if a TCP payload is delivered by multiple IP fragments, the application software can calculate the pseudo-header checksum value from the first fragment and add the payload checksums delivered with the status for all fragments to verify the TCP datagram checksum.

### 31.4.8.6 IPv6 Support

The following sections describe the IPv6 support.

#### 31.4.8.6.1 Receive Processing

An Ethernet frame of type 0x86DD identifies an IP Version 6 frame (IPv6) frame. If an IPv6 frame is received, the first IP header is inspected (first ten words) which is available in every IPv6 frame.

If the receive SHIFT16 function is enabled, the IP header is aligned on a 32-bit boundary allowing more efficient processing (see [Section 31.4.8.3, “32-bit Ethernet Payload Alignment”](#)).

For TCP and UDP datagrams the pseudo-header checksum calculation is performed and verified.

To assist in protocol-specific checksum verification, the one's-complement sum is always calculated on the IP payload (all bytes following the IP header) and provided with the frame status word. For example, if extension headers were present, their sums can be subtracted in software from the checksum to isolate the TCP/UDP datagram checksum, if required.

#### 31.4.8.6.2 Transmit Processing

For IPv6 transmission the SHIFT16 function is supported to process 32-bit aligned datagrams.

IPv6 has no IP header checksum; therefore, the IP checksum insertion configuration is ignored.

The protocol checksum is inserted only if the next header of the IP header is a known protocol (TCP, UDP, or ICMP). If a known protocol is detected, the checksum over all bytes following the IP header is calculated and inserted in the correct position.

The pseudo-header checksum calculation is performed for TCP and UDP datagrams accordingly.

### 31.4.9 Resets and Stop Controls

The following sections describe the resets and stop controls.

#### 31.4.9.1 Hardware Reset

To reset the core, set ENETn\_ECR[RESET].

#### 31.4.9.2 Soft Reset

When ENETn\_ECR[ETHER\_EN] is cleared during operation, the following occurs:

- DMA, buffer descriptor, and FIFO control logic are reset, including the buffer descriptor and FIFO pointers
- A currently ongoing transmit is terminated by asserting MII\_TXER to the PHY

- A currently ongoing transmit FIFO write from the application is terminated by stopping the write to the FIFO, and all further data from the application is ignored. All subsequent writes are ignored until reenabled.
- A currently ongoing receive FIFO read is terminated. The RxBD has arbitrary values in this case.

### 31.4.9.3 Hardware Freeze

When the processor enters debug mode and ENET $n$ \_ECR[DBGEN] is set, the MAC enters a freeze state where it stops all transmit and receive activities gracefully. The following happens when the MAC enters hardware freeze:

- A currently ongoing receive transaction on the receive application interface is completed as normal. No further frames are read from the FIFO.
- A currently ongoing transmit transaction on the transmit application interface is completed as normal (i.e. until writing end-of-packet (eop)).
- A currently ongoing MII frame receive is completed normally. After that, no further frames are accepted from the MII.
- A currently ongoing MII frame transmit is completed normally. After that, no further frames are transmitted.

### 31.4.9.4 Graceful Stop

During a graceful stop any currently ongoing transactions are completed normally and no further frames are accepted. The MAC can resume from a graceful stop without the need for a reset (e.g. clearing ETHER\_EN is not required). The following conditions lead to a graceful stop of the MAC transmit or receive datapaths.

#### 31.4.9.4.1 Graceful Transmit Stop (GTS)

When gracefully stopped, the MAC is no longer reading frame data from the transmit FIFO and has completed any ongoing transmission. In any of the following conditions, the transmit datapath stops after an ongoing frame transmission has been completed normally.

- ENET $n$ \_TCR[GTS] is set by software
- ENET $n$ \_TCR[TFC\_PAUSE] is set by software requesting a pause frame transmission. The status (and register bit) is cleared after the pause frame has been sent.
- A pause frame was received stopping the transmitter. The stopped situation is terminated when the pause timer expires or a pause frame with zero quanta is received.
- MAC is placed in sleep mode by software or the processor entering stop mode (see [Section 31.4.7.1, “Sleep Mode”](#)).
- The MAC is in hardware freeze mode

When the transmitter has reached its stopped state, the following events occur:

- The GRA interrupt is asserted, when transitioned into stopped
- In hardware freeze mode, the GRA interrupt does not wait for the application write completion and asserts when the transmit state machine (line side of TX FIFO) reaches its stopped state.

### 31.4.9.4.2 Graceful Receive Stop (GRS)

When gracefully stopped, the MAC is no longer writing frames into the receive FIFO. The receive datapath stops after any ongoing frame reception has been completed normally, if any of the following conditions occur:

- MAC is placed in sleep mode (by software or the processor is in stop mode). The MAC continues to receive frames and hunt for magic packets if enabled (see [Section 31.4.7, “Magic Packet Detection”](#)). However, no frames are written into the receive FIFO, and therefore are not forwarded to the application.
- The MAC is in hardware freeze mode. The MAC does not accept any frames from the MII.

When the receive datapath is stopped the following events occur:

- If the RX is in the stopped state, ENETn\_RCR[GRS] is set
- The GRA interrupt is asserted when the transmitter and receiver are stopped
- Any ongoing receive transaction to the application (RX FIFO read) continues normally until the frame is completed (end of packet (eop)). After this, the following occurs:
  - When sleep mode is active, all further frames are discarded, flushing the RX FIFO
  - In hardware freeze mode, no further frames are delivered to the application and they stay in the receive FIFO.

#### NOTE

The assertion of GRS does not wait for an ongoing transaction on the application side of the FIFO (FIFO read).

### 31.4.9.4.3 Graceful Stop Interrupt (GRA)

The graceful stopped interrupt (GRA) is asserted for the following conditions:

- In sleep mode, the interrupt asserts only after both TX and RX datapaths are stopped
- In hardware freeze mode, the interrupt asserts only after both TX and RX datapaths are stopped
- The MAC transmit datapath is stopped for any other condition (GTS, TFC\_PAUSE, pause received)

The GRA interrupt is triggered only once when the stopped state is entered. If the interrupt is cleared while the stop condition persists, no further interrupt is triggered.

## 31.4.10 IEEE 1588 Functions

To allow for IEEE 1588 or similar time synchronization protocol implementations, the MAC is combined with a time-stamping module to support precise time stamping of incoming and outgoing frames. Set ENETn\_ECR[1588EN] to enable 1588 support.

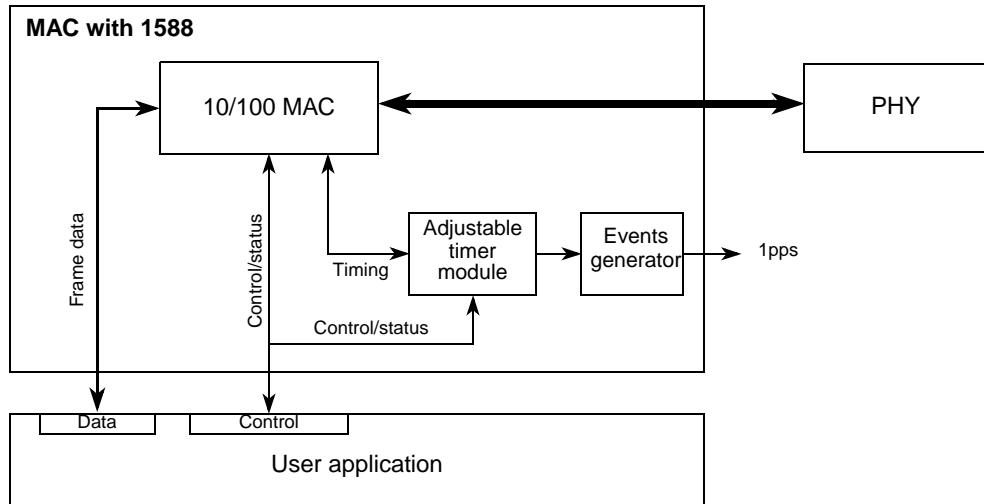


Figure 31-34. IEEE 1588 Functions Overview

### 31.4.10.1 Adjustable Timer Module

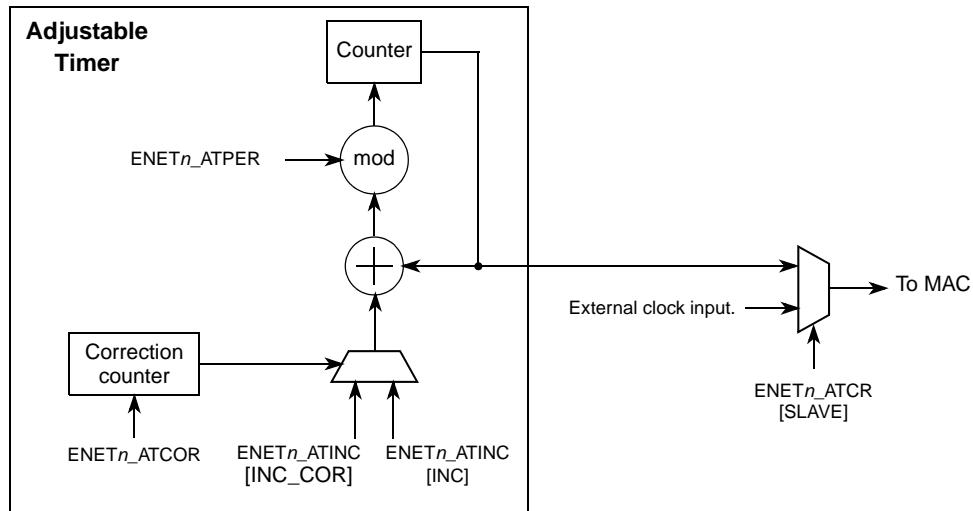
The adjustable timer module (TSM) implements the free running counter (FRC), which generates the timestamps. The FRC operates with the time-stamping clock, which can be set to any value depending on your system requirements. However, choose a period which is an integer value (e.g. 5ns, 6ns, 8ns) to implement a precise timer.

Through dedicated correction logic, the timer can be adjusted to allow synchronization to a remote master and provide a synchronized timing reference to the local system. The timer can be configured to cause an interrupt after a fixed time period to allow synchronization of software timers or perform other synchronized system functions.

The timer is usually used to implement a period of one second; hence, its value ranges from 0 to  $(1 \times 10^9) - 1$ . The period event can trigger an interrupt and software can maintain the seconds and hours time values as necessary.

#### 31.4.10.1.1 Adjustable Timer Implementation

The adjustable timer consists of a programmable counter/accumulator and a correction counter. The periods of both counters and its increment rate are freely configurable allowing very fine tuning of the timer. See [Section 31.4.10.2, “Timer Synchronization for Multi-Port Implementations,”](#) for external clock input options.



**Figure 31-35. Adjustable Timer Implementation Detail**

The counter produces the current time. During each time-stamping clock cycle a constant value is added to the current time as programmed in ENET<sub>n</sub>\_ATINC. The value depends on the chosen time-stamping clock frequency. For example, if it operates at 125 MHz setting the increment to eight represents 8 ns.

The period, configured in ENET<sub>n</sub>\_ATPER, defines the modulo when the counter wraps. In a typical implementation the period is set to  $1 \times 10^9$  so the counter wraps every second, and hence all timestamps represent the absolute nanoseconds within the one second period. When the period is reached, the counter wraps to start again respecting the period modulo. This means it does not necessarily start from zero, but instead the counter is loaded with the value (Current + Inc -  $(1 \times 10^9)$ ), assuming the period is set to  $1 \times 10^9$ .

The correction counter operates fully independently and increments by one with each time-stamping clock cycle. When it reaches the value configured in ENET<sub>n</sub>\_ATCOR, it restarts and instructs the timer once to increment by the correction value, instead of the normal value. The normal and correction increments are configured in ENET<sub>n</sub>\_ATINC. To speed up the timer, set the correction increment more than the normal increment value. To slow down the timer, set the correction increment less than the normal increment value. The correction counter only defines the distance of the corrective actions, not the amount. This allows very fine corrections and low jitter (in the range of 1 ns) independent of the chosen clock frequency.

By enabling slave mode (ENET<sub>n</sub>\_ATCR[SLAVE] = 1) the timer is ignored and the current time is externally provided from one of the external modules as defined by MISCCR3[ENETCLK] in the CCM module. This is useful if multiple modules within the system must operate from a single timer (see [Section 31.4.10.2, “Timer Synchronization for Multi-Port Implementations”](#)). When slave mode is enabled, you still must set ENET<sub>n</sub>\_ATINC[INC] to the value of the master, since it is used for internal comparisons.

### 31.4.10.2 Timer Synchronization for Multi-Port Implementations

Additional inputs are available to provide a timer value for all time-stamping functions. This is necessary to synchronize the two MACs to a single reference timer. MISCCR3[ENETCLK] in the CCM module

configures the clock used as an input to the time-stamping functions. To operate the MAC in slave mode, ENETn\_ATCR[SLAVE] disables the internal adjustable timer and uses the externally provided timer.

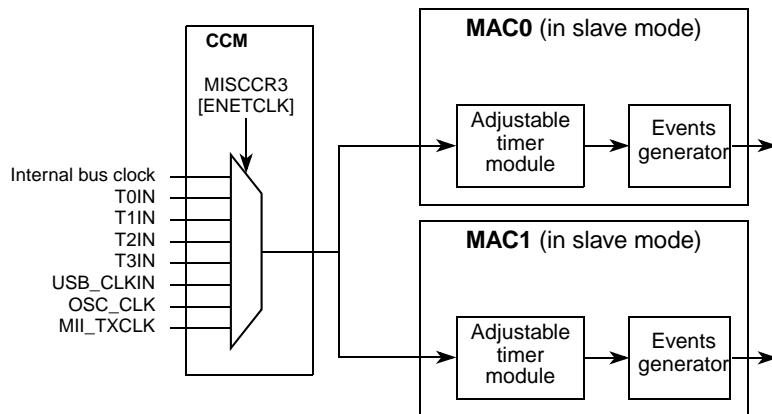


Figure 31-36. 1588 Multiple MAC implementation

### 31.4.10.3 Transmit Timestamping

Only 1588 event frames need to be time-stamped on transmit. The client application (e.g. the MAC driver) should detect 1588 event frames and set TxBD[TS] together with the frame.

If TxBD[TS] is set, the MAC records the timestamp for the frame in ENETn\_ATSTMP. ENETn\_EIR[TS\_AVAIL] is set to indicate that a new timestamp is available.

Software implements a handshaking procedure by setting TxBD[TS] when it transmits the frame it needs a timestamp for and then waits for ENETn\_EIR[TS\_AVAIL] to know when the timestamp is available. It then can read the timestamp from ENETn\_ATSTMP. This is done for all event frames. Other frames do not use TxBD[TS] and, therefore, do not interfere with the timestamp capture.

### 31.4.10.4 Receive Timestamping

When a frame is received, the MAC latches the value of the timer when the frame's SFD field is detected and provides the captured timestamp on RxBD[1588 timestamp]. This is done for all received frames.

### 31.4.10.5 Time Synchronization

The adjustable timer module is available to synchronize the local clock of a node to a remote master. It implements a free running 32-bit counter, and also contains an additional correction counter. The correction counter increases or decreases the rate of the free running counter, enabling very fine granular changes of the timer for synchronization, yet adding only very low jitter when performing corrections.

The application software implements, in a slave scenario, the required control algorithm setting the correction to compensate for local oscillator drifts and locking the timer to the remote master clock on the network.

The timer and all timestamp-related information should be configured to show the true nanoseconds value of a second (i.e. the timer is configured to have a period of one second). Hence, the values range from 0 to

$(1 \times 10^9) - 1$ . In this application, the seconds counter is implemented in software using an interrupt function that is executed when the nanoseconds counter wraps at  $1 \times 10^9$ .

### 31.4.11 FIFO Thresholds

The core FIFO thresholds are fully programmable to dynamically change the FIFO operation. For example, store and forward transfer can be enabled by a simple change in the FIFO threshold registers. The thresholds are defined in 64-bit words.

#### 31.4.11.1 Receive FIFO

Four programmable thresholds are available, which can be set to any value to control the core operation as follows.

**Table 31-88. Receive FIFO Thresholds Definition**

Register	Description
ENETn_RSFL [RX_SECTION_FULL]	When the FIFO level reaches the ENETn_RSFL value, the MAC status signal is asserted to indicate that data is available in the receive FIFO (cut-through operation). Once asserted, if the FIFO empties below the threshold set with ENETn_RAEM and if the end-of-frame is not yet stored in the FIFO, the status signal is deasserted again. If a frame has a size smaller than the threshold (i.e. an end-of-frame is available for the frame), the status is also asserted. To enable store and forward on the receive path, clear ENETn_RSFL. the MAC status signal is asserted only when a complete frame is stored in the receive FIFO. When programming a non-zero value to ENETn_RSFL (cut-through operation) it should be greater than ENETn_RAEM.
ENETn_RAEM [RX_ALMOST_EMPTY]	When the FIFO level reaches the ENETn_RAEM value, and the end-of-frame has not been received, the core receive read control stops the FIFO read (and subsequently stops transferring data to the MAC client application). It continues to deliver the frame, if again more data than the threshold or the end-of-frame is available in the FIFO. Set ENETn_RAEM to a minimum of six.
ENETn_RAFL [RX_ALMOST_FULL]	When the FIFO level comes close to the maximum, so that there is no more space for at least ENETn_RAFL number of words, the MAC control logic stops writing data in the FIFO and truncates the received frame to avoid FIFO overflow. The corresponding error status is set when the frame is delivered to the application. Set ENETn_RAFL to a minimum of 4.
ENETn_RSEM [RX_SECTION_EMPTY]	When the FIFO level reaches the ENETn_RSEM value, an indication is sent to the MAC transmit logic, which generates a XOFF pause frame. This indicates FIFO congestion to the remote Ethernet client. When the FIFO level goes below the value programmed in ENETn_MRBR, an indication is sent to the MAC transmit logic, which generates a XON pause frame. This indicates the FIFO congestion is cleared to the remote Ethernet client. Clearing ENETn_RSEM disables any pause frame generation.

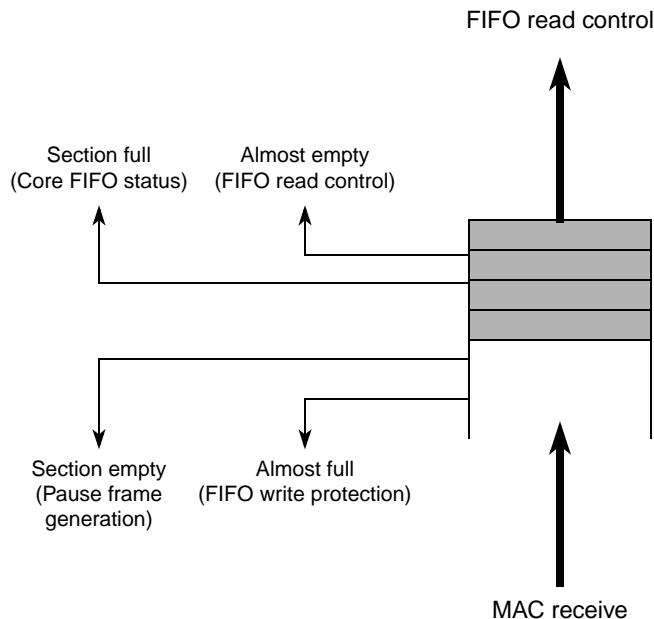


Figure 31-37. Receive FIFO Overview

### 31.4.11.2 Transmit FIFO

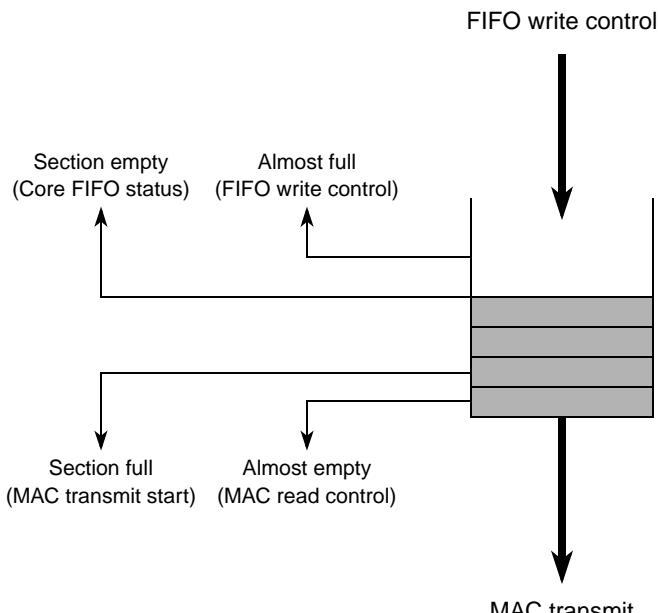
Four programmable thresholds are available which control the core operation as described below.

Table 31-89. Transmit FIFO Thresholds Definition

Register	Description
ENET $n$ _TAEM [TX_ALMOST_EMPTY]	When the FIFO level reaches the ENET $n$ _TAEM value and no end-of-frame is available for the frame, the MAC transmit logic avoids a FIFO underflow by stopping FIFO reads and transmitting the Ethernet frame with an MII error indication. Set ENET $n$ _TAEM to a minimum of 4.
ENET $n$ _TAFL [TX_ALMOST_FULL]	When the FIFO level approaches the maximum, so that there is no more space for at least ENET $n$ _TAFL number of words, the MAC deasserts its control signal to the application. If the application does not react on this signal, the FIFO write control logic avoids FIFO overflow by truncating the current frame and setting the error status. As a result, the frame is transmitted with an MII error indication. Set ENET $n$ _TAFL to a minimum of 4. Larger values allow more latency for the application to react on the MAC control signal deassertion, before the frame is truncated. A typical setting is 8, which offers 3–4 clock cycles of latency to the application to react on the MAC control signal deassertion.

**Table 31-89. Transmit FIFO Thresholds Definition (continued)**

Register	Description
ENET <sub>n</sub> _TSEM [TX_SECTION_EMPTY]	When the FIFO level reaches the ENET <sub>n</sub> _TSEM value, a MAC status signal is deasserted to indicate that the transmit FIFO is getting full. This gives the application an indication to slow or stop its write transaction to avoid a buffer overflow. This is a pure indication function to the application. It has no effect within the MAC. When ENET <sub>n</sub> _TSEM is 0, the signal is never deasserted.
ENET <sub>n</sub> _TFWR	When the FIFO level reaches the ENET <sub>n</sub> _TFWR value and when STRFWD is cleared, the MAC transmit control logic starts frame transmission before the end-of-frame is available in the FIFO (cut-through operation). If a complete frame has a size smaller than the ENET <sub>n</sub> _TFWR threshold, the MAC also transmits the frame to the line. To enable store and forward on the transmit path, set STRFWD. In this case, the MAC starts to transmit data only when a complete frame is stored in the transmit FIFO.

**Figure 31-38. Transmit FIFO Overview**

### 31.4.12 Loopback Options

The core implements external and internal loopback options, which are controlled by the following ENET<sub>n</sub>\_RCR register bits:

**Table 31-90. Loopback Options**

Register Bit	Description
LOOP	Internal MII loopback. The MAC transmit is returned to the MAC receive. No data is transmitted to the external interfaces.

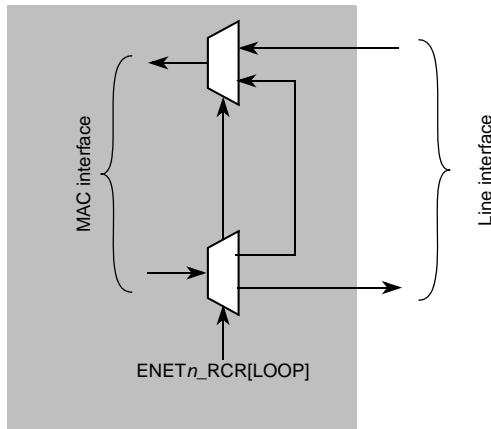


Figure 31-39. Loopback Options

### 31.4.13 Legacy Buffer Descriptors

To support the Ethernet controller on previous ColdFire devices, legacy FEC buffer descriptors are available. To enable legacy support, clear ENETn\_ECR[1588EN].

#### 31.4.13.1 Legacy Receive Buffer Descriptor

The following figure shows the legacy FEC receive buffer descriptor. [Table 31-91](#) contains the descriptions for each field.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Offset + 0	E	RO1	W	RO2	L	—	—	M	BC	MC	LG	NO	—	CR	OV	TR
Offset + 2 Data length																
Offset + 4 Rx data buffer pointer - A[31:16]																
Offset + 6 Rx data buffer pointer - A[15:0]																

Figure 31-40. Legacy FEC Receive Buffer Descriptor (RxBD)

#### 31.4.13.2 Legacy Transmit Buffer Descriptor

The following figure shows the legacy FEC transmit buffer descriptor. [Table 31-92](#) contains the descriptions for each field.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Offset + 0	R	TO1	W	TO2	L	TC	ABC <sup>1</sup>	—	—	—	—	—	—	—	—	—
Offset + 2 Data Length																
Offset + 4 Tx Data Buffer Pointer - A[31:16]																
Offset + 6 Tx Data Buffer Pointer - A[15:0]																

Figure 31-41. Legacy FEC Transmit Buffer Descriptor (TxD)

<sup>1</sup> This bit is not supported by the uDMA.

### 31.4.14 Enhanced Buffer Descriptors

This section provides a description of the enhanced operation of the driver/DMA via the buffer descriptors. It is followed by a detailed description of the receive and transmit descriptor fields. To enable the enhanced features, set ENETn\_ECR[1588EN].

#### 31.4.14.1 Enhanced Receive Buffer Descriptor

This section discusses the enhanced uDMA receive buffer descriptor.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Offset + 0	E	RO1	W	RO2	L	—	—	M	BC	MC	LG	NO	—	CR	OV	TR
Offset + 2	Data length															
Offset + 4	Rx data buffer pointer - A[31:16]															
Offset + 6	Rx data buffer pointer - A[15:0]															
Offset + 8	ME	—	—	—	—	PE	CE	UC	INT	—	—	—	—	—	—	—
Offset + A	—	—	—	—	—	—	—	—	—	—	ICE	PCR	—	VLAN	IPV6	FRAG
Offset + C	Header length					—	—	—	Protocol type							
Offset + E	Payload checksum															
Offset + 10	BDU	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Offset + 12	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Offset + 14	1588 timestamp [31:16]															
Offset + 16	1588 timestamp [15:0]															
Offset + 18	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Offset + 1A	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Offset + 1C	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
Offset + 1E	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 31-42. Enhanced uDMA Receive Buffer Descriptor (RxBD)

Table 31-91. Receive Buffer Descriptor Field Definitions

Word	Field	Description
Offset + 0	15 E	Empty. Written by the MAC (=0) and user (=1). 0 The data buffer associated with this BD is filled with received data, or data reception has aborted due to an error condition. The status and length fields have been updated as required. 1 The data buffer associated with this BD is empty, or reception is currently in progress.
Offset + 0	14 RO1	Receive software ownership. This field is reserved for use by software. This read/write bit is not modified by hardware, nor does its value affect hardware.

**Table 31-91. Receive Buffer Descriptor Field Definitions (continued)**

<b>Word</b>	<b>Field</b>	<b>Description</b>
Offset + 0	13 W	Wrap. Written by user. 0 The next buffer descriptor is found in the consecutive location 1 The next buffer descriptor is found at the location defined in ENETn_RDSR
Offset + 0	12 RO2	Receive software ownership. This field is reserved for use by software. This read/write bit is not modified by hardware, nor does its value affect hardware.
Offset + 0	11 L	Last in frame. Written by the uDMA. 0 The buffer is not the last in a frame. 1 The buffer is the last in a frame.
Offset + 0	10–9	Reserved, must be cleared.
Offset + 0	8 M	Miss. Written by the MAC. This bit is set by the MAC for frames accepted in promiscuous mode, but flagged as a miss by the internal address recognition. Therefore, while in promiscuous mode, you can use the M-bit to quickly determine whether the frame was destined to this station. This bit is valid only if the L and PROM bits are set. 0 The frame was received because of an address recognition hit 1 The frame was received because of promiscuous mode
Offset + 0	7 BC	Set if the DA is broadcast (FFFF_FFFF_FFFF).
Offset + 0	6 MC	Set if the DA is multicast and not BC.
Offset + 0	5 LG	Rx frame length violation. Written by the MAC. A frame length greater than RCR[MAX_FL] was recognized. This bit is valid only if the L bit is set. The receive data is not altered in any way unless the length exceeds TRUNC_FL bytes.
Offset + 0	4 NO	Receive non-octet aligned frame. Written by the MAC. A frame that contained a number of bits not divisible by 8 was received, and the CRC check that occurred at the preceding byte boundary generated an error or a PHY error occurred. This bit is valid only if the L bit is set. If this bit is set, the CR bit is not set.
Offset + 0	3	Reserved, must be cleared.
Offset + 0	2 CR	Receive CRC or frame error. Written by the MAC. This frame contains a PHY or CRC error and is an integral number of octets in length. This bit is valid only if the L bit is set.
Offset + 0	1 OV	Overrun. Written by the MAC. A receive FIFO overrun occurred during frame reception. If this bit is set, the other status bits, M, LG, NO, CR, and CL lose their normal meaning and are zero. This bit is valid only if the L bit is set.
Offset + 0	0 TR	Set if the receive frame is truncated (frame length > TRUNC_FL). If the TR bit is set, the frame must be discarded and the other error bits must be ignored as they may be incorrect.
Offset + 2	15–0 Data Length	Data length. Written by the MAC. Data length is the number of octets written by the MAC into this BD's data buffer if L is cleared (the value is equal to EMRBR), or the length of the frame including CRC if L is set. It is written by the MAC once as the BD is closed.
Offset + 4	15–0 A[31:16]	RX data buffer pointer, bits [31:16] <sup>1</sup>
Offset + 6	15–0 A[15:0]	RX data buffer pointer, bits [15:0]
Offset + 8	15 ME	MAC error. This bit is written by the uDMA. This bit means that the frame stored in the system memory was received with an error. This bit is only valid when the L bit is set.

**Table 31-91. Receive Buffer Descriptor Field Definitions (continued)**

<b>Word</b>	<b>Field</b>	<b>Description</b>
Offset + 8	14–11	Reserved, must be cleared.
Offset + 8	10 PE	PHY Error. This bit is written by the uDMA. Set to “1” when the frame was received with an Error character on the PHY interface. The frame is invalid. This bit is valid only when the L bit is set.
Offset + 8	9 CE	Collision. This bit is written by the uDMA. Set when the frame was received with a collision detected during reception. The frame is invalid and sent to the user application. This bit is valid only when the L bit is set.
Offset + 8	8 UC	Unicast. This bit is written by the uDMA. This bit means that the frame is unicast. This bit is valid regardless of if the L bit is set.
Offset + 8	7 INT	Generate RXB/RXF interrupt. This bit is set by the user. This bit indicates that the uDMA is to generate an interrupt on the <i>dma_int_rxb / dma_int_rxf</i> event.
Offset + 8	6–0	Reserved, must be cleared.
Offset + A	15–6	Reserved, must be cleared.
Offset + A	5 ICE	IP header checksum error. This is an accelerator option. This bit is written by the uDMA. Set when either a non-IP frame is received or the IP header checksum was invalid. This bit is only valid if the L bit is set.
Offset + A	4 PCR	Protocol checksum error. This is an accelerator option. This bit is written by the uDMA. Set when the checksum of the protocol is invalid or an unknown protocol is found and checksumming could not be performed. This bit is only valid if the L bit is set.
Offset + A	3	Reserved, must be cleared.
Offset + A	2 VLAN	VLAN. This is an accelerator option. This bit is written by the uDMA. This bit means that the frame has a VLAN tag. This bit is valid only if the L bit is set.
Offset + A	1 IPv6	IPv6 Frame. This bit is written by the uDMA. This bit indicates that the frame has a IPv6 frame type. If this bit is not set it means that an IPv4 or other protocol frame was received. This bit is valid only if the L bit is set.
Offset + A	0 FRAG	IPv4 Fragment. This is an accelerator option. This bit is written by the uDMA. This bit indicates that the frame is an IPv4 fragment frame. This bit is only valid when the L bit is set.
Offset + C	15–11 Header length	Header length. This is an accelerator option. This field is written by the uDMA. This field is the sum of 32 bit words found within the IP and its following protocol headers. If an IP datagram with an unknown protocol is found the value is the length of the IP header. If no IP frame or an erroneous IP header is found, the value is 0. The following values are minimum values if no header options exist in the respective headers: <ul style="list-style-type: none"> <li>• ICMP/IP: 6 (5 IP header, 1 ICMP header)</li> <li>• UDP/IP: 7 (5 IP header, 2 UDP header)</li> <li>• TCP/IP: 10 (5 IP header, 5 TCP header)</li> </ul> This field is only valid if the L bit is set.
Offset + C	10–8	Reserved, must be cleared.
Offset + C	7–0 Protocol type	Protocol type. This is an accelerator option. The 8-bit protocol field found within the IP header of the frame. Only valid if ICE is cleared. This bit is only valid if the L bit is set.
Offset + E	15–0 Payload checksum	Internet payload checksum. This is an accelerator option. The one's complement sum of the payload section of the IP frame. The sum is calculated over all data following the IP header until the end of the IP payload. This field is valid only when the L bit is set.

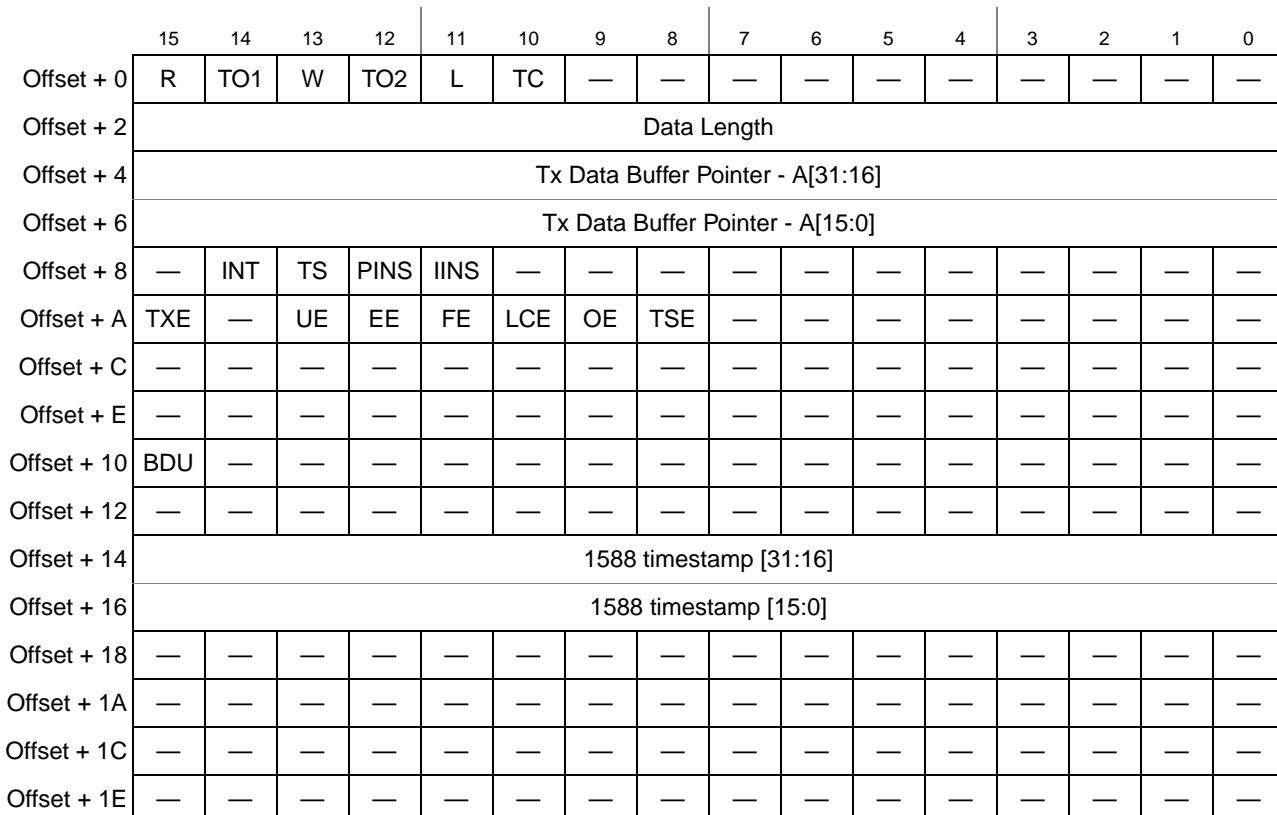
**Table 31-91. Receive Buffer Descriptor Field Definitions (continued)**

Word	Field	Description
Offset + 10	15 BDU	Last buffer descriptor update done. Indicates that the last BD data has been updated by uDMA. This bit is written by the user (=0) and uDMA (=1).
Offset + 10	14–0	Reserved, must be cleared.
Offset + 12	15–0	Reserved, must be cleared.
Offset + 14	15–0	This value is written by the uDMA. It is only valid if the L bit is set.
Offset + 16	1588 timestamp	
Offset + 18 – Offset + 1E	15–0	Reserved, must be cleared.

<sup>1</sup> The receive buffer pointer, containing the address of the associated data buffer, must always be evenly divisible by 16. The buffer must reside in memory external to the MAC. The Ethernet controller never modifies this value.

### 31.4.14.2 Enhanced Transmit Buffer Descriptor

This section discusses the enhanced uDMA transmit buffer descriptor.

**Figure 31-43. Enhanced Transmit Buffer Descriptor (TxD)**

**Table 31-92. Enhanced Transmit Buffer Descriptor Field Definitions**

<b>Word</b>	<b>Field</b>	<b>Description</b>
Offset + 0	15 R	Ready. Written by the MAC and you. 0 The data buffer associated with this BD is not ready for transmission. You are free to manipulate this BD or its associated data buffer. The MAC clears this bit after the buffer has been transmitted or after an error condition is encountered. 1 The data buffer, prepared for transmission by you, has not been transmitted or currently transmits. You may write no fields of this BD after this bit is set.
Offset + 0	14 TO1	Transmit software ownership. This field is reserved for software use. This read/write bit is not modified by hardware nor does its value affect hardware.
Offset + 0	13 W	Wrap. Written by user. 0 The next buffer descriptor is found in the consecutive location 1 The next buffer descriptor is found at the location defined in ETDSR.
Offset + 0	12 TO2	Transmit software ownership. This field is reserved for use by software. This read/write bit is not modified by hardware nor does its value affect hardware.
Offset + 0	11 L	Last in frame. Written by user. 0 The buffer is not the last in the transmit frame 1 The buffer is the last in the transmit frame
Offset + 0	10 TC	Transmit CRC. Written by user (only valid if L is set). 0 End transmission immediately after the last data byte 1 Transmit the CRC sequence after the last data byte This bit is valid only when the L bit is set.
Offset + 0	9 ABC	Append bad CRC. <b>Note:</b> This bit is not supported by the uDMA and is ignored.
Offset + 0	8–0	Reserved, must be cleared.
Offset + 2	15–0 Data Length	Data length, written by user. Data length is the number of octets the MAC should transmit from this BD's data buffer. It is never modified by the MAC. Bits [15:5] are used by the DMA engine; bits[4:0] are ignored.
Offset + 4	15–0 A[31:16]	Tx data buffer pointer, bits [31:16]. The transmit buffer pointer, containing the address of the associated data buffer, must always be evenly divisible by 8. The buffer must reside in memory external to the MAC. This value is never modified by the Ethernet controller.
Offset + 6	15–0 A[15:0]	Tx data buffer pointer, bits [15:0]
Offset + 8	15	Reserved, must be cleared.
Offset + 8	14 INT	Generate interrupt. This bit is written by the user. This bit is valid regardless of the L bit and must be the same for all EBD for a given frame. The uDMA does not update this value.
Offset + 8	13 TS	Timestamp. This bit is written by the user. This indicates that the uDMA is to generate a timestamp frame to the MAC. This bit is valid regardless of the L bit and must be the same for all EBD for the given frame. The uDMA does not update this value.
Offset + 8	12 PINS	Insert protocol specific checksum. This bit is written by the user. If set, the MAC's IP accelerator calculates the protocol checksum and overwrites the corresponding checksum field with the calculated value. The checksum field must be cleared by the application generating the frame. The uDMA does not update this value. This bit is valid regardless of the L bit and must be the same for all EBD for a given frame.

**Table 31-92. Enhanced Transmit Buffer Descriptor Field Definitions (continued)**

Word	Field	Description
Offset + 8	11 IINS	Insert IP header checksum. This bit is written by the user. If set, the MAC's IP accelerator calculates the IP header checksum and overwrites the corresponding header field with the calculated value. The checksum field must be cleared by the application generating the frame. The uDMA does not update this value. This bit is valid regardless of the L bit and must be the same for all EBD for a given frame.
Offset + 8	10–0	Reserved, must be cleared.
Offset + A	15 TXE	Transmit error occurred. This bit is written by the uDMA. This bit indicates that there was a transmit error of some sort reported with the frame. Effectively this bit is an OR of the other error bits including UE, EE, FE, LCE, OE, and TSE. This bit is only valid when the L bit is set.
Offset + A	14	Reserved, must be cleared.
Offset + A	13 UE	Underflow error. This bit is written by the uDMA. This bit indicates that the MAC reported an underflow error on transmit. This bit is only valid when the L bit is set.
Offset + A	12 EE	Excess Collision error. This bit is written by the uDMA. This bit indicates that the MAC reported an excess collision error on transmit. This bit is only valid when the L bit is set.
Offset + A	11 FE	Frame with error. This bit is written by the uDMA. This bit indicates that the MAC reported that the uDMA reported an error when providing the packet. This bit is only valid when the L bit is set.
Offset + A	10 LCE	Late collision error. This bit is written by the uDMA. This bit indicates that the MAC reported that there was a Late Collision on transmit. This bit is only valid when the L bit is set.
Offset + A	9 OE	Overflow error. This bit is written by the uDMA. This bit indicates that the MAC reported that there was a FIFO overflow condition on transmit. This bit is only valid when the L bit is set.
Offset + A	8 TSE	Timestamp error. This bit is written by the uDMA. This bit indicates that the MAC reported a different frame type then a timestamp frame. This bit is only valid when the L bit is set.
Offset + A	7–0	Reserved, must be cleared.
Offset + C	15–0	Reserved, must be cleared.
Offset + E	15–0	Reserved, must be cleared.
Offset + 10	15 BDU	Last buffer descriptor update done. Indicates that the last BD data has been updated by uDMA. This bit is written by the user (=0) and uDMA (=1).
Offset + 10	14–0	Reserved, must be cleared.
Offset + 12	15–0	Reserved, must be cleared.
Offset + 14	15–0	This value is written by the uDMA. It is only valid if the L bit is set.
Offset + 16	1588 timestamp	
Offset + 18 – Offset + 1E	15–0	Reserved, must be cleared.

### 31.4.15 Client FIFO Application Interface

The FIFO interface is completely asynchronous from the Ethernet line, and the transmit and receive interface can operate at a different clock rate.

All transfers to/from the user application are handled independent of the core operation, and the core provides a simple interface to user applications based on a two-signal handshake.

### 31.4.15.1 Data Structure Description

The data structure defined in the following tables for the FIFO interface must be respected to ensure proper data transmission on the Ethernet line. Byte 0 is sent to and received from the line first.

**Table 31-93. FIFO Interface Data Structure**

	63	56	55	48	47	40	39	32	31	24	23	16	15	8	7	0
Word 0	Byte 7	Byte 6	Byte 5	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0								
Word 1	Byte 15	Byte 14	Byte 13	Byte 12	Byte 11	Byte 10	Byte 9	Byte 8								
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

The size of a frame on the FIFO interface may not be a modulo of 64-bit. The user application may not care about the Ethernet frame formats in full detail. It needs to provide and receive an Ethernet frame with the following structure:

- Ethernet MAC destination address
- Ethernet MAC source address
- Optional 802.1q VLAN Tag (VLAN type and info field)
- Ethernet length/type field
- Payload

Frames on the FIFO interface do not contain preamble and SFD fields, which are inserted and discarded by the MAC on transmit and receive, respectively.

- On receive, CRC and frame padding can be stripped or passed through transparently.
- On transmit, padding and CRC can be provided by the user application, or appended automatically by the MAC independent for each frame. No size restrictions apply.

#### NOTE

On transmit, if ENETn\_TCR[ADDINS] is set, bytes 6–11 of each frame can be set to any value, since the MAC overwrites the bytes with the MAC address programmed in the ENETn\_PAUR and ENETn\_PALR registers.

**Table 31-94. FIFO Interface Frame Format**

Byte Number	Field
0–5	Destination MAC address
6–11	Source MAC address
12–13	Length/type field
14–N	Payload data

VLAN-tagged frames are also supported on both transmit and receive and implement additional information (VLAN type and info).

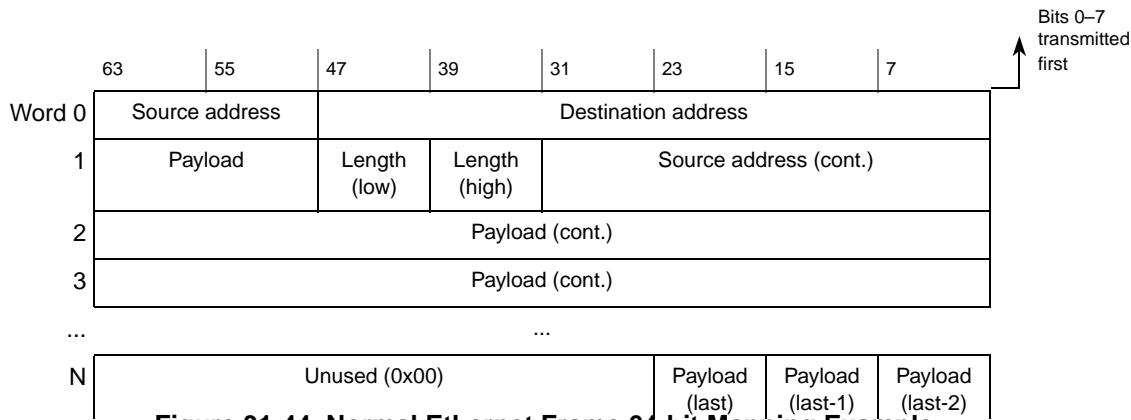
**Table 31-95. FIFO Interface VLAN Frame Format**

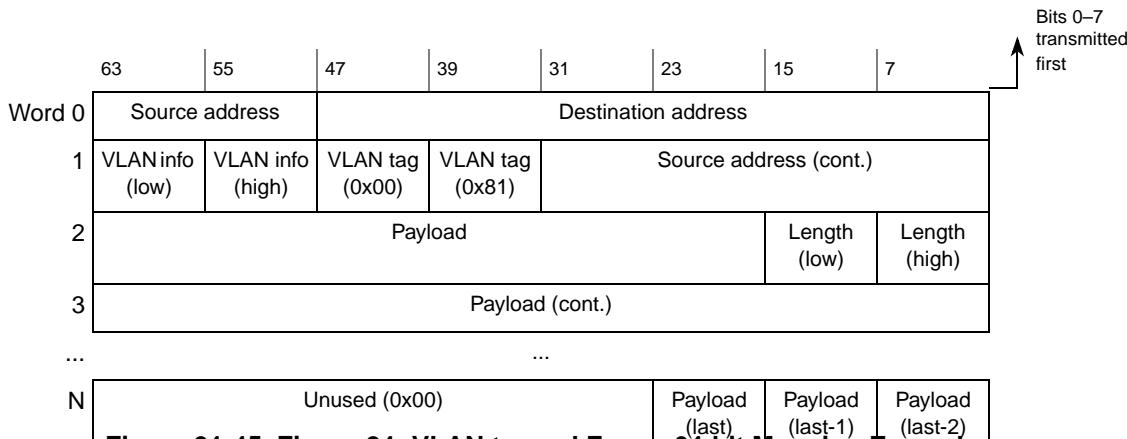
Byte Number	Field
0–5	Destination MAC address
6–11	Source MAC address
12–15	VLAN tag and info
16–17	Length/type field
18–N	Payload data

#### NOTE

The standard defines that the LSB of the MAC address is sent/received first, while for all the other header fields (i.e. length/type, VLAN tag, VLAN info and pause quanta), the MSB is sent/received first.

#### 31.4.15.2 Data Structure Examples



**Figure 31-45. Figure 24: VLAN tagged Frame 64-bit Mapping Example**

If CRC forwarding is enabled ( $\text{CRCFWD} = 0$ ), the last four valid octets of the frame contain the FCS field. The non-significant bytes of the last word can have any value.

### 31.4.15.3 Frame Status

A MAC layer status word and an accelerator status word is available in the receive buffer descriptor. See [Section 31.4.14, “Enhanced Buffer Descriptors”](#) for details. The status is available with each frame with the last data of the frame.

If the frame status contains a MAC layer error (e.g., CRC or length error), RxBD[ME] is also set with the last data of the frame.

## 31.4.16 FIFO Protection

The following sections describe the FIFO protection mechanisms.

### 31.4.16.1 Transmit FIFO Underflow

During a frame transfer, when the transmit FIFO reaches the almost empty threshold with no end-of-frame indication stored in the FIFO, the MAC logic:

- Stops reading data from the FIFO
- Asserts the MII error signal (MII\_TXER) (1) to indicate that the fragment already transferred is not valid
- Deasserts the MII transmit enable signal (MII\_TXEN) to terminate the frame transfer (2)

After an underflow, when the application completes the frame transfer (3), the MAC transmit logic discards any new data available in the FIFO until the end of packet is reached (4) and sets TxBD[UE].

The MAC starts to transfer data on the MII interface when the application sends a new frame with a start of frame indication (5).

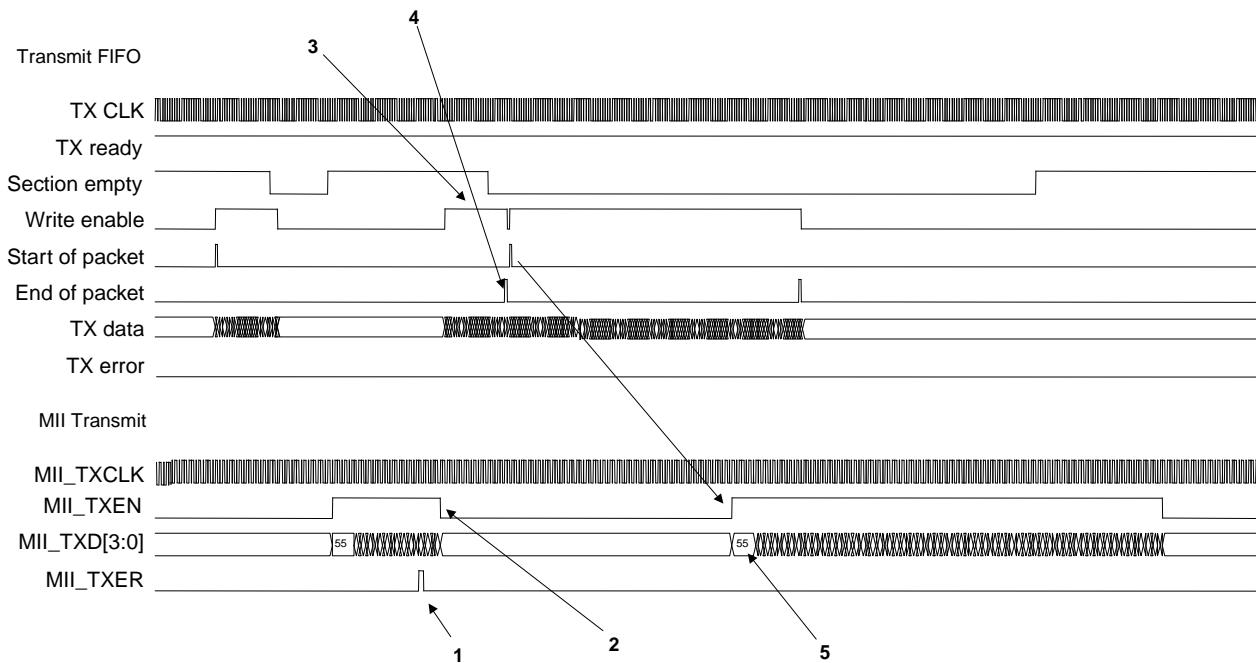


Figure 31-46. Transmit FIFO Underrun Protection

### 31.4.16.2 Transmit FIFO Overflow

On the transmit path, when the FIFO reaches the programmable almost full threshold, the MAC ready signal is deasserted. The application should stop sending new data. However, if the application keeps sending data, the transmit FIFO overflows, corrupting previously-stored contents. The core logic sets TxBD[OE] for the next frame transmitted to indicate this overflow occurrence.

#### NOTE

Overflow is a fatal error and must be addressed by resetting the core or clearing ENETn\_ECR[ETHER\_EN] to clear the FIFOs and prepare for normal operation again.

### 31.4.16.3 Receive FIFO Overflow

During a frame reception, if the client application is not able to receive data (1), the MAC receive control truncates the incoming frame, when the FIFO reaches the programmable almost full threshold to avoid an overflow. The frame is subsequently received on the FIFO interface with an error indication (RxBD[ME] set together with receive end-of-packet) (2) with the truncation error status bit set (3).

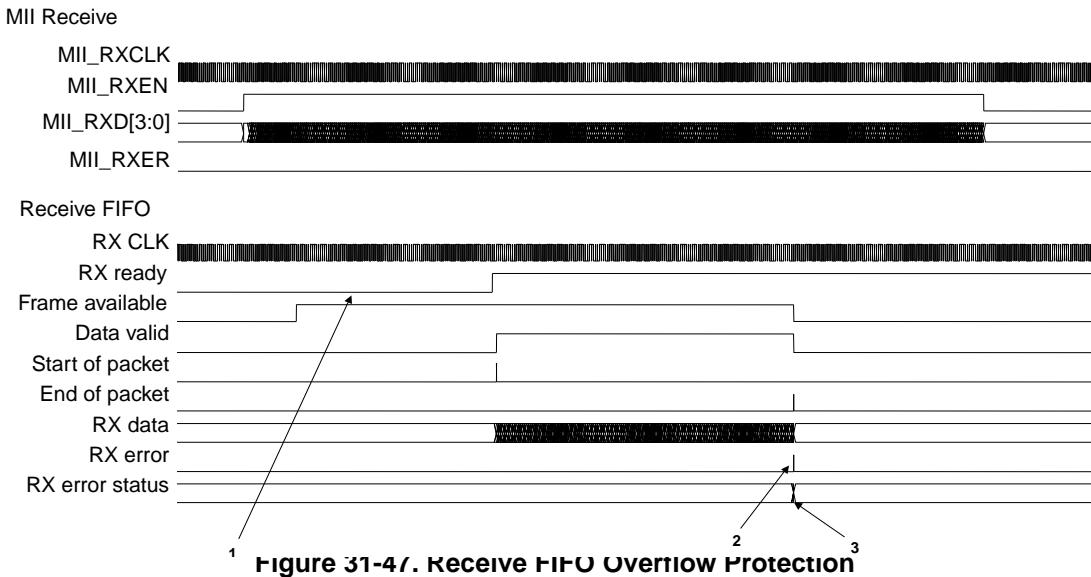


Figure 31-47. Receive FIFO Overflow Protection

### 31.4.17 PHY Management Interface

The MDIO interface is a two-wire management interface. The MDIO management interface implements a standardized method to access the PHY device management registers. The core implements a master MDIO interface, which can be connected to up to 32 PHY devices.

#### 31.4.17.1 MDIO Frame Format

The core MDIO master controller communicates with the slave (PHY device) using frames that are defined in the following table. A complete frame has a length of 64 bits (optional 32-bit preamble, 14-bit command, 2-bit bus direction change, 16-bit data). Each bit is transferred on the rising edge of the MDIO clock (MDC signal). The core PHY management interface supports the standard MDIO specification (IEEE803.2 Clause 22).

Table 31-96. MDIO Frame Formats (Read/Write)

Type	Command					TA	Data		Idle
	PRE	ST	OP	Addr1	Addr2		MSB	LSB	
Read	1...1	01	10	xxxxx	xxxxx	Z0	xxxxxxxxxxxxxx	Z	
Write	1...1	01	01	xxxxx	xxxxx	10	xxxxxxxxxxxxxx	Z	

**Table 31-97. MDIO Frame Field Descriptions**

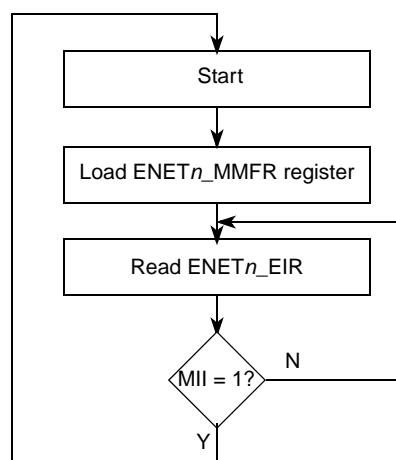
Field	Description
PRE	Preamble. 32 bits of logical ones sent prior to every transaction when ENETn_MSCR[DIS_PRE] is cleared. If DIS_PRE is set, the preamble is not generated.
ST	Start indication, programmed with ENETn_MMFR[ST] <ul style="list-style-type: none"> <li>• Standard MDIO (Clause 22): 01</li> </ul>
OP	Opcode defines if a read or write operation is performed, programmed with ENETn_MMFR[OP]. <ul style="list-style-type: none"> <li>01 Write operation</li> <li>10 Read operation</li> </ul>
Addr1	The PHY device address, programmed with ENETn_MMFR[PA]. Up to 32 devices can be addressed.
Addr2	Register address, programmed with ENETn_MMFR[RA]. Each PHY can implement up to 32 registers.
TA	Turnaround time, programmed with ENETn_MMFR[TA]. Two bit-times are reserved for read operations to switch the data bus from write to read for read operations. The PHY device presents its register contents in the data phase and drives the bus from the second bit of the turnaround phase.
Data	16 bits of data, set to ENETn_MMFR[DATA], written to or read from the PHY
Idle	Between frames the MDIO data signal is tri-stated.

### 31.4.17.2 MDIO Clock Generation

The MDC clock is generated from the internal bus clock divided by the value programmed in ENETn\_MSCR[MII\_SPEED].

### 31.4.17.3 MDIO Operation

To perform a MDIO access, set the MDIO command register (ENETn\_MMFR) according to the description provided in [Section 31.3.6, “MII Management Frame Register \(ENETn\\_MMFR\)”](#). To check when the programmed access completes, read the ENETn\_EIR[MII] bit.

**Figure 31-48. MDIO Access Overview**

### 31.4.18 MII Interface

#### 31.4.18.1 MII Interface — Transmit

On transmit, all data transfers are synchronous to MII\_TXCLK rising edge. The MII data enable signal MII\_TXEN is asserted to indicate the start of a new frame and remains asserted until the last byte of the frame is present on the MII\_TxD[3:0] bus. Between frames, MII\_TXEN remains deasserted.

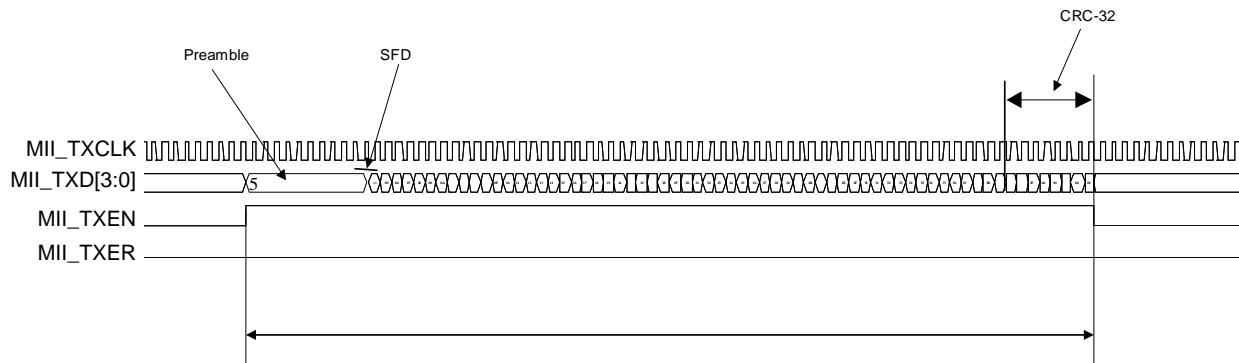


Figure 31-49. MII Transmit Operation

If a frame is received on the FIFO interface with an error (e.g., RxBD[ME] set) the frame is subsequently transmitted with the MII\_TXER error signal for one clock cycle at any time during the packet transfer.

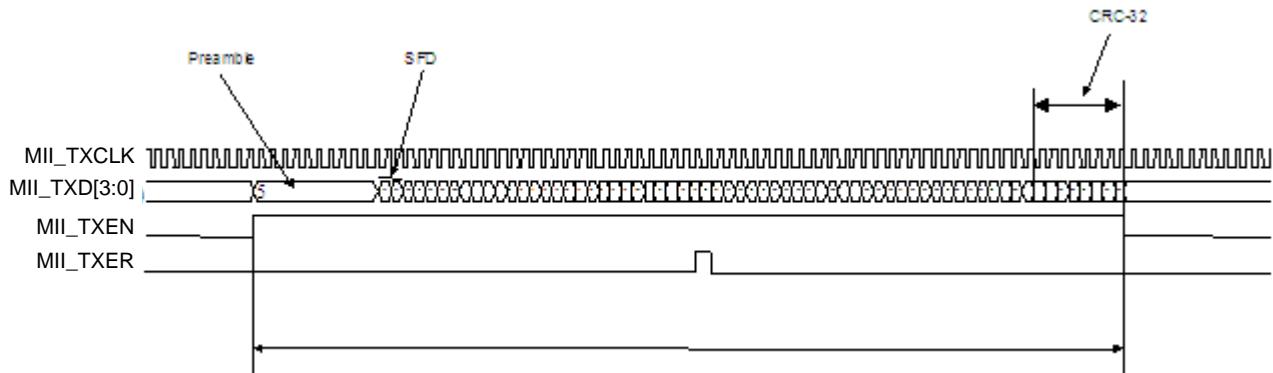


Figure 31-50. MII Transmit Operation — Errored Frame

#### 31.4.18.1.1 Transmit with Collision — Half Duplex

When a collision is detected during a frame transmission (MII\_COL asserted), the MAC stops the current transmission, sends a 32-bit jam pattern, and re-transmits the current frame (see [Section 31.4.4.1, “Collision Detection in Half Duplex Mode”](#) for details).

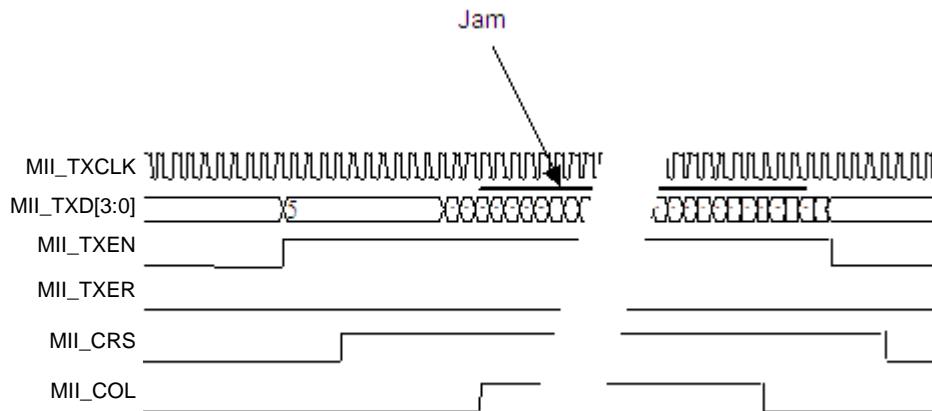


Figure 31-51. MII Transmit Operation — Transmission with Collision

### 31.4.18.2 MII Interface — Receive

On receive all signals are sampled on the MII\_RXCLK rising edge. The MII data enable signal, MII\_RXDV, is asserted by the PHY to indicate the start of a new frame and remains asserted until the last byte of the frame is present on MII\_RXD[3:0] bus. Between frames, MII\_RXDV remains de-asserted.

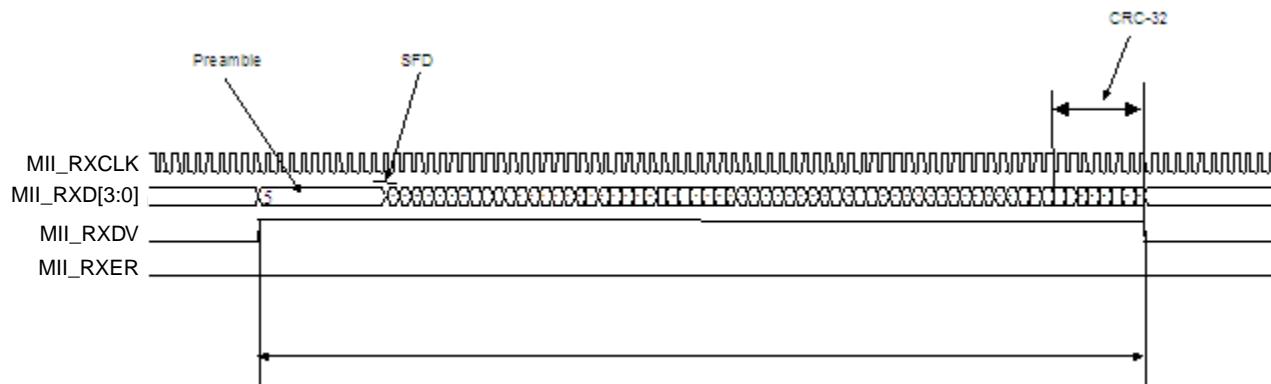


Figure 31-52. MII Receive Operation

If the PHY detects an error on the frame received from the line, the PHY asserts the MII error signal, MII\_RXER, for at least one clock cycle at any time during the packet transfer.

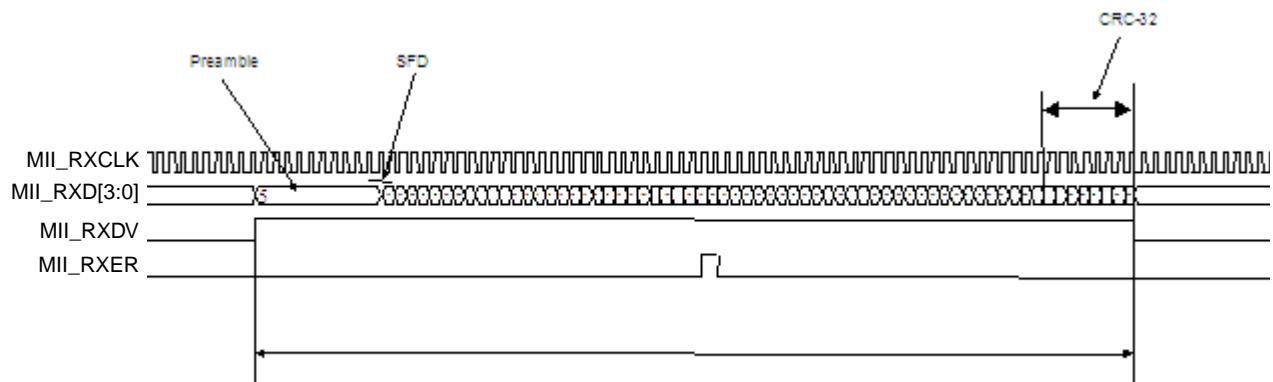


Figure 31-53. MII Receive Operation — Errored Frame

A frame received on the MII interface with a PHY error indication is subsequently transferred on the FIFO interface with RxBD[ME] set.

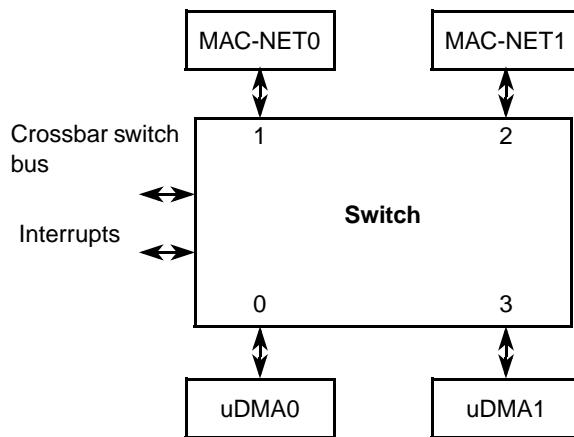


# Chapter 32

## Ethernet Switch

### 32.1 Introduction

The switch core is seamlessly connected to the MAC-NET core and DMA controllers. For control and configuration, the switch implements a register interface and multiple maskable interrupts.



The switch port assignment is listed in [Table 32-1](#).

**Table 32-1. Port Assignment**

Switch Port	Assignment
0	DMA 0
1	MAC-NET 0
2	MAC-NET 1
3 (bypass port)	DMA 1

### 32.1.1 Block Diagram

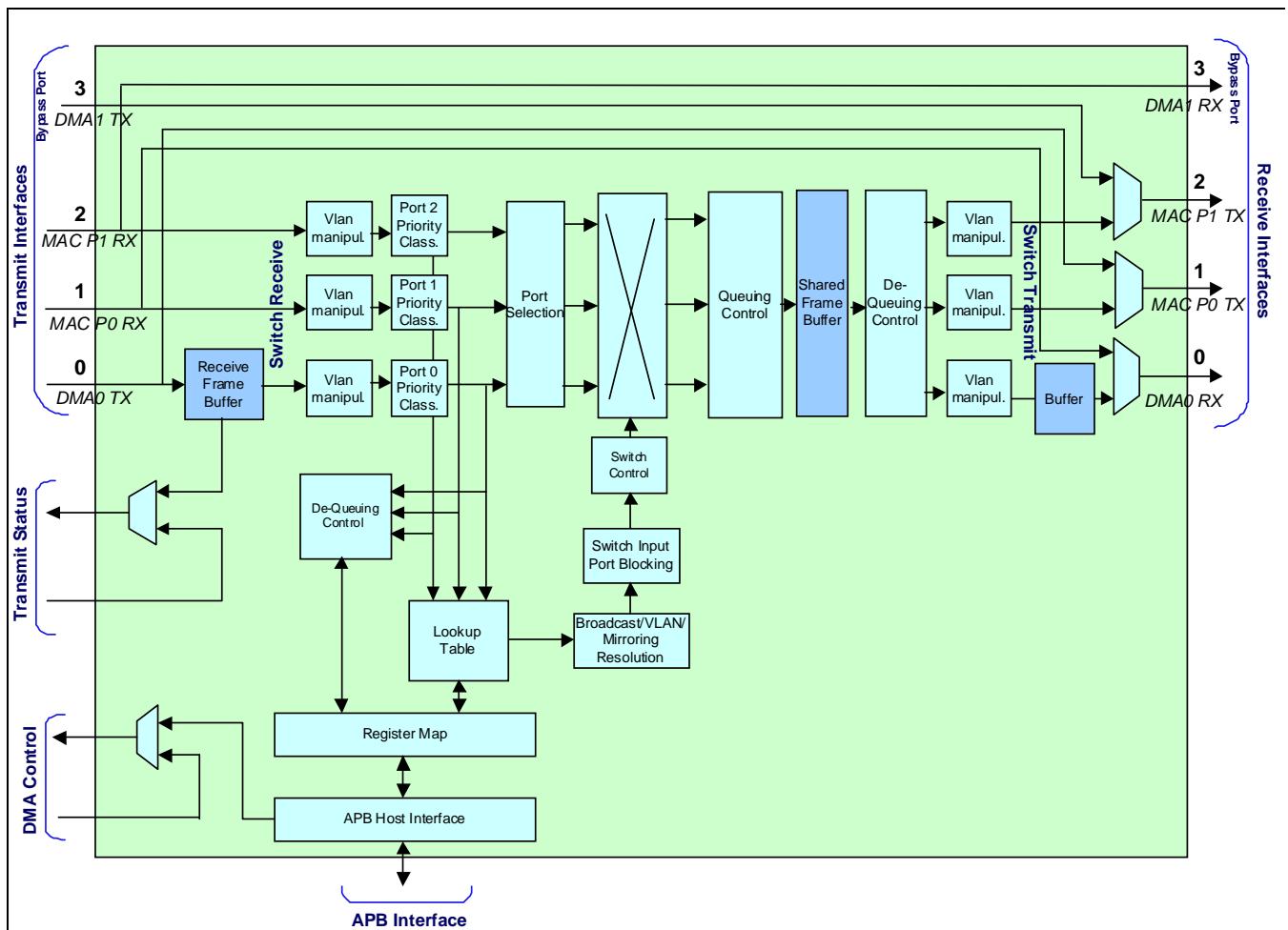


Figure 32-1. Block Diagram

#### NOTE

- The left side is termed “Transmit Interfaces” as the interfaces are driven by the DMA transmit interfaces in pass-through mode. If the switch is enabled, these interfaces connect to the switch internal receive interfaces. The right side is named “Receive Interfaces” as the interfaces represent (are driven from) the respective MAC receive interfaces in pass-through mode connected to the DMA RX. If the switch is enabled these are driven by the switch internal transmit interfaces. See [Section 32.1, “Introduction”](#) for a description of the operational modes.
- All descriptions related to switch functions refer to the switch internal receive/transmit interfaces.
- Receive Frame buffer (DMA0 TX interface). Can hold at least one complete frame transferred from DMA0 to the switch. This is a local 64-bit wide FIFO to absorb a burst from the DMA, provide the necessary transaction handshake to the DMA, and forward the frame then to the switch logic.

- Decoupling buffer only (DMA0 RX interface) to absorb switch latency (e.g. 16/32 words) resulting from a rdy deassertion from DMA0.
- The APB host interface module implements an indirect addressing scheme to the internal registers to allow arbitrary length access cycles (e.g. address table read/write and configuration register read/write). However the DMA control registers are directly mapped and accessed through APB.

### 32.1.2 Features

- Integrated Ethernet switch engine compatible with 10/100 MAC-NET core
- Three port switch with a fourth DMA bypass port
- Can be configured to operate as a 3-port switch (switch mode) or as two independent ports (passthrough mode)
- Filters and forward traffic at wire-speed on all ports
- Per-queue tail-drop congestion management
- Implements hardware switching look-up mechanism providing a learning capacity of up to 2K MAC addresses
- Supports configurable VLAN switching when MAC address lookup should be omitted
- Classification and priority assignment based on port number, MAC address, IPv4 DiffServ code point field, IPv6 Class of Service and VLAN Priority (IEEE802.1q)
- Efficient output queue frame buffering with shared Frame buffer of 24 Kbyte
- Each port implements four priority queues with configurable weighted round-robin selection
- Support Ethernet multicast and broadcast with flooding control to avoid unnecessary duplication of frames
- Programmable multicast destination port mask to restrict frame duplication for individual multicast addresses
- Multicast and broadcast resolution with VLAN domain filtering providing a strict separation of up to 32 VLANs
- IP snooping with programmable protocol and port number registers
- Programmable ingress and egress VLAN tag addition, removal and manipulation supporting single and double-tagged VLAN frames
- Event and status signals which can monitor port activity, severe error conditions, or any user-specific event
- Programmable firmware operation with static or dynamic (learning, aging) switching tables
- Switch firmware source available to provide customer-specific software development capability
- Support for IEEE 1588 precise time-stamping applications
- Supports aggregation and redundant backplane applications

## 32.2 Modes of Operation

The switch, controlled with the configuration pin sx\_ena, can be programmed to operate in two modes:

- Passthrough mode — The switch logic is disabled and bypassed

- Switch mode — The switch logic is enabled

### 32.2.1 Passthrough Mode

When the sx\_ena configuration signal is negated, the switch logic is bypassed and can be totally powered down and disabled with, for example, the switch clocks clk and pclk stopped and the switch reset signals reset\_clk and reset\_pclk asserted.

The switch APB interface and interrupt signals are disabled and should not be used. To control the frame transfer from DMA0 and DMA1, the MAC-NET 0 and the MAC-NET 1 APB interfaces and interrupt signals should be used.

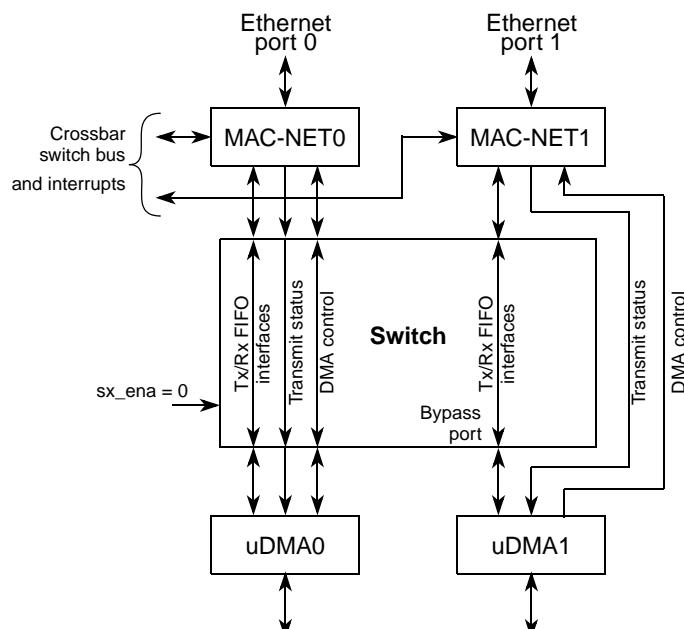


Figure 32-2. Passthrough Mode Configuration Overview

### 32.2.2 Switch Mode

When the switch is programmed to operate in switch mode (sx\_ena set to 1), the bypass mode (port 1) interface is disabled and should not be used.

Frame transfers to and from the line are performed on port 0 only (DMA 0). The transmit status signals are generated from the switch port 0 receive buffer and the DMA control signals from the switch register space. The MAC-NET 0 and MAC-NET 1 transmit status and DMA control signals are not used.

The MAC-NET 0 and MAC-NET 1 APB interfaces and interrupts are enabled and can monitor the line activity and gather the line statistic information.

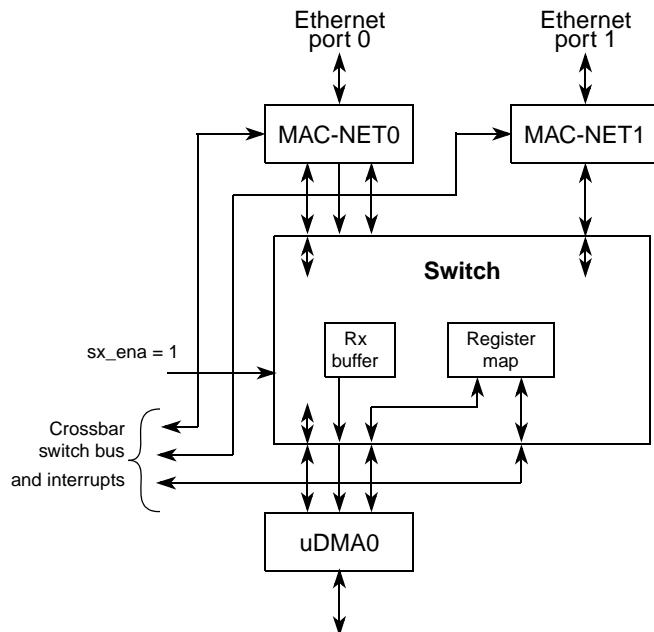


Figure 32-3. Switch Mode Configuration Overview

### 32.2.2.1 Port 0 Input Buffer

A dedicated input buffer of at least 2 KB storage is implemented at the port 0 (DMA 0) input interface, when the switch is operating in switch mode. In bypass mode, this buffer is bypassed.

The input buffer is normally operated in store-and-forward mode, absorbing a DMA burst and forwarding the frame to the switch (internally) when the complete frame is stored in the input buffer.

### 32.2.2.2 Port 0 Input Backpressure/Congestion Indication

When frames are transferred from DMA0 to the switch's port 0 transmit interface, the input buffer indicates when data is to be written to the port 0 input. When the buffer reaches an almost full threshold, it indicates a stop request to DMA0. DMA0 may write a few more words (typically up to four), and then stops writing.

In addition, a special backpressure mechanism for port 0 is included to pause DMA0 transfers when the output queues' shared memory (see [Section 32.4.10, “Output Frame Queuing”](#)) becomes full to a programmable threshold. Respecting the output queues' shared memory fill level can avoid the switch (due to memory congestion) discarding frames written by the DMA0. The threshold is configured through ESW\_P0BCT.

If the shared memory has less than ESW\_P0BCT number of free cells available, the switch stops serving port 0. That is, the switch does not start to read a frame from the port 0 input buffer. The port 0 input buffer continues to accept data from DMA0 until it becomes full.

**NOTE**

The backpressure only considers the total amount of memory available, not a specific queue. Hence, it may still happen that an outgoing frame from DMA0 is discarded by the switch, if the output queue for the frame is congested while the total amount of memory has free space available.

The backpressure threshold (ESW\_P0BCT) should be set higher than the memory full threshold (ESW\_LMT) to stop the DMA0 before a memory full situation.

### 32.3 Memory Map/Register Definition

**Table 32-2. Switch Memory Map**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
<b>Ethernet Switch Configuration and Control</b>					
0xFC0D_C000	Revision (ESW_REV)	32	R	See section	<a href="#">32.3.1/32-9</a>
0xFC0D_C004	Scratch register (ESW_SCR)	32	R/W	0x0000_0000	<a href="#">32.3.2/32-9</a>
0xFC0D_C008	Port enable register (ESW_PER)	32	R/W	0x0000_0000	<a href="#">32.3.3/32-9</a>
0xFC0D_C010	VLAN verify (ESW_VLANV)	32	R/W	0x0000_0000	<a href="#">32.3.4/32-10</a>
0xFC0D_C014	Default broadcast resolution (ESW_DBCR)	32	R/W	0x0000_0000	<a href="#">32.3.5/32-10</a>
0xFC0D_C018	Default multicast resolution (ESW_DMCR)	32	R/W	0x0000_0000	<a href="#">32.3.6/32-11</a>
0xFC0D_C01C	Blocking and learning enable (ESW_BKLR)	32	R/W	0x0000_0000	<a href="#">32.3.7/32-11</a>
0xFC0D_C020	Bridge management port configuration (ESW_BMPC)	32	R/W	0x0000_0000	<a href="#">32.3.8/32-12</a>
0xFC0D_C024	Mode configuration (ESW_MODE)	32	R/W	0x0000_0000	<a href="#">32.3.9/32-13</a>
0xFC0D_C028	VLAN input manipulation select (ESW_VIMSEL)	32	R/W	0x0000_0000	<a href="#">32.3.10/32-14</a>
0xFC0D_C02C	VLAN output manipulation select (ESW_VOMSEL)	32	R/W	0x0000_0000	<a href="#">32.3.11/32-14</a>
0xFC0D_C030	VLAN input manipulation enable (ESW_VIMEN)	32	R/W	0x0000_0000	<a href="#">32.3.12/32-15</a>
0xFC0D_C034	VLAN tag ID (ESW_VID)	32	R/W	0x0000_8100	<a href="#">32.3.13/32-15</a>
<b>Mirroring Control</b>					
0xFC0D_C040	Mirror control register (ESW_MCR)	32	R/W	0x0000_0000	<a href="#">32.3.14/32-16</a>
0xFC0D_C044	Egress port definitions (ESW_EGMAP)	32	R/W	0x0000_0000	<a href="#">32.3.15/32-17</a>
0xFC0D_C048	Ingress port definitions (ESW_INGMAP)	32	R/W	0x0000_0000	<a href="#">32.3.16/32-17</a>
0xFC0D_C04C	Ingress source MAC address low (ESW_INGSAL)	32	R/W	0x0000_0000	<a href="#">32.3.17/32-18</a>
0xFC0D_C050	Ingress source MAC address high (ESW_INGSAH)	32	R/W	0x0000_0000	<a href="#">32.3.17/32-18</a>
0xFC0D_C054	Ingress destination MAC address low (ESW_INGDAL)	32	R/W	0x0000_0000	<a href="#">32.3.17/32-18</a>
0xFC0D_C058	Ingress destination MAC address high (ESW_INGDAH)	32	R/W	0x0000_0000	<a href="#">32.3.17/32-18</a>
0xFC0D_C05C	Egress source MAC address low (ESW_EGSAL)	32	R/W	0x0000_0000	<a href="#">32.3.17/32-18</a>

**Table 32-2. Switch Memory Map (continued)**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0D_C060	Egress source MAC address high (ESW_EGSAH)	32	R/W	0x0000_0000	<a href="#">32.3.17/32-18</a>
0xFC0D_C064	Egress destination MAC address low (ESW_EGDAL)	32	R/W	0x0000_0000	<a href="#">32.3.17/32-18</a>
0xFC0D_C068	Egress destination MAC address high (ESW_EGDAH)	32	R/W	0x0000_0000	<a href="#">32.3.17/32-18</a>
0xFC0D_C06C	Mirror count value (ESW_MINVAL)	32	R/W	0x0000_0000	<a href="#">32.3.18/32-18</a>
<b>Output Queue Memory Manager Status and Configuration</b>					
0xFC0D_C080	Memory manager status (ESW_MMSR)	32	R/W	0x0060_00zA (z=0 or 4)	<a href="#">32.3.19/32-19</a>
0xFC0D_C084	Low memory threshold (ESW_LMT)	32	R/W	0x0000_0009	<a href="#">32.3.20/32-20</a>
0xFC0D_C088	Lowest number of free cells (ESW_LFC)	32	R/W	0x0000_0000	<a href="#">32.3.21/32-21</a>
0xFC0D_C08C	Port congestion status (ESW_PCSR)	32	R	Undefined	<a href="#">32.3.22/32-21</a>
0xFC0D_C090	Switch input and output interface status (ESW_IOSR)	32	R	Undefined	<a href="#">32.3.23/32-21</a>
0xFC0D_C094	Queue weights (ESW_QWT)	32	R/W	0x0000_0000	<a href="#">32.3.24/32-22</a>
0xFC0D_C09C	Port 0 Backpressure Congestion Threshold (ESW_P0BCT)	32	R/W	0x0000_0009	<a href="#">32.3.25/32-23</a>
<b>Forced Forwarding for Port 0</b>					
0xFC0D_C0BC	Port 0 forced forwarding enable (ESW_FFEN)	32	R/W	0x0000_0000	<a href="#">32.3.26/32-23</a>
<b>TCP/UDP Port Snooping</b>					
0xFC0D_C0C0 – 0xFC0D_C0DC	Port snooping registers (ESW_PSNP1–8)	32	R/W	0x0000_0000	<a href="#">32.3.27/32-24</a>
<b>IP Snooping</b>					
0xFC0D_C0E0 – 0xFC0D_C0FC	IP snooping registers (ESW_IPSNP1–8)	32	R/W	0x0000_0000	<a href="#">32.3.28/32-25</a>
<b>Port Configurations</b>					
0xFC0D_C100 – 0xFC0D_C108	Port 0–2 VLAN priority resolution map (ESW_PnVRES)	32	R/W	0x0000_0000	<a href="#">32.3.29/32-25</a>
0xFC0D_C140	IPv4/v6 priority resolution table (ESW_IPRES)	32	R/W	0x0000_0000	<a href="#">32.3.30/32-26</a>
0xFC0D_C180 – 0xFC0D_C188	Port <i>n</i> priority resolution configuration (ESW_PnRES)	32	R/W	0x0000_0000	<a href="#">32.3.31/32-27</a>
0xFC0D_C200 – 0xFC0D_C208	Port <i>n</i> VLAN ID (ESW_PnID)	32	R/W	0x0000_0000	<a href="#">32.3.32/32-28</a>

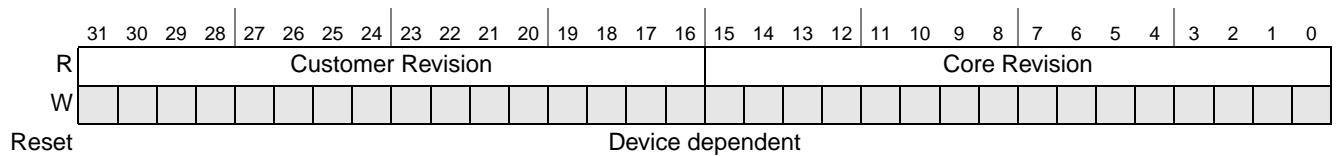
**Table 32-2. Switch Memory Map (continued)**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0D_C280 — 0xFC0D_C2FC	VLAN domain resolution entry 0–31 (ESW_VRES0–31)	32	R/W	0x0000_0000	<a href="#">32.3.33/32-28</a>
<b>Statistics</b>					
0xFC0D_C300	Number of discarded frames (ESW_DISCN)	32	R	0x0000_0000	<a href="#">32.3.34/32-29</a>
0xFC0D_C304	Bytes of discarded frames (ESW_DISCB)	32	R	0x0000_0000	<a href="#">32.3.34/32-29</a>
0xFC0D_C308	Number of non-discard frames (ESW_NDISCN)	32	R	0x0000_0000	<a href="#">32.3.34/32-29</a>
0xFC0D_C30C	Bytes of non-discard frames (ESW_NDISCB)	32	R	0x0000_0000	<a href="#">32.3.34/32-29</a>
<b>Port Statistics</b>					
0xFC0D_C310 + $n \times 0x10$	Port $n$ output queue congestion (ESW_PnOQC)	32	R	0x0000_0000	<a href="#">32.3.35/32-29</a>
0xFC0D_C314 + $n \times 0x10$	Port $n$ mismatching VLAN ID (ESW_PnMVID)	32	R	0x0000_0000	<a href="#">32.3.35/32-29</a>
0xFC0D_C318 + $n \times 0x10$	Port $n$ missing VLAN tag (ESW_PnMVTAG)	32	R	0x0000_0000	<a href="#">32.3.35/32-29</a>
0xFC0D_C31C + $n \times 0x10$	Port $n$ blocked (ESW_PnBL)	32	R	0x0000_0000	<a href="#">32.3.35/32-29</a>
<b>Interrupts and DMA Control</b>					
0xFC0D_C400	Interrupt status register (ESW_ISR)	32	R/W	0x0000_0000	<a href="#">32.3.36/32-29</a>
0xFC0D_C404	Interrupt mask register (ESW_IMR)	32	R/W	0x0000_0000	<a href="#">32.3.37/32-31</a>
0xFC0D_C408	Receive descriptor ring pointer (ESW_RDSR)	32	R/W	0x0000_0000	<a href="#">32.3.38/32-31</a>
0xFC0D_C40C	Transmit descriptor ring pointer (ESW_TDSR)	32	R/W	0x0000_0000	<a href="#">32.3.39/32-32</a>
0xFC0D_C410	Maximum receive buffer size (ESW_MRBR)	32	R/W	0x0000_0000	<a href="#">32.3.40/32-32</a>
0xFC0D_C414	Receive descriptor active (ESW_RDAR)	32	R/W	0x0000_0000	<a href="#">32.3.41/32-33</a>
0xFC0D_C418	Transmit descriptor active (ESW_TDAR)	32	R/W	0x0000_0000	<a href="#">32.3.42/32-33</a>
<b>Learning Interface</b>					
0xFC0D_C500	Learning records A0 & B1 (ESW_LREC0)	32	R	0x0000_0000	<a href="#">32.3.43/32-34</a>
0xFC0D_C504	Learning record B1 (ESW_LREC1)	32	R	0x0000_0000	<a href="#">32.3.43/32-34</a>
0xFC0D_C508	Learning data available status (ESW_LSR)	32	R	0x0000_0000	<a href="#">32.3.44/32-35</a>
<b>MAC Address Lookup Memory</b>					
0xFC0E_0000 — 0xFC0E_3FFC	MAC address lookup table	32	RW	Undefined	<a href="#">32.3.45/32-35</a>

### **32.3.1 Revision Register (ESW\_REV)**

Address: 0xFC0D\_C000 (ESW\_REV)

Access: User read/write



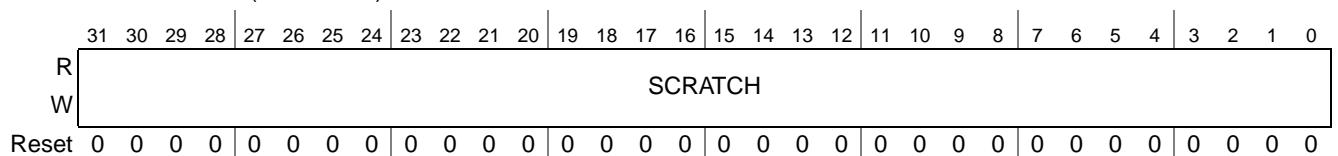
**Figure 32-4. Revision Register (ESW\_REV)**

### **32.3.2 Scratch Register (ESW\_SCR)**

The scratch register provides a memory location to test register access. It returns all data written to it in inverted form.

Address: 0xFC0D\_C004 (ESW\_SCR)

Access: User read/write



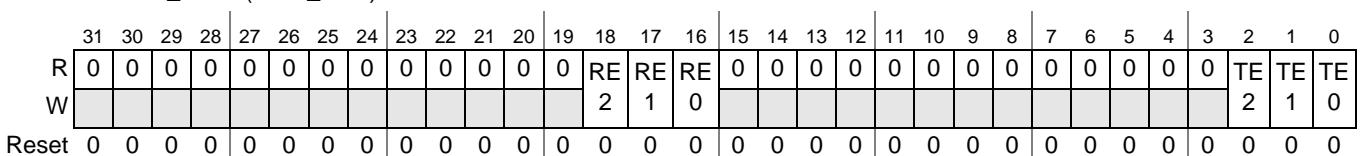
**Figure 32-5. Scratch Register (ESW\_SCR)**

### 32.3.3 Port Enable Register (ESW\_PER)

The port enable register independently enables the transmit and receive direction for each port.

Address: 0xFC0D\_C008 (ESW\_PER)

Access: User read/write



**Figure 32-6. Port Enable Register (ESW PER)**

**Table 32-3. ESW PER Field Descriptions**

Field	Description
31–19	Reserved, must be cleared.
18, 17, 16 REn	Enable receive on port <i>n</i> . 0 Disable. The input is ignored and never selected for frame reception. 1 Enable. The port is selected and a frame is accepted if it indicates data available.
15–3	Reserved, must be cleared.
2, 1, 0 TEn	Enable transmit on port <i>n</i> . 0 Disable. All frames forwarded to the port are discarded. 1 Enable. A frame can be forwarded to the port.

### 32.3.4 VLAN Verify (ESW\_VLANV)

Address: 0xFC0D\_C010 (ESW\_VLANV)

Access: User 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	0	0	0	0	0	0	0	0	0	0	0	0	0	DU	DU	DU	0	0	0	0	0	0	0	0	0	0	0	VV	VV	VV	0	
W														2	1	0											2	1		0		
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	

Figure 32-7. VLAN Verify Register (ESW\_VLANV)

Table 32-4. ESW\_VLANV Field Descriptions

Field	Description
31–19	Reserved, must be cleared.
18, 17, 16 DUn	Discard unknown. 0 Received frames with unknown VLAN IDs are not discarded 1 Received frames with an unknown VLAN ID or no VLAN tag are discarded and not forwarded (i.e. the default bcast is ignored)
15–3	Reserved, must be cleared.
2, 1, 0 VVn	Verify VLAN domain. 0 Frames are routed to the output port without VLAN domain checking 1 A frame is accepted from the port as valid only when the input and output ports are members of the VLAN domain of the frame.

### 32.3.5 Default Broadcast Resolution (ESW\_DBCR)

The default output port list for broadcast/flooding resolution (see [Section 32.4.9.2, “Broadcast/Multicast/VLAN Domain Resolution”](#)).

Address: 0xFC0D\_C014 (ESW\_DBCR)

Access: User 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	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	P2	P1	P0		
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		

Figure 32-8. Default Broadcast Resolution Register (ESW\_DBCR)

Table 32-5. ESW\_DBCR Field Descriptions

Field	Description
31–3	Reserved, must be cleared.
2, 1, 0 Pn	Default broadcast resolution 0 A frame with the corresponding VLAN ID is not switched to that port 1 Indicates that each port is a member of the VLAN and frames with the corresponding VLAN ID can be switched to the port

### 32.3.6 Default Multicast Resolution (ESW\_DMCR)

ESW\_DMCR is used for broadcast/flooding resolution (see [Section 32.4.9.2, “Broadcast/Multicast/VLAN Domain Resolution”](#)).

Address: 0xFC0D\_C018 (ESW\_DMCR)

Access: User 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	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	P2	P1	P0		
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		

Figure 32-9. Default Multicast Resolution Register (ESW\_DMCR)

Table 32-6. ESW\_DMCR Field Descriptions

Field	Description																											
31–3	Reserved, must be cleared.																											
2, 1, 0 Pn	Default multicast resolution. When the received frame carries a multicast address, the default output port list used instead of ESW_DBCR.																											

### 32.3.7 Blocking and Learning Enable (ESW\_BKLR)

ESW\_BKLR independently defines the blocking and learning states for each port.

Address: 0xFC0D\_C01C (ESW\_BKLR)

Access: User 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	0	0	0	0	0	0	0	0	0	0	0	0	0	LD	LD	LD	0	0	0	0	0	0	0	0	0	0	0	BE	BE	BE		
W														2	1	0											2	1	0			
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		

Figure 32-10. Default Blocking and Learning Enable Register (ESW\_BKLR)

Table 32-7. ESW\_BKLR Field Descriptions

Field	Description																											
31–19	Reserved, must be cleared.																											
18, 17, 16 LDn	Disable learning. 0 Enable. 1 Disable. Only bridge protocol data unit frames are learned. Other frames are ignored for learning.																											
15–3	Reserved, must be cleared.																											
2, 1, 0 BEn	Enable blocking. 0 Disable. 1 Enable. Only bridge protocol data units are accepted on that input, all other frames are discarded.																											

### 32.3.8 Bridge Management Port Configuration (ESW\_BMPC)

ESW\_BMPC enables and defines the management port that receives bridge protocol frames.

Address: 0xFC0D_C020 (ESW_BMPC)																Access: User read/write																						
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16																						
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	PORTMASK																					
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																						
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	PRIORITY	0	DIS	EN	MSG TX	0	PORT															
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																						
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																						

Figure 32-11. Bridge Management Port Configuration Register (ESW\_BMPC)

Table 32-8. ESW\_BMPC Field Descriptions

Field	Description
31–19	Reserved, must be cleared.
18–16 PORTMASK	Portmask for transmission of BPDU frames as defined in <a href="#">Section 32.4.9.5.4, “Management Frame Forwarding”</a> . When the management port transmits a BPDU frame to the switch, it is forwarded to all ports in this portmask (bit16=port0, bit17=port1, bit 18=port2).
15–13 PRIORITY	Priority to use for transmitted management frames. Used to e.g. put a management frame in a high-priority output queue for fast delivery.
12–8	Reserved, must be cleared.
7 DIS	If set, BPDU frames are discarded always. Setting has no effect, when the enable bit is set.
6 EN	If set, all BPDU frames are forwarded exclusively to the management port specified in bits 3:0. If cleared, BPDU frames are forwarded as any other frame, or discarded if the discard bit is set.
5 MSGTX	Set (latched) when a BPDU frame as defined in <a href="#">Section 32.4.9.5.4, “Management Frame Forwarding”</a> was transmitted from the management port to any output port. Bit is reset by writing into the register.
4	Reserved, must be cleared.
3–0 PORT	The Port number of the port that should act as a management port. Relevant to all functions that forward frames to the management port (i.e. BPDU processing, snooping). Note: It must be set 0 in this switch configuration (Port 0 to DMA0 is the management port).

### 32.3.9 Mode Configuration Register (ESW\_MODE)

ESW\_MODE defines several global configuration settings.

Address: 0xFC0D_C024 (ESW_MODE)																Access: User read/write									
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	0	0	0	0	0	0	0	0	
	STAT	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
W	RST																								
	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
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	POCT	CRC	STOP	0	0	0	0	0	0	0	0	0	0	0	0	SW	EN	SW	RST
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

Figure 32-12. Mode Configuration Registers (ESW\_MODE)

Table 32-9. ESW\_MODE Field Descriptions

Field	Description
31 STATRST	Reset Statistics Counters Command. When set during a write, all statistics counters are cleared. When set, all other bits are ignored and do not influence the currently stored value in the register (i.e. it is not necessary to read/preserve the register contents prior to writing this bit).
30–10	Reserved, must be cleared.
9 P0CT	Enable Port0 input buffer cut-through mode. When cleared (0, default) the input buffer operates in store&forward mode, which is the recommended mode of operation.
8 CRCTRAN	When enabled (1) the MAC ports are expected to process frames to/from the switch including CRC. Frames from a MAC port to a MAC port will then have their respective ff_rx_crc_fwd1/2 pin (wired to MAC's ff_tx_crc_fwd) asserted, indicating that no CRC should be appended. However, even when enabled, the DMA0 port is not expected to provide frames with crc: Frames from port 0 (DMA0 transmit) are always forwarded with the crc option as defined by the input ff_tx_crc_fwd0, independent of the CRCTRAN setting. This means, if the DMA0 port will never transmit frames with CRC, ff_tx_crc_fwd0 can be wired to permanently 0. Note that the MAC configuration bit RCR(CRC_FWD) must be set to ensure the MAC forwards received frames with CRC to the switch. When disabled (0, default) the output pins ff_rx_crc_fwd1/2 to the MACs are always 0 to ensure a CRC is appended to outgoing frames no matter from which port they were received. DMA0 still can influence the crc append through its ff_tx_crc_fwd0 input pin if required. Note: The ff_tx_crc_fwd0 input to the switch must be valid throughout the complete frame (from sop to eop). This is in contrast to the MAC definition for this signal, which requires validity during eop only.
7 STOP	Controls toplevel output pin stop_en. No internal function.
6–2	Reserved, must be cleared.
1 SWEN	Controls toplevel output pin switch_en. When deasserted (0), all DMA registers are cleared.
0 SWRST	Controls toplevel output pin switch_reset. No internal function.

### 32.3.10 VLAN Input Manipulation Select (ESW\_VIMSEL)

ESW\_VIMSEL defines behavior of the VLAN input manipulation function, if such a function is present on an input port. ESW\_VIMSEL has effect only if enabled by the corresponding port bit in ESW\_VIMEN.

Address: 0xFC0D\_C028 (ESW\_VIMSEL)

Access: User 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	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																												IM	IM	IM		
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		

Figure 32-13. VLAN Input Manipulation Select Register (ESW\_VIMSEL)

Table 32-10. ESW\_VIMSEL Field Descriptions

Field	Description
31–6	Reserved, must be cleared.
5–4, 3–2, 1–0 IM <sub>n</sub>	Input manipulation select for port <i>n</i> . 00 Mode 1, single tag passthrough 01 Mode 2, single tag overwrite 10 Mode 3, double tag passthrough 11 Mode 4, double tag overwrite

### 32.3.11 VLAN Output Manipulation Select (ESW\_VOMSEL)

ESW\_VOMSEL defines behavior of the VLAN output manipulation function, if such a function is present on an output port.

Address: 0xFC0D\_C02C (ESW\_VOMSEL)

Access: User 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	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																												OM	OM	OM		
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		

Figure 32-14. VLAN Output Manipulation Select Register (ESW\_VOMSEL)

Table 32-11. ESW\_VOMSEL Field Descriptions

Field	Description
31–6	Reserved, must be cleared.
5–4, 3–2, 1–0 OM <sub>n</sub>	Output manipulation select for port <i>n</i> . 00 No output manipulation 01 Mode 1, strip mode 10 Mode 2, tag through 11 Mode 3, transparent

### 32.3.12 VLAN input manipulation enable (ESW\_VIMEN)

ESW\_VIMEN enables the input processing according to the ESW\_VIMSEL for a port.

Address: 0xFC0D\_C030 (ESW\_VIMSEL)

Access: User 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	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	EN	EN	EN	0	
W																											2	1	0			
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	

Figure 32-15. VLAN Input Manipulation Enable Register (ESW\_VIMEN)

Table 32-12. ESW\_VIMEN Field Descriptions

Field	Description
31–3	Reserved, must be cleared.
2, 1, 0 EN <sub>n</sub>	Input manipulation enable for port <i>n</i> . 0 Disable. ESW_VIMSEL has no effect and the frames are processed unmodified. 1 Enable

### 32.3.13 VLAN Tag ID (ESW\_VID)

The VLAN type field value to expect to identify a VLAN-tagged frame. A valid 802.1Q VLAN-tagged frame must use the value 0x8100.

Address: 0xFC0D\_C034 (ESW\_VID)

Access: User 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																																
W																																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	

Figure 32-16. VLAN Tag ID Register (ESW\_VID)

Table 32-13. ESW\_VID Field Descriptions

Field	Description
31–0 TAG	ID to identify a VLAN-tagged frame. A valid 802.1Q VLAN-tagged frame must use the value 0x8100.

### 32.3.14 Mirror control register (ESW\_MCR)

The mirror control register defines port mirroring and filtering conditions.

Address: 0xFC0D_C040 (ESW_MCR)																Access: User read/write				
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	0	0	0	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	PORT			
W	0	0	0	0	0	EG DA	EG SA	ING DA	ING SA	EG MAP	ING MAP	MEN	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 32-17. Mirror Control Register (ESW\_MCR)

Table 32-14. ESW\_MCR Field Descriptions

Field	Description
31–11	Reserved, must be cleared.
10 EGDA	If set, only frames transmitted on an egress port with a destination address matching the value in ESW_EGDA{L,H} are mirrored. Other frames are not mirrored. <b>Note:</b> If the egress map is not enabled (EGMAP = 0) then any frame with a matching destination address is mirrored.
9 EGSA	If set, only frames transmitted on an egress port with a source address matching the value in ESW_EGSA{L,H} are mirrored. Other frames are not mirrored. <b>Note:</b> If the egress map is not enabled (EGMAP = 0) then any frame with a matching source address is mirrored.
8 INGDA	If set, only frames received on an ingress port with a destination address matching the value in ESW_INGDA{L,H} are mirrored. Other frames are not mirrored. <b>Note:</b> If the ingress map is not enabled (INGMAP = 0) then any frame with a matching destination address is mirrored.
7 INGSA	If set, only frames received on an ingress port with a source address matching the value in ESW_INGS{L,H} are mirrored. Other frames are not mirrored. <b>Note:</b> If the ingress map is not enabled (INGMAP = 0) then any frame with a matching source address is mirrored.
6 EGMAP	Egress map enable. 0 Egress port map has no effect 1 Egress map is enabled. A frame forwarded to an output port that has a bit set in the egress map is mirrored.
5 INGMAP	Ingress map enable. 0 Ingress port map has no effect 1 Ingress map is enabled. A frame received on an ingress port that has a bit set in the ingress map is mirrored.

**Table 32-14. ESW\_MCR Field Descriptions (continued)**

Field	Description
4 MEN	Mirroring enable. 0 Disabled 1 Enabled
3–0 PORT	The port number that should act as the mirror port and receive all mirrored frames.

### 32.3.15 Egress Port Definitions (ESW\_EGMAP)

Address: 0xFC0D\_C044 (ESW\_EGMAP)

Access: User 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	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	EG	EG	EG	0	
W																													2	1	0		
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	0

**Figure 32-18. Egress Port Definitions Register (ESW\_EGMAP)****Table 32-15. ESW\_EGMAP Field Descriptions**

Field	Description
31–3	Reserved, must be cleared.
2, 1, 0 EG <sub>n</sub>	Port mirroring egress for port <i>n</i> . 0 Disable 1 Enabled. Frames destined for this port are mirrored to the mirror port.

### 32.3.16 Ingress Port Definitions (ESW\_INGMAP)

Address: 0xFC0D\_C048 (ESW\_INGMAP)

Access: User 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	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	ING	ING	ING	0	
W																													2	1	0		
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	0

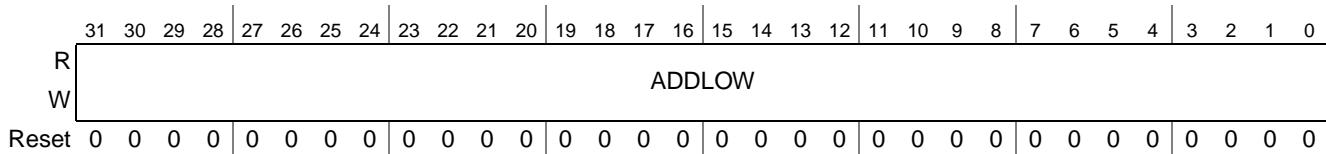
**Figure 32-19. Ingress Port Definitions Register (ESW\_INGMAP)****Table 32-16. ESW\_INGMAP Field Descriptions**

Field	Description
31–3	Reserved, must be cleared.
2, 1, 0 ING <sub>n</sub>	Port mirroring ingress for port <i>n</i> . 0 Disable 1 Enabled. Frames from this port are mirrored to the mirror port.

### 32.3.17 Ingress and Egress MAC Address Registers

Address: 0xFC0D\_C04C (ESW\_INGSAL)  
0xFC0D\_C054 (ESW\_INGDAL)  
0xFC0D\_C05C (ESW\_EGSAL)  
0xFC0D\_C064 (ESW\_EGDAL)

Access: User read/write



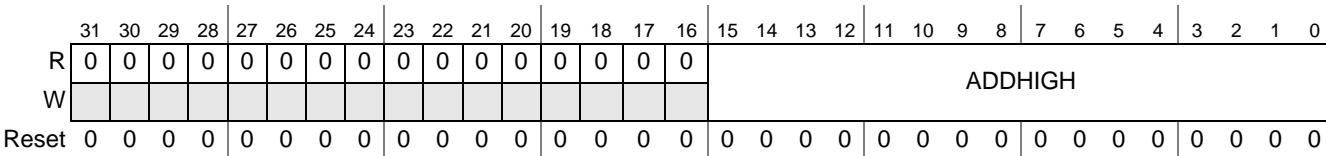
**Figure 32-20. Ingress and Egress MAC Address Low Registers**

**Table 32-17. ESW\_ING{S,D}AL and ESW\_EG{S,D}AL Field Descriptions**

Field	Description
31-0 ADDLOW	First four bytes of the ingress/egress MAC address for mirror filtering.

Address: 0xFC0D\_C050 (ESW\_INGSAH)  
0xFC0D\_C058 (ESW\_INGDAH)  
0xFC0D\_C060 (ESW\_EGSAH)  
0xFC0D\_C068 (ESW\_EGDAH)

Access: User read/write



**Figure 32-21. Ingress and Egress MAC Address High Register**

**Table 32-18. ESW\_ING{S,D}AH and ESW\_EG{S,D}AH Field Descriptions**

Field	Description
31–16	Reserved, must be cleared.
15–0 ADDHIGH	First two bytes of the ingress/egress MAC address for mirror filtering.

### 32.3.18 Mirror Count Value (ESW\_MINVAL)

## **NOTE**

If the egress filtering port map is active, every forwarded frame is considered. Otherwise, frames are counted only if the mirroring decision indicated that the frame should be mirrored.

Address: 0xFC0D\_C06C (ESW\_MCVAL)

Access: User 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	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																																
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		

Figure 32-22. Mirror Count Value Register (ESW\_MCVAL)

Table 32-19. ESW\_MCVAL Field Descriptions

Field	Description																												
31–8	Reserved, must be cleared.																												
7–0 COUNT	Count value for mirror filtering. Every $n^{\text{th}}$ frame is forwarded to the mirror port if enabled. 0x00 Every frame forwarded 0x01 Every frame forwarded 0x02 Every second frame forwarded ... 0xFF Every 255 <sup>th</sup> frame forwarded																												

### 32.3.19 Memory Manager Status Register (ESW\_MMSR)

All latched bits are cleared upon a write with any content to the register.

Address: 0xFC0D\_C080 (ESW\_MMSR)

Access: User read/write

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

Figure 32-23. Memory Manager Status Register (ESW\_MMSR)

Table 32-20. ESW\_MMSR Field Descriptions

Field	Description																												
31–24	Reserved, must be cleared.																												
23–16 CELLS_AVAIL	Real-time indication of currently available cells in memory.																												
15–7	Reserved, must be cleared.																												
6 DQGRNT	Indication of if currently inputs are de-queued. Should be set always and is cleared when the memory becomes full (see the MEMFULL bit). <b>Note:</b> The bit is cleared upon reset. However, set shortly after when the memory manager completes initialization.																												
5–4	Reserved, must be cleared.																												

**Table 32-20. ESW\_MMSR Field Descriptions (continued)**

Field	Description
3 MFLATCH	Latched version of mem_full. Is kept set even when mem_full is cleared again. The bit is cleared when the register is written.
2 MEMFULL	Current memory full indication. The memory is full when less than the programmed minimum cell threshold is available in memory. This is not an error and the memory controller is working fine. It just indicates that the switch does no longer serve its input ports to avoid memory overrun (no_cell error).
1 NOCELL	Set, when memory has exceeded the maximum available number of cells. The event is latched and the bit stays set if the event is no longer active. This is a fatal error and must never happen during operation. The minimum cells threshold must be increased if it happens. The bit is always set after reset (during initialization) and must be cleared when the busy initialization (see bit 0) indication is cleared.  IMPORTANT NOTE: When this bit is set any time during operation (after initialization completed) the switch is in an inoperable state and must be reset completely to restore correct operation. If such an event happens the ESW_LMT setting must be increased during initialization to avoid such situation. The bit is cleared when the register is written.
0 BUSY	When set (1), Memory controller is initializing. The initialization is only preparing the internal data structures within the controller, it does not initialize the shared memory used for frame storage as this is not required. It is asserted after reset and stays set until the memory controller is ready to store frames. The switch must not be enabled before initialization of the memory controller has been completed.

### 32.3.20 Low Memory Threshold (ESW\_LMT)

If the number of cells available in memory is less than ESW\_LMT, the switch discards frames. Choose a value for at least two full-sized frames. A memory overflow due to a too low threshold is a fatal error and may require a device reset.

Address: 0xFC0D\_C084 (ESW\_LMT)

Access: User 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	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																																
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	1	0	0	

**Figure 32-24. Low Memory Threshold Register (ESW\_LMT)****Table 32-21. ESW\_LMT Field Descriptions**

Field	Description
31–8	Reserved, must be cleared.
7–0 THRESH	Low memory threshold. The value of this field is the number of 256 bytes. 0x01 0.25 KB (1 × 256 bytes) 0x02 0.5 KB (2 × 256 bytes) ... 0x09 2.25 KB (9 × 256 bytes) ... <b>Note:</b> Choose a value for at least two full-sized frames.

### 32.3.21 Lowest Number of Free Cells (ESW\_LFC)

ESW\_LFC indicates the lowest number of free cells reached in memory during operation since it was last cleared.

Address: 0xFC0D\_C088 (ESW\_LFC)

Access: User read/write

**Figure 32-25. Low Number of Free Cells Register (ESW\_LFC)**

## Table 32-22. ESW\_LFC Field Descriptions

Field	Description
31-0 COUNT	Lowest number of free cells reached in memory during operation since it was last cleared. This register is reset to the maximum by writing any value to it.

### **32.3.22 Port Congestion Status (ESW\_PCSR)**

Address: 0xFC0D\_C08C (ESW\_PCSR)

Access: User read-only

**Figure 32-26. Port Congestion Status Register (ESW\_PCSR)**

**Table 32-23. ESW\_PCSR Field Descriptions**

Field	Description
31–3	Reserved, must be cleared.
2, 1, 0 PC $n$	Port congestion status for port $n$ . 0 Not congested 1 Congested

### 32.3.23 Input/Output Interface Status Register (ESW IOSR)

Address: 0xFC0D\_C090 (ESW\_IOSR)

Access: User read-only

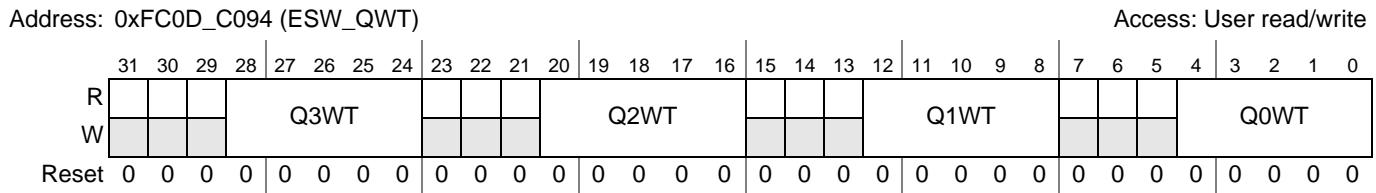
**Figure 32-27. Input/Output Interface Register (ESW IOSR)**

**Table 32-24. ESW\_IOSR Field Descriptions**

Field	Description
31–19	Reserved, must be cleared.
18, 17, 16 IR $n$	Input data available for port $n$ . 0 Not available 1 Data available
15–3	Reserved, must be cleared.
2, 1, 0 OR $n$	Output ready to accept data for port $n$ . 0 Not ready 1 Ready

### 32.3.24 Queue Weights (ESW\_QWT)

ESW\_QWT defines the weight for the corresponding queue for all ports. Setting all weights to zero implements a strict priority scheme.

**Figure 32-28. Queue Weights Register (ESW\_QWT)****Table 32-25. ESW\_QWT Field Descriptions**

Field	Description
31–29	Reserved, must be cleared
28–24 Q3WT	Queue 3 weight. Defines the weight for the corresponding queue for all ports. A higher weight gives more priority to the queue. Queue 3 is the highest priority queue. Valid values are 0–30.
23–21	Reserved, must be cleared
20–16 Q2WT	Queue 2 weight. Defines the weight for the corresponding queue for all ports. A higher weight gives more priority to the queue. Valid values are 0–30.
15–13	Reserved, must be cleared
12–8 Q1WT	Queue 1 weight. Defines the weight for the corresponding queue for all ports. A higher weight gives more priority to the queue. Valid values are 0–30.
7–5	Reserved, must be cleared
4–0 Q0WT	Queue 0 weight. Defines the weight for the corresponding queue for all ports. A higher weight gives more priority to the queue. Valid values are 0–30.

### 32.3.25 Port 0 Backpressure Congestion Threshold (ESW\_P0BCT)

ESW\_P0BCT defines the congestion threshold for port 0 backpressure. If the total output queues' shared memory (see [Section 32.4.10, “Output Frame Queuing”](#)) has less than this amount of free cells available, the switch stops serving the port 0 input buffer. This eventually fills the input buffer.

Address: 0xFC0D\_C09C (ESW\_P0BCT)

Access: User 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	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																																
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	1	0	0	1	

Figure 32-29. Port 0 Backpressure Congestion Threshold Register (ESW\_P0BCT)

Table 32-26. ESW\_P0BCT Field Descriptions

Field	Description
31–8	Reserved, must be cleared.
7–0 THRESH	Defines the congestion threshold for port 0 backpressure. Clearing this field disables the function (no backpressure). <b>Note:</b> Set this field higher than the memory full threshold (ESW_LMT) to stop the DMA0 before a memory full situation.

### 32.3.26 Port 0 Forced Forwarding Enable (ESW\_FFEN)

ESW\_FFEN forces forwarding for port 0 frames (i.e. frames transmitted from the DMA0 to the port 0 of the switch).

Address: 0xFC0D\_C0BC (ESW\_FFEN)

Access: User 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	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																																
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	

Figure 32-30. Forced Forwarding Port 0 Register (ESW\_FFEN)

Table 32-27. ESW\_FFEN Field Descriptions

Field	Description
31–4	Reserved, must be cleared.
3–2 FD	When FEN is set, this field defines if the port 0 frame should be forwarded to the MAC at ports 1 and 2. 00 Do not forward. Frame is processed normally. 01 Forward to port 1 only 10 Forward to port 2 only 11 Forward to both ports <b>Note:</b> It is possible to forward to one or both MAC ports. If neither bit is set, FEN is ignored and the frame is processed normally. This can be used to implement a handshake, as FEN is still reset but no further action occurs.

**Table 32-27. ESW\_FFEN Field Descriptions (continued)**

Field	Description
1	Reserved, must be cleared.
0 FEN	When set, the next frame received from port 0 (the local DMA port) is forwarded to the ports defined in FD. The bit resets to zero automatically when one frame from port 0 has been processed by the switch (i.e. has been read from the port 0 input buffer; see <a href="#">Figure 32-1</a> ). Therefore, the bit must be set again as necessary. See also <a href="#">Section 32.4.8.2, “Forced Forwarding”</a> for a description.

### 32.3.27 Port Snooping Registers (ESW\_PSNP1–8)

ESW\_PSNP $n$  defines the TCP/UDP port number snooping function configuration. There are eight registers available to set independent snooping destinations.

Address:	0xFC0D_C0C0 (ESW_PSNP1) 0xFC0D_C0C4 (ESW_PSNP2) 0xFC0D_C0C8 (ESW_PSNP3) 0xFC0D_C0CC (ESW_PSNP4)	0xFC0D_C0D0 (ESW_PSNP5) 0xFC0D_C0D4 (ESW_PSNP6) 0xFC0D_C0D8 (ESW_PSNP7) 0xFC0D_C0DC (ESW_PSNP8)	Access: User 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 W	P <small>ORT</small> _C <small>O</small> P <small>A</small> R <small>E</small>	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   CS CD MODE EN	
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 32-31. Port Snoop Registers (ESW\_PSNP $n$ )****Table 32-28. ESW\_PSNP $n$  Field Descriptions**

Field	Description
31–16 PORT_COMP ARE	The 16-bit port number to compare within the TCP or UDP header of a frame. Note: it is possible to set both the compare source/dest bits. The result is OR'ed, meaning if any of the fields match the compare value, the frame is processed as defined by the mode bits.
15–5	Reserved, must be cleared.
4 CS	When set, the TCP or UDP source port number field within the frame is compared with the compare value provided in 31:16.
3 CD	When set, the TCP or UDP destination port number field within the frame is compared with the compare value provided in PORT_COMPARE.
2–1 MODE	Defines the forwarding that should occur, when an IP frame is received and the protocol field matches the protocol value. 00 Forward frame to designated management port only 01 Copy to management port and forward normally 10 Discard 11 Reserved <b>Note:</b> The management port is defined in ESW_BMPC register.
0 EN	When set (1) the entry contains valid data and the function is active. If a match with the TCP/UDP destination port value occurs, the frame is processed as defined by the mode setting. All other bits of the register are interpreted by the snooping function only if the enable bit is set. Otherwise the settings are ignored. When written with 0 will also force bits 3,4 to 0. Defaults to 0 upon reset.

### 32.3.28 IP snooping registers (ESW\_IPSNP1–8)

**ESW\_IPSNP $n$**  defines the IP snooping function configuration. There are eight registers available to set independent snooping for different protocols.

**Figure 32-32. IP Snoop Register (ESW\_IPSNPn)**

**Table 32-29. ESW\_IPSNP*n* Field Descriptions**

Field	Description
31–16	Reserved, must be cleared.
15–8 PROTOCOL	The 8-bit protocol value to match with the incoming frame's IP header protocol field.
7–3	Reserved, must be cleared.
2–1 MODE	Defines the forwarding that should occur, when an IP frame is received and the protocol field matches the protocol value (see the PROTOTCOL bits).
	00 Forward frame to designated management port only
	01 Copy to management port and forward normally
	10 Discard
	11 Reserved
	<b>Note:</b> The management port is defined in register ESW_BMPC.
0 EN	When set (1) the entry contains valid data and the function is active. If a match with the protocol value occurs, the frame is processed as defined by the mode setting. All other bits of the register are interpreted by the IP snooping function only if the enable bit is set. Otherwise the settings are ignored. Defaults to 0 upon reset.

### 32.3.29 Port 0–2 VLAN Priority Resolution Map (ESW\_PnVRES)

The ESW\_PnVRES registers implement a 3-bit to 3-bit VLAN priority mapping capability. For each port, one register is provided. The current frame's 3-bit VLAN priority field is used as an index and the

corresponding priority is taken from the respective position of the register giving the final classification for the frame.

Address: 0xFC0D\_C100 (ESW\_P0VRES)  
0xFC0D\_C104 (ESW\_P1VRES)  
0xFC0D\_C108 (ESW\_P2VRES)

Access: User read/write

**Figure 32-33. VLAN Priority Registers (ESW\_PnVRES)**

**Table 32-30. ESW PnVRES Field Descriptions**

Field	Description
31–24	Reserved, must be cleared.
23–0 PRIn	The current frame's 3-bit VLAN priority field is an index to the corresponding PRIn field to give the final priority classification for the frame.

### **32.3.30 IPv4/v6 Priority Resolution Table (ESW\_IPRES)**

Address: 0xFC0D\_C140 (ESW\_IPRES)

Access: User read/write

**Figure 32-34. IP Priority Resolution Register (ESW\_IPRES)**

**Table 32-31. ESW\_IPRES Field Descriptions**

Field	Description
31 READ	Must be cleared to write values in the tables. When set during register writes, the IPv6 select and address bits are stored in the register only and the priority bits are ignored and not written into the addressed table. When the register is read, the priority bits represent the value read from the table always.
30–15	Reserved, must be cleared.
14–13 PRI2	The priority information to write into the addressed table entry. These 2 bits represent the output priority selected when the frame is received on port 2. When reading from the register, the bits show the value from the addressed table entry (address from last write operation).
12–11 PRI1	The priority information to write into the addressed table entry. These 2 bits represent the output priority selected when the frame is received on port 1. When reading from the register, the bits show the value from the addressed table entry (address from last write operation).

**Table 32-31. ESW\_IPRES Field Descriptions (continued)**

Field	Description
10–9 PRI0	The priority information to write into the addressed table entry. These 2 bits represent the output priority selected when the frame is received on port 0. 00=priority 0 (will be forwarded to output queue 0) 01=priority 1 (output queue 1) 10=priority 2 (output queue 2) 11=priority 3 (output queue 3) When reading from the register, the bits show the value from the addressed table entry (address from last write operation).
8 IPV4SEL	If set during a write, the IPv4 table is accessed. Valid address values range from 0 to 63. If cleared, the IPv6 table is accessed. Valid address values range from 0 to 255.
7–0 ADDRESS	The address of the priority entry to read or write for a frame received on port n. The IPv4 priority table has 64 entries. The IPv6 table has 256 entries. See also <a href="#">Section 32.4.5.2.1, “Classification Table Programming Model”</a> for a description of the mapping table.

### 32.3.31 Port n Priority Resolution Configuration (ESW\_PnRES)

ESW\_PnRES defines which priority information should be used for priority resolution.

Address: 0xFC0D\_C180 (ESW\_P0RES)  
 0xFC0D\_C184 (ESW\_P1RES)  
 0xFC0D\_C188 (ESW\_P2RES)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
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
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	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 32-35. Priority Configuration Registers (ESW\_PnRES)****Table 32-32. ESW\_PnRES Field Descriptions**

Field	Description
31–4	Reserved, must be cleared.
6–4 DFLT_PRI	The default priority of a frame received on port n, if none of the priority resolutions could define a priority of the frame. Up to 3 bits can be implemented.
3	Reserved, must be cleared.
2 MAC	Enable MAC based priority resolution for frame received on port n. If set, the priority information found within the MAC address table is used.
1 IP	Enable IP priority resolution for frame received on port n. If set, the IP DiffServ/COS field is used and priority is resolved according to the ESW_IPRES setting for the port. If cleared, IP Diffserv/COS fields are ignored.
0 VLAN	Enable VLAN priority resolution for frame received on port n. If set, the VLAN tag field of a frame is inspected and priority is resolved according to the setting programmed in ESW_PnVRES for the port on which the frame was received. If cleared, VLAN priority is ignored.

### 32.3.32 Port *n* VLAN ID (ESW\_P*n*ID)

ESW\_P*n*ID defines the VLAN ID field for VLAN input manipulation function of a port (if it exists).

Address: 0xFC0D\_C200 (ESW\_P0ID)  
 0xFC0D\_C204 (ESW\_P1ID)  
 0xFC0D\_C208 (ESW\_P2ID)

Access: User 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	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																																
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		

Figure 32-36. VLAN Priority Registers (ESW\_P*n*ID)

Table 32-33. ESW\_P*n*ID Field Descriptions

Field	Description
31–16	Reserved, must be cleared.
15–0 VLANID	VLAN ID field for the VLAN input manipulation function.

### 32.3.33 VLAN Domain Resolution 0–31 (ESW\_VRES0–31)

#### NOTE

The VLAN table is always searched completely. Therefore, the table entries do not need to be written in any order.

Address: 0xFC0D\_C280 (ESW\_VRES0)  
 0xFC0D\_C284 (ESW\_VRES1)

Access: User read/write

...

0xFC0D\_C2FC (ESW\_VRES31)

	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	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	P2	P1	P0	
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	

Figure 32-37. VLAN Priority Registers (ESW\_VRES*n*)

Table 32-34. ESW\_VRES*n* Field Descriptions

Field	Description
31–15	Reserved, must be cleared.
14–3 VLANID	VLAN identifier
2, 1, 0 P <i>n</i>	One bit per port that is member of the VLAN identified with the 12-bit VLAN ID of the entry.

### 32.3.34 Statistics Registers

- ESW\_DISCN — Total number of incoming frames processed but discarded in the switch
- ESW\_DISCB — Sum of bytes of frames counted in ESW\_DISCN
- ESW\_NDISCN — Total number of incoming frames processed and not discarded
- ESW\_NDISCB — Sum of bytes of frames counted in ESW\_NDISCN

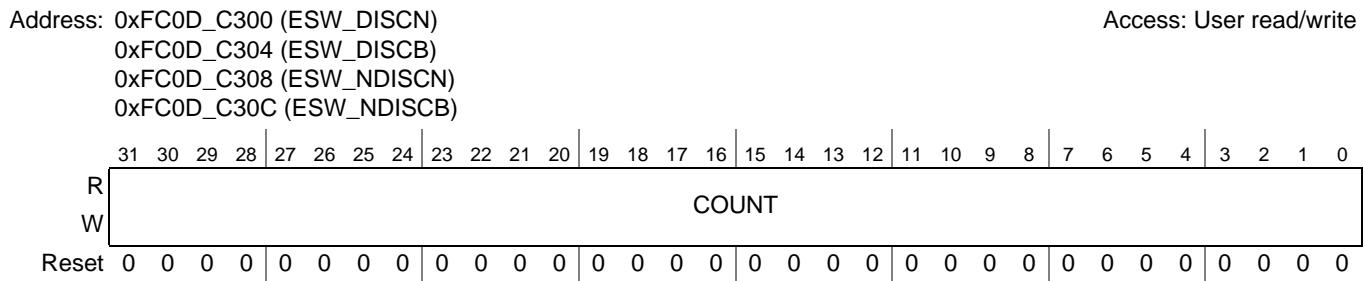


Figure 32-38. Statistics Registers

### 32.3.35 Port Statistics Registers

- ESW\_PnOQC — Port 0 outgoing frames discarded due to output queue congestion
- ESW\_PnMVID — Port 0 incoming frames discarded due to mismatching or missing VLAN ID while VLAN verification was enabled. See ESW\_VLANV.
- ESW\_PnMVTAG — Port 0 incoming frames discarded due to missing VLAN tag. See ESW\_VLANV.
- ESW\_PnBL — Port 0 incoming frames discarded (after learning) as port is configured in blocking mode

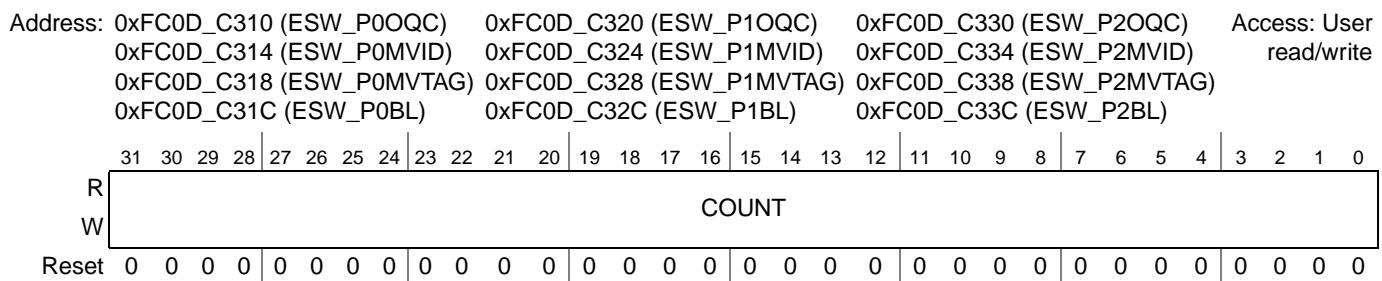


Figure 32-39. Port Statistics Registers

### 32.3.36 Interrupt Status Register (ESW\_ISR)

ESW\_ISR indicates the interrupt status. To clear a bit write a one to it. The bit stays set if the event condition persists.

Address: 0xFC0D\_C400 (ESW\_ISR) Access: User read/write

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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	LRN	OD2	OD1	OD0	QM	TXF	TXB	RXF	RXB	EBERR
W						w1c										
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 32-40. Interrupt Status Register (ESW\_ISR)

Table 32-35. ESW\_ISR Field Descriptions

Field	Description
31–10	Reserved, must be cleared.
9 LRN	Learning Record available in registers LNR_REC_0 and LNR_REC_1 (Signal ipi_lrn_int asserted). Note: this interrupt can be very frequent on a heavy loaded network. It is not recommended to use this interrupt source as interrupt but rather implement a slow background task polling the bit to perform learning.
8 OD2	Outgoing frames discarded due to output Queue congestion on Port 2 or port is disabled (ESW_PER). Asserts ipi_od2_int
7 OD1	Outgoing frames discarded due to output Queue congestion on Port 1 or port is disabled (ESW_PER). Asserts ipi_od1_int
6 OD0	Outgoing frames discarded due to output Queue congestion on Port 0 or port is disabled (ESW_PER). Asserts ipi_od0_int
5 QM	Low Memory Threshold. Asserted if the memory became congested and number of free cells dropped below threshold ESW_LMT (Signal ipi_qm_int asserted). Note: will become asserted after reset immediately due to memory initialization.
4 TXF	Transmit frame interrupt. This bit indicates a frame has been transmitted and the last corresponding buffer descriptor has been updated (Signal ipi_txf_int asserted).
3 TXB	Transmit buffer interrupt. This bit indicates a transmit buffer descriptor has been updated (Signal ipi_txb_int asserted).
2 RXF	Receive frame interrupt. This bit indicates a frame has been received and the last corresponding buffer descriptor has been updated (Signal ipi_rxf_int asserted).
1 RXB	Receive buffer interrupt. This bit indicates a receive buffer descriptor not the last in the frame has been updated (Signal ipi_rxb_int asserted).
0 EBERR	Ethernet bus error. This bit indicates a system bus error occurs when a DMA transaction is underway (Signal ipi_eberr_int asserted).

### 32.3.37 Interrupt Mask Register (ESW\_IMR)

Address: 0xFC0D\_C404 (ESW\_IMR)

Access: User read/write

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

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	LRN	OD2	OD1	OD0	QM	TXF	TXB	RXF	RXB	EB ERR
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 32-41. Interrupt Mask Register (ESW\_IMR)

Table 32-36. ESW\_IMR Field Descriptions

Field	Description
31–10	Reserved, must be cleared.
9–0 See Figure 32-41	Each bit corresponds to an interrupt source defined by the ESW_ISR register. The corresponding ESW_IMR bit determines whether an interrupt condition can generate an interrupt. At each processor clock, ESW_ISR samples the signal generated by the interrupting source. Each ESW_ISR bit reflects the state of the interrupt signal even if the corresponding ESW_IMR bit is set. 0 The corresponding interrupt source is masked 1 The corresponding interrupt source is not masked and an interrupt can occur

### 32.3.38 Receive Descriptor Ring Pointer (ESW\_RDSR)

ESW\_RDSR points to the start of the circular receive buffer descriptor queue in external memory. This pointer must be 32-bit aligned; however, it is recommended it be made 128-bit aligned (evenly divisible by 16). This register is not reset and must be initialized prior to operation.

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

Figure 32-42. Receive Descriptor Ring Start Register (ESW\_RDSR)

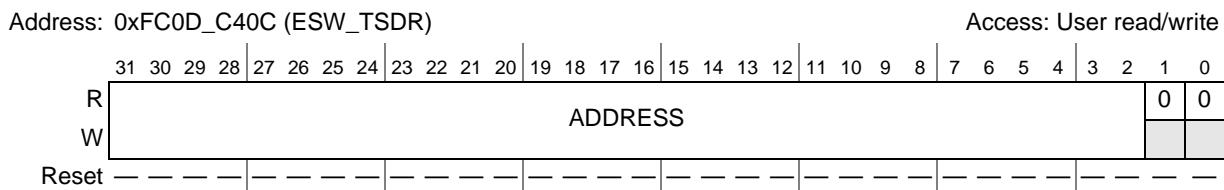
Table 32-37. ESW\_RDSR Field Descriptions

Field	Description
31–2 ADDRESS	Pointer to start of receive buffer descriptor queue.
3–0	Reserved, must be cleared.

### 32.3.39 Transmit Descriptor Ring Pointer (ESW\_TDSR)

`ESW_TSDR` provides a pointer to the start of the circular transmit buffer descriptor queue in external memory. This pointer must be 32-bit aligned; however, it is recommended it be made 128-bit aligned (evenly divisible by 16). You should write zeros to bits 1 and 0. Hardware ignores non-zero values in these two bit positions.

This register is undefined at reset and must be initialized prior to operation.



**Figure 32-43. Transmit Buffer Descriptor Ring Start Register (ESW\_TDSR)**

**Table 32-38. ESW\_TDSR Field Descriptions**

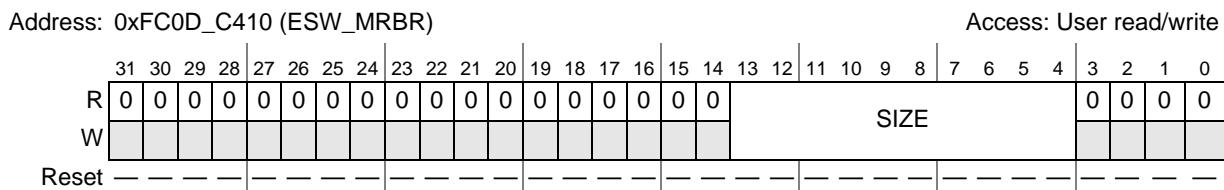
Field	Description
31–2 ADDRESS	Pointer to start of transmit buffer descriptor queue.
1–0	Reserved, must be cleared.

### 32.3.40 Maximum receive buffer size (ESW\_MRBR)

The ESW\_MRBR dictates the maximum size of all receive buffers. This value should take into consideration that the receive CRC is always written into the last receive buffer. To allow one maximum size frame per buffer, ESW\_MRBR must be set to ENETn\_RCR[MAX\_FL] or larger. To properly align the buffer, ESW\_MRBR must be evenly divisible by 16. To ensure this, bits 3–0 are forced low.

To minimize bus utilization (descriptor fetches), it is recommended that ESW\_MRBR be greater than or equal to 256 bytes.

The ESW\_MRBR register is undefined at reset and must be initialized by the user.



**Figure 32-44. Receive Buffer Size Register (ESW\_MRBR)**

**Table 32-39. ESW\_MRBR Field Descriptions**

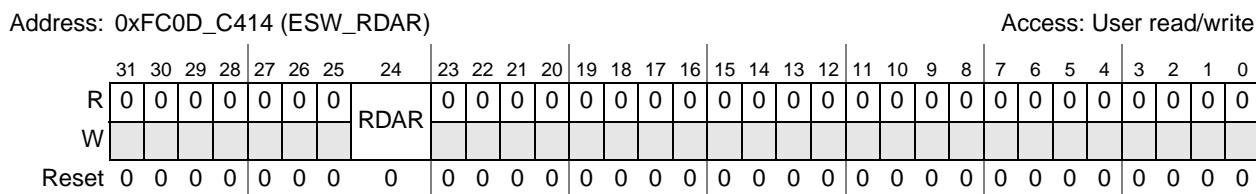
Field	Description
31–14	Reserved, must be cleared.
13–4 SIZE	Maximum size of receive buffer size in bytes. To minimize bus utilization (descriptor fetches), set this field to 256 bytes (0x10) or larger. 0x010 256 + 15 bytes (minimum size recommended) 0x011 272 + 15 bytes ... 0x3FF 16,368 + 15 bytes
3–0	Reserved, must be cleared.

### 32.3.41 Receive Descriptor Active Register (ESW\_RDAR)

ESW\_RDAR is a command register, written by the user, indicating the receive descriptor ring is updated (the driver produced empty receive buffers with the empty bit set).

When the register is written, the RDAR bit is set. This is independent of the data actually written by the user. When set, the FEC polls the receive descriptor ring and processes receive frames (provided ENET<sub>n</sub>\_ECR[ETHER\_EN] is also set). After the MAC polls a receive descriptor whose empty bit is not set, the MAC clears the RDAR bit and ceases receive descriptor ring polling until the user sets the bit again, signifying that additional descriptors are placed into the receive descriptor ring.

The ESW\_RDAR register is cleared at reset and when ENET<sub>n</sub>\_ECR[ETHER\_EN] is cleared.

**Figure 32-45. Receive Descriptor Active Register (ESW\_RDAR)****Table 32-40. ESW\_RDAR Field Descriptions**

Field	Description
31–25	Reserved, must be cleared.
24 RDAR	Set to 1 when this register is written, regardless of the value written. Cleared by the FEC device when no additional empty descriptors remain in the receive ring. Also cleared when ENET <sub>n</sub> _ECR[ETHER_EN] is cleared.
23–0	Reserved, must be cleared.

### 32.3.42 Transmit Descriptor Active Register (ESW\_TDAR)

The ENET<sub>n</sub>\_TDAR are command registers which the user writes to indicate the transmit descriptor ring is updated (transmit buffers have been produced by the driver with the ready bit set in the buffer descriptor).

When the register is written, the TDAR bit is set. This value is independent of the data actually written by the user. When set, the MAC polls the transmit descriptor ring and processes transmit frames (provided ENET $n$ \_ECR[ETHER\_EN] is also set). After the MAC polls a transmit descriptor that is a ready bit not set, MAC clears TDAR and ceases transmit descriptor ring polling until you set the bit again, signifying additional descriptors are placed into the transmit descriptor ring.

The ENET $n$ \_TDAR registers are cleared at reset, when ENET $n$ \_ECR[ETHER\_EN] is cleared, or when ENET $n$ \_ECR[RESET] is set.

Address: 0xFC0D_C418 (ESW_TDAR)																																
R	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
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	

Figure 32-46. Transmit Descriptor Active Register (ENET $n$ \_TDAR)

Table 32-41. ENET $n$ \_TDAR Field Descriptions

Field	Description
31–25	Reserved, must be cleared.
24 TDAR	Set to 1 when this register is written, regardless of the value written. Cleared by the MAC device when no additional ready descriptors remain in the transmit ring. Also cleared when ENET $n$ _ECR[ETHER_EN] is cleared.
23–0	Reserved, must be cleared.

### 32.3.43 Learning Record Registers (ESW\_LREC0 and ESW\_LREC1)

ESW\_LREC0 must be read first, followed by reading ESW\_LREC1.

Address: 0xFC0D_C500 (ESW_LREC0)																																
R	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
W	MACADDR0																															
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		

Figure 32-47. Learning Record Register 0 (ESW\_LREC0)

Table 32-42. ESW\_LREC0 Field Descriptions

Field	Description
31–0 MAC_ADDR0	Lower 32-Bit of the Frame MAC Address. 7:0 = first octet, 31:24=4th octet. Note: this register must be read first, before reading ESW_LREC1

Address: 0xFC0D\_C504 (ESW\_LREC1)

Access: User 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	0	0	0	0	0	0	SW	PORT	HASH								MACADDR1																
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		

Figure 32-48. Learning Record Register 1 (ESW\_LREC1)

Table 32-43. ESW\_LREC1 Field Descriptions

Field	Description
31–26	Reserved, must be cleared.
25–24 SWPORT	Port number on which the Frame is received.
23–16 HASH	The 8-bit Hash value
15–0 MAC_ADDR1	Upper 16-Bit of the Frame MAC Address. 7:0=5th octet, 15:8=6th octet.

### 32.3.44 Learning Data Status Register (ESW\_LSR)

Address: 0xFC0D\_C508 (ESW\_LSR)

Access: User 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	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	DA		
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		

Figure 32-49. Learning Data Status (ESW\_LSR)

Table 32-44. ESW\_LSR Field Descriptions

Field	Description
31–1	Reserved, must be cleared.
0 DA	Indicates if the learning record is valid and can be read. 0 Learning record invalid 1 Learning record valid

### 32.3.45 Look-up Memory Table

This is the  $2048 \times 64$ -bit entry MAC address lookup table. An entry of 64 bits is split into two 32-bit words. The low address (0xFC0E\_0000, 0xFC0E\_0008, 0xFC0E\_0010, ...) represents the lower 32 bits (31:0) and the high address (0xFC0E\_0004, 0xFC0E\_000c, 0xFC0E\_0014, ...) represents the higher 32 bits (63:32) of an entry. See [Section 32.4.7.2, “Address Memory”](#) for the structure of a 64-bit entry.

Each entry must be written or read with the low address accessed first followed by the high address. The table should be initialized by software during system startup.

## 32.4 Functional Description

The switch implements the following main functions:

- Input/output VLAN processing
- IP snooping
- Input frame parsing and priority extraction
- Input port selection
- Output port(s) resolution
- Frame queuing
- Output queue scheduling

### 32.4.1 VLAN Input Processing Function

The VLAN input processing function is used on each switch input port to inspect and manipulate the VLAN tag of frames entering the switch. It performs the following functions:

- Input frame parsing
- VLAN tag insertion or manipulation

Based on the information of the input processing function the frame can be switched to the corresponding output port or is discarded.

#### 32.4.1.1 Terms and Definitions

- VLAN information — The 16-bit field following the VLAN type field within a frame
- VLAN ID — The lower 12 bits of the VLAN information field
- VLAN priority — The upper 3 bits of the VLAN information field that prioritizes incoming frames. A value of 0 represents lowest priority; a value of 7 represents highest priority.

#### 32.4.1.2 Configuration Information

The switch management provides the following information to configure and control the operation of the function:

- ESW\_PnID — 16 bit value. The VLAN information field (VLAN-ID and priority) used for tag insertion operations.
- Mode of operation — There are different modes of operation, which define how incoming frames must be processed for a port. The function can be enabled and configured individually per port. See the ESW\_VIMEN and ESW\_VIMSEL registers.

#### NOTE

If the VLAN input processing function is not enabled (ESW\_VIMEN = 0) the mode setting has no effect.

### 32.4.1.3 Modes of Operation

#### 32.4.1.3.1 Frame Processing

The VLAN input processing function modifies the frames before they enter the switching engine. If a VLAN tag is inserted, the switch only acts on the inserted VLAN tag (e.g. priority). Any original tag that was found in the frame before the modification, if any, has no effect within the switch.

In addition, if VLAN verification is enabled for a port (see the ESW\_VLANV register), the VLAN ID used for insertion (ESW\_PnID) must also be configured in the global VLAN resolution table (see the ESW\_VRESn register). This ensures the switch accepts frames, which contain the inserted tag.

When a tag is inserted in any of the modes, it is always inserted as the first tag (outer) and its information field is set as programmed in the ESW\_PnID register for the port *n* where the frame is received.

#### 32.4.1.3.2 Mode 1 — Single Tagging with Passthrough

Mode 1 inserts a tag only if the frame is untagged. If the frame is already tagged, the frame is unmodified.

#### 32.4.1.3.3 Mode 2 — Single Tagging with Replace

If untagged, add the tag. If single tagged, overwrite it.

#### 32.4.1.3.4 Mode 3 — Double Tagging with Passthrough

Insert a tag on untagged and tagged frames. This results in a single-tagged frame when an untagged is received, and a double-tagged frame, when a single-tagged frame is received. When a double-tagged frame is received, the frame is unmodified.

#### 32.4.1.3.5 Mode 4 — Double Tagging with Replace

Insert tag on untagged and single-tagged frames. If a double-tagged frame is received, overwrite the outer tag.

### 32.4.2 IP Snooping

The switch supports programmable snooping for up to eight programmable IP protocols. If the protocol field of an IPv4 or IPv6 frame matches one of the programmed values and snooping is enabled for that entry, the frame can be processed as follows:

- Forward to designated management port only
- Copy to management port and normal forward/flood
- Discard on match

The management port is identified by the port number set in the ESW\_BMPC register. The function is configured using ESW\_IPSNP1–8.

The snooping function can be enabled/disabled individually for each of the entries. If no protocol matches, a match occurs but snooping is disabled, or the frame is coming from the management port itself, the frame is processed normally.

**NOTE**

Snooping respects any optional VLAN tags (i.e. extracts next after last VLAN tag).

### **32.4.3 TCP/UDP Port Number Snooping**

Programmable snooping for up to eight programmable TCP or UDP port numbers. If the source or destination port number field within an TCP/IP or UDP/IP frame (IPv4 and IPv6) matches the compare value and snooping is enabled for that entry, the frame can be processed as follows:

- Forward to designated management port only
- Copy to management port and normal forward/flood
- Discard on match

The management port is identified by the port number set in the ESW\_BMPC register. The function is configured using ESW\_IPSNP1–8.

The snooping function can be enabled/disabled individually for each of the entries. If no entry matches, a match occurs but snooping is disabled, or the frame is coming from the management port itself, the frame is processed normally.

**NOTE**

Port number snooping is possible only if the IP header ends up to ten words (40 bytes) after the MAC header. If the IP header ends later (e.g. IPv6 + VLAN or IPv4 + >20 byte options) the port numbers cannot be parsed any more and the port number snooping is ignored (protocol-based snooping is not affected by this limit).

For IPv6 frames the port number can only be compared if the UDP or TCP header is the very next header to the IPv6 header (i.e. it does not detect such headers if any extension headers are present in an IPv6 frame before the TCP or UDP header).

### **32.4.4 VLAN Output Processing Function**

The VLAN output processing function is used on a switch output port to manipulate the VLAN tag of the outgoing frames that leave the switch. Frames are processed based on the output processing mode and the number of tags in the frame.

#### **32.4.4.1 Configuration Information**

The switch management provides the information on operating mode to configure and control the operation of the function using the ESW\_VOMSEL register of a port. There are three different modes of operation, which define how the outgoing frames should be processed.

##### **32.4.4.1.1 Mode 0 — Disabled**

No frame manipulation occurs.

#### 32.4.4.1.2 Mode 1 — Strip Mode

In strip mode, all the tags (single or double) are removed from incoming frame.

#### 32.4.4.1.3 Mode 2 — Tag Through Mode

In tag through mode, the inner tag is passed through while the outer tag is removed for a double-tagged frame. The following rules apply:

- When a single-tagged frame is received, strip the tag from the frame.
- When a double-tagged frame is received, strip the outer tag from the frame.

#### 32.4.4.1.4 Mode 3 — Transparent Mode

In transparent mode, a single-tagged frame is unchanged. The following rules apply:

- When a single-tagged frame is received, frame is unchanged.
- When a double-tagged frame is received, strip the outer tag from the frame.

### 32.4.5 Frame Classification and Priority Resolution

When a frame is received on an input port, several pieces are extracted from the frame (Ethernet MAC address, VLAN tag, and IP headers) to determine the frame type and perform the relevant classification actions.

In addition, the MAC address table can provide a priority indication for the destination MAC address if the switch management has programmed the address table accordingly (static entry).

The frame is classified in up to four priority levels (0 = lowest, 3 = highest) and is eventually queued in the corresponding priority queue at the output port.

#### 32.4.5.1 VLAN Priority Look-Up

An eight-entry programmable priority table is implemented on each port. The ESW\_PnVRES registers contain the priority mapping for port  $n$ .

The switch uses 3-bit priority field from the VLAN tag information to extract the corresponding bits from the table, which indicates which priority the frame is finally classified.

The index in the mapping table is the three bits of the first octet of the VLAN tag data (bit 5 (prio0) is the lsb and bit 7 (prio2) is the msb).

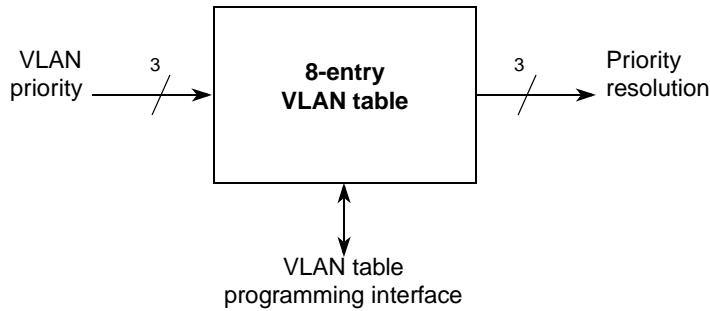


Figure 32-50. VLAN Table Overview

### 32.4.5.2 IPv4 and IPv6 Priority Look Up

The switch can classify IPv4 and IPv6 frames:

- A 64-entry table is implemented per port to classify the IPv4 frames
  - The frame's six-bit DiffServ field is provided and the table returns the 3-bit priority information
- A 256-entry table is implemented per port to classify IPv6 Frames (IP COS tables)
  - The eight-bit class of service field is provided and the table returns the 3-bit priority information

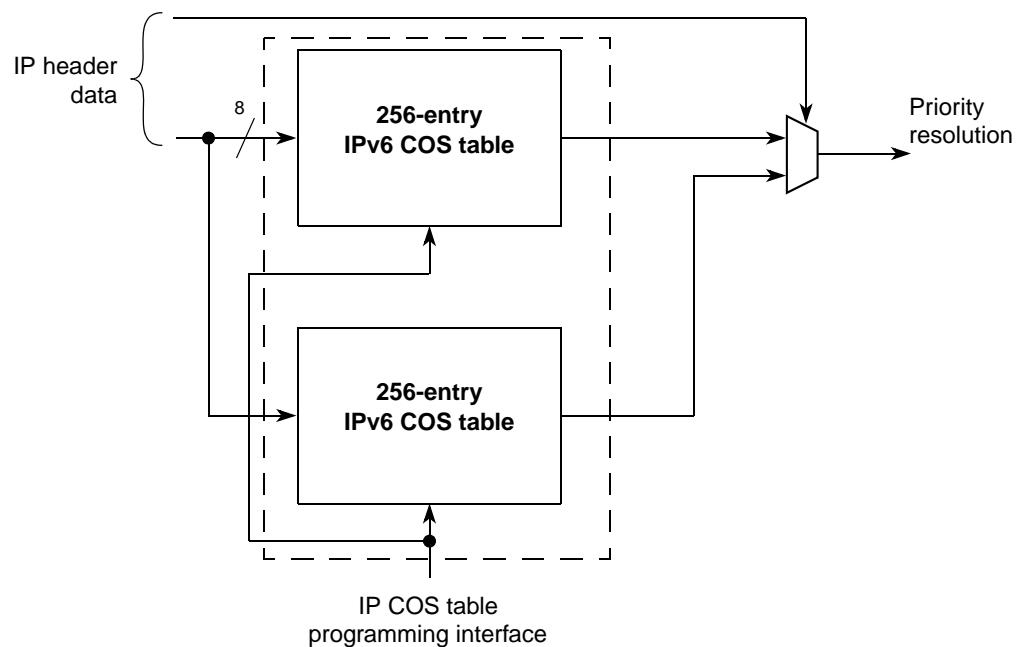
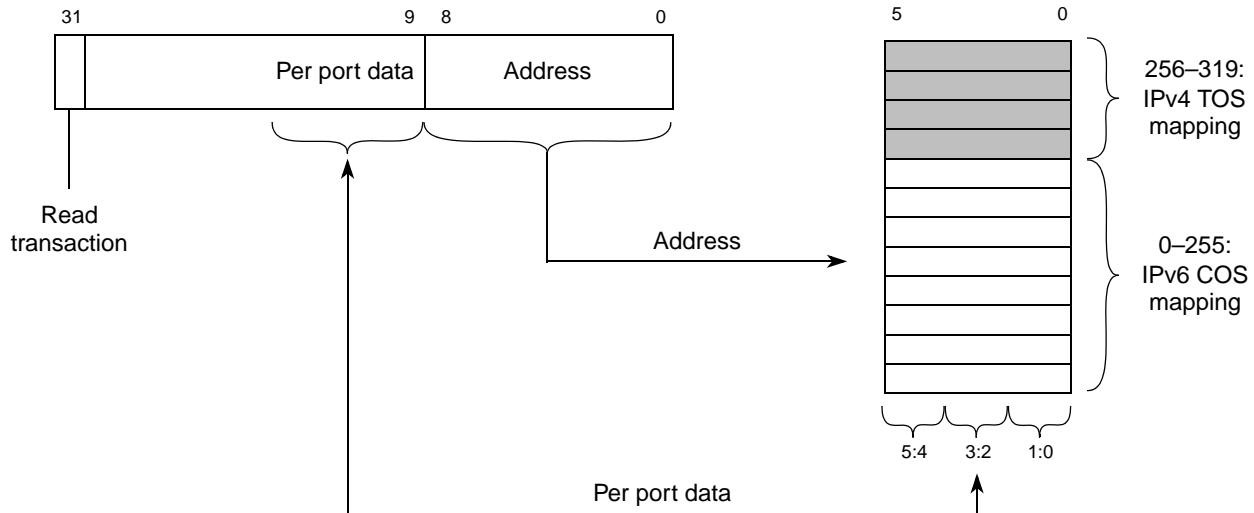


Figure 32-51. IP COS Tables Overview

### 32.4.5.2.1 Classification Table Programming Model

An indirect addressing scheme is implemented to program the mapping tables using a single register (ESW\_IPRES).



**Figure 32-52. ESW\_IPRES Mapping Table Programming Model**

The table is implemented in a single 320-deep table used for IPv4 6-bit TOS and IPv6 8-bit COS mappings.

- The first 256 entries represent the IPv6 COS field mapping. The received COS field of a frame is used to address a row from 0–255. The value stored is read and used as priority for the frame.
- The last 64 entries represent the IPv4 DiffServ field mapping. The received DiffServ (upper 6-bits of TOS) value of a frame is used to address a row from 256–319.
- Each entry of the table provides the priority mapping in bits 1:0 for port 0 (00 = queue0, 01 = queue1, 10 = queue2, 11 = queue3), bits 3:2 for port 1, and bits 5:4 for port 2.

To write a table row into the table the address is provided in bits 8:0 and the data in bits 13:9, where bit 9 represents bit 0 of the data and bit 13 represents bit 5 of the data.

To read a table row, the read bit must be set when writing into the ESW\_IPRES register. This triggers a read transaction to the address provided in bits 8:0 of the register. After this, reading the register provides the data returned from the table for this address.

When writing an entry into the table, software can only write the mapping for all ports in one write transaction. Therefore software must implement a read-modify-write scheme, to:

1. Read the current table contents
2. Modify the priority bits for the port of interest without modifying the other ports bits
3. Write back the complete data word into the table

### 32.4.5.3 Priority Resolution

The priority resolution function is, on each port independently, programmable with the registers ESW\_PnRES to enable or disable VLAN, IP, or MAC address-based classification (see [Section 32.3.31, “Port n Priority Resolution Configuration \(ESW\\_PnRES\)”](#)).

The priority resolution follows the following ruleset depending on which classifications are enabled (ESW\_PnRES) and which fields are found within the frame:

- If IP classification is enabled and an IP header found, map the priority according to the ESW\_IPRES table
- Else, if VLAN classification is enabled and a VLAN tag is found, map the priority according to the ESW\_PnVRES table
- Else, if MAC classification is enabled and MAC address found, take the priority from address table, if it is a static entry
- Else, use default priority as specified in ESW\_PnRES

### **32.4.5.4 Bridge Control Protocol Identification**

To allow for implementation of bridge control protocols like the spanning tree protocol, all control frames (bridge protocol data units, BPDU) are marked when they enter the switch. The mark then can be used by the input port blocking function to drop the frame after the address learning (see [Section 32.4.9.4, “Protocol Snooping”](#)).

In addition, the function can be configured to pass all frames or to pass only control frames (e.g. covering spanning tree port states blocking, listening, and learning) and discard all other frames.

### **32.4.6 Input Port Selection**

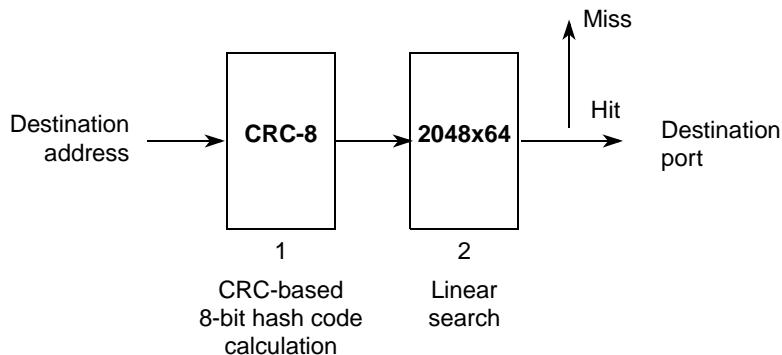
The port selection constantly polls all input ports for available data. If any data is available, the port is selected and frame data is read from the input. After one frame is read, another port is selected, even if more data is available on the current port.

This means for the application on a port atlantic input interface, that it is not allowed to perform back-to-back frame transfers to the switch. Instead the application must wait for a new selection after one frame has been transferred.

### **32.4.7 Layer 2 Look-Up Engine**

A hash code is calculated using the frame destination's MAC address. It is used as an entry (address) to a table, which contains MAC addresses with destination port number and validity information.

As one hash code value can represent more than one MAC address, the memory implements for each pointer, up to eight MAC address entries, which are searched linearly.

**Figure 32-53. Port Look-Up Overview**

### 32.4.7.1 Hash Code

For a MAC address table up to 2048 entries, an 8-bit hash value is calculated from the 48-bit destination MAC address. The hash code uses a CRC-8:

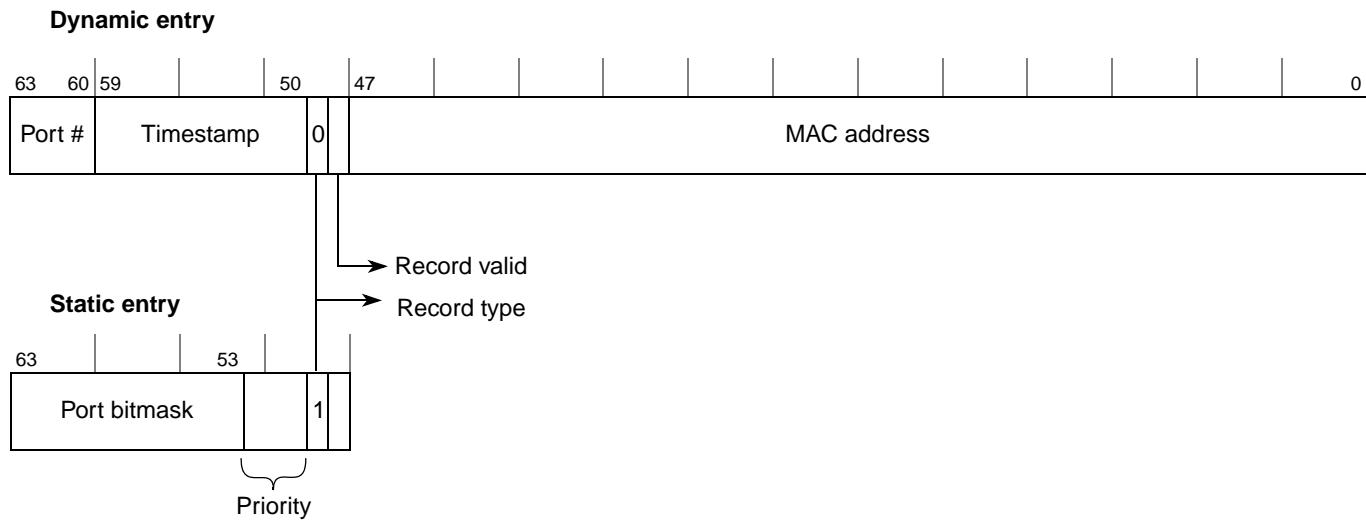
$$x^8 + x^2 + x + 1 \text{ (0x07)}$$

### 32.4.7.2 Address Memory

The address memory is divided into blocks. Each block contains eight records, which contain 64 bits of information each. Each record contains the 48-bit MAC address and provides the necessary forwarding information, and priority or timestamp information.

Two types of records are defined:

- Dynamic record — The dynamic entry provides the MAC address together with a 10-bit timestamp and destination port number. These entries are created by the learning function based on received frames to enable forwarding of frames to dedicated ports. Dynamic entries are deleted by the aging function if not updated.
- Static multiport/priority record — Switch management can also write static entries in the table, which can include priority information and multiple destination ports for forwarding. The MAC address can be unicast or multicast. These records can be used to specify the ports to participate in a specific multicast domain or to assign a MAC address based priority to a frame. The aging and learning functions ignore static records.



**Figure 32-54. Address Memory Record Types**

The record's bit 49 decides which type of record is found in the table:

- If 0, a dynamic entry is available. The 10-bit timestamp and 4-bit port number are given in the upper bits of the record.
- If 1, a static entry is available with a 3-bit priority field followed by a 3-bit port bit mask. The record bit 53 represents port 0, bit 54 port 1, and bit 55 port 2. The frame is forwarded to all ports whose port bitmask is set. The source port is removed dynamically from the bitmask during forwarding (a frame is never forwarded to the port where it came from).

The 48-bit MAC address is stored with the first octet in bits 7:0 and the sixth octet in 47:40 of the record.

### 32.4.8 Layer 2 Lookup Tasks Overview

#### 32.4.8.1 MAC Address Lookup

The 48-bit destination MAC address of each frame received by the switch, on any of its interfaces, is extracted by the hardware and provided to the look-up engine together with the physical interface number. If the frame carries a VLAN tag, the tag information is also extracted and provided to the look-up engine.

If the received frame is a unicast or multicast frame, a two-stage lookup process is implemented. The look-up engine first calculates the hash value from the MAC address. The hash code is used as an entry to the switch address table. The look-up function can provide three results with tree different associated actions performed by the switch hardware:

1. The address is in the table and associated with a correct port number:  
The switch forwards the frame only to the looked up port.
2. The address is in the table but is associated with the port on which it was received:  
The switch discards the frame and does not forward it to any port.
3. The address is not found in the table:

The switch engine sends the received frame to all ports except the port on which it was received (flooding).

If a broadcast frame is received, the switch hardware sends the received frame to all output ports, except the one from which it was received (flooding).

#### NOTE

Flooding and additional frame filtering can be controlled (for example, to avoid the duplication of critical information to unwanted destinations) with the mechanism described in [Section 32.4.9.2, “Broadcast/Multicast/VLAN Domain Resolution”](#).

### 32.4.8.2 Forced Forwarding

The MAC address lookup result can be overwritten using the forced forwarding configuration available in the ESW\_FFEN register. This feature is available only for frames coming from the local port (port 0).

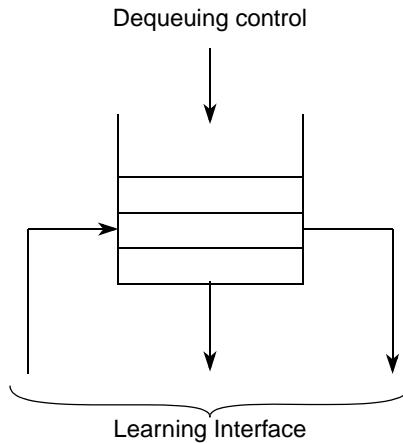
When forced forwarding is enabled for a frame, the frame is forwarded to the forced destination ports, ignoring any results from the MAC destination address lookup. Forced forwarding only replaces the MAC lookup function, all other filtering functions (e.g. VLAN verification) act as normal.

### 32.4.8.3 Learning

The switch hardware extracts the source MAC address of each frame received on each of the switch ports and provides it (via the ESW\_LREC0 and ESW\_LREC1 registers) to the switch firmware which implements the learning task. The ESW\_LSR register indicates availability of learning data.

#### 32.4.8.3.1 Learning Interface

The interface implements a FIFO buffer that stores up to 32 words of 32-bit each.



**Figure 32-55. Learning Interface Overview**

For each frame processed by the switch engine, two 32-bit records (record A0 and record B1) are written in the information FIFO. Record A is written first.

The MAC address available in records A0 and B1 is the source MAC address of the frame. Record A holds the first four bytes of the frame source address, and record B contains the last two bytes.

The hash code in record B is calculated with the source MAC address and the same hash polynomial used for look-up, as defined in [Section 32.4.7.1, “Hash Code”](#).

The 4-bit port number defines the port/MAC address association.

Frame record A	MAC address															
Frame record B	Reserved Port # Hash code MAC address (cont'd)															

**Figure 32-56. Frame Information Records — 8-Bit Hash Values**

When information for at least one frame (two records) is available, the status indication in ESW\_LSR register is set. To read the frame records, read the ESW\_LREC0 register (record A) first followed by the ESW\_LREC1 register (record B).

#### NOTE

Reading ESW\_LREC1 triggers the retrieval of the next record pair from the FIFO, if any.

The learning task (software) uses this information and then executes as follows:

1. For every frame received, the source address with port and timestamp information is stored in the address lookup table. The following information is stored for each entry:
  - MAC address
  - Time stamp: a 10-bit value, determines the age of an entry
  - Port number: a 4-bit value, indicates the port the frame was received
2. If the MAC address table is full, a new entry replaces the oldest entry with an identical matching hash value.

#### 32.4.8.4 Migration

If the firmware receives a MAC address, which is already in the switch table but is associated to a different physical port number, the current entry is overwritten with the new information and the timestamp is set to the current time.

#### 32.4.8.5 Aging

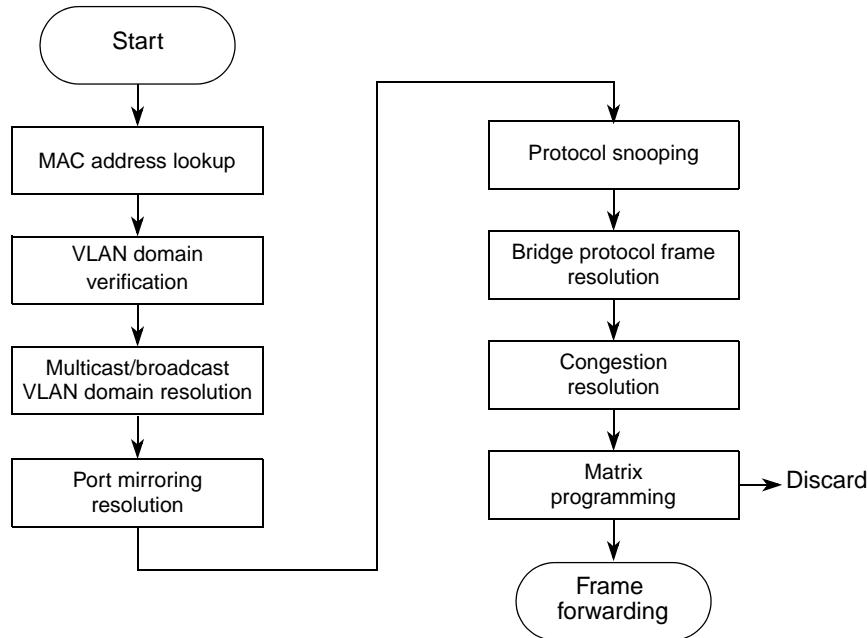
Aging refers to deleting old entries within the table. It proceeds as follows:

1. The 10-bit timestamp is stored with all the entries and is updated each time the source address appears.
2. If a record is not updated for a longer period of time, it is removed from the table if it is a dynamic entry.
3. Static entries are not affected by the aging process and are always kept.

This process runs continuously in the background of the firmware. The aging period is software-controlled and programmable, defaulting to four seconds per step. This gives a range of 4 seconds to 68 minutes.

### 32.4.9 Frame-Forwarding Tasks

When an input port is selected the frame is forwarded to its corresponding output port. Output port resolution and switching is based on the information from the two-stage MAC address look-up (see [Section 32.4.8, “Layer 2 Lookup Tasks Overview”](#)) followed by additional resolution functions to allow frame duplication and flooding control. These are described in the following sections.



**Figure 32-57. Frame Forwarding Tasks Overview**

#### 32.4.9.1 VLAN Domain Verification

When the L2 MAC address lookup is successful and identifies a dedicated output port for the frame, the output and input port can be verified to be in the correct VLAN domain if VLAN verification is enabled for the port and the frame contains a VLAN tag. The VLAN resolution table (see [Section 32.4.9.2.1, “VLAN Resolution Table”](#)) is used as follows:

- If the frame's VLAN ID is in the table but the output port number is not a member of the VLAN domain, or the input port is not a member of the VLAN domain, the frame is marked invalid and is eventually discarded.
- If the frame's VLAN ID is in the table and the output and input ports are members of the VLAN domain, the frame is forwarded normally.
- If the frame's VLAN ID is not found in the VLAN table or the frame has no VLAN tag, the frame is forwarded normally (default broadcast domain), or if the discard bit for the port is set (also in register ESW\_VLANV) it is discarded.

### 32.4.9.2 Broadcast/Multicast/VLAN Domain Resolution

To ensure that traffic within VLAN channels are always routed to the correct ports, for example to avoid the duplication of critical information through a network, the switch implements a resolution mechanism that, for any frame that is switched to multiple ports, checks the VLAN ID provided with the current frame.

The VLAN resolution mechanism searches the VLAN resolution table (see ESW\_VRESn registers), which stores up to 32 unique VLAN IDs, each associated to a port bit mask. The resolution mechanism is used for the following conditions:

- Unicast frames with a destination MAC address that are not in the table of the layer 2 engine
- Multicast frames with a destination MAC address that are not in the table of the layer 2 engine
- Any broadcast frame

#### 32.4.9.2.1 VLAN Resolution Table

The VLAN resolution table (ESW\_VRESn) provides a unique VLAN ID/port bit mask association for up to 32 VLANs. A default entry (ESW\_DBCR) provides an additional port bit mask. The port bit mask implements one bit for each port.

Each port bit indicates, if set, that it is member of the VLAN and frames with the corresponding VLAN ID can be switched to the port. If the port bit is cleared, a frame with the corresponding VLAN ID is not switched to that port. If no VLAN ID matches, the default mask is applied.

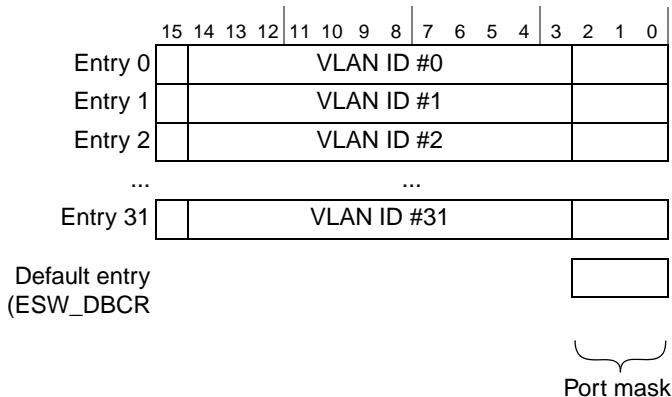


Figure 32-58. VLAN Resolution Table Overview

#### 32.4.9.2.2 VLAN Switching / Resolution Mechanism

The VLAN table is used for VLAN domain verification (see [Section 32.4.9.1, “VLAN Domain Verification”](#)) and VLAN resolution. Once the frame has passed any VLAN domain verification (i.e. will not be discarded by the verification function already) the forwarding resolution applies.

- If the destination MAC address (Unicast or Multicast) is found in the MAC address table and
  - the frame carries a VLAN tag that is found in the VLAN table, the frame can be forwarded only to the ports within the VLAN domain and will be discarded if the destination port is not member of the VLAN domain.

- else if the frame carries a VLAN tag that is not found in the VLAN table, or does not contain a VLAN tag, it is forwarded as indicated by the lookup table (note that VLAN domain verification can be configured to discard the frame in this case if enabled).
- If the destination MAC address (Unicast or Multicast) is not found in the MAC address table, or if the destination address is the Broadcast address, the frame is forwarded according to the following rules:
  - If the frame carries a VLAN tag, the VLAN resolution table is searched for a matching VLAN ID and the frame is sent to all ports that are associated with the VLAN ID.
  - If the frame carries a VLAN tag and the VLAN ID does not match any entry in the VLAN Resolution Table, the frame is forwarded to all ports that are enabled to receive broadcast frames by the default entry.
  - If the frame does not carry a VLAN tag the frame is forwarded to all ports that are enabled to receive broadcast frames by the default entry.
  - The frame is discarded, if it cannot be associated with any VLAN group and if the default (broadcast) group has been set to all zero.

To disable the VLAN resolution set all VLAN IDs to 0xFFFF or (0x000 if that ID is not used) and all port mask bits to 1. If the VLAN resolution is disabled, normal port flooding is implemented as described in [Section 32.4.5, “Frame Classification and Priority Resolution”](#). The default entry can still be used to restrict broadcast to only dedicated ports, if not programmed to all 1s.

### 32.4.9.3 Port Mirroring

The function allows duplicating traffic to a dedicated mirror port. Any one of the ports can be assigned to act as a mirror port (register ESW\_MCR).

The mirror port then is always added to the list of output ports and therefore receives a copy of the frame, if any of the following rules matches with the currently processed frame:

- Ingress Port Number Match  
When a frame is received on port N and the corresponding bit in the register ESW\_INGMAP is set to 1, the frame is mirrored.
- Egress Port Number Match  
When a frame is forwarded to port N and the corresponding bit in the register ESW\_EGMAP is set to 1, the frame will be mirrored.
- MAC Ingress SA Match  
When the Ingress Port Number match succeeded (see above) and the MAC source address matches the ESW\_INGSA{L,H}, the frame will be mirrored.
- MAC Ingress DA Match  
When the Ingress Port Number match succeeded (see above) and the MAC destination address matches the ESW\_INGDA{L,H}, the frame will be mirrored.
- MAC Egress SA Match  
When the Egress Port Number match succeeded (see above) and the MAC source address matches the ESW\_EGSA{L,H}, the frame will be mirrored.

- MAC Egress DA Match

When the Egress Port Number match succeeded (see above) and the MAC destination address matches the ESW\_EGDA{L,H}, the frame will be mirrored.

In addition, a counter is implemented (Register MIRROR\_COUNT) that allows specifying that only every Nth frame that matches any of the above criteria is mirrored. If the counter is set to 1 or 0, every frame is mirrored that matches any of the above criteria.

### **32.4.9.4 Protocol Snooping**

The incoming frames are parsed for IPv4 and IPv6 headers and UDP/TCP if available. The snooping function can be programmed to redirect specific protocols exclusively to the management port. See [Section 32.4.2, “IP Snooping”](#) and [Section 32.4.3, “TCP/UDP Port Number Snooping”](#) for a description of the snooping options.

The snooping is active only for frames received from the external ports. When a frame is transmitted from the management port itself, snooping does not apply and the frames are forwarded normally (MAC lookup).

### **32.4.9.5 Bridge Protocol Frame Resolution**

To implement bridge control protocols like the Spanning Tree protocol, the following control functions are performed by the Protocol Frame Resolution function:

#### **32.4.9.5.1 Input Port Blocking**

The input port blocking function is used to avoid forwarding of frames after address learning. The firmware can program the ESW\_BKLR register and if a frame is received on port  $n$  that should be blocked ( $BE_n = 1$ ) and the frame is not a bridge protocol frame (see below), the frame is marked for discard and is not forwarded to any output port.

#### **32.4.9.5.2 Input Port Learning Disable**

To reduce processing load from the firmware, a port can be configured for exclusion from learning (see the ESW\_BKLR register).

When learning is disabled on a port no source address extraction happens for incoming frames, with the exception of incoming BPDU frames. BPDU frame source addresses are always extracted and forwarded to the learning interface.

#### **32.4.9.5.3 Management Port Forwarding**

If enabled, bridge protocol frames are always forwarded to the dedicated management port (see the ESW\_BMPC register) independent of any address lookup or other resolution functions.

Bridge protocol frames are identified by its destination address being any of the following:

- 01-80-C2-00-00-00 to 01-80-C2-00-00-0F (Spanning Tree, IEEE 802.1d, Table 7-9)
- 01-80-C2-00-00-10 (Bridge Management Address, 802.1d, Table 7-10)

- 01-80-C2-00-00-20 to 01-80-C2-00-00-2F (Generic Attribute Registration Protocol, 802.1d, Table 12-1)

#### **32.4.9.5.4 Management Frame Forwarding**

If the management port transmits Frames, they are forwarded according to the port mask defined in the configuration register ESW\_BMPC. A handshaking mechanism is implemented that can be used by the firmware to configure the destination port mask on a frame-by-frame basis for management frames.

Note: VLAN domain verification/discard (see the ESW\_VLANV register) should be switched off for the management port to avoid that the switch discards management frames.

#### **32.4.9.6 Congestion Resolution**

The congestion resolution function is used whenever an output port is not available and data needs to be sent to that port. An output port is defined to be available if the port is enabled (bit in ESW\_PER set 1) and the output buffer (shared memory) is not congested. If, for a port, one of these conditions is not valid, the port is not available and frames cannot be switched to that port.

The congestion resolution function determines whether the frame should be processed further or discarded according to the following rules:

##### **32.4.9.6.1 Unique Destination (one input to one output)**

If the output port is enabled and can accept a frame the frame will be forwarded normally.

In any other case the frame will be discarded. If a frame switched to port N, the counter ESW\_PnOQC is incremented.

##### **32.4.9.6.2 Multiple Destinations (Flooding)**

After broadcast / flooding resolution a frame needs to be switched to multiple output ports.

- Output disabled: All disabled ports are removed from the list of outputs.
- Output congestion: If any of the outputs cannot accept a frame (As indicated by the output queue management for the port, implementation specific) it is also removed from the list of outputs.

If no output port is left in the list of outputs, the frame is discarded.

If a frame switched to port N, the counter ESW\_PnOQC is incremented.

#### **32.4.9.7 Switching**

After the output port(s) have been determined, the switch control enables the corresponding path through the Switch Matrix and the frame is forwarded to the output queue(s).

In a similar fashion, if a Frame should be switched to multiple ports (e.g. Broadcast), the switch control enables the corresponding paths through the Switch Matrix and the frame is forwarded to all the destination output ports.

### 32.4.10 Output Frame Queuing

The memory controller implements a shared memory architecture to store Frames of arbitrary size for multiple destination ports.

Each destination port implements 4 priority queues. The memory controller implements a single write input port and three output ports with the capability to perform virtual frame duplication on the output ports (Multiple reads on multiple ports of a single frame stored in the buffer).

A single large memory, partitioned in 256 byte cells, is implemented to efficiently share the available space for small and large frames without leaving large unused spaces when storing small frames.

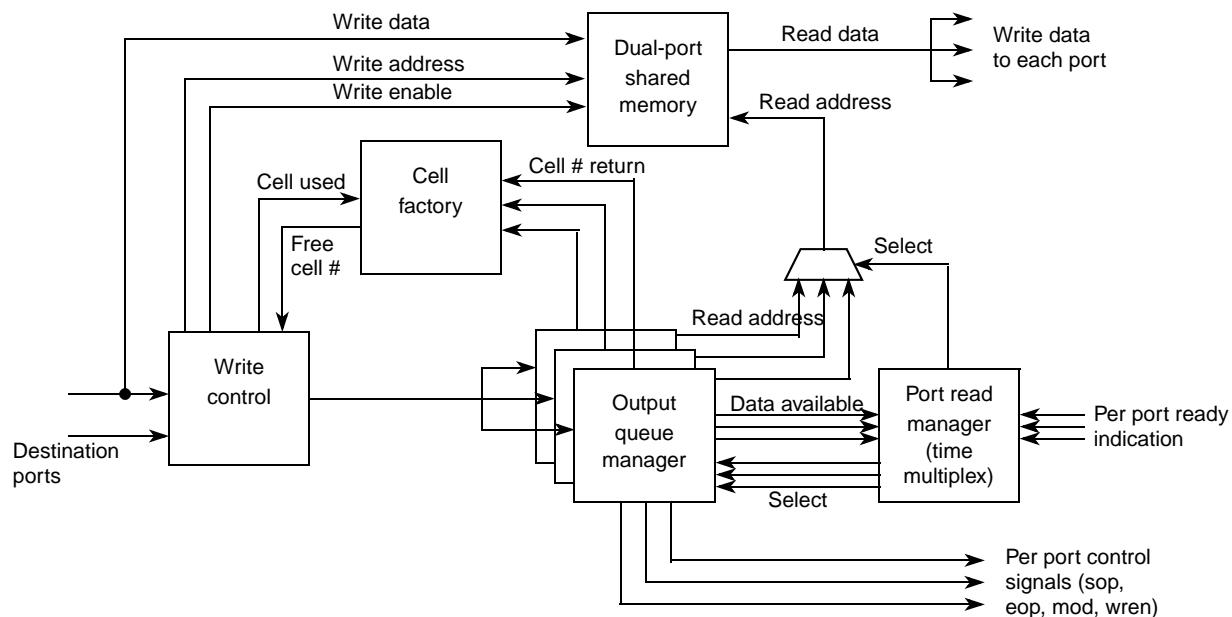


Figure 32-59. Memory Controller Overview

#### 32.4.10.1 Cell and Queue Concept

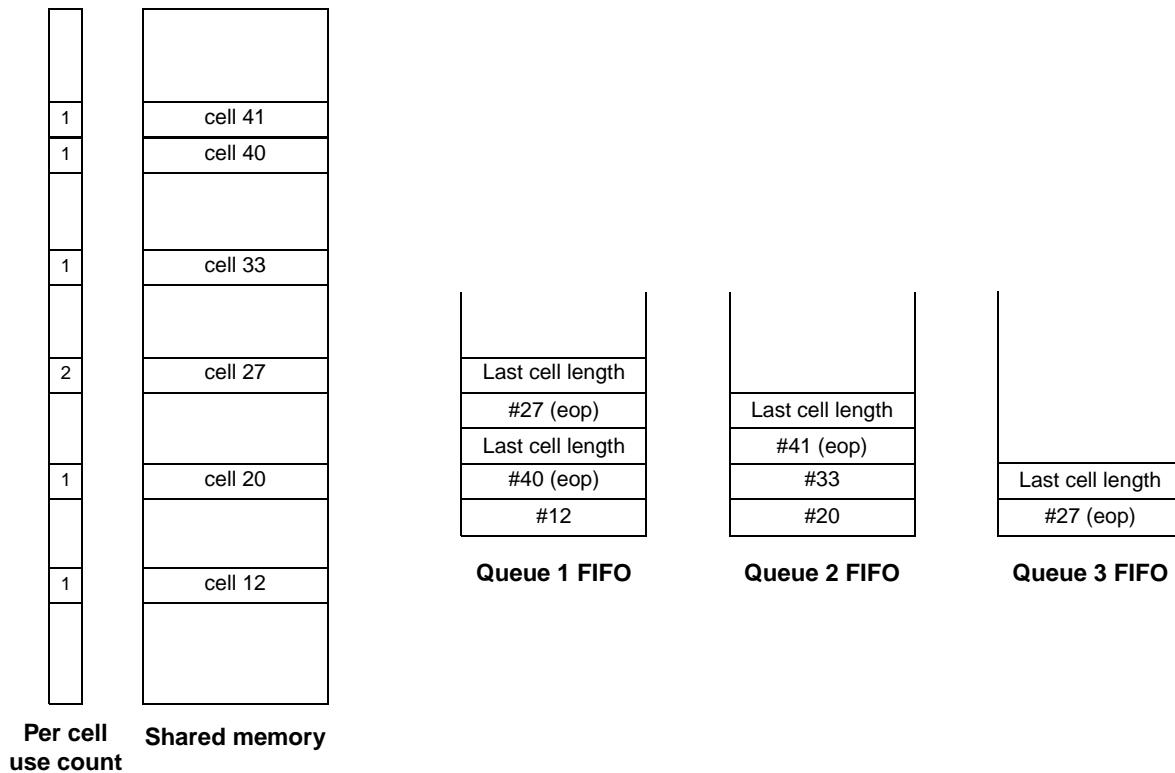
The shared memory is partitioned in 256 byte cells using a 32-bit datapath implementation. This results in a cell holding 64 32-bit words.

Incoming frames are stored, partitioned in cells, in the shared memory and only the cell numbers are managed by the individual port queues.

Due to the arbitrary length of incoming frames, the last cell may not be fully utilized. A frame can spread from one to any number of cells. The number of bytes used in the last cell is also stored in the queue FIFO together with the individual cell numbers.

Cells can be stored anywhere in the shared memory. A single frame must not necessarily be stored in consecutive cells but instead can be scattered over the complete memory at arbitrary positions. The start of a cell is fixed to a 64-word boundary (i.e. the memory start address of a cell is simply the cell # multiplied by 64).

Per port, a queue FIFO is implemented that stores the cell numbers for the frames. The number of bytes used in the last cell is also stored in the queue FIFO together with the individual cell numbers.



**Figure 32-60. Cell Storage Concept**

The example in [Figure 32-60](#) shows the storage of 3 frames with one frame duplicated and stored in 2 queues (orange in above picture).

### 32.4.10.2 Write Control Module

The Write Control Module receives frames from the Switch engine, partitions and stores the frames in the shared memory. The cell numbers used to store the frame are forwarded to the port output managers.

### 32.4.10.3 Cell Factory Module

The Cell Factory implements the cell management, it always provides a free cell number to the write control module so the write control module can immediately start writing into memory to avoid any write latency.

### 32.4.10.4 Output Queue Manager

The Output Queue Manager implements, per port, the individual queue FIFOs (One queue FIFO per priority). The queue FIFOs are addressed by the priority information extracted by the Switch classification engine. Per port, eight prioritized queues are implemented.

When more than one queue has data available, the read logic selects one of the queues with a Weighted Fair Queuing scheduling algorithm.

#### **32.4.10.4.1 Weighted Fair Queuing Scheduling Algorithm**

The weight of each queue, common for all ports, can be configured between 0 and 30 with the register ESW\_QWT. A Queue with a higher weight is served more often than a queue with lower weight.

Queue 0 represents the lowest priority, Queue 3 the highest priority queue.

The scheduler first serves the queue with the highest weight. Each time the scheduler serves a queue, the weight of the other queues is increased and the weight of the selected queue is reset (decreased) to its programmed weight value. This guarantees that all queues are served eventually.

When multiple queues have the same weight the queue with the higher number is served first.

If all weights are programmed to 0 (default) a strict priority scheme is active where the higher priority queues are served as long as they are not empty.

#### **32.4.10.5 Congestion Management**

The Write control logic is protected against memory overflow. When data is written is the cell factory has no more free cells (Number of available cells less than value programmed in register ESW\_LMT), the frame is discarded or terminated with an error (i.e. forwarded to the output queue manager with the end-of-packet and error indication).

If the congestion persists, the switch resolves the congestion as specifies in [Section 32.4.9.6, “Congestion Resolution”](#).

### **32.4.11 Reset and Stop Functions**

#### **32.4.11.1 Stop Controls**

The register ESW\_MODE offers several bits that control output pins. In addition some controls have an effect on internal logic functions:

- stop\_en: no internal function.
- switch\_en: when de-asserted, all DMA registers are cleared.
- switch\_reset: no internal function.

An external logic may use the controls to disable or enable the switch function as necessary.

#### **32.4.11.2 Port Disable**

The switch toplevel offers a disable input for each port (port\_dis(2:0). When a pin is asserted (1), the corresponding port enable bits within register ESW\_PER for both transmit and receive will be cleared.

This results in the following behavior:

- If the port-enable bits of port0 are cleared, it also resets the input buffer and output buffer at the port0 DMA interface.
  - The ready output to DMA0 (ff\_tx\_rdy0) will be asserted allowing the application to continue writing data at the interface, which will be ignored (application flush). No further transmitted frame status (tx\_ts\_val0) will be given (i.e. for any currently stored if any, as well as the currently ignored).
  - If the transmit enable is cleared while the interface is currently transferring a frame to the DMA, the frame is aborted (output buffer reset). The eop is not produced. Therefore the connected DMA module must be reset to ensure proper restart after re-enabling the port.
- If any port's transmit enable bit is cleared, the shared memory will continue delivering the frames stored currently for a port as normal (i.e. flushing the memory). New frames will be discarded before they are written into the shared memory. That is, no invalid frame will appear on the MAC interfaces after disabling or re-enabling a port.
- If any port's receive enable bit is cleared while a frame is transferred, this frame will be aborted with an error internally. The port's ready indication will stay asserted (output ff\_tx\_rdy=1) to flush any application data. Reenabling the port at any time will ignore any input data until a sop starts a new frame.

### 32.4.11.3 Port 0 Input Protection

The port 0 input buffer is protected for application errors that abort a frame without writing a proper eop to the interface. The next frame then written to the port 0 transmit interface will be concatenated with whatever data was already written before, but the frame will be marked with an error and hence will be forwarded and transmitted with an error indication (mii tx error).

If the port0 input buffer is reset (by deasserting ESW\_PER receive enable bit) while a frame is transferred to the switch internally, the frame transfer will be aborted in a clean way (producing an eop with error indication) to avoid blocking the switch.

### 32.4.11.4 Port 1/2 Input Protection

Ports 1 and 2 are protected for a 2nd SOP in case the MAC is reset in the middle of a receive transaction to the switch hence did not produce a proper EOP to the switch. The next frame will be concatenated with whatever was provided to the switch before and marked with an error.

If the MAC is stopped in the middle of a transaction, the switch is blocked, waiting for the EOP, not serving any of the other ports. Clearing the ESW\_PER receive enable bit in this situation will terminate the frame with an error internally to the switch hence remove the blocking condition.

### 32.4.11.5 DMA Bus Error

When the DMA bus error input (dma\_eberr\_int) is asserted, the DMA registers are all cleared. If the corresponding interrupt was enabled the ipi\_eberr\_int pin will be asserted.

# Chapter 33

## FlexCAN

### 33.1 Introduction

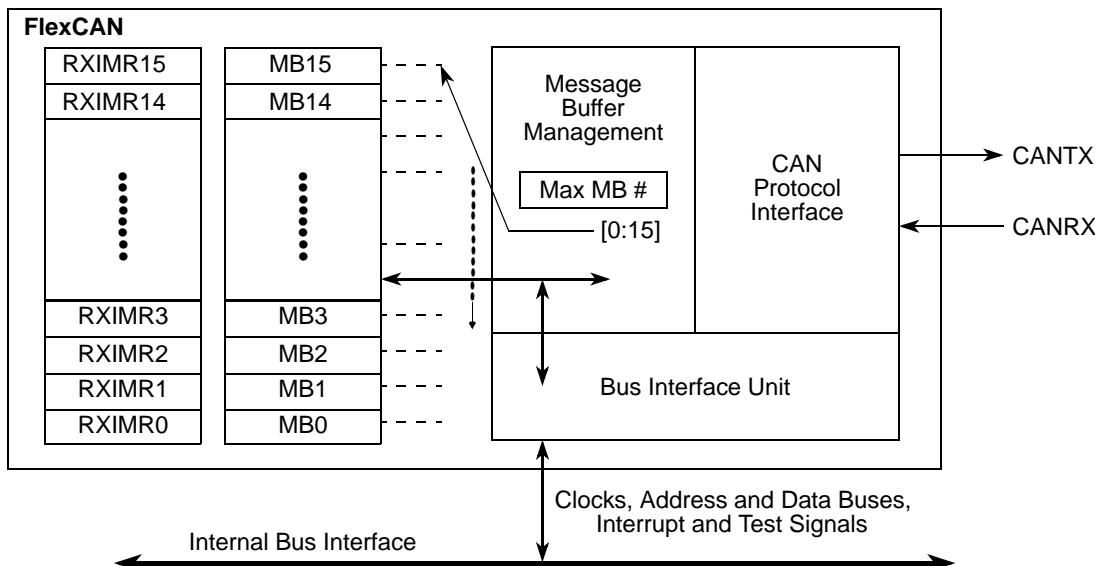
The MCF5441x devices contain two FlexCAN modules.

The FlexCAN is a communication controller implementing the controller area network (CAN) protocol, an asynchronous communications protocol used in automotive and industrial control systems. It is a high speed (1 Mbps), short distance, priority-based protocol that can communicate using a variety of mediums (such as fiber optic cable or an unshielded twisted pair of wires). The FlexCAN supports the standard and extended identifier (ID) message formats specified in the CAN protocol specification, revision 2.0, part B.

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. A general working knowledge of the CAN protocol revision 2.0 is assumed in this document. For details, refer to the CAN protocol revision 2.0 specification.

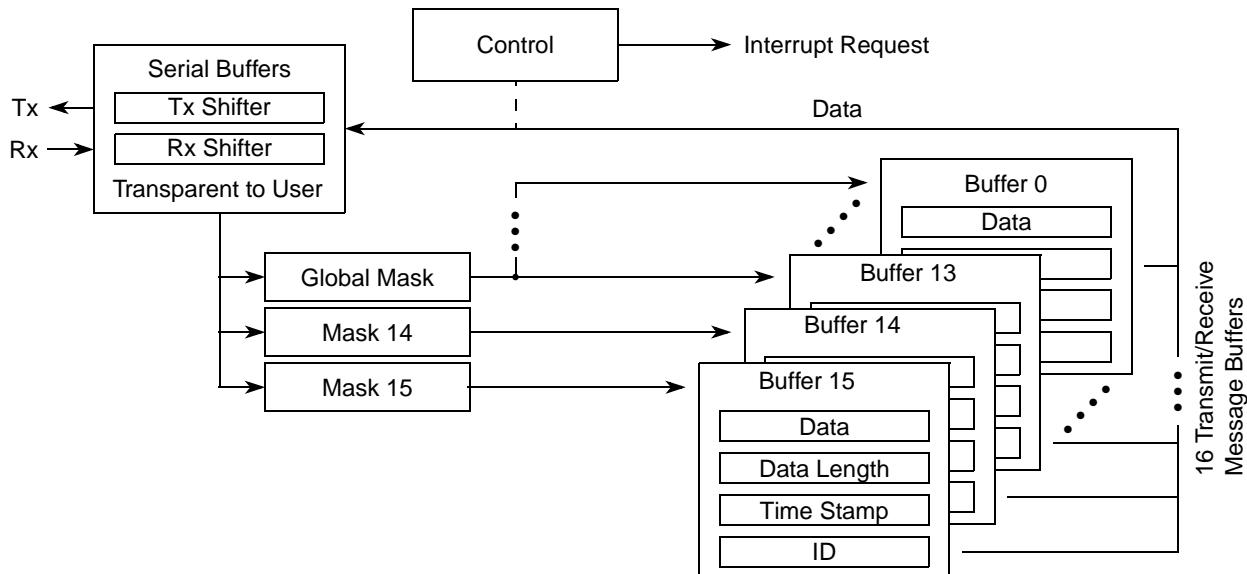
#### 33.1.1 Block Diagram

A block diagram describing the various submodules of the FlexCAN module is shown in [Figure 33-1](#). Each submodule is described in detail in subsequent sections.

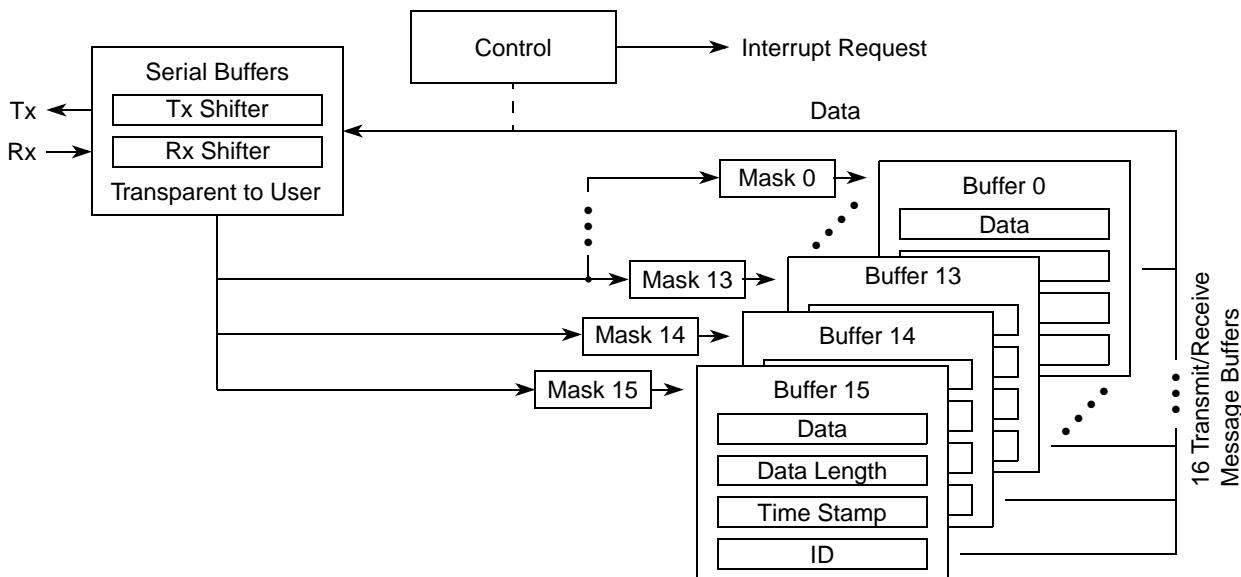


**Figure 33-1. FlexCAN Block Diagram**

The message buffer architecture is shown in [Figure 33-2](#) and [Figure 33-3](#). [Figure 33-3](#) shows the MB architecture when individual masks are used, while [Figure 33-2](#) shows the legacy configuration.



**Figure 33-2. FlexCAN Message Buffer Architecture (CANMCR[BCC] = 0)**



**Figure 33-3. FlexCAN Message Buffer Architecture (CANMCR[BCC] = 1)**

### 33.1.1.1 The CAN System

A typical CAN system is shown below in [Figure 33-4](#). Each CAN station is connected physically to the CAN bus through a transceiver. The transceiver provides the transmit drive, waveshaping, and receive/compare functions required for communicating on the CAN bus. It can also provide protection against damage to the FlexCAN caused by a defective CAN bus or defective stations.

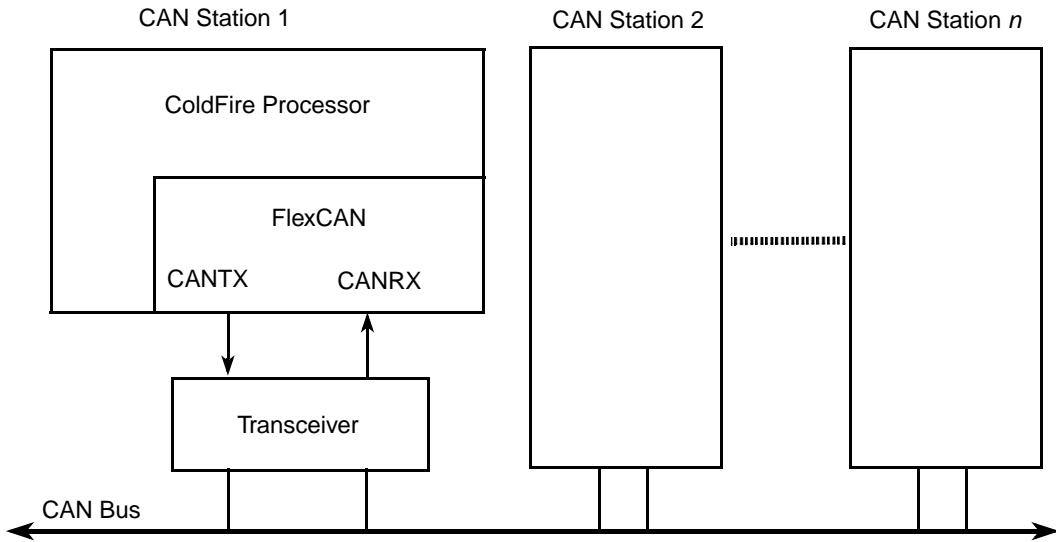


Figure 33-4. Typical CAN System

### 33.1.2 Features

Following are the main features of the FlexCAN module:

- Full implementation of the CAN protocol specification version 2.0B
  - Standard data and remote frames (up to 109 bits long)
  - Extended data and remote frames (up to 127 bits long)
  - 0–8 bytes data length
  - Programmable bit rate up to 1 Mbps
  - Content-related addressing
- Up to 16 flexible message buffers of zero to eight bytes data length, each configurable as Rx or Tx, all supporting standard and extended messages
- Listen-only mode capability
- Individual mask registers for each message buffer
- Reception queue support
- Programmable transmission priority scheme: lowest ID or lowest buffer number
- 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)
- Open network architecture
- Multimaster bus
- High immunity to EMI
- Short latency time due to an arbitration scheme for high-priority messages

### 33.1.3 Modes of Operation

#### 33.1.3.1 Normal Mode

In normal mode, the module operates receiving and/or transmitting message frames, errors are managed normally, and all the CAN protocol functions are enabled. User and supervisor modes differ in the access to some restricted control registers.

#### 33.1.3.2 Freeze Mode

Freeze mode is entered by setting:

- CANMCR $n$ [FRZ], and
- CANMCR $n$ [HALT], or by asserting the  $\overline{\text{BKPT}}$  signal.

After entry into freeze mode is requested, the FlexCAN waits until an intermission or idle condition exists on the CAN bus, or until the FlexCAN enters the error passive or bus off state. After one of these conditions exists, the FlexCAN waits for the completion of all internal activity such as arbitration, matching, move-in, and move-out. When this happens, the following events occur:

- The FlexCAN stops transmitting/receiving frames.
- The prescaler is disabled, thus halting all CAN bus communication.
- The FlexCAN ignores its Rx pins and drives its Tx pins as recessive.
- The FlexCAN loses synchronization with the CAN bus and the NOTRDY and FRZACK bits in CANMCR $n$  are set.
- The CPU is allowed to read and write the error counter registers (in other modes they are read-only).

After engaging one of the mechanisms to place the FlexCAN in freeze mode, the user must wait for the FRZACK bit to be set before accessing any other registers in the FlexCAN; otherwise, unpredictable operation may occur. In freeze mode, all memory mapped registers are accessible.

To exit freeze mode, the  $\overline{\text{BKPT}}$  line must be negated or the HALT bit in CANMCR $n$  must be cleared. After freeze mode is exited, the FlexCAN resynchronizes with the CAN bus by waiting for 11 consecutive recessive bits before beginning to participate in CAN bus communication.

#### 33.1.3.3 Module Disabled Mode

This mode disables the FlexCAN module; it is entered by setting CANMCR $n$ [MDIS]. If the module is disabled during freeze mode, it shuts down the system clocks, sets the LPMACK bit, and clears the FRZACK bit.

If the module is disabled during transmission or reception, FlexCAN does the following:

- Waits to be in 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 such as arbitration, matching, move-in, and move-out to finish
- Ignores its Rx input pin and drives its Tx pin as recessive

- Shuts down the system clocks

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 disabled. Exiting from this mode is done by negating the MDIS bit, which resumes the clocks and negate the LPMACK bit.

### 33.1.3.4 Loop-back Mode

The module enters this mode when the LPB bit in the control register is set. 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. Transmit and receive interrupts are generated.

### 33.1.3.5 Listen-only Mode

In listen-only mode, transmission is disabled, all error counters are frozen and the module operates in a CAN error passive mode. Only messages acknowledged by another CAN station are received. If FlexCAN detects a message that has not been acknowledged, it flags a BIT0 error (without changing the REC), as if it was trying to acknowledge the message. Because the module does not influence the CAN bus in this mode, the device is capable of functioning like a monitor or for automatic bit-rate detection.

## 33.2 External Signal Description

Each FlexCAN module has two I/O signals connected to the external MPU pins: CAN0TX, CAN0RX, CAN1TX, and CAN1RX. CAN $n$ TX transmits serial data to the CAN bus transceiver, while CAN $n$ RX receives serial data from the CAN bus transceiver.

## 33.3 Memory Map/Register Definition

The FlexCAN module address space is split into 128 bytes starting at the base address, 256 bytes starting at the base address + 0x80, and 256 bytes starting at the base address + 0x880. Out of the lower 128 bytes, only part is occupied by various registers. The second block of 256 bytes are fully used for the message buffer structures, as described in [Section 33.3.9, “Message Buffer Structure.”](#) The upper 256 bytes is used by the individual masking registers.

**Table 33-1. FlexCAN Memory Map**

Address	Register	Width (bits)	Affected by Hard Reset	Affected by Soft Reset	Access	Reset Value	Section/Page
FlexCAN0 FlexCAN1							
<b>Supervisor-only Access Registers</b>							
0xFC02_0000 0xFC02_4000	FlexCAN Module Configuration Register (CANMCR $n$ )	32	Y	Y	R/W	0xD890_000F	<a href="#">33.3.1/33-7</a>
<b>Supervisor/User Access Registers</b>							
0xFC02_0004 0xFC02_4004	FlexCAN Control Register (CANCTRL $n$ )	32	Y	N	R/W	0x0000_0000	<a href="#">33.3.2/33-9</a>
0xFC02_0008 0xFC02_4008	Free Running Timer (TIMER $n$ )	32	Y	Y	R/W	0x0000_0000	<a href="#">33.3.3/33-12</a>
0xFC02_0010 0xFC02_4010	Rx Global Mask (RXGMASK $n$ )	32	Y	N	R/W	0x1FF_FFFF	<a href="#">33.3.4/33-13</a>
0xFC02_0014 0xFC02_4014	Rx Buffer 14 Mask (RX14MASK $n$ )	32	Y	N	R/W	0x1FFF_FFFF	<a href="#">33.3.4/33-13</a>
0xFC02_0018 0xFC02_4018	Rx Buffer 15 Mask (RX15MASK $n$ )	32	Y	N	R/W	0x1FFF_FFFF	<a href="#">33.3.4/33-13</a>
0xFC02_001C 0xFC02_401C	Error Counter Register (ERRCNT $n$ )	32	Y	Y	R/W	0x0000_0000	<a href="#">33.3.6/33-16</a>
0xFC02_0020 0xFC02_4020	Error and Status Register (ERRSTAT $n$ )	32	Y	Y	R/W	0x0000_0000	<a href="#">33.3.6/33-16</a>
0xFC02_0028 0xFC02_4028	Interrupt Mask Register (IMASK $n$ )	32	Y	Y	R/W	0x0000_0000	<a href="#">33.3.7/33-18</a>
0xFC02_0030 0xFC02_4030	Interrupt Flag Register (IFLAG $n$ )	32	Y	Y	R/W	0x0000_0000	<a href="#">33.3.8/33-18</a>
0xFC02_0080 0xFC02_4080	Message Buffers 0–15 (MB0–15)	2048	N	N	R/W	—	<a href="#">33.3.9/33-19</a>
0xFC02_0880 0xFC02_4880	Rx Individual Mask Registers (RXIMR $n$ 0–15)	2048	N	N	R/W	—	<a href="#">33.3.11/33-25</a>

**NOTE**

The FlexCAN has no hard-wired protection against invalid bit/field programming within its registers. Specifically, no protection is provided if the programming does not meet CAN protocol requirements.

Programming the FlexCAN control registers is typically done during system initialization, prior to the FlexCAN becoming synchronized with the CAN bus. The configuration registers can be changed after synchronization by halting the FlexCAN module. This is done when the user sets the CANMCR $n$ [HALT] bit. The FlexCAN responds by setting the CANMCR $n$ [NOTRDY] bit.

### 33.3.1 FlexCAN Configuration Register (CANMCR $n$ )

CANMCR $n$  defines global system configurations, such as the module operation mode 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.

Address: 0xFC02_0000 (CANMCR0) 0x000 (CANMCR1)																Access: Supervisor read/write																																					
<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="width: 2.5%;"><b>R</b></td><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td></tr> <tr> <td><b>W</b></td><td>MDIS</td><td>FRZ</td><td>FEN</td><td>HALT</td><td>NOT RDY</td><td>0</td><td>SOFT RST</td><td>FRZ ACK</td><td>SUPV</td><td>0</td><td>WARN _EN</td><td>LPM ACK</td><td>0</td><td>DOZE</td><td>SRX DIS</td><td>BCC</td></tr> </table>																<b>R</b>	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	<b>W</b>	MDIS	FRZ	FEN	HALT	NOT RDY	0	SOFT RST	FRZ ACK	SUPV	0	WARN _EN	LPM ACK	0	DOZE	SRX DIS	BCC				
<b>R</b>	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16																																					
<b>W</b>	MDIS	FRZ	FEN	HALT	NOT RDY	0	SOFT RST	FRZ ACK	SUPV	0	WARN _EN	LPM ACK	0	DOZE	SRX DIS	BCC																																					
Reset																1	0	0	1																																		
<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="width: 2.5%;"><b>R</b></td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr> <tr> <td><b>W</b></td><td>0</td><td>0</td><td>LPRI_O_EN</td><td>AEN</td><td>0</td><td>0</td><td colspan="2" rowspan="2" style="text-align: center;">IDAM</td><td>0</td><td>0</td><td>0</td><td>0</td><td colspan="4" style="text-align: center;">MAXMB</td></tr> </table>																<b>R</b>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	<b>W</b>	0	0	LPRI_O_EN	AEN	0	0	IDAM		0	0	0	0	MAXMB				0	0	0	0
<b>R</b>	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																																					
<b>W</b>	0	0	LPRI_O_EN	AEN	0	0	IDAM		0	0	0	0	MAXMB																																								
Reset																1	1	1	1																																		

Figure 33-5. FlexCAN Configuration Register (CANMCR $n$ )

Table 33-2. CANMCR $n$  Field Descriptions

Field	Description
31 MDIS	Module disable. This bit controls whether FlexCAN is enabled or not. When disabled, FlexCAN shuts down the FlexCAN clocks that drive the CAN interface and Message Buffer sub-module. This is the only bit in CANMCR $n$ not affected by soft reset. See <a href="#">Section 33.1.3.3, "Module Disabled Mode,"</a> for more information. 0 Enable the FlexCAN module, clocks enabled 1 Disable the FlexCAN module, clocks disabled
30 FRZ	Freeze mode enable. When set, the FlexCAN can enter freeze mode when the $\overline{\text{BKPT}}$ line is asserted or the HALT bit is set. Clearing this bit causes the FlexCAN to exit freeze mode. Refer to <a href="#">Section 33.1.3.2, "Freeze Mode,"</a> for more information. 0 FlexCAN ignores the $\overline{\text{BKPT}}$ signal and the CANMCR $n$ [HALT] bit. 1 FlexCAN module enabled to enter debug mode.
29 FEN	FIFO enable. 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. 0 FIFO not enabled 1 FIFO enabled
28 HALT	Halt FlexCAN. Setting this bit puts the FlexCAN module into freeze mode. It has the same effect as assertion of the $\overline{\text{BKPT}}$ signal. This bit is set after reset and should be cleared after initializing the message buffers and control registers. FlexCAN message buffer receive and transmit functions are inactive until this bit is cleared. While in freeze mode, the CPU has write access to the error counter register (ERRCNT $n$ ) that is otherwise read-only. 0 The FlexCAN operates normally 1 FlexCAN enters freeze mode if FRZ equals 1
27 NOTRDY	FlexCAN not ready. This bit indicates that the FlexCAN is in disable or freeze mode. This bit is read-only and it is cleared after the FlexCAN exits these modes. 0 FlexCAN is in normal mode, listen-only mode, or loop-back mode. 1 FlexCAN is in disable or freeze mode.
26	Reserved, must be cleared.

**Table 33-2. CANMCR $n$  Field Descriptions (continued)**

Field	Description
25 SOFRST	<p>Soft reset. When set, the FlexCAN resets its internal state machines (sequencer, error counters, error flags, and timer) and the host interface registers (CANMCR<math>n</math> [except the MDIS bit], TIMER, ERRCNT, ERRSTAT, IMASK, and IFLAG).</p> <p>The configuration registers that control the interface with the CAN bus are not changed (CANCTRL<math>n</math>, RXGMASK<math>n</math>, RX14MASK<math>n</math>, RX15MASK<math>n</math>). Message buffers are also not changed. This allows SOFRST to be used as a debug feature while the system is running.</p> <p>Because 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 SOFRST bit remains set while reset is pending and is automatically cleared when reset completes. The user should poll this bit to know when the soft reset has completed.</p> <ul style="list-style-type: none"> <li>0 Soft reset cycle completed</li> <li>1 Soft reset cycle initiated</li> </ul>
24 FRZACK	<p>Freeze acknowledge. Indicates that the FlexCAN module has entered freeze mode. The user should poll this bit after freeze mode has been requested, to know when the module has actually entered freeze mode. When freeze mode is exited, this bit is cleared after the FlexCAN prescaler is enabled. This is a read-only bit.</p> <ul style="list-style-type: none"> <li>0 The FlexCAN has exited freeze mode and the prescaler is enabled.</li> <li>1 The FlexCAN has entered freeze mode, and the prescaler is disabled.</li> </ul>
23 SUPV	<p>Supervisor/user data space. Places the FlexCAN registers in supervisor or user data space.</p> <ul style="list-style-type: none"> <li>0 Registers with access controlled by the SUPV bit are accessible in user or supervisor privilege mode.</li> <li>1 Registers with access controlled by the SUPV bit are restricted to supervisor mode.</li> </ul>
22	Reserved, must be cleared.
21 WRN_EN	<p>Warning interrupt enable. Enables the generation of the TWRN_INT and RWRN_INT flags in the Error and Status Register.</p> <ul style="list-style-type: none"> <li>0 TWRN_INT and RWRN_INT bits are always zero, independent of the values in the error counters. No warning interrupt is ever generated.</li> <li>1 TWRN_INT and RWRN_INT bits are set when the respective error counter transition from &lt;96 to = 96</li> </ul>
20 LPMACK	<p>Low power mode acknowledge. Indicates that FlexCAN is disabled. Disabled mode cannot be entered until all current transmission or reception processes have finished, so the CPU can poll the LPMACK bit to know when the FlexCAN has actually entered low power mode. See <a href="#">Section 33.1.3.3, “Module Disabled Mode,”</a> and <a href="#">Chapter 9, “Power Management,”</a> for more information. This bit is read-only.</p> <ul style="list-style-type: none"> <li>0 FlexCAN not disabled.</li> <li>1 FlexCAN is in disabled mode.</li> </ul>
18 DOZE	<p>Doze mode enable. This bit defines whether FlexCAN is allowed to enter low power mode when doze mode is requested at MPU level. This bit is automatically reset when FlexCAN wakes up from doze mode upon detecting activity on the CAN bus (self wake-up enabled).</p> <ul style="list-style-type: none"> <li>0 FlexCAN is not enabled to enter low power mode when doze mode is requested</li> <li>1 FlexCAN is enabled to enter low power mode when doze mode is requested</li> </ul>
19	Reserved, must be cleared.
17 SRX_DIS	<p>Self reception disable. Defines whether FlexCAN is allowed to receive frames transmitted by itself. If this bit is set, frames transmitted by the module are not 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 is generated due to the frame reception.</p> <ul style="list-style-type: none"> <li>0 Self reception enabled</li> <li>1 Self reception disabled</li> </ul>

**Table 33-2. CANMCR $n$  Field Descriptions (continued)**

Field	Description
16 BCC	<p>Backwards compatibility configuration. This bit is provided to support backwards compatibility with legacy FlexCAN software. When this bit is cleared, the following configuration is applied:</p> <ul style="list-style-type: none"> <li>• Individual Rx ID masking is disabled. Instead of individual ID masking per MB, the FlexCAN uses its previous masking scheme with RXGMASK, RX14MASK, and RX15MASK.</li> <li>• The reception queue feature is disabled. Upon receiving a message, if the first MB with a matching ID remains occupied by a previous unread message, FlexCAN does not look for another matching MB. It overrides this MB with the new message and set the CODE field to 0110 (overrun).</li> </ul> <p>Upon reset this bit is cleared, allowing legacy software to work without modification.</p> <p>0 Individual Rx masking and queue feature are disabled 1 Individual Rx masking and queue feature are enabled</p>
15–14	Reserved, must be cleared.
13 LPRIO_EN	<p>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.</p> <p>0 Local priority disabled 1 Local priority enabled</p>
12 AEN	<p>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.</p> <p>0 Abort disabled 1 Abort enabled</p>
11–10	Reserved, must be cleared.
9–8 IDAM	<p>ID acceptance mode. Identifies the format of the elements of the Rx FIFO filter table All elements below are configured at the same time by this field (they are all the same format).</p> <p>00 Format A — One full ID (standard or extended) per filter element 01 Format B — Two full standard IDs or two partial 14-bit extended IDs per filter element 10 Format C — Four partial 8-bit IDs (standard or extended) per filter element 11 Format D — All frames rejected</p>
7–4	Reserved, must be cleared.
3–0 MAXMB	<p>Maximum number of message buffers. Defines the maximum number of message buffers that take part in the matching and arbitration process. The reset value (0xF) is equivalent to 16 message buffer (MB) configuration. This field should be changed only while the module is in freeze mode.</p> <p><b>Note:</b> Maximum MBs in Use = MAXMB + 1</p>

### 33.3.2 FlexCAN Control Register (CANCTRL $n$ )

CANCTRL $n$  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. 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 disabled or in freeze mode. Exceptions are the BOFFMSK, ERRMSK, and BOFFREC bits, which can be accessed at any time.

Address: 0xFC02\_0004 (CANCTRL0)  
0xFC02\_4004 (CANCTRL1)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R W	PRESDIV								RJW		PSEG1			PSEG2		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R W	15 MSK	14 MSK	13 SRC	12 LPB	11 _MSK	10 _MSK	9 _MSK	8 _MSK	7 SMP	6 BOFF REC	5 TSYN	4 LBUF	3 LOM	2 PROPSEG	1 	0 
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 33-6. FlexCAN Control Register (CANCTRLn)

Table 33-3. CANCTRLn Field Descriptions

Field	Description
31–24 PRESDIV	Prescaler division factor. Defines the ratio between the clock source frequency (set by CLK_SRC bit) and the serial clock (S clock) frequency. The S clock period defines the time quantum of the CAN protocol. For the reset value, the S clock frequency is equal to the clock source frequency. The maximum value of this register is 0xFF, that gives a minimum S clock frequency equal to the clock source frequency divided by 256. For more information refer to <a href="#">Section 33.3.21, “Protocol Timing”</a> .
	$\text{S clock frequency} = \frac{f_{\text{sys}/2} \text{ or EXTAL}}{\text{PRESDIV} + 1}$ Eqn. 33-1
23–22 RJW	Resynchronization jump width. Defines the maximum number of time quanta (one time quantum is equal to the S clock period) that a bit time can be changed by one resynchronization. The valid programmable values are 0–3.
	Resync jump width = (RJW + 1) time quanta Eqn. 33-2
21–19 PSEG1	Phase buffer segment 1. 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) time quanta Eqn. 33-3
18–16 PSEG2	Phase buffer segment 2. 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) time quanta Eqn. 33-4
15 BOFFMSK	Bus off interrupt mask. 0 Bus off interrupt disabled 1 Bus off interrupt enabled
14 ERRMSK	Error interrupt mask. 0 Error interrupt disabled 1 Error interrupt enabled
13 CLK_SRC	Clock source. Selects the clock source for the CAN interface to be fed to the prescalar. This bit should only be changed while the module is disabled. 0 Clock source is EXTAL 1 Clock source is the internal bus clock, $f_{\text{sys}/2}$

**Table 33-3. CANCTRL $n$  Field Descriptions (continued)**

Field	Description
12 LPB	Loop back. 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. Transmit and receive interrupts are generated. 0 Loop back disabled 1 Loop back enabled
11 TWRN_MSK	Tx warning interrupt mask. 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 CANMCR is cleared and it is read as zero when WRN_EN is cleared. 0 Tx warning interrupt disabled 1 Tx warning interrupt enabled
10 RWRN_MSK	Rx warning interrupt mask. 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 CANMCR is cleared and it is read as zero when WRN_EN is cleared. 0 Rx warning interrupt disabled 1 Rx warning interrupt enabled
9–8	Reserved, must be cleared.
7 SMP	Sampling mode. Determines whether the FlexCAN module samples each received bit one time or three times to determine its value. 0 One sample, taken at the end of phase buffer segment 1, is used to determine the value of the received bit. 1 Three samples are used to determine the value of the received bit. The samples are taken at the normal sample point and at the two preceding periods of the S-clock; a majority rule is used.
6 BOFFREC	Bus off recovery mode. Defines how FlexCAN recovers from bus off state. If this bit is cleared, automatic recovering from bus off state occurs according to the <i>CAN Specification 2.0B</i> . If the bit is set, automatic recovering from bus off is disabled and the module remains in bus off state until the bit is cleared by the user. If the bit is cleared before 128 sequences of 11 recessive bits are detected on the CAN bus, then bus off recovery happens as if the BOFFREC bit had never been set. If the bit is cleared after 128 sequences of 11 recessive bits occurred, FlexCAN re-synchronizes to the bus by waiting for 11 recessive bits before joining the bus. After clearing, the BOFFREC bit can be set again during bus off, but it is only effective the next time the module enters bus off. If BOFFREC was cleared when the module entered bus off, setting it during bus off is not effective for the current bus off recovery. 0 Automatic recovering from bus off state enabled, according to CAN Spec 2.0B 1 Automatic recovering from bus off state disabled
5 TSYN	Timer synchronize mode. Enables the mechanism that resets the free-running timer each time a message is received in Message Buffer 0. This feature provides the means to synchronize multiple FlexCAN stations with a special SYNC message (global network time). 0 Timer synchronization disabled. 1 Timer synchronization enabled. <b>Note:</b> There can be a bit clock skew of four to five counts between different FlexCAN modules that are using this feature on the same network.
4 LBUF	Lowest buffer transmitted first. Defines the ordering mechanism for message buffer transmission. 0 Message buffer with lowest ID is transmitted first 1 Lowest numbered buffer is transmitted first

**Table 33-3. CANCTRL*n* Field Descriptions (continued)**

Field	Description
3 LOM	<p>Listen-only mode. 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. Only messages acknowledged by another CAN station is received. If FlexCAN detects a message that has not been acknowledged, it flags a BIT0 error (without changing the REC), as if it was trying to acknowledge the message.</p> <p>0 FlexCAN module is in normal active operation; listen-only mode is deactivated            1 FlexCAN module is in listen-only mode operation</p>
2-0 PROPSEG	<p>Propagation segment. Defines the length of the propagation segment in the bit time. The valid programmable values are 0–7.</p> <p style="text-align: right;"><i>Eqn. 33-5</i></p>

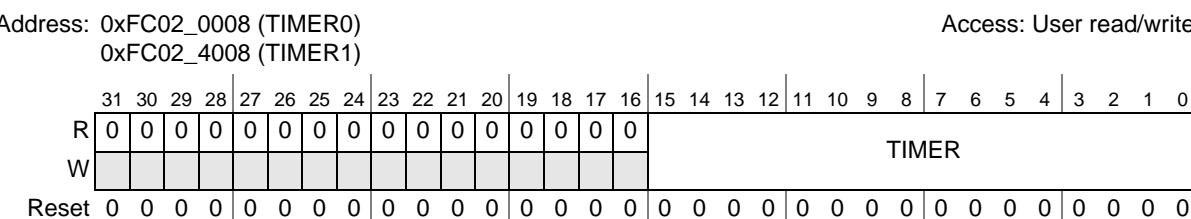
### 33.3.3 FlexCAN Free Running Timer Register (TIMERn)

This register represents a 16-bit free running counter that can be read and written to 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 received or transmitted bit. 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 (ID) field of any frame on the CAN bus. This captured value is written into the **TIMESTAMP** 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, 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 takes some time to be actually written to the register. If desired, software can poll the register to discover when the data was actually written.



**Figure 33-7. FlexCAN Timer Register (TIMERn)**

**Table 33-4. TIMERn Field Descriptions**

Field	Description
31–16	Reserved, must be cleared.
15–0 TIMER	Free running timer. Captured at the beginning of the identifier (ID) field of any frame on the CAN bus. This captured value is written into the TIMESTAMP entry in a message buffer after a successful reception or transmission of a message.

### 33.3.4 Rx Mask Registers (RXGMASKn, RX14MASKn, RX15MASKn)

#### NOTE

These registers are provided for legacy software by clearing. For more configurability use the individual masking registers instead by setting CANMCR[BCC]. See [Section 33.3.11, “Rx Individual Masking Registers \(RXIMR0–15\),”](#) for more details.

These registers are used as acceptance masks for received frame IDs if CANMCR[BCC] is cleared. (If CANMCR[BCC] is set, these registers are reserved and do not affect FlexCAN operation.) Three masks are defined: a global mask (RXGMASKn) used for Rx buffers 0–13 and two separate masks for buffers 14 (RX14MASKn) and 15 (RX15MASKn). The meaning of each mask bit is the following:

MI<sub>n</sub> bit = 0: The corresponding incoming ID bit is don’t care.

MI<sub>n</sub> bit = 1: The corresponding ID bit is checked against the incoming ID bit, to see if a match exists.

These masks are used for standard and extended ID formats. The value of the mask registers should not be changed while in normal operation (only while in freeze mode), as locked frames that matched a message buffer (MB) through a mask may be transferred into the MB (upon release) but may no longer match.

See [Section 33.4.2, “Mask Misalignment for Rx FIFO”,](#) for an issue regarding the mask registers when using the receive FIFO.

**Table 33-5. Mask Examples for Normal/Extended Messages**

	Base ID ID28.....ID18	IDE	Extended ID ID17.....ID0	Match
MB2-ID	1 1 1 1 1 1 1 0 0 0	0		
MB3-ID	1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
MB4-ID	0 0 0 0 0 0 1 1 1 1	0		
MB5-ID	0 0 0 0 0 0 1 1 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
MB14-ID	1 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	
Rx_Global_Mask	1 1 1 1 1 1 1 1 1 0		1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1	
Rx_Msg in <sup>1</sup>	1 1 1 1 1 1 1 0 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	MB3 <sup>1</sup>
Rx_Msg in <sup>2</sup>	1 1 1 1 1 1 1 0 0 1	0		MB2 <sup>2</sup>
Rx_Msg in <sup>3</sup>	1 1 1 1 1 1 1 0 0 1	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 0	3
Rx_Msg in <sup>4</sup>	0 1 1 1 1 1 1 0 0 0	0		4

**Table 33-5. Mask Examples for Normal/Extended Messages (continued)**

	Base ID ID28.....ID18	IDE	Extended ID ID17.....ID0	Match
Rx_Msg in <sup>5</sup>	0 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	MB14 <sup>5</sup>
RX14MASK	0 1 1 1 1 1 1 1 1 1		1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0	
Rx_Msg in <sup>6</sup>	1 0 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	6
Rx_Msg in <sup>7</sup>	0 1 1 1 1 1 1 0 0 0	1	0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	MB14 <sup>7</sup>

<sup>1</sup> Match for Extended Format (MB3).

<sup>2</sup> Match for Normal Format. (MB2).

<sup>3</sup> Mismatch for MB3 because of ID0

<sup>4</sup> Mismatch for MB2 because of ID28

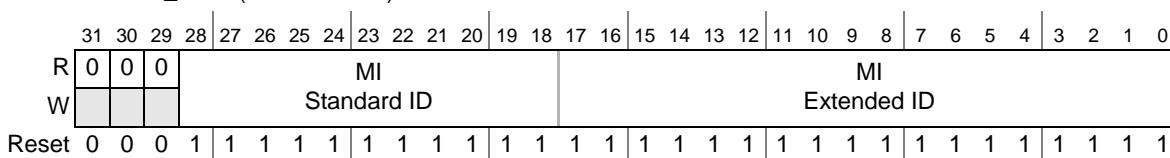
<sup>5</sup> Mismatch for MB3 because of ID28. Match for MB14 (Uses RX14MASKn)

<sup>6</sup> Mismatch for MB14 because of ID23 (Uses RX14MASKn)

- Mismatch for MB14 because of ID27

Address: 0xFC02\_0010 (RXGMASK0)  
0xFC02\_0014 (RX14MASK0)  
0xFC02\_0018 (RX15MASK0)  
0xFC02\_4010 (RXGMASK1)  
0xFC02\_4014 (RX14MASK1)  
0xFC02\_4018 (RX15MASK1)

Access: User read/write



**Figure 33-8. FlexCAN Rx Mask Registers (RXGMASK $n$ , RX14MASK $n$ , RX15MASK $n$ )**

**Table 33-6. RXxxMASKn Field Descriptions**

Field	Description
31–29	Reserved, must be cleared.
28–18 MI28–18	Standard ID mask bits. These bits are the same mask bits for the Standard and Extended Formats.
17–0 MI17–0	Extended ID mask bits. These bits are used to mask comparison only in Extended Format.

### 33.3.5 FlexCAN Error Counter Register (ERRCNT $n$ )

This register has two 8-bit fields reflecting the value of two FlexCAN error counters: transmit error counter (TXECTR) and receive error counter (RXECTR). 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  $\text{ERRCNT}_n$  register while in freeze mode is an indirect operation. The data is first written to an auxiliary register, 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 takes 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 TXECTR or RXECTR increases to be greater than or equal to 128, the FLTCONF field in the error and status register (ERRSTAT $n$ ) is updated to reflect error-passive state.
- If the FlexCAN state is error-passive, and TXECTR or RXECTR decrements to a value less than or equal to 127 while the other already satisfies this condition, the ERRSTAT $n$ [FLTCONF] field is updated to reflect error-active state.
- If the value of TXECTR increases to be greater than 255, the ERRSTAT $n$ [FLTCONF] field is updated to reflect bus off state, and an interrupt may be issued. The value of TXECTR is then reset to zero.
- If FlexCAN is in bus off state, then TXECTR is cascaded together with another internal counter to count the 128th occurrences of 11 consecutive recessive bits on the bus. Hence, TXECTR is reset to zero and counts in a manner where the internal counter counts 11 such bits and then wraps around while incrementing the TXECTR. When TXECTR reaches the value of 128, the ERRSTAT $n$ [FLTCONF] field is updated to be error-active, and both error counters are reset to zero. At any instance of a dominant bit following a stream of less than 11 consecutive recessive bits, the internal counter resets itself to zero without affecting the TXECTR value.
- If during system start-up, only one node is operating, then its TXECTR increases in each message it is trying to transmit, as a result of acknowledge errors (indicated by the ERRSTAT $n$ [ACKERR] bit). After the transition to error-passive state, the TXECTR does not increment anymore by acknowledge errors. Therefore, the device never goes to the bus off state.
- If the RXECTR 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.

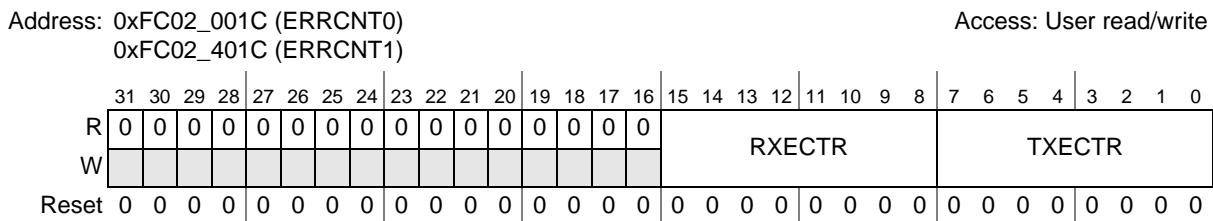


Figure 33-9. FlexCAN Error Counter Register (ERRCNT $n$ )

Table 33-7. ERRCNT $n$  Field Descriptions

Field	Description
31–16	Reserved, must be cleared.

**Table 33-7. ERRCNT<sub>n</sub> Field Descriptions (continued)**

Field	Description
15–8 RXECTR	Receive error counter. Indicates current number of receive errors.
7–0 TXECTR	Transmit error counter. Indicates current number of transmit errors.

### 33.3.6 FlexCAN Error and Status Register (ERRSTAT<sub>n</sub>)

ERRSTAT<sub>n</sub> reflects various error conditions, some general status of the device, and is the source of three interrupts to the CPU. The reported error conditions (bits 15:10) are those occurred since the last time the CPU read this register. The read action clears bits 15–10. Bits 9–3 are status bits.

Most bits in this register are read only, except for BOFFINT and ERRINT, which are interrupt flags that can be cleared by writing 1 to them. Writing 0 has no effect. Refer to [Section 33.4.1, “Interrupts.”](#)

Address: 0xFC02\_0020 (ERRSTAT0)  
0xFC02\_4020 (ERRSTAT1)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	TWRN_INT	RWRN_INT
W															w1c	w1c
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	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														w1c	w1c	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 33-10. FlexCAN Error and Status Register (ERRSTAT<sub>n</sub>)****Table 33-8. ERRSTAT<sub>n</sub> Field Descriptions**

Field	Description
31–18	Reserved, must be cleared.
17 TWRN_INT	Tx warning interrupt flag. If the WRN_EN bit in MCR is set, the TWRN_INT bit is set when the TX_WRN flag transitions 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 one to it. Writing zero has no effect. 0 No such occurrence 1 The Tx error counter transition from < 96 to = 96
16 RWRN_INT	Rx warning interrupt flag. If the WRN_EN bit in MCR is set, the RWRN_INT bit is set when the RX_WRN flag transitions 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 one to it. Writing zero has no effect. 0 No such occurrence 1 The Rx error counter transition from < 96 to = 96

**Table 33-8. ERRSTAT $n$  Field Descriptions (continued)**

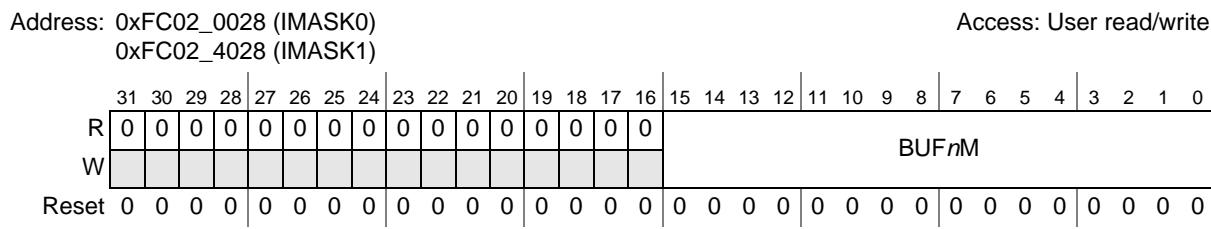
Field	Description
15 BIT1ERR	Bit1 error. Indicates inconsistency between the transmitted and received bit in a message. 0 No transmit bit error 1 At least one bit sent as recessive was received as dominant <b>Note:</b> The transmit bit error field is not modified during the arbitration field or the ACK slot bit time of a message, or by a transmitter that detects dominant bits while sending a passive error frame.
14 BIT0ERR	Bit0 error. Indicates inconsistency between the transmitted and received bit in a message. 0 No transmit bit error 1 At least one bit sent as dominant was received as recessive
13 ACKERR	Acknowledge error. Indicates whether an acknowledgment has been correctly received for a transmitted message. 0 No ACK error was detected since the last read of this register. 1 An ACK error was detected since the last read of this register.
12 CRCERR	Cyclic redundancy check error. Indicates whether or not a CRC error has been detected by the receiver. 0 No CRC error was detected since the last read of this register. 1 A CRC error was detected since the last read of this register.
11 FRMERR	Message form error. 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 form error was detected since the last read of this register. 1 A form error was detected since the last read of this register.
10 STFERR	Bit stuff error. 0 No bit stuffing error was detected since the last read of this register. 1 A bit stuffing error was detected since the last read of this register.
9 TXWRN	Transmit error status flag. Reflects the status of the FlexCAN transmit error counter. 0 Transmit error counter < 96 1 TXErrCounter ≥ 96
8 RXWRN	Receiver error status flag. Reflects the status of the FlexCAN receive error counter. 0 Receive error counter < 96 1 RxErrCounter ≥ 96
7 IDLE	Idle status. Indicates when there is activity on the CAN bus. 0 The CAN bus is not idle. 1 The CAN bus is idle.
6 TXRX	Transmit/receive status. Indicates when the FlexCAN module is transmitting or receiving a message. TXRX has no meaning when IDLE equals 1. 0 The FlexCAN is receiving a message if IDLE equals 0. 1 The FlexCAN is transmitting a message if IDLE equals 0.
5–4 FLTCONF	Fault confinement state. Indicates the confinement state of the FlexCAN module, as shown below. If the CANCTRL $n$ [LOM] bit is set, FLTCONF indicates error-passive. Because the CANCTRL $n$ register is not affected by soft reset, the FLTCONF field is not affected by soft reset if the LOM bit is set. 00 Error active 01 Error passive 1x Bus off
3	Reserved, must be cleared.
2 BOFFINT	Bus off interrupt. Used to request an interrupt when the FlexCAN enters the bus off state. The user must write a 1 to clear this bit. Writing 0 has no effect. 0 No bus off interrupt requested. 1 This bit is set when the FlexCAN state changes to bus off. If the CANCTRL $n$ [BOFFMSK] bit is set an interrupt request is generated. This interrupt is not requested after reset.

**Table 33-8. ERRSTAT $n$  Field Descriptions (continued)**

Field	Description
1 ERRINT	Error interrupt. Indicates that at least one of the ERRSTAT $n$ [15:10] bits is set. The user must write a 1 to clear this bit. Writing 0 has no effect. 0 No error interrupt request. 1 At least one of the error bits is set. If the CANCTRL $n$ [ERRMSK] bit is set, an interrupt request is generated.
0	Reserved, must be cleared.

### 33.3.7 Interrupt Mask Register (IMASK $n$ )

IMASK $n$  contains one interrupt mask bit per buffer. It enables the CPU to determine which buffer generates an interrupt after a successful transmission/reception (when the corresponding IFLAG $n$  bit is set).

**Figure 33-11. FlexCAN Interrupt Mask Register (IMASK $n$ )****Table 33-9. IMASK $n$  Field Descriptions**

Field	Description
31–16	Reserved, must be cleared.
15–0 BUFnM	Buffer interrupt mask. Enables the respective FlexCAN message buffer (MB0 to MB15) interrupt. These bits allow the CPU to designate which buffers generate interrupts after successful transmission/reception. 0 The interrupt for the corresponding buffer is disabled. 1 The interrupt for the corresponding buffer is enabled. <b>Note:</b> Setting or clearing an IMASK $n$ bit can assert or negate an interrupt request, if the corresponding IFLAG $n$ bit is set.

### 33.3.8 Interrupt Flag Register (IFLAG $n$ )

IFLAG $n$  contains one interrupt flag bit per buffer. Each successful transmission/reception sets the corresponding IFLAG $n$  bit and, if the corresponding IMASK $n$  bit is set, generates an interrupt.

The interrupt flag is cleared by writing a 1, while writing 0 has no effect.

When the AEN bit in the CANMCR is set (Abort enabled), while the IFLAG2 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 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.

Address: 0xFC02\_0030 (IFLAG0)  
0xFC02\_4030 (IFLAG1)

Access: User read/write

**Figure 33-12. FlexCAN Interrupt Flags Register (IFLAGn)**

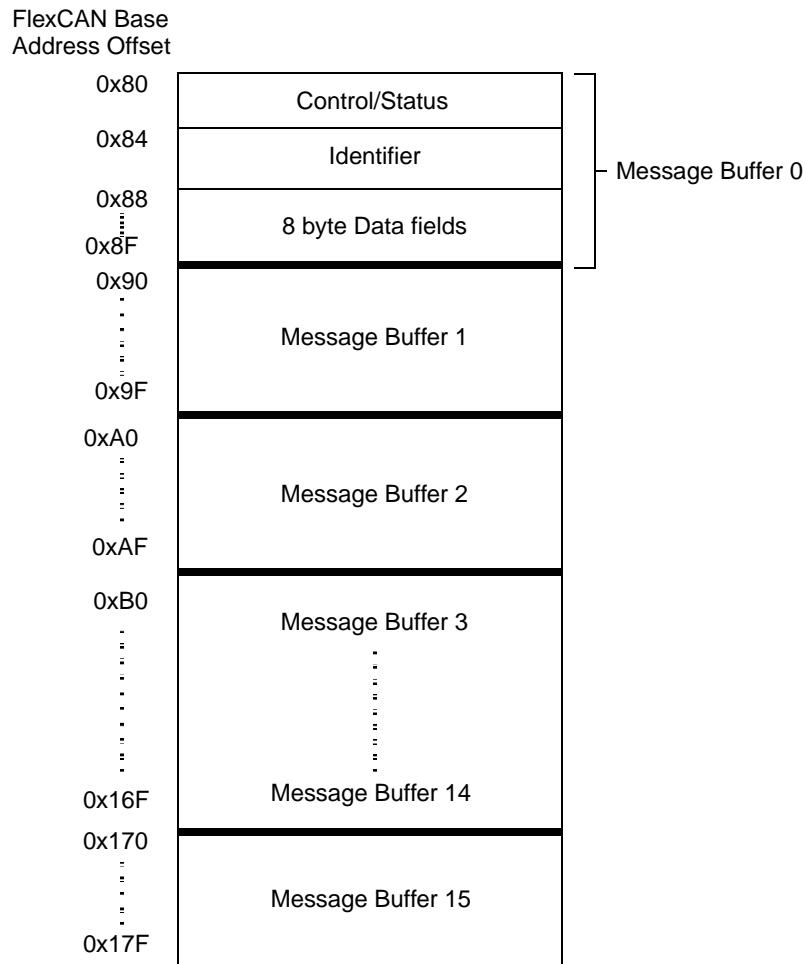
**Table 33-10. IFLAG*n* Field Descriptions**

Field	Description
31–16	Reserved, must be cleared.
15–8 BUF15I– BUF8I	<p>Buffer interrupt flag. Indicates a successful transmission/reception for the corresponding message buffer. If the corresponding IMASK<math>n</math> bit is set, an interrupt request is generated. The user must write a 1 to clear an interrupt flag; writing 0 has no effect.</p> <p>0 No such occurrence. 1 The corresponding buffer has successfully completed transmission or reception.</p>
7 BUF7I	<p>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).</p> <p>0 No such occurrence 1 MB7 completed transmission/reception or FIFO overflow</p>
6 BUF6I	<p>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).</p> <p>0 No such occurrence 1 MB6 completed transmission/reception or FIFO almost full</p>
5 BUF5I	<p>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.</p> <p>0 No such occurrence 1 MB5 completed transmission/reception or frames available in the FIFO</p>
4–0 BUF4I– BUF0I	<p>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.</p> <p>0 No such occurrence 1 Corresponding MB completed transmission/reception</p>

### 33.3.9 Message Buffer Structure

The message buffer memory map starts at an offset of 0x80 from the FlexCAN's base address (CAN0: 0xFC02\_0000 or CAN1: 0xFC02\_4000). The 256-byte message buffer space is fully used by the 16 message buffer structures.

Each message buffer consists of a control and status field that configures the message buffer, an identifier field for frame identification, and up to 8 bytes of data.

**Figure 33-13. FlexCAN Message Buffer Memory Map**

The message buffer structure used by the FlexCAN module is shown in [Figure 33-14](#). Standard and extended frames used in the *CAN Specification Version 2.0, Part B* are represented. A standard frame is represented by the 11-bit standard identifier, and an extended frame is represented by the combined 29-bits of the standard identifier (11 bits) and the extended identifier (18 bits).

	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				
0x0					CODE		SRR	IDE	RTR		LENGTH																									
0x4	PRI	O																																		
0x8																																				
0xC																																				

**Figure 33-14. Message Buffer Structure for Extended and Standard Frames**

**Table 33-11. Message Buffer Field Descriptions**

Field	Description
31–28	Reserved, must be cleared.
27–24 CODE	Message buffer code. 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 <a href="#">Table 33-12</a> and <a href="#">Table 33-13</a> . See <a href="#">Section 33.3.12, “Functional Overview,”</a> for additional information.
23	Reserved, must be cleared.
22 SRR	Substitute remote request. Fixed recessive bit, used only in extended format. It must be set by the user for transmission (Tx Buffers) and is stored with the value received on the CAN bus for Rx receiving buffers. It can be received as 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
21 IDE	ID extended bit. Identifies whether the frame format is standard or extended. 0 Standard frame format 1 Extended frame format
20 RTR	Remote transmission request. 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
19–16 LENGTH	Length of data in bytes. Indicates the length (in bytes) of the Rx or Tx data; data is located in offset 0x8 through 0xF of the MB space (see <a href="#">Figure 33-14</a> ). In reception, this field is written by the FlexCAN module, copied from the DLC (data length code) field of the received frame. DLC is defined by the <i>CAN Specification</i> and refers to the data length of the actual frame before it is copied into the message buffer. In transmission, this field is written by the CPU and is used as the DLC field value of the frame to be transmitted. When RTR is set, the frame to be transmitted is a remote frame and is transmitted without the DATA field, regardless of the LENGTH field.
15–0 TIME STAMP	Free-running counter time stamp. Stores the value of the free-running timer which is captured when the beginning of the identifier (ID) field appears on the CAN bus.
31–29 PRIO	Local priority. Only used when LPPIO_EN bit is set in CANMCR 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 <a href="#">Section 33.3.14, “Arbitration Process”</a> .
28–0 ID	Standard frame identifier: In standard frame format, only the 11 most significant bits (28 to 18) are used for frame identification in receive and transmit cases. The 18 least significant bits are ignored.
	Extended frame identifier: In extended frame format, all bits (the 11 bits of the standard frame identifier and the 18 bits of the extended frame identifier) are used for frame identification in receive and transmit cases.
31–24, 23–16, 15–8, 7–0 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 provides the data to be transmitted within the frame.

**Table 33-12. 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 control & status (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 remains FULL.
		0110	If the MB is FULL and a new frame should be written into this MB before the CPU had time to read it, the MB is overwritten, and the code is automatically updated to OVERRUN.
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 is overwritten again, and the code remains OVERRUN.
0XY1 <sup>1</sup>	BUSY: Flexcan is updating the contents of the MB with a new receive frame. The CPU should not try to 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).

<sup>1</sup> For transmit message buffers (see Table 33-13), the BUSY bit should be ignored upon read, except when CANMCR[AEN] is set.

**Table 33-13. Message Buffer Code for Tx Buffers**

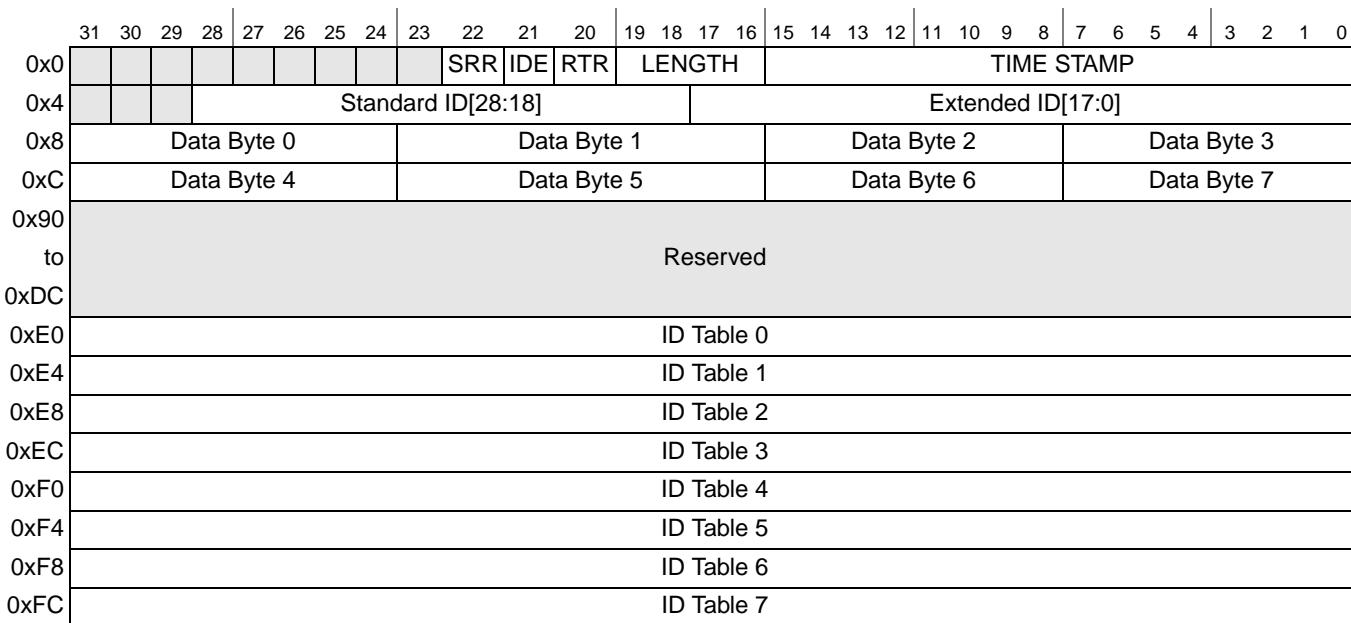
MBn[RTR]	Initial Tx Code	Code After Successful Transmission	Description
X	1000	—	INACTIVE: Message buffer not ready for transmit and participates in the arbitration process.
X	1001	—	ABORT: MB was configured as Tx and CPU aborted the transmission. This code is only valid when AEN bit in CANMCR is set. MB does not participate in the arbitration process.
0	1100	1000	Data frame to be transmitted once, unconditionally. After transmission, the MB automatically returns to the INACTIVE state.

**Table 33-13. Message Buffer Code for Tx Buffers (continued)**

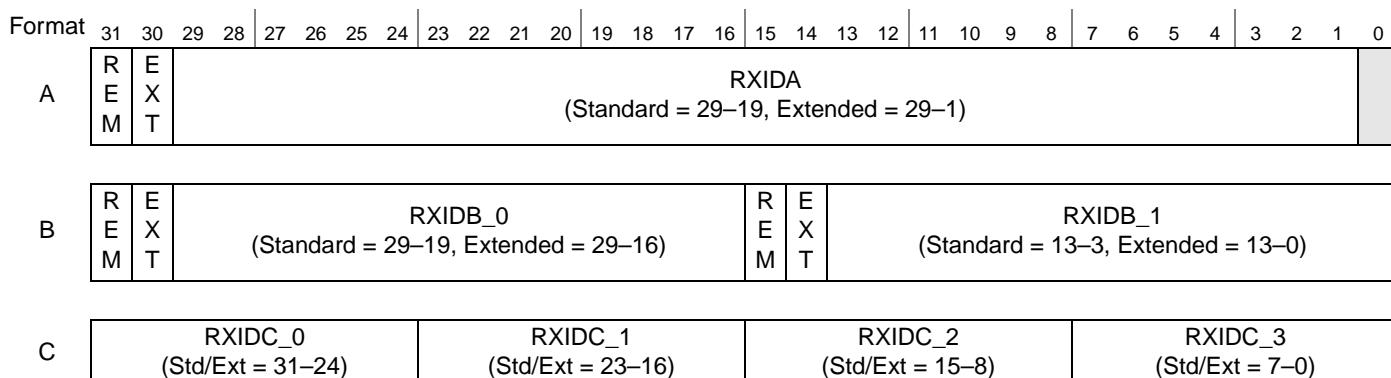
<b>MBn[RTR]</b>	<b>Initial Tx Code</b>	<b>Code After Successful Transmission</b>	<b>Description</b>
1	1100	0100	Remote frame to be transmitted unconditionally once, and message buffer becomes an Rx message buffer with the same ID for data frames.
0	1010	1010	Transmit a data frame when a remote request frame with the same ID is received. This message buffer participates simultaneously in 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 message buffer 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 automatically written to the message buffer as a result of match to a remote request frame. The data frame is transmitted unconditionally once, and then the code automatically returns to 1010. The CPU can also write this code with the same effect.

### 33.3.10 Rx FIFO Structure

When the FEN bit is set in the CANMCR, the memory area from 0x80–0xFC (which is normally occupied by MBs 0 to 7) is used by the reception FIFO engine. [Figure 33-15](#) 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 33-16](#) shows the three different formats that the elements of the ID table can assume, depending on the IDAM field of the MCR. Note that all elements of the table must have the same format. See [Section 33.3.18, “Rx FIFO”](#), for more information.



**Figure 33-15. Rx FIFO Structure**



**Figure 33-16.** ID Table 0-7

**Table 33-14. Message Buffer Field Descriptions**

Field	Description
REM	Remote frame. 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 are used for frame identification. In the extended frame format, all bits are used.

**Table 33-14. Message Buffer Field Descriptions (continued)**

Field	Description
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) 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.

### 33.3.11 Rx Individual Masking Registers (RXIMR0–15)

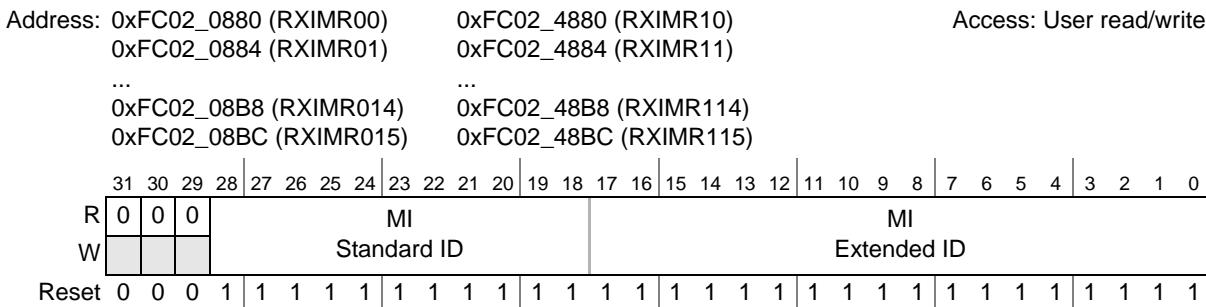
These registers are used as acceptance masks for received frame IDs if CANMCR[BCC] is set. (If CANMCR[BCC] is clear, these registers are reserved and do not affect FlexCAN operation.) One mask register is provided for each message buffer for individual ID masking per MB. The meaning of each mask bit is the following:

MI<sub>n</sub> bit = 0: The corresponding incoming ID bit is don't care.

MI<sub>n</sub> bit = 1: The corresponding ID bit is checked against the incoming ID bit, to see if a match exists.

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. Also, they can only be accessed by the CPU while the module is in freeze mode (CANMCR[FRZ, HALT] are set). Out of freeze mode, write accesses are blocked and read accesses return all zeros. Furthermore, if the CANMCR[BCC] bit cleared, any read or write operation to these registers results in access error.

These masks are used for standard and extended ID formats.

**Figure 33-17. FlexCAN Rx Individual Masking Registers (RXIMR0–15)****Table 33-15. RXxxMASKn Field Descriptions**

Field	Description
31–29	Reserved, must be cleared.
28–18 MI28–18	Standard ID mask bits. These bits are the same mask bits for the Standard and Extended Formats.
17–0 MI17–0	Extended ID mask bits. These bits are used to mask comparison only in Extended Format.

### 33.3.12 Functional Overview

The FlexCAN module is flexible in that each one of its 16 message buffers (MBs) can be assigned as a transmit buffer or a receive buffer. Each MB, which is up to 8 bytes long, is also assigned an interrupt flag bit that indicates successful completion of transmission or reception.

An arbitration algorithm decides the prioritization of MBs to be transmitted based on the message ID or the MB ordering. 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. A reception queue can be implemented by programming the same ID on more than one receiving MB. Data coherency mechanisms are implemented to guarantee data integrity during MB manipulation by the CPU.

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 33-12](#)). Similarly, a Tx MB with a 1000 code is inactive (refer to [Table 33-13](#)). An MB not programmed with 0000 or 1000 is temporarily deactivated (does not participate in the current arbitration/matching run) when the CPU writes to the C/S field of that MB.

### 33.3.13 Transmit Process

The CPU prepares or changes an MB for transmission by writing the following:

1. Control/status word to hold Tx MB inactive (CODE = 1000)
2. ID word
3. Data bytes
4. Control/status word (active CODE, LENGTH)

#### NOTE

The first and last steps are mandatory.

The first write to the control/status word is important in case there was pending reception or transmission. The write operation immediately deactivates the MB, removing it from any currently ongoing arbitration or ID matching processes, giving time for the CPU to program the rest of the MB (see [Section 33.3.17.2, “Message Buffer Deactivation”](#)). After the MB is activated in the fourth step, it participates in 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 (TIMER $n$ ) is written into the message buffer’s time stamp field, the code field in the control and status word is updated, a status flag is set in the IFLAG $n$  register, and an interrupt is generated if allowed by the corresponding IMASK $n$  register bit. The new code field after transmission depends on the code that was used to activate the MB in step four (see [Table 33-13](#)).

### 33.3.14 Arbitration Process

The arbitration process is an algorithm executed by the message buffer management (MBM) that scans the entire MB memory looking for the highest priority message to be transmitted. All MBs programmed as

transmit buffers are scanned to find the lowest ID or the lowest MB number or the highest priority, depending on the CANCTRL $n$ [LBUF, LPRIOR\_EN] bits.

#### NOTE

If CANCTRL $n$ [LBUF] is cleared, 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.

The arbitration process is triggered in the following events:

- During the CRC field of 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 set, the LPRIOR\_EN bit has no effect and the lowest numbered buffer is transmitted first.

When LBUF and LPRIOR\_EN are cleared, the MB with the lowest ID is transmitted first but.

When LBUF is cleared and LPRIOR\_EN is set, the PRIO bits augment the ID used during the arbitration process. With this extended ID concept, arbitration is 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 determines the priority of transmission. If two or more MBs have the same priority (3 extra bits) and the same regular ID, the lowest MB is transmitted first.

After 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 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 8 data bytes, even if the data length code (DLC) value is bigger.

### 33.3.15 Receive Process

The CPU prepares or changes an MB for frame reception by writing the following:

1. 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. If backwards compatibility is

desired (AEN in MCR cleared), write ‘1000’ to the Code field to inactivate the MB, but then the pending frame may be transmitted without notification. If the MB already programmed as a receiver, write ‘0000’ to the Code field of the Control and Status word to keep the MB inactive.

2. ID word
3. Control/status word to mark the Rx MB as active and empty (CODE = 0100)

#### **NOTE**

The first and last steps are mandatory.

After the MB is activated in the third step, it is able to receive CAN frames that match the programmed ID. At the end of a successful reception:

- The value of the free running timer (TIMERn) 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 33-12](#)), and
- A status flag is set in the IFLAGn register and an interrupt is generated if allowed by the corresponding IMASKn bit.

The CPU should read a receive frame from its MB by reading the following:

1. Control/status word (mandatory—activates internal lock for this buffer)
2. ID (optional—needed only if a mask was used)
3. Data field words
4. Free-running timer (Releases internal lock —optional)

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. 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.

The CPU should synchronize to frame reception by an IFLAGn bit for the specific MB (see [Section 33.3.8, “Interrupt Flag Register \(IFLAGn\)”](#)), and not by the control/status word CODE field for that MB. Polling the CODE field does not work because after a frame was received and the CPU services the MB (by reading the C/S word followed by unlocking the MB), the CODE field does not return to EMPTY. It remains FULL, as explained in [Table 33-12](#). 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 poll by directly reading the C/S word of the MBs. Instead, read the IFLAGn register.

The received identifier 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.

#### **33.3.15.1 Self-Received Frames**

Self-received frames are frames that are sent by the FlexCAN and received by itself. The FlexCAN sends a frame externally through the physical layer onto the CAN bus. If the ID of the frame matches the ID of the FlexCAN MB, the frame is received by the FlexCAN. Such a frame is a self-received frame. FlexCAN

does not receive frames transmitted by itself if another device on the CAN bus has an ID that matches the FlexCAN Rx MB ID. Also, if SRX\_DIS in CANMCR is set, FlexCAN does not store frames transmitted by itself in any MB, even if it contains a matching MB, and no interrupt flag or interrupt signal is generated due to the frame reception.

### 33.3.16 Matching Process

The matching process is an algorithm that scans the entire MB memory looking for Rx MBs programmed with the same ID as the one received from the CAN bus. Only MBs programmed to receive participate in the matching process for received frames.

While the ID, DLC and data fields are retrieved from the CAN bus, they are stored temporarily in the serial message buffer. The matching process takes place during the CRC field. If a matching ID is found in one of the MBs, the contents of the SMB are transferred to the matched MB during the sixth 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.

An MB with a matching ID is free to receive a new frame if the MB is not locked (see [Section 33.3.17.3, “Locking and Releasing Message Buffers”](#)). The CODE field is EMPTY, FULL, or OVERRUN but the CPU has already serviced the MB (read the C/S word and then unlocked the MB).

For example, suppose that there are two MBs with the same ID and FlexCAN starts receiving messages with that ID. These MBs are the second and the fifth in the array. When the first message arrives, the matching algorithm finds 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 finds MB number 2 again. However, it is not free to receive, so it keeps 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 to allow more time to the CPU for servicing the MBs. By programming more than one MB with the same ID, received messages are 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 CANMCR[BCC] bit is cleared, the matching algorithm stops at the first MB with a matching ID that it finds, whether this MB is free or not. As a result, the message queueing feature does not work if the BCC bit is cleared.

Matching to a range of IDs is possible by using ID acceptance masks. FlexCAN supports individual masking per MB. Please refer to [Section 33.3.11, “Rx Individual Masking Registers \(RXIMR0–15\)”](#). FlexCAN also supports an alternate masking scheme with only three mask registers (RXGMASK, RX14MASK, and RX15MASK) for backwards compatibility. This alternate masking scheme is enabled when the CANMCR[BCC] bit is cleared. 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.

See [Section 33.4.2, “Mask Misalignment for Rx FIFO”](#), for an issue regarding the mask registers when using the receive FIFO.

### 33.3.17 Message Buffer Managing

To maintain data coherency and FlexCAN proper operation, the CPU must obey the rules described in [Section 33.3.13, “Transmit Process”](#) and [Section 33.3.15, “Receive Process.”](#) Any form of CPU accessing a MB structure within FlexCAN other than those specified may cause FlexCAN to behave in an unpredictable way.

#### 33.3.17.1 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 setting the AEN bit in the CANMCR.

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 set 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).

### 33.3.17.2 Message Buffer Deactivation

If the CPU wants to change the function of an active MB, the recommended procedure is to put the module into freeze mode and then change the CODE field of that MB. This is a safe procedure because the FlexCAN waits for pending CAN bus and MB moving activities to finish before entering freeze mode. Nevertheless, a mechanism is 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 C/S word of an MB causes that MB to be excluded from the transmit or receive processes during the current matching or arbitration round. This mechanism is called MB deactivation. It 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, that MB is deactivated.

Even with the coherence mechanism described above, writing to the C/S 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 continues looking for another matching MB within the ones it has not scanned yet. If it can not find one, the message is 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 is lost even if the second matching MB was free to receive.
- If a Tx MB containing the lowest ID is deactivated after the FlexCAN has scanned it, the FlexCAN looks for another winner within the MBs that it has not yet scanned. Therefore, it may transmit an MB that may not have the lowest ID at the time because a lower ID might be present 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.

### 33.3.17.3 Locking and Releasing Message Buffers

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 EMPTY1 (0100). Also, Tx MBs can not be locked.

Suppose, for example, that 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 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 remains in the SMB waiting for the MB to be unlocked, and only then, is it 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 is no indication of lost messages 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 set. 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 cleared.

If the BUSY bit is set or if the MB is empty, then reading the control and status word does not lock the MB.

**NOTE**

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 is not transferred to the MB anymore.

### 33.3.18 Rx FIFO

The receive-only FIFO is enabled by asserting the FEN bit in the CANMCR. 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 33.3.10, “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 six 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 five 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

eight 32-bit registers that can be configured to one of the following formats (see also [Section 33.3.10, ‘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 RXIMR8, continue to affect the regular MBs, starting from MB8. If the BCC 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 RX14MASK, element 7 is affected by RX15MASK and the other elements (0 to 5) are affected by RXGMASK.

### **33.3.19 CAN Protocol Related Frames**

#### **33.3.19.1 Remote Frames**

The remote frame is a message frame transmitted to request a data frame. The FlexCAN can be configured to transmit a data frame automatically in response to a remote frame, or to transmit a remote frame and then wait for the responding data frame to be received.

When transmitting a remote frame, the user initializes a message buffer as a transmit message buffer with the RTR bit set. After this remote frame is transmitted successfully, the transmit message buffer automatically becomes a receive message buffer, with the same ID as the remote frame that was transmitted.

When a remote frame is received by the FlexCAN, the remote frame ID is compared to the IDs of all transmit message buffers programmed with a CODE of 1010. If there is an exact matching ID, the data frame in that message buffer is transmitted. If the RTR bit in the matching transmit message buffer is set, the FlexCAN transmits a remote frame as a response.

A received remote frame is not stored in a receive message buffer. It is only used to trigger the automatic transmission of a frame in response. The mask registers are not used in remote frame ID matching. All ID bits (except RTR) of the incoming received frame must match for the remote frame to trigger a response transmission. The matching message buffer immediately enters the internal arbitration process, but is considered as a normal Tx MB, with no higher priority. The data length of this frame is independent of the data length code (DLC) field in the remote frame that initiated its transmission.

#### **33.3.19.2 Overload Frames**

Overload frame transmissions are not initiated by the FlexCAN unless certain conditions are detected on the CAN bus. These conditions include detection of a dominant bit in the following:

- First or second bit of intermission
- Seventh (last) bit of the end-of-frame (EOF) field in receive frames
- Eighth (last) bit of the error frame delimiter or overload frame delimiter

### 33.3.20 Time Stamp

The value of  $\text{TIMER}_n$  is sampled at the beginning of the identifier field on the CAN bus. For a message being received, the time stamp is stored in the **TIMESTAMP** entry of the receive message buffer at the time the message is written into that buffer. For a message being transmitted, the **TIMESTAMP** entry is written into the transmit message buffer after the transmission has completed successfully.

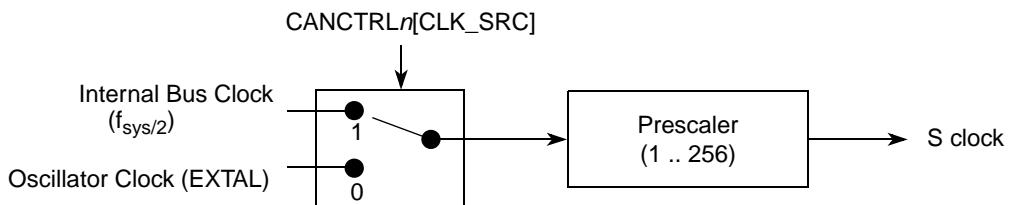
The free-running timer can optionally be reset upon the reception of a frame into message buffer 0. This feature allows network time synchronization to be performed. See the **CANCTRL $_n$ [TSYN]** bit.

### 33.3.21 Protocol Timing

The FlexCAN module **CANCTRL $_n$**  register configures the bit timing parameters required by the CAN protocol. The **CLK\_SRC**, **PRESDIV**, **RJW**, **PSEG1**, **PSEG2**, and the **PROPSEG** fields allow the user to configure the bit timing parameters.

The **CANCTRL $_n$ [CLK\_SRC]** bit defines whether the module uses the internal bus clock or the output of the crystal oscillator via the **EXTAL** pin. The crystal oscillator clock should be selected when a tight tolerance (up to 0.1%) is required for the CAN bus timing. The crystal oscillator clock has better jitter performance than PLL generated clocks. The value of this bit should not be changed, unless the module is in disable mode (**CANMCR $_n$ [MDIS]** bit is set)

The **PRESDIV** field controls a prescaler that generates the serial clock (S-clock), whose period defines the time quantum used to compose the CAN waveform. A time quantum is the atomic unit of time managed by the CAN engine.



**Figure 33-18. CAN Engine Clocking Scheme**

$$f_{Tq} = \frac{f_{sys/2} \text{ or } EXTAL}{(PRESDIV + 1)}$$

**Eqn. 33-6**

A bit time is subdivided into three segments<sup>1</sup> (see [Figure 33-19](#) and [Table 33-16](#)):

- **SYNC\_SEG**: Has a fixed length of one time quantum. Signal edges are expected to happen within this section.

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.

- Time Segment 1: 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 CANCTRL*n* register so that their sum (plus 2) is in the range of 4 to 16 time quanta.
- Time Segment 2: Represents the phase segment 2 of the CAN standard. It can be programmed by setting the PSEG2 field of the CANCTRL*n* register (plus 1) to be 2 to 8 time quanta long.

$$\text{Bit Rate} = \frac{f_{Tq}}{\text{(number of Time Quanta)}}$$

Eqn. 33-7

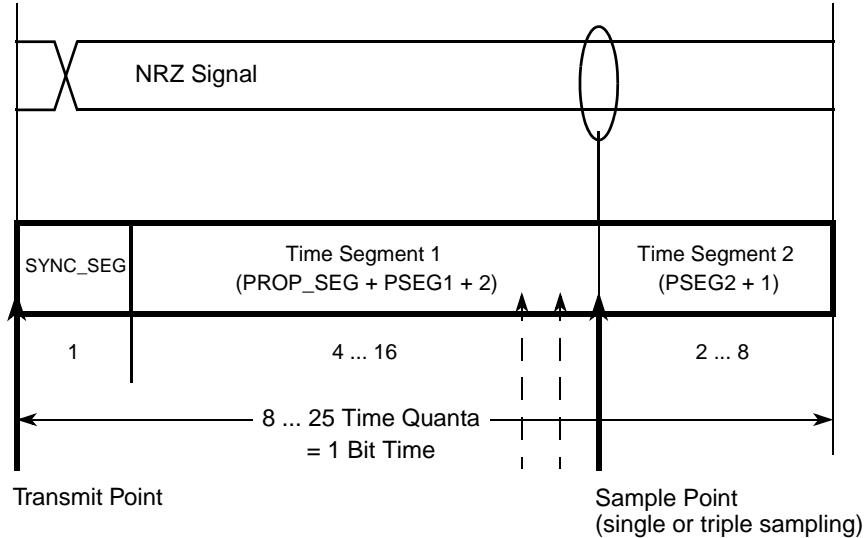


Figure 33-19. Segments within the Bit Time

Table 33-16. 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 33-17 gives an overview of the CAN compliant segment settings and the related parameter values.

#### NOTE

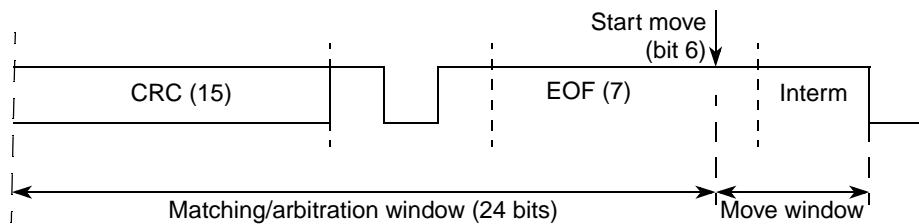
It is the user's responsibility to ensure the bit time settings are in compliance with the CAN standard. For bit time calculations, use an IPT (Information Processing Time) of 2, which is the value implemented in the FlexCAN module

**Table 33-17. 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

### 33.3.22 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 the following figure.

**Figure 33-20. Figure 19. 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 33-17](#)
- 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 33-18](#)

**Table 33-18. 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 33-18](#) 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, PROPSSEG, 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.

## 33.4 Initialization/Application Information

Initialization of the FlexCAN includes the initial configuration of the message buffers and configuration of the CAN communication parameters following a reset, as well as any reconfiguration that may be required during operation. The FlexCAN module may be reset in three ways:

- Device level hard reset—resets all memory mapped registers asynchronously
- Device level soft reset—resets some of the memory mapped registers synchronously (refer to [Table 33-1](#) to see which registers are affected by soft reset)
- CANMCR $n$ [SOFT\_RST] bit—has the same effect as the device 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 CANMCR $n$ [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, CANCTRL $n$ [CLK\_SRC], should be selected while the module is in disable mode. After the clock source is selected and the module is enabled (CANMCR $n$ [MDIS] bit cleared), the FlexCAN automatically enters freeze mode. In freeze mode, the FlexCAN is un-synchronized to the CAN bus, the CANMCR $n$  register's HALT and FRZ bits are set, the internal state machines are disabled, and the CANMCR $n$  register's FRZ\_ACK and NOT\_RDY bits are set. The CAN $n$ TX pin is in recessive state and the FlexCAN does not initiate any transmission or reception of CAN frames. The message buffers are not affected by reset, so they are not automatically initialized.

For any configuration change/initialization, the FlexCAN must be in freeze mode (see [Section 33.1.3.2, “Freeze Mode”](#)). The following is a generic initialization sequence applicable to the FlexCAN module:

1. Initialize the CANMCR register
  - a) Enable individual filtering per MB and reception queue features by setting the BCC bit
  - b) Enable the warning interrupts by setting the WRN\_EN bit
  - c) If required, disable frame self reception by setting the SRX\_DIS bit
  - d) Enable the FIFO by setting the FEN bit
  - e) Enable the abort mechanism by setting the AEN bit
  - f) Enable the local priority feature by setting the LPPIO\_EN bit

2. Initialize all operation modes in the CANCTRL register.
  - a) Initialize the bit timing parameters PROPSEG, PSEGS1, PSEG2, and RJW.
  - b) Select the S-clock rate by programming the PRESDIV field.
  - c) Select the internal arbitration mode via the LBUF bit.
3. Initialize message buffers.
  - a) The control/status word of all message buffers must be written as an active or inactive message buffer
  - b) If FIFO was enabled, the 8-entry ID table must be initialized
  - c) All other entries in each message buffer should be initialized as required
4. Initialize the RX individual mask registers for acceptance mask as needed.
5. Initialize FlexCAN interrupt handler.
  - a) Initialize the interrupt controller registers for any needed interrupts. See [Chapter 17, “Interrupt Controller Modules,”](#) for more information.
  - b) Set the required mask bits in the IMASK $n$  register (for all message buffer interrupts) and the CANCTRL $n$  (for bus off and error interrupts).
6. Clear the CANMCR $n$ [HALT] bit. At this point, the FlexCAN attempts to synchronize with the CAN bus.

### 33.4.1 Interrupts

There are 19 interrupt sources for the FlexCAN module. An interrupt for each of the 16 MBs. Plus, a combined interrupt for all 16 MBs is generated by logically OR’ing all the interrupt sources from the MBs. In this case, the CPU must read the IFLAG $n$  register to determine which MB caused the interrupt. The other interrupt sources (bus off and error) act in the same manner, and are located in the ERRSTAT $n$  register. The bus off and error interrupt mask bits are located in the CANCTRL $n$  register.

### 33.4.2 Mask Misalignment for Rx FIFO

During CAN message reception by FlexCAN, RXGMASK is an acceptance mask for most of the Rx Message Buffers (MB). When the FIFO enable bit in the FlexCAN Module Configuration Register (CANx\_MCR[FEN]) is set, the RXGMASK also applies to most of the elements of the ID filter table. However, there is a misalignment between the position of the ID field in the Rx MB and in RXIDA, RXIDB and RXIDC fields of the ID Tables. In fact RXIDA filter in the ID Tables is shifted one bit to the left from Rx MBs ID position as shown below:

- Rx MB ID = bits 28–0 of ID word corresponding to message ID bits 31–3
- RXIDA = bits 29–1 of ID Table corresponding to message ID bits 31–3

Note that the mask bits’ one-to-one correspondence occurs with the filters bits, not with the incoming message ID bits. This leads the RXGMASK to affect Rx MB and Rx FIFO filtering in different ways. For example, if the user intends to mask bit 7 of the ID filter of Message Buffers then configure RXGMASK to 0xFFFF\_FFEF. As result, bit 7 of the ID field of the incoming message is ignored during filtering process for Message Buffers. This very same configuration of RXGMASK leads bit 7 of RXIDA to be “don’t care” and thus bit 6 of the ID field of the incoming message is ignored during filtering process for Rx FIFO.

Similarly, RXIDB and RXIDC filters have multiple misalignments with regards to position of ID field in Rx MBs, which can lead to erroneous masking during filtering process for either Rx FIFO or MBs. RX14MASK (Rx 14 Mask) and RX15MASK (Rx 15 Mask) have the same structure as the RXGMASK. This includes the misalignment problem between the position of the ID field in the Rx MBs and in RXIDA, RXIDB and RXIDC fields of the ID Tables.

### 33.4.2.1 Work Around

It is recommended that one of the following actions be taken to avoid problems:

- Do not enable the RxFIFO. If CANx\_MCR[FEN]=0 then the Rx FIFO is disabled and thus the masks RXGMASK, RX14MASK and RX15MASK do not affect it.
- Enable Rx Individual Mask Registers. If the Backwards Compatibility Configuration bit in the FlexCAN Module Configuration Register (CANx\_MCR[BCC]) is set then the Rx Individual Mask Registers (RXIMR0-63) are enabled and thus the masks RXGMASK, RX14MASK and RX15MASK are not used.
- Do not use masks RXGMASK, RX14MASK and RX15MASK (i.e. let them in reset value which is 0xFFFF\_FFFF) when CANx\_MCR[FEN]=1 and CANx\_MCR[BCC]=0. In this case, filtering processes for both Rx MBs and Rx FIFO are not affected by those masks.
- Do not configure any MB as Rx (i.e. let all MBs as either Tx or inactive) when CANx\_MCR[FEN]=1 and CANx\_MCR[BCC]=0. In this case, the masks RXGMASK, RX14MASK and RX15MASK can be used to affect ID Tables without affecting filtering process for Rx MBs.

# Chapter 34

## Motor Control Pulse-Width Modulator (mcPWM)

### 34.1 Introduction

#### 34.1.1 Overview

The pulse width modulator module (PWM) contains four PWM submodules, each able to control a single half-bridge power stage. There are also three fault channels.

This PWM is capable of controlling most motor types:

- AC induction motors (ACIM)
- Permanent magnet AC motors (PMAC)
- Brushless (BLDC) and brush DC motors (BDC)
- Switched (SRM) and variable reluctance motors (VRM)
- Stepper motors

#### NOTE

The PWM\_X $n$  and OUT\_TRIG $n$  signals are not output from this device. However, they are available to the on-chip analog-to-digital convertor for triggering a conversion sequence.

### 34.1.2 Block Diagram

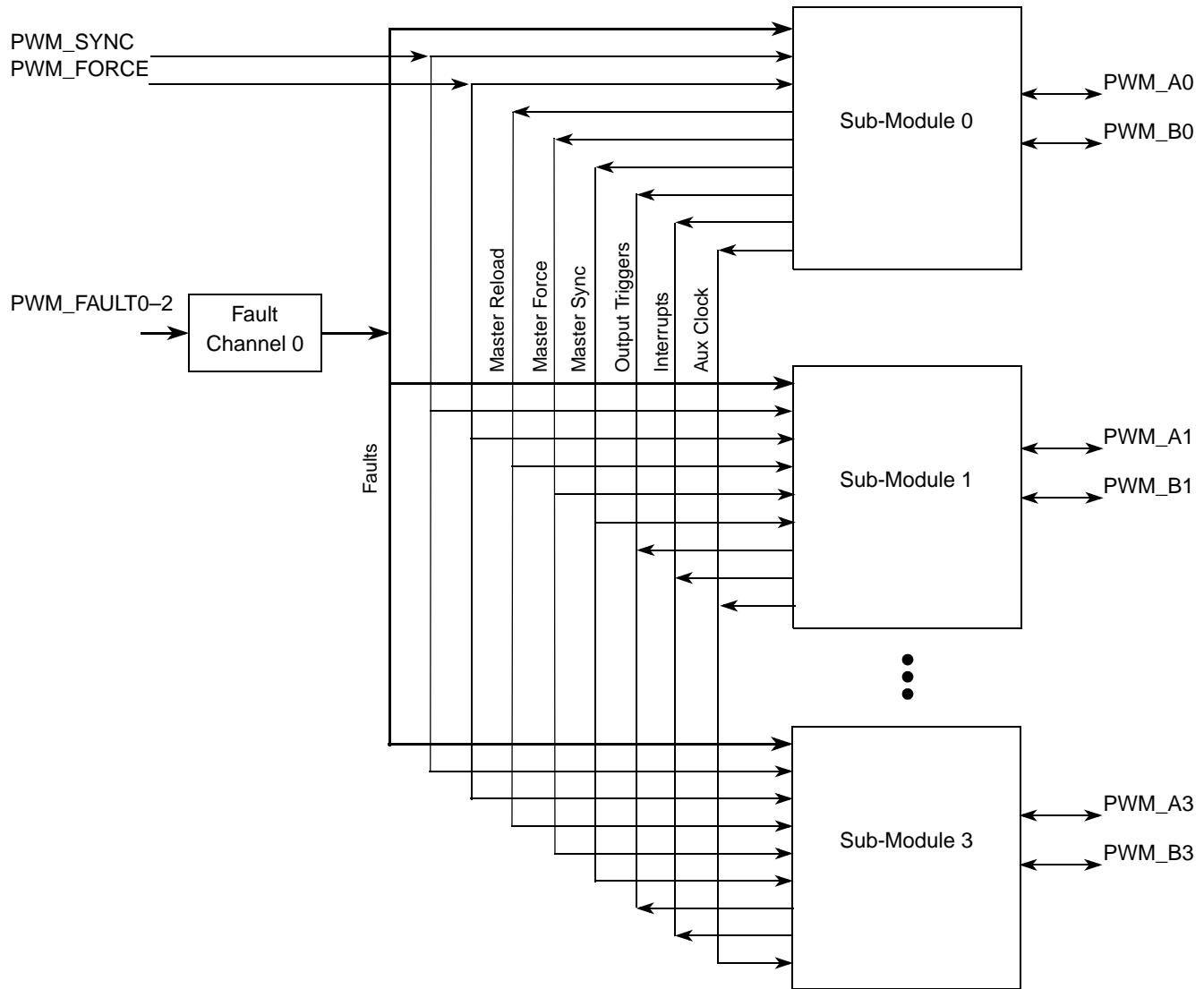


Figure 34-1. PWM Block Diagram

### 34.1.3 Features

- 16-bit resolution for center-aligned, edge-aligned, and asymmetrical PWMs
- PWM outputs can operate as complimentary pairs or independent channels
- Can accept signed numbers for PWM generation
- Independent control of both edges of each PWM output
- Synchronization to external hardware or other PWM supported
- Double-buffered PWM registers
  - Integral reload rates from 1 to 16
  - Half-cycle reload capability

- Multiple output trigger events can be generated per PWM cycle via hardware
- Support for double-switching PWM outputs
- Fault inputs can be assigned to control multiple PWM outputs
- Programmable filters for fault inputs
- Independently programmable PWM output polarity
- Independent top and bottom deadtime insertion
- Each complementary pair can operate with its own PWM frequency and deadtime values
- Individual software-control for each PWM output
- All outputs can be programmed to change simultaneously via a “Force Out” event
- PWM\_X signals can optionally output a third PWM signal from each submodule
- Channels not used for PWM generation can be used for buffered output compare functions
- Channels not used for PWM generation can be used for input capture functions
- Enhanced dual edge capture functionality
- The option to supply the source for each complementary PWM signal pair from any of the following:
  - External digital pin
  - Internal timer channel
  - External ADC input, accounting for values set in ADC high and low limit registers

### 34.1.4 Modes of Operation

Take care when using this module in certain chip operating modes. Some motors (3-phase AC motors) require regular software updates for proper operation. Failure to do so could result in destroying the motor or inverter. Because of this, PWM outputs are placed in their inactive states in STOP mode, and optionally under WAIT and debug modes. PWM outputs are reactivated (assuming they were active to begin with) when these modes are exited.

**Table 34-1. Modes When PWM Operation is Restricted**

Mode	Description
STOP	Peripheral and CPU clocks are stopped. PWM outputs are driven inactive.
WAIT	CPU clocks are stopped while peripheral clocks continue to run. PWM outputs are driven inactive as a function of PWM_SMnCR2[WAITEN].
DEBUG	CPU and peripheral clocks continue to run, but CPU may stall for periods of time. PWM outputs are driven inactive as a function of the PWM_SMnCR2[DBGEN] bit.

## 34.2 External Signal Descriptions

**Table 34-2. PWM Signal Descriptions**

Signal	I/O	Description
PWM_A[3:0] PWM_B[3:0]	I/O	External PWM pair. They can be independent output PWM signals or a complementary pair. When not needed as an output, they can be used for input capture.
PWMFAULT[2:0]	I	Fault inputs for disabling selected PWM outputs
PWM_SYNC	I	External synchronization signal. Allows a source external to the PWM to initialize the PWM counter.
PWM_FORCE	I	External output force signal. Allows a source external to the PWM to force an update of the PWM outputs. For example, simultaneously switching all PWM outputs on a commutation boundary for trapezoidal control of a BLDC motor. The boundary can be established by external logic or an on-chip timer.
PWM_EXTA[3:0]	I	Alternate PWM control signals. These pins allow an alternate source to control the PWMA outputs. Although typically, the PWM_EXTA $n$ input is used for the generation of a complementary pair. Typical connections include ADC results registers, timer outputs, GPIO inputs, and comparator outputs.

## 34.3 Memory Map/Register Description

Each of the four submodules contain a set of registers for configuring their respective PWM signals. The base addresses are indicated in [Table 34-3](#). There are also a set of registers for configuring the PWM outputs and the fault channels.

**Table 34-3. Submodule Base Addresses**

Submodule #	Base Address
0	0xEC08_8000
1	0xEC08_8050
2	0xEC08_80A0
3	0xEC08_80F0

**Table 34-4. PWM Memory Map**

Address	Register	Width	Access	Reset Value	Section/Page
<b>Submodule <math>n</math> registers (<math>n = 0\text{--}3</math>)</b>					
0xEC08_8000 + ( $n \times 0x50$ )	Counter register (PWM_SM $n$ CNT)	16	R	0x0000	<a href="#">34.3.1/34-7</a>
0xEC08_8002 + ( $n \times 0x50$ )	Initial Count Register (PWM_SM $n$ INIT)	16	R/W	0x0000	<a href="#">34.3.2/34-7</a>
0xEC08_8004 + ( $n \times 0x50$ )	Control Register 2 (PWM_SM $n$ CR2)	16	R/W	0x0000	<a href="#">34.3.3/34-7</a>

**Table 34-4. PWM Memory Map (continued)**

Address	Register	Width	Access	Reset Value	Section/Page
0xEC08_8006 + (n × 0x50)	Control Register 1 (PWM_SMnCR1)	16	R/W	0x0000	<a href="#">34.3.4/34-9</a>
0xEC08_8008 + (n × 0x50)	Value Register 0 (PWM_SMnVAL0)	16	R/W	0x0000	<a href="#">34.3.5/34-10</a>
0xEC08_800A + (n × 0x50)	Value Register 1 (PWM_SMnVAL1)	16	R/W	0x0000	<a href="#">34.3.5/34-10</a>
0xEC08_800C + (n × 0x50)	Value Register 2 (PWM_SMnVAL2)	16	R/W	0x0000	<a href="#">34.3.5/34-10</a>
0xEC08_800E + (n × 0x50)	Value Register 3 (PWM_SMnVAL3)	16	R/W	0x0000	<a href="#">34.3.5/34-10</a>
0xEC08_8010 + (n × 0x50)	Value Register 4 (PWM_SMnVAL4)	16	R/W	0x0000	<a href="#">34.3.5/34-10</a>
0xEC08_8012 + (n × 0x50)	Value Register 5 (PWM_SMnVAL5)	16	R/W	0x0000	<a href="#">34.3.5/34-10</a>
0xEC08_8018 + (n × 0x50)	Output Control Register (PWM_SMnOCR)	16	R/W	See Section	<a href="#">34.3.6/34-11</a>
0xEC08_801A + (n × 0x50)	Status Register (PWM_SMnSR)	16	R/W	0x0000	<a href="#">34.3.7/34-13</a>
0xEC08_801C + (n × 0x50)	Interrupt Enable Register (PWM_SMnIER)	16	R/W	0x0000	<a href="#">34.3.8/34-14</a>
0xEC08_801E + (n × 0x50)	DMA Enable Register (PWM_SMnDMAEN)	16	R/W	0x0000	<a href="#">34.3.9/34-14</a>
0xEC08_8020 + (n × 0x50)	Output Trigger Control Register (PWM_SMnOTCR)	16	R/W	0x0000	<a href="#">34.3.10/34-15</a>
0xEC08_8022 + (n × 0x50)	Fault Disable Mapping Register (PWM_SMnDISMAP)	16	R/W	0xFFFF	<a href="#">34.3.11/34-16</a>
0xEC08_8024 + (n × 0x50)	Deadtime Count Register 0 (PWM_SMnDTCNT0)	16	R/W	0x07FF	<a href="#">34.3.12/34-17</a>
0xEC08_8026 + (n × 0x50)	Deadtime Count Register 1 (PWM_SMnDTCNT1)	16	R/W	0x07FF	<a href="#">34.3.12/34-17</a>
0xEC08_8028 + (n × 0x50)	Capture Control Register A (PWM_SMnCCRA)	16	R/W	0x0000	<a href="#">34.3.13/34-17</a>
0xEC08_802A + (n × 0x50)	Capture Compare Register A (PWM_SMnCCMPA)	16	R/W	0x0000	<a href="#">34.3.14/34-19</a>
0xEC08_802C + (n × 0x50)	Capture Control Register B (PWM_SMnCCRB)	16	R/W	0x0000	<a href="#">34.3.13/34-17</a>
0xEC08_802E + (n × 0x50)	Capture Compare Register B (PWM_SMnCCMPB)	16	R/W	0x0000	<a href="#">34.3.14/34-19</a>
0xEC08_8030 + (n × 0x50)	Capture Control Register X (PWM_SMnCCRX)	16	R/W	0x0000	<a href="#">34.3.13/34-17</a>

**Table 34-4. PWM Memory Map (continued)**

Address	Register	Width	Access	Reset Value	Section/Page
0xEC08_8032 + (n × 0x50)	Capture Compare Register X (PWM_SMnCCMPX)	16	R/W	0x0000	<a href="#">34.3.14/34-19</a>
0xEC08_8034 + (n × 0x50)	Capture Value 0 Register (PWM_SMnCVAL0)	16	R	0x0000	<a href="#">34.3.15/34-20</a>
0xEC08_8036 + (n × 0x50)	Capture Value 0 Cycle Register (PWM_SMnCCYC0)	16	R	0x0000	<a href="#">34.3.16/34-21</a>
0xEC08_8038 + (n × 0x50)	Capture Value 1 Register (PWM_SMnCVAL1)	16	R	0x0000	<a href="#">34.3.15/34-20</a>
0xEC08_803A + (n × 0x50)	Capture Value 1 Cycle Register (PWM_SMnCCYC1)	16	R	0x0000	<a href="#">34.3.16/34-21</a>
0xEC08_803C + (n × 0x50)	Capture Value 2 Register (PWM_SMnCVAL2)	16	R	0x0000	<a href="#">34.3.15/34-20</a>
0xEC08_803E + (n × 0x50)	Capture Value 2 Cycle Register (PWM_SMnCCYC2)	16	R	0x0000	<a href="#">34.3.16/34-21</a>
0xEC08_8040 + (n × 0x50)	Capture Value 3 Register (PWM_SMnCVAL3)	16	R	0x0000	<a href="#">34.3.15/34-20</a>
0xEC08_8042 + (n × 0x50)	Capture Value 3 Cycle Register (PWM_SMnCCYC3)	16	R	0x0000	<a href="#">34.3.16/34-21</a>
0xEC08_8044 + (n × 0x50)	Capture Value 4 Register (PWM_SMnCVAL4)	16	R	0x0000	<a href="#">34.3.15/34-20</a>
0xEC08_8046 + (n × 0x50)	Capture Value 4 Cycle Register (PWM_SMnCCYC4)	16	R	0x0000	<a href="#">34.3.16/34-21</a>
0xEC08_8048 + (n × 0x50)	Capture Value 5 Register (PWM_SMnCVAL5)	16	R	0x0000	<a href="#">34.3.15/34-20</a>
0xEC08_804A + (n × 0x50)	Capture Value 5 Cycle Register (PWM_SMnCCYC5)	16	R	0x0000	<a href="#">34.3.16/34-21</a>
<b>Configuration registers</b>					
0xEC08_8140	Output Enable Register (PWM_OUTEN)	16	R/W	0x0000	<a href="#">34.3.17/34-21</a>
0xEC08_8142	Output Mask Register (PWM_MASK)	16	R/W	0x0000	<a href="#">34.3.18/34-22</a>
0xEC08_8144	Software Controlled Output Register (PWM_SWCOUT)	16	R/W	0x0000	<a href="#">34.3.19/34-23</a>
0xEC08_8146	Deadtime Source Select Register (PWM_DTSS)	16	R/W	0x0000	<a href="#">34.3.20/34-23</a>
0xEC08_8148	Master Control Register (PWM_MCR)	16	R/W	0x0000	<a href="#">34.3.21/34-24</a>
<b>Fault channel registers</b>					
0xEC08_814C	Fault Control Register (PWM_FCR)	16	R/W	0x0000	<a href="#">34.3.22/34-25</a>
0xEC08_814E	Fault Status Register (PWM_FSR)	16	R/W	See Section	<a href="#">34.3.23/34-26</a>
0xEC08_8150	Fault Filter Register (PWM_FFILT)	16	R/W	0x0000	<a href="#">34.3.24/34-27</a>

### 34.3.1 Counter Register (PWM\_SMnCNT)

Address:	0xEC08_8000 (PWM_SM0CNT) 0xEC08_8050 (PWM_SM1CNT)	0xEC08_80A0 (PWM_SM2CNT) 0xEC08_80F0 (PWM_SM3CNT)	Access:	User read-only

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

Figure 34-2. Counter Register (PWM\_SMnCNT)

Table 34-5. PWM\_SMnCNT Field Descriptions

Field	Description
15–0 CNT	Displays the state of the signed 16-bit submodule counter. This register is not byte-accessible.

### 34.3.2 Initial Count Register (PWM\_SMnINIT)

Address:	0xEC08_8002 (PWM_SM0INIT) 0xEC08_8052 (PWM_SM1INIT)	0xEC08_80A2 (PWM_SM2INIT) 0xEC08_80F2 (PWM_SM3INIT)	Access:	User read/write

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

Figure 34-3. Initial Count Register (PWM\_SMnINIT)

Table 34-6. PWM\_SMnINIT Field Descriptions

Field	Description
15–0 INIT	Defines the initial count value for the PWM in PWM clock periods. This is the value loaded into the submodule counter when local sync, master sync, or master reload is asserted (based on PWM_SMnCR2[INIT_SEL]) or when PWM_FORCE is asserted and PWM_SMnCR2[FRCEN] is set. For PWM operation, the buffered contents of this register are loaded into the counter at the start of each PWM cycle. This register is not byte-accessible.  <b>Note:</b> This register is buffered. The value written does not take effect until PWM_MCR[LDOCK] is set and the next PWM load cycle begins or PWM_SMnCR1[LDMOD] is set. This register cannot be written when LDOCK is set. Reading INIT reads the value in a buffer and not necessarily the value the PWM generator is currently using.

### 34.3.3 Control Register 2 (PWM\_SMnCR2)

Address:	0xEC08_8004 (PWM_SM0CR2) 0xEC08_8054 (PWM_SM1CR2)	0xEC08_80A4 (PWM_SM2CR2) 0xEC08_80F4 (PWM_SM3CR2)	Access:	User read/write

15	14	13	12		11	10	9	8		7	6	5	4		3	2	1	0	
R					PWM	PWM				FRC					RELO				
W	DBG EN	WAIT EN	INDEP	PWM 23_ INIT	45_ INIT	X_ INIT	INIT_SEL		EN	0	FOR CE				AD SEL	CLK_SEL			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-4. Control Register 2 (PWM\_SMnCR2)

Table 34-7. PWM\_SMnCR2 Field Descriptions

Field	Description
15 DBGEN	<p>Debug enable.</p> <p>0 The PWM outputs are disabled until debug mode is exited. At that point the PWM pins resume operation as programmed in the PWM registers.</p> <p>1 The PWM continues to run while the processor is in debug mode</p> <p><b>Note:</b> For certain types of motors (3-phase AC), leave this bit cleared. Failure to do so could result in damage to the motor or inverter. For other types of motors (DC motors), this bit might safely be set. PWM parameter updates do not occur in debug mode. Any motors requiring such updates should be disabled during debug mode. If in doubt, leave this bit cleared.</p>
14 WAITEN	<p>Wait enable.</p> <p>0 The PWM outputs are disabled until WAIT mode is exited. At that point the PWM pins resume operation as programmed in the PWM registers.</p> <p>1 The PWM continues to run while the chip is in wait mode. In wait mode, the peripheral clock continues to run but the CPU clock does not.</p> <p><b>Note:</b> For certain types of motors (3-phase AC), leave this bit cleared. Failure to do so could result in damage to the motor or inverter. For other types of motors (DC motors), this bit might safely be set. PWM parameter updates do not occur in wait mode. Any motors requiring such updates should be disabled during wait mode. If in doubt, leave this bit cleared.</p>
13 INDEP	<p>Independent or complementary pair operation. Determines if the PWMA and PWMB channels are independent PWMs or a complementary PWM pair.</p> <p>0 PWMA and PWMB form a complementary PWM pair</p> <p>1 PWMA and PWMB outputs are independent PWMs</p>
12 PWM23_INIT	PWM23 initial value. Determines the initial value for PWM23 and the value it is forced when FORCE_INIT is asserted. See <a href="#">Figure 34-40</a> for details on how FORCE_INIT is derived.
11 PWM45_INIT	PWM45 initial value. Determines the initial value for PWM45 and the value it is forced when FORCE_INIT is asserted. See <a href="#">Figure 34-40</a> for details on how FORCE_INIT is derived.
10 PWMX_INIT	PWMX initial value. Determines the initial value for PWMX and the value it is forced when FORCE_INIT is asserted. See <a href="#">Figure 34-40</a> for details on how FORCE_INIT is derived.
9–8 INIT_SEL	<p>Initialization control select. Controls the source of the INIT signal which goes to the counter.</p> <p>00 Local sync (PWM_X) causes initialization</p> <p>01 Master reload from submodule 0 causes initialization. Do not use this setting in PWM_SM0CR2 as it forces the INIT signal to logic 0.</p> <p>10 Master sync from submodule 0 causes initialization. Do not use this setting in PWM_SM0CR2 as it forces the INIT signal to logic 0.</p> <p>11 PWM_SYNC causes initialization</p>
7 FRCEN	Force initialization enable. Allows the FORCE signal to initialize the counter without regard to INIT_SEL. This is a software-controlled initialization.
6 FORCE	<p>Force initialization. If FORCE_SEL is 000, setting this bit results in a force out event. This causes the following:</p> <ul style="list-style-type: none"> <li>The PWMA and PWMB output pins assume values based on the SEL23 and SEL45 bits</li> <li>If FRCEN is set, the counter value is initialized with the PWM_SMnINIT register value</li> </ul>

**Table 34-7. PWM\_SMnCR2 Field Descriptions (continued)**

Field	Description
5–3 FORCE_SEL	Force source select. Determines the source of the force output signal for this submodule. 000 The local FORCE bit from this submodule 001 The master force signal from submodule 0. Do not use this setting in PWM_SM0CR2 as it holds the force output signal to logic 0. 010 The local reload signal from this submodule 011 The master reload signal from submodule 0. Do not use this setting in PWM_SM0CR2 as it holds the force output signal to logic 0. 100 The local sync signal from this submodule 101 The master sync signal from submodule 0. Do not use this setting in PWM_SM0CR2 as it holds the force output signal to logic 0. 110 The external force signal, PWM_FORCE, causes updates 111 Reserved
2 RELOAD_SEL	Reload source select. Determines the source of the reload signal for this submodule. 0 The local reload signal is used to reload registers 1 The master reload signal from submodule 0 is used to reload registers. Use the LDOKE bit in submodule 0, since the local LDOKE is ignored. Do not use this setting in PWM_SM0CR2 as it forces the reload signal to logic 0.
1–0 CLK_SEL	Clock source select. Determines the source of the clock signal for this submodule's prescaler and counter. 00 Peripheral bus clock 01 EXT_CLK. For this device, EXT_CLK is any of the DMA timer outputs as chosen by a register in the CCM module (MISCCR[PWM_ECS]). See <a href="#">Chapter 10, "Chip Configuration Module (CCM)"</a> , for more details. 10 Submodule 0's clock (AUX_CLK). Do not use this setting in PWM_SM0CR2 as it forces the clock to logic 0. 11 Reserved

### 34.3.4 Control Register 1 (PWM\_SMnCR1)

Address: 0xEC08_8006 (PWM_SM0CR) 0xEC08_8056 (PWM_SM1CR)				0xEC08_80A6 (PWM_SM2CR) 0xEC08_80F6 (PWM_SM2CR)				Access: User read/write			
15	14	13	12	11	10	9	8	7	6	5	4
R W	LDFQ				HALF	FULL	DT		0	PRSC	
Reset	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0	0	0	0

**Figure 34-5. Control Register 1 (PWM\_SMnCR1)**

Table 34-8. PWM\_SMnCR1 Field Descriptions

Field	Description
15–12 LDFQ	<p>Load frequency. Selects the PWM load frequency. A PWM opportunity is determined by HALF and FULL.</p> <p><b>Note:</b> This field takes effect when the current load cycle is complete, regardless of the state of the LDOCK bit. Reading this field reads the buffered values and not necessarily the values currently in effect.</p> <ul style="list-style-type: none"> <li>0000 Every PWM opportunity</li> <li>0001 Every 2 PWM opportunities</li> <li>...</li> <li>1111 Every 16 PWM opportunities</li> </ul>
11 HALF	<p>Half-cycle reload. A half cycle is defined by when the submodule counter matches PWM_SMnVAL0 and is not necessarily halfway through the PWM cycle. HALF or FULL must be set to move the buffered data into the registers used by the PWM generators. If both bits are set, then reloads can occur twice per cycle.</p> <ul style="list-style-type: none"> <li>0 Half-cycle reloads disabled</li> <li>1 Half-cycle reloads enabled</li> </ul>
10 FULL	<p>Full-cycle reload. A full cycle is defined by when the submodule counter matches the PWM_SMnVAL1 register. HALF or FULL must be set to move the buffered data into the registers used by the PWM generators. If both bits are set, then reloads can occur twice per cycle.</p> <ul style="list-style-type: none"> <li>1 Full-cycle reloads enabled</li> <li>0 Full-cycle reloads disabled</li> </ul>
9–8 DT	Deadtime. Reflects the sampled values of the PWMX input at the end of each deadtime. Sampling occurs at the end of deadtime 0 for DT[0] and the end of deadtime 1 for DT[1].
7	Reserved, must be cleared.
6–4 PRSC	<p>Prescaler. Selects the divide ratio of the PWM clock frequency selected by PWM_SMnCR2[CLK_SEL].</p> <ul style="list-style-type: none"> <li>000 <math>f_{clk}</math></li> <li>001 <math>f_{clk}/2</math></li> <li>010 <math>f_{clk}/4</math></li> <li>...</li> <li>111 <math>f_{clk}/128</math></li> </ul> <p><b>Note:</b> Reading this field reads the buffered values and not necessarily the values currently in effect. This field takes affect at the beginning of the next PWM cycle and only when LDOCK or LDMD is set. This field cannot be written when LDOCK is set.</p>
3	Reserved, must be cleared.
2 LDMD	<p>Load mode select. Selects the timing of loading the buffered registers for this submodule.</p> <ul style="list-style-type: none"> <li>0 At the next PWM reload if LDOCK is set</li> <li>1 Immediately upon LDOCK being set</li> </ul>
1	Reserved, must be cleared.
0 DBLEN	<p>Double switching enable.</p> <ul style="list-style-type: none"> <li>0 Disable</li> <li>1 Enable</li> </ul>

### 34.3.5 Value Registers (PWM\_SMnVAL0–5)

The six value registers per submodule define various counter values for PWM operation:

- PWM\_SMnVAL0—Defines the mid-cycle reload point for the PWM in PWM clock periods. This value also defines when the PWM\_X signal is set and the local sync signal is reset.

- PWM\_SM<sub>n</sub>VAL1 — Defines the modulo count value (maximum count) for the submodule counter. Upon reaching this count value, the counter reloads itself with the contents of PWM\_SM<sub>n</sub>INIT and asserts the local sync signal while resetting PWM\_X
- PWM\_SM<sub>n</sub>VAL2 — Defines the count value to set PWM23 high
- PWM\_SM<sub>n</sub>VAL3 — Defines the count value to set PWM23 low
- PWM\_SM<sub>n</sub>VAL4 — Defines the count value to set PWM45 high
- PWM\_SM<sub>n</sub>VAL5 — Defines the count value to set PWM45 low

Address: 0xEC08_8008 (PWM_SM0VAL0)	0xEC08_80A8 (PWM_SM2VAL0)	Access: User read/write
0xEC08_800A (PWM_SM0VAL1)	0xEC08_80AA (PWM_SM2VAL1)	
0xEC08_800C (PWM_SM0VAL2)	0xEC08_80AC (PWM_SM2VAL2)	
0xEC08_800E (PWM_SM0VAL3)	0xEC08_80AE (PWM_SM2VAL3)	
0xEC08_8010 (PWM_SM0VAL4)	0xEC08_80B0 (PWM_SM2VAL4)	
0xEC08_8012 (PWM_SM0VAL5)	0xEC08_80B2 (PWM_SM2VAL5)	
0xEC08_8058 (PWM_SM1VAL0)	0xEC08_80F8 (PWM_SM3VAL0)	
0xEC08_805A (PWM_SM1VAL1)	0xEC08_80FA (PWM_SM3VAL1)	
0xEC08_805C (PWM_SM1VAL2)	0xEC08_80FC (PWM_SM3VAL2)	
0xEC08_805E (PWM_SM1VAL3)	0xEC08_80FE (PWM_SM3VAL3)	
0xEC08_8060 (PWM_SM1VAL4)	0xEC08_8100 (PWM_SM3VAL4)	
0xEC08_8062 (PWM_SM1VAL5)	0xEC08_8102 (PWM_SM3VAL5)	
15      14      13      12   11      10      9      8   7      6      5      4   3      2      1      0		
R	VAL	
W		
Reset	0      0      0      0   0      0      0      0   0      0      0      0   0      0      0      0	

Figure 34-6. Value Registers (PWM\_SM<sub>n</sub>VAL0–5)Table 34-9. PWM\_SM<sub>n</sub>VAL0–5 Field Descriptions

Field	Description
15–0 VAL	These registers are not byte-accessible. <b>Note:</b> The value registers are buffered. The value written does not take effect until the LDOOK bit is set and the next PWM load cycle begins or LDMOD is set. They cannot be written when LDOOK is set. Reading the value registers reads the value in a buffer, and is not necessarily the value the PWM generator is currently using.

### 34.3.6 Output Control Register (PWM\_SM<sub>n</sub>OCR)

The PWM\_SM<sub>n</sub>OCR registers control the output pins of each submodule.

Address: 0xEC08_8018 (PWM_SM0OCR)	0xEC08_80B8 (PWM_SM2OCR)	Access: User read/write
0xEC08_8068 (PWM_SM1OCR)	0xEC08_8108 (PWM_SM3OCR)	
15      14      13      12   11      10      9      8   7      6      5      4   3      2      1      0		
R	PWM A_IN      PWM B_IN      PWM X_IN      0      0      POLA      POLB      POLX      0      0      PWMAFS      PWMBFS      PWMXFS	
W		
Reset	U      U      U      0   0      0      0      0   0      0      0      0   0      0      0      0	

Figure 34-7. Output Control Register (PWM\_SM<sub>n</sub>OCR)

**Table 34-10. PWM\_SMnOCR Field Descriptions**

<b>Field</b>	<b>Description</b>
15 PWMA_IN	Reflects the value currently being driven into the PWM_A input.
14 PWMB_IN	Reflects the value currently being driven into the PWM_B input.
13 PWMX_IN	Reflects the value currently being driven into the PWM_X input.
12–11	Reserved, must be cleared.
10 POLA	PWM_A output polarity. 0 Output not inverted. A high level on the pin represents the on/active state. 1 Output inverted. A low level on the pin represents the on/active state.
9 POLB	PWM_B output polarity. 0 Output not inverted. A high level on the pin represents the on/active state. 1 Output inverted. A low level on the pin represents the on/active state.
8 POLX	PWM_X output polarity. 0 Output not inverted. A high level on the pin represents the on/active state. 1 Output inverted. A low level on the pin represents the on/active state.
7–6	Reserved, must be cleared.
5–4 PWMAFS	PWM_A fault state. Determines the PWM output level during fault conditions and stop mode. If PWM_SMnCR2[WAITEN, DBGEN] are cleared, also defines the output state during wait and debug modes, respectively. 00 Output is forced to 0 prior to consideration of output polarity control 01 Output is forced to 1 prior to consideration of output polarity control 1x Output is tristated
3–2 PWMBFS	PWM_B fault state. Determines the PWM output level during fault conditions and stop mode. If PWM_SMnCR2[WAITEN, DBGEN] are cleared, also defines the output state during wait and debug modes, respectively. 00 Output is forced to 0 prior to consideration of output polarity control 01 Output is forced to 1 prior to consideration of output polarity control 1x Output is tristated
1–0 PWMXFS	PWM_X fault state. Determines the PWM output level during fault conditions and stop mode. If PWM_SMnCR2[WAITEN, DBGEN] are cleared, also defines the output state during wait and debug modes, respectively. 00 Output is forced to logic 0 state prior to consideration of output polarity control 01 Output is forced to logic 1 state prior to consideration of output polarity control 1x Output is tristated

### 34.3.7 Status Register (PWM\_SMnSR)

Address: 0xEC08_801A (PWM_SM0SR) 0xEC08_806A (PWM_SM1SR)				0xEC08_80BA (PWM_SM2SR) 0xEC08_810A (PWM_SM3SR)				Access: User read/write								
R	15	14	13	12	RF	CFA1	CFA0	CFB1	CFB0	CFX1	CFX0	w1c	w1c	w1c	w1c	w1c
W					RF	0	0	0	0	0	0	0	0	0	0	0
Reset	0	0	0	0								0	0	0	0	0

Figure 34-8. Status Register (PWM\_SMnSR)

Table 34-11. PWM\_SMnSR Field Descriptions

Field	Description
15	Reserved, must be cleared.
14 RUF	Registers updated. Indicates when one of the PWM_SMnINIT, PWM_SMnVALm, or PWM_SMnCR1[PRSC] registers has been written resulting in non-coherent data in the set of double-buffered registers. Clear this bit by a proper reload sequence: a reload signal while LDOK is set. 0 No register update has occurred since last reload 1 At least one of the double-buffered registers has been updated since the last reload
13 REF	Reload error. Indicates when a reload cycle occurs while LDOK is cleared and the double-buffered registers are in a non-coherent state (RUF is set). Write one to clear this bit. 0 No reload error occurred 1 Reload signal occurred with non-coherent data and LDOK is cleared
12 RF	Reload. Indicates the beginning of every reload cycle, regardless of the state of LDOK. <ul style="list-style-type: none"><li>• If PWM_SMnDMAEN[VALDE] is cleared (non-DMA mode), write one to this bit to clear it.</li><li>• If PWM_SMnDMAEN[VALDE] is set (DMA mode), the DMA done signal clears this bit.</li></ul> 0 No new reload cycle since last RF clearing 1 New reload cycle since last RF clearing
11 CFA1 10 CFA0	Capture flag A1/A0. Indicates when the word count of CA1CNT or CA0CNT exceeds the value of the CFAWM field. <ul style="list-style-type: none"><li>• If PWM_SMnDMAEN[CA1DE/CA0DE] is cleared (non-DMA mode), write one to each bit to clear it.</li><li>• If PWM_SMnDMAEN[CA1DE/CA0DE] is set (DMA mode), the DMA done signal clears each bit.</li></ul>
9 CFB1 8 CFB0	Capture flag B1/B0. Indicates when the word count of CB1CNT or CB0CNT exceeds the value of the CFBWM field. <ul style="list-style-type: none"><li>• If PWM_SMnDMAEN[CB1DE/CB0DE] is cleared (non-DMA mode), write one to each bit to clear it.</li><li>• If PWM_SMnDMAEN[CB1DE/CB0DE] is set (DMA mode), the DMA done signal clears each bit.</li></ul>
7 CFX1 6 CFX0	Capture flag X1/X0. Indicates when the word count of CX1CNT or CX0CNT exceeds the value of the CFXWM field. <ul style="list-style-type: none"><li>• If PWM_SMnDMAEN[CX1DE/CX0DE] is cleared (non-DMA mode), write one to each bit to clear it.</li><li>• If PWM_SMnDMAEN[CX1DE/CX0DE] is set (DMA mode), the DMA done signal clears each bit.</li></ul>
5–0 CMF	Compare. Indicates when the submodule's counter value matches the value of one of its PWM_SMnVALm registers. Clear these bits by writing a 1 to a bit position. 0 No compare event has occurred for a particular PWM_SMnVALm value 1 A compare event has occurred for a particular PWM_SMnVALm value

### 34.3.8 Interrupt Enable Register (PWM\_SMnIER)

Each bit in PWM\_SMnSR requests an interrupt if the corresponding bit in the PWM\_SMnIER is set.

												Access: User read/write				
				0xEC08_80AC (PWM_SM2IER) 0xEC08_810C (PWM_SM3IER)												
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	0	0	REIE	RIE	CA1 IE	CA0 IE	CB1 IE	CB0 IE	CX1 IE	CX0 IE			CMPIE			
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-9. Interrupt Enable Register (PWM\_IER)

Table 34-12. PWM\_IER Field Descriptions

Field	Description
15–14	Reserved, must be cleared
13 REIE	Reload error interrupt enable.
12 RIE	Reload interrupt enable.
11 CA1IE	Capture A 1 interrupt enable. <b>Note:</b> Do not set this bit and PWM_SMnDMAEN[CA1DE].
10 CA0IE	Capture A 0 interrupt enable. <b>Note:</b> Do not set this bit and PWM_SMnDMAEN[CA0DE].
9 CB1IE	Capture B 1 interrupt enable. <b>Note:</b> Do not set this bit and PWM_SMnDMAEN[CB1DE].
8 CB0IE	Capture B 0 interrupt enable. <b>Note:</b> Do not set this bit and PWM_SMnDMAEN[CB0DE].
7 CX1IE	Capture X 1 interrupt enable. <b>Note:</b> Do not set this bit and PWM_SMnDMAEN[CX1DE].
6 CX0IE	Capture X 0 interrupt enable. <b>Note:</b> Do not set this bit and PWM_SMnDMAEN[CX0DE].
5–0 CMPIE	Compare interrupt enables.

### 34.3.9 DMA Enable Register (PWM\_SMnDMAEN)

The PWM module may request DMA service for certain operations.

												Access: User read/write				
				0xEC08_80BE (PWM_SM2DMAEN) 0xEC08_810E (PWM_SM3DMAEN)												
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	0	0	0	0	0	0	VAL DE	FAND	CAPTDE	CA1 DE	CA0 DE	CB1 DE	CB0 DE	CX1 DE	CX0 DE	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-10. DMA Enable Register (PWM\_SMnDMAEN)

**Table 34-13. PWM\_SMnDMAEN Field Descriptions**

<b>Field</b>	<b>Description</b>
15–10	Reserved, must be cleared.
9 VALDE	Value registers DMA enable. Enables DMA write requests to the PWM_SMnVALm registers when PWM_SMnSR[RF] is set. 0 Disabled 1 Enabled
8 FAND	FIFO watermark AND control. If CAPTDE is set to watermark mode (CAPTDE = 01), this bit determines the operation of the FIFO watermarks selected in CA <sub>n</sub> DE, CB <sub>n</sub> DE, and CX <sub>n</sub> DE. 0 OR operation is used 1 AND operation is used
7–6 CAPTDE	Capture DMA enable source select. Selects the source of enabling the DMA read requests for the capture FIFOs. 00 Read DMA requests disabled 01 Exceeding a FIFO watermark sets the DMA read request. At least one of the Cx <sub>n</sub> DE bits must be set to determine which watermarks the DMA request is sensitive to. 10 A local sync (PWM_SMnVAL1 matches counter) sets the read DMA request 11 A local reload (RF set) sets the read DMA request
5 CA1DE	Capture A1 FIFO DMA enable. Enables DMA read requests for the Capture A1 FIFO data when CFA1 is set. <b>Note:</b> Do not set this bit and CA1IE.
4 CA0DE	Capture A0 FIFO DMA enable. Enables DMA read requests for the Capture A0 FIFO data when CFA0 is set. <b>Note:</b> Do not set this bit and CA0IE.
3 CB1DE	Capture B1 FIFO DMA enable. Enables DMA read requests for the Capture B1 FIFO data when CFB1 is set. <b>Note:</b> Do not set this bit and CB1IE.
2 CB0DE	Capture B0 FIFO DMA enable. Enables DMA read requests for the Capture B0 FIFO data when CFB0 is set. <b>Note:</b> Do not set this bit and CB0IE.
1 CX1DE	Capture X1 FIFO DMA enable. Enables DMA read requests for the Capture X1 FIFO data when CFX1 is set. <b>Note:</b> Do not set this bit and CX1IE.
0 CX0DE	Capture X0 FIFO DMA enable. Enables DMA read requests for the Capture X0 FIFO data when CFX0 is set. <b>Note:</b> Do not set this bit and CX0IE.

### 34.3.10 Output Trigger Control Register (PWM\_SMnOTCR)

This register enables the generation of OUT\_TRIG0 and OUT\_TRIG1 if the counter value matches one or more of the PWM\_SMnVAL0–5 registers. The output triggers are assigned to the value registers as follows:

**Table 34-14. OUT\_TRIGn vs. PWM\_SMnVAL[5:0]**

<b>Output Trigger</b>	<b>PWM_SMnVAL[5:0]</b>					
	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
OUT_TRIG0		✓		✓		✓
OUT_TRIG1	✓		✓		✓	

## Motor Control Pulse-Width Modulator (mcPWM)

Address: 0xEC08_8020 (PWM_SM0OTCR) 0xEC08_8070 (PWM_SM1OTCR)				0xEC08_80C0 (PWM_SM2OTCR) 0xEC08_8110 (PWM_SM3OTCR)				Access: User read/write								
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 34-11. Output Trigger Control Register (PWM\_SMnOTCR)

Table 34-15. PWM\_SMnOTCR Field Descriptions

Field	Description
15–6	Reserved, must be cleared
5–0 OTEN	Output trigger enables. The OUT_TRIGGER signals are only asserted when the counter value matches the PWM_SMnVALm value. Therefore, up to six triggers can be generated per PWM cycle per submodule. 0 Disabled 1 Enabled

### 34.3.11 Fault Disable Mapping Register (PWM\_SMnDISMAP)

This register determines which PWM pins are disabled by the fault protection inputs, illustrated in Section 34.4.8.11, “Fault Protection”.

Address: 0xEC08_8022 (PWM_SM0DISMAP) 0xEC08_8072 (PWM_SM1DISMAP)				0xEC08_80C2 (PWM_SM2DISMAP) 0xEC08_8112 (PWM_SM3DISMAP)				Access: User read/write								
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	1	1	1	1	DISX				DISB				DISA			

Reset 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Figure 34-12. Fault Disable Mapping Register (PWM\_SMnDISMAP)

Table 34-16. PWM\_SMnDISMAP Field Descriptions

Field	Description
15–12	Reserved, must be set.
11–8 DISX	PWM_X fault disable mask. Each of the bits of this field corresponds to the PWM_FAULTn inputs. 0 The corresponding PWM_FAULTn has no effect on the PWM_X output 1 If the corresponding PWM_FAULTn input signal is asserted, the PWM_X output is turned off
7–4 DISB	PWM_B fault disable mask. Each of the bits of this field corresponds to the PWM_FAULTn inputs. 0 The corresponding PWM_FAULTn has no effect on the PWM_B output 1 If the corresponding PWM_FAULTn input signal is asserted, the PWM_B output is turned off
3–0 DISA	PWM_A fault disable mask. Each of the bits of this field corresponds to the PWM_FAULTn inputs. 0 The corresponding PWM_FAULTn has no effect on the PWM_A output 1 If the corresponding PWM_FAULTn input signal is asserted, the PWM_A output is turned off

### 34.3.12 Deadtime Count Registers (PWM\_SMnDTCNT $m$ )

Deadtime operation is only applicable to complementary channel operation (PWM\_SMnCR2[INDEP] = 0). The 12-bit values written to these registers are in terms of peripheral bus clock cycles regardless of the setting of PWM\_SMnCR1[PRSC] or PWM\_SMnCR2[CLK\_SEL]. Reset sets all bits of the field, selecting a deadtime of 4095 bus clock cycles. These registers are not byte-accessible.

**Figure 34-13. Deadtime Count Registers (PWM\_SMnDTCNT $m$ )**

**Table 34-17. PWM SM<sub>n</sub>DTCNT<sub>m</sub> Field Descriptions**

Field	Description
15–11	Reserved, must be cleared.
10–0 DT	DTCNT0[DT] controls the deadtime during 0 to 1 transitions of the PWMA output (assuming normal polarity). DTCNT1[DT] controls the deadtime during 0 to 1 transitions of the complementary PWMB output.

### 34.3.13 Capture Control Registers (PWM\_SMnCCRx)

**Figure 34-14. Capture Control Registers (PWM SMnCCRx)**

**Table 34-18. PWM\_SMnCCRx Field Descriptions**

<b>Field</b>	<b>Description</b>
15–13 C1CNT	Capture A1/B1/X1 FIFO word count. Reflects the number of words in the capture A1/B1/X1 FIFO.
12–10 C0CNT	Capture A0/B0/X0 FIFO word count. Reflects the number of words in the capture A0/B0/X0 FIFO.
9–8 CFWM	Capture FIFOs watermark. Defines the watermark level for capture A/B/X FIFOs. The capture flags in PWM_SMnSR are not set until the word count of the corresponding FIFO is greater than this watermark level.
7 EDGCNT _EN	Edge counter A/B/X enable. Enables the edge counter which counts rising and falling edges on the PWM input signal. 0 Edge counter disabled and held in reset 1 Edge counter enabled
6 INP_SEL	Input select . Selects between the raw PWM input signal and the output of the edge counter/compare circuitry as the source for the input capture circuit. 0 Raw PWM input signal selected as source 1 Output of edge counter/compare selected as source <b>Note:</b> When this bit is set, the internal edge counter is enabled and the rising or falling edges specified by the EDG0 and EDG1 fields are ignored. The software must still write a value other than 00 in EDG0 or EDG1 to enable one or both of the capture registers.
5–4 EDG1	Edge 1. Determines which input edges cause a capture event on input capture 1. 00 Disabled 01 Capture falling edges 10 Capture rising edges 11 Capture any edge
3–2 EDG0	Edge 0. Determines which input edges cause a capture event on input capture 0. 00 Disabled 01 Capture falling edges 10 Capture rising edges 11 Capture any edge

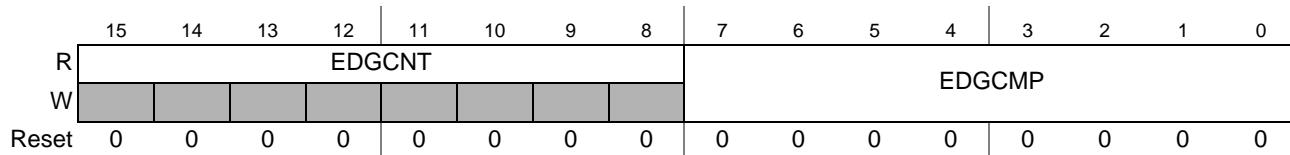
**Table 34-18. PWM\_SMnCCR<sub>x</sub> Field Descriptions (continued)**

Field	Description							
1 ONE SHOT	One shot mode. Selects between free running and one shot mode for the input capture circuitry.							
	<b>ONE SHOT</b>	<b>EDG0</b>	<b>EDG1</b>	Description				
	0	0	1	<ul style="list-style-type: none"> <li>Captures repeat indefinitely on the enabled capture circuit</li> </ul>				
		1	0	<ul style="list-style-type: none"> <li>Capture circuit 0 is armed after ARM is set</li> <li>After capture circuit 0 performs a capture, it is disarmed and capture circuit 1 is armed</li> <li>After capture circuit 1 performs a capture, it is disarmed and capture circuit 0 is re-armed</li> <li>The process repeats indefinitely</li> </ul>				
	1	0	1	<ul style="list-style-type: none"> <li>A single capture occurs on the enabled capture circuit and the ARM bit is cleared</li> </ul>				
		1	0	<ul style="list-style-type: none"> <li>Capture circuit 0 is armed after ARM is set</li> <li>After capture circuit 0 performs a capture, it is disarmed and capture circuit 1 is armed</li> <li>After capture circuit 1 performs a capture, it is disarmed and the ARM bit is cleared</li> <li>No further captures are performed until the ARM bit is set again.</li> </ul>				
0 ARM	Arm. Starts the input capture process. This bit can be cleared at any time to disable input capture operation. This bit is self-cleared when in one-shot mode and the enabled capture circuit has had a capture event. 0 Input capture operation is disabled 1 Input capture operation as specified by the EDG <sub>n</sub> bits is enabled.							

### 34.3.14 Capture Compare Registers (PWM\_SMnCCMP<sub>x</sub>)

Address: 0xEC08\_802A (PWM\_SM0CCMPA)      0xEC08\_80CA (PWM\_SM2CCMPA)      Access: User read/write  
 0xEC08\_802E (PWM\_SM0CCMPB)      0xEC08\_80CE (PWM\_SM2CCMPB)  
 0xEC08\_8032 (PWM\_SM0CCMPX)      0xEC08\_80D2 (PWM\_SM2CCMPX)

0xEC08\_807A (PWM\_SM1CCMPA)      0xEC08\_811A (PWM\_SM3CCMPA)  
 0xEC08\_807E (PWM\_SM1CCMPB)      0xEC08\_811E (PWM\_SM3CCMPB)  
 0xEC08\_8082 (PWM\_SM1CCMPX)      0xEC08\_8122 (PWM\_SM3CCMPX)

**Figure 34-15. Capture Compare Registers (PWM\_SMnCCMP<sub>x</sub>)**

**Table 34-19. PWM\_SMnCCMPx Field Descriptions**

Field	Description
15–8 EDGCNT	Edge counter. Reflects the edge counter value for each input capture circuitry.
7–0 EDGCM	Edge compare. Indicates the compare value associated with the edge counter for the input capture circuitry.

### 34.3.15 Capture Value Registers (PWM\_SMnCVALm)

Address: 0xEC08_8034 (PWM_SM0CVAL0)	0xEC08_80D4 (PWM_SM2CVAL0)	Access: User read-only																																																																		
0xEC08_8038 (PWM_SM0CVAL1)	0xEC08_80D8 (PWM_SM2CVAL1)																																																																			
0xEC08_803C (PWM_SM0CVAL2)	0xEC08_80DC (PWM_SM2CVAL2)																																																																			
0xEC08_8040 (PWM_SM0CVAL3)	0xEC08_80E0 (PWM_SM2CVAL3)																																																																			
0xEC08_8044 (PWM_SM0CVAL4)	0xEC08_80E4 (PWM_SM2CVAL4)																																																																			
0xEC08_8048 (PWM_SM0CVAL5)	0xEC08_80E8 (PWM_SM2CVAL5)																																																																			
0xEC08_8084 (PWM_SM1CVAL0)	0xEC08_8124 (PWM_SM3CVAL0)																																																																			
0xEC08_8088 (PWM_SM1CVAL1)	0xEC08_8128 (PWM_SM3CVAL1)																																																																			
0xEC08_808C (PWM_SM1CVAL2)	0xEC08_812C (PWM_SM3CVAL2)																																																																			
0xEC08_8090 (PWM_SM1CVAL3)	0xEC08_8130 (PWM_SM3CVAL3)																																																																			
0xEC08_8094 (PWM_SM1CVAL4)	0xEC08_8134 (PWM_SM3CVAL4)																																																																			
0xEC08_8098 (PWM_SM1CVAL5)	0xEC08_8138 (PWM_SM3CVAL5)																																																																			
<table border="1"> <tr> <td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>9</td><td>8</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td> </tr> <tr> <td>R</td><td></td><td></td><td></td><td colspan="10">CAPTVAL</td><td></td><td></td><td></td><td></td> </tr> <tr> <td>W</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> <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> </tr> </table>			15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	R				CAPTVAL														W																Reset	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				CAPTVAL																																																																
W																																																																				
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																																																					

**Figure 34-16. Capture Value Registers (PWM\_SMnCVALm)****Table 34-20. PWM\_SMnCVALm Field Descriptions**

Field	Description														
15–0 CAPTVAL	Capture value. Stores the value captured from the submodule counter. The corresponding PWM_SMnCCR <sub>X</sub> [EDG0,EDG1] bit shown below determines exactly when this capture occurs. This register is not byte-accessible. <table border="1" style="margin-top: 10px;"> <tr> <th>Capture Register</th> <th>EDG bit</th> </tr> <tr> <td>PWM_SMnCVAL0</td> <td>PWM_SMnCCR<sub>X</sub>[EDG0]</td> </tr> <tr> <td>PWM_SMnCVAL1</td> <td>PWM_SMnCCR<sub>X</sub>[EDG1]</td> </tr> <tr> <td>PWM_SMnCVAL2</td> <td>PWM_SMnCCRA[EDGA0]</td> </tr> <tr> <td>PWM_SMnCVAL3</td> <td>PWM_SMnCCRA[EDGA1]</td> </tr> <tr> <td>PWM_SMnCVAL4</td> <td>PWM_SMnCCRB[EDGB0]</td> </tr> <tr> <td>PWM_SMnCVAL5</td> <td>PWM_SMnCCRB[EDGB1]</td> </tr> </table> <p><b>Note:</b> This is actually a 4-deep FIFO and not a single register.</p>	Capture Register	EDG bit	PWM_SMnCVAL0	PWM_SMnCCR <sub>X</sub> [EDG0]	PWM_SMnCVAL1	PWM_SMnCCR <sub>X</sub> [EDG1]	PWM_SMnCVAL2	PWM_SMnCCRA[EDGA0]	PWM_SMnCVAL3	PWM_SMnCCRA[EDGA1]	PWM_SMnCVAL4	PWM_SMnCCRB[EDGB0]	PWM_SMnCVAL5	PWM_SMnCCRB[EDGB1]
Capture Register	EDG bit														
PWM_SMnCVAL0	PWM_SMnCCR <sub>X</sub> [EDG0]														
PWM_SMnCVAL1	PWM_SMnCCR <sub>X</sub> [EDG1]														
PWM_SMnCVAL2	PWM_SMnCCRA[EDGA0]														
PWM_SMnCVAL3	PWM_SMnCCRA[EDGA1]														
PWM_SMnCVAL4	PWM_SMnCCRB[EDGB0]														
PWM_SMnCVAL5	PWM_SMnCCRB[EDGB1]														

### 34.3.16 Capture Value Cycle Registers (PWM\_SMnCCYC $m$ )

Address:	0xEC08_8036 (PWM_SM0CCYC0) 0xEC08_803A (PWM_SM0CCYC1) 0xEC08_803E (PWM_SM0CCYC2) 0xEC08_8042 (PWM_SM0CCYC3) 0xEC08_8046 (PWM_SM0CCYC4) 0xEC08_804A (PWM_SM0CCYC5)	0xEC08_80D6 (PWM_SM2CCYC0) 0xEC08_80DA (PWM_SM2CCYC1) 0xEC08_80DE (PWM_SM2CCYC2) 0xEC08_80E2 (PWM_SM2CCYC3) 0xEC08_80E6 (PWM_SM2CCYC4) 0xEC08_80EA (PWM_SM2CCYC5)	Access: User read-only
	0xEC08_8086 (PWM_SM1CCYC0) 0xEC08_808A (PWM_SM1CCYC1) 0xEC08_808E (PWM_SM1CCYC2) 0xEC08_8092 (PWM_SM1CCYC3) 0xEC08_8096 (PWM_SM1CCYC4) 0xEC08_809A (PWM_SM1CCYC5)	0xEC08_8126 (PWM_SM3CCYC0) 0xEC08_812A (PWM_SM3CCYC1) 0xEC08_812E (PWM_SM3CCYC2) 0xEC08_8132 (PWM_SM3CCYC3) 0xEC08_8136 (PWM_SM3CCYC4) 0xEC08_813A (PWM_SM3CCYC5)	
R	15 14 13 12	11 10 9 8	7 6 5 4
W	0 0 0 0	0 0 0 0	0 0 0 0
Reset	0 0 0 0	0 0 0 0	0 0 0 0
			CYC
			3 2 1 0

Figure 34-17. Capture Value Registers (PWM\_SMnCCYC $m$ )Table 34-21. PWM\_SMnCCYC $m$  Field Descriptions

Field	Description
15–0 CYC	Stores the cycle number corresponding to the value captured in PWM_SMnCVAl $m$ . This field resets to 0 and increments each time the counter is loaded with the PWM_SMnINIT value. <b>Note:</b> This is actually a 4-deep FIFO and not a single register.

### 34.3.17 Output Enable Register (PWM\_OUTEN)

Address:	0xEC08_8140 (PWM_OUTEN)	Access: User read/write
R	15 14 13 12	11 10 9 8
W	0 0 0 0	0 0 0 0
Reset	0 0 0 0	0 0 0 0
	PWMA_EN	PWMB_EN
		PWMX_EN
		3 2 1 0

Figure 34-18. Output Enable Register (PWM\_OUTEN)

Table 34-22. PWM\_OUTEN Field Descriptions

Field	Description
15–12	Reserved, must be cleared.
11–8 PWMA_EN	PWMA output enable. Enables the PWM_A outputs of each submodule. Clear these bits when a PWM_A pin is configured for input capture. 0 Disabled 1 Enabled

**Table 34-22. PWM\_OUTEN Field Descriptions (continued)**

Field	Description
7–4 PWMB_EN	PWMB output enable. Enables the PWM_B outputs of each submodule. Clear these bits when a PWM_B pin is configured for input capture. 0 Disabled 1 Enabled
3–0 PWMX_EN	PWMX output enable. Enables the PWM_X outputs of each submodule. Clear these bits when a PWM_X pin is configured for input capture. 0 Disabled 1 Enabled

### 34.3.18 Mask Register (PWM\_MASK)

#### NOTE

The MASKx bits are double buffered and do not take effect until a FORCE\_OUT event occurs within the appropriate submodule. Refer to [Figure 34-41](#) to see how FORCE\_OUT is generated. Reading MASKx reads the buffered value and not necessarily the value currently in effect.

Address: 0xEC08\_8142 (PWM\_MASK)

Access: User read/write

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

**Figure 34-19. Mask Register (PWM\_MASK)****Table 34-23. PWM\_MASK Field Descriptions**

Field	Description
15–12	Reserved, must be cleared.
11–8 MASKA	PWMA masks. Masks the PWM_A outputs of each submodule forcing the output to logic 0 prior to consideration of the output polarity. 0 Output normal 1 Output masked
7–4 MASKB	PWMB masks. Masks the PWM_B outputs of each submodule forcing the output to logic 0 prior to consideration of the output polarity. 0 Output normal 1 Output masked
3–0 MASKX	PWMX masks. Masks the PWM_X outputs of each submodule forcing the output to logic 0 prior to consideration of the output polarity. 0 Output normal 1 Output masked

### 34.3.19 Software-Controlled Output Register (PWM\_SWCOUT)

#### NOTE

These bits are double buffered and do not take effect until a FORCE\_OUT event occurs within the appropriate submodule. Refer to [Figure 34-41](#) to see how FORCE\_OUT is generated. Reading these bits reads the buffered value and not necessarily the value currently in effect.

Address: 0xEC08_8144 (PWM_SWCOUT)																Access: User read/write							
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0							
W	0	0	0	0	0	0	0	0	OUT 23_3	OUT 45_3	OUT 23_2	OUT 45_2	OUT 23_1	OUT 45_1	OUT 23_0	OUT 45_0							
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							

**Figure 34-20. Software-Controlled Output Register (PWM\_SWCOUT)**

**Table 34-24. PWM\_SWCOUT Field Descriptions**

Field	Description
15–8	Reserved, must be cleared.
7 OUT23_3	Software controlled output 23_3. When SEL23 for submodule 3 is 0b10, this bit allows software control of the signal supplied to the submodule's deadtime generator. 0 A logic 0 is supplied to the deadtime generator of submodule 3 instead of PWM23 1 A logic 1 is supplied to the deadtime generator of submodule 3 instead of PWM23
6 OUT45_3	Software controlled output 45_3. See OUT23_3 bit description.
5 OUT23_2	Software controlled output 23_2. See OUT23_3 bit description.
4 OUT45_2	Software controlled output 45_2. See OUT23_3 bit description.
3 OUT23_1	Software controlled output 23_1. See OUT23_3 bit description.
2 OUT45_1	Software controlled output 45_1. See OUT23_3 bit description.
1 OUT23_0	Software controlled output 23_0. See OUT23_3 bit description.
0 OUT45_0	Software controlled output 45_0. See OUT23_3 bit description.

### 34.3.20 Deadtime Source Select Register (PWM\_DTSS)

#### NOTE

The deadtime source select bits are double buffered and do not take effect until a FORCE\_OUT event occurs within the appropriate submodule. Refer to [Figure 34-41](#) to see how FORCE\_OUT is generated. Reading these bits reads the buffered value and not necessarily the value currently in effect.

Address: 0xEC08_8146 (PWM_DTSS)								Access: User read/write								
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	SEL23_3	SEL45_3	SEL23_2	SEL45_2	SEL23_1	SEL45_1	SEL23_0	SEL45_0								
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-21. Deadtime Source Select Register (PWM\_DTSS)

Table 34-25. PWM\_DTSS Field Descriptions

Field	Description
15–14 SEL23_3	PWM23_3 control select. Selects possible overrides to the generated PWM23 signal in submodule 3 that is passed to the deadtime logic upon the occurrence of a FORCE_OUT event in that submodule. See <a href="#">Figure 34-41</a> . 00 Generated PWM23_3 signal is used by the deadtime logic 01 Inverted generated PWM23_3 signal is used by the deadtime logic 10 OUT23_3 bit is used by the deadtime logic 11 PWM_EXTA3 signal is used by the deadtime logic
13–12 OUT45_3	PWM45_3 control select. Selects possible overrides to the generated PWM45 signal in submodule 3 that is passed to the deadtime logic upon the occurrence of a FORCE_OUT event in that submodule. See <a href="#">Figure 34-41</a> . 00 Generated PWM45_3 signal is used by the deadtime logic 01 Inverted generated PWM45_3 signal is used by the deadtime logic 10 OUT45_3 bit is used by the deadtime logic 11 Reserved
11–10 SEL23_2	PWM23_2 control select. See SEL23_3 bit description.
9–8 SEL45_2	PWM45_2 control select. See SEL45_3 bit description.
7–6 SEL23_1	PWM23_1 control select. See SEL23_3 bit description.
5–4 SEL45_1	PWM45_1 control select. See SEL45_3 bit description.
3–2 SEL23_0	PWM23_0 control select. See SEL23_3 bit description.
1–0 SEL45_0	PWM45_0 control select. See SEL45_3 bit description.

### 34.3.21 Master Control Register (PWM\_MCR)

The PWM\_MCR contains four 4-bit fields. Each bit within the fields controls the corresponding submodule.

Address: 0xEC08_8148 (PWM_MCR)								Access: User read/write								
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	IPOL				RUN				0	0	0	0		LDOK		
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 34-22. Master Control Register (PWM\_MCR)

**Table 34-26. PWM\_MCR Field Descriptions**

Field	Description
15–12 IPOL	Current polarity. Selects between PWM23 and PWM45 as the source for the generation of the complementary PWM pair output. This bit is ignored in independent mode (PWM_SMnCR2[INDEP]=1). 0 PWM23 1 PWM45 <b>Note:</b> This bit does not take effect until a FORCE_OUT event takes place in the appropriate submodule. Reading this bit reads the buffered value and not necessarily the value currently in effect.
11–8 RUN	Enables the clocks to the PWM generator. In submodules other than 0, when CLK_SEL is 10 the local RUN bit is ignored; this indicates that the AUX_CLK from submodule 0 is being used by this submodule. A reset clears RUN. 0 PWM generator disabled. The corresponding submodule's counter is reset. 1 PWM generator enabled <b>Note:</b> For proper initialization of the LDOCK and RUN bits, see <a href="#">Section 34.5, “Initialization/Application Information”</a> .
7–4 CLDOK	Clear load okay. This bit is self clearing and always reads as a 0. 0 No effect 1 Write a 1 to clear the corresponding LDOCK. If a reload occurs with LDOCK set at the same time that CLDOK is written, then the reload is not performed and LDOCK is cleared.
3–0 LDOCK	Load okay. Loads PWM_SMnCR1[PRSC], PWM_SMnINIT, and PWM_SMnVALm into a set of buffers. The buffered prescaler divisor, submodule counter modulus value, and PWM pulse width take effect: <ul style="list-style-type: none"><li>• at the next PWM reload if LDMOD is cleared</li><li>• immediately if LDMOD is set</li></ul> Set LDOCK by reading it when it is zero and then writing a logic one to it. The PWM_SMnCR1[PRSC], PWM_SMnINIT, and PWM_SMnVALm registers cannot be written while LDOCK is set. LDOCK is automatically cleared after the new values are loaded, or can be manually cleared by writing a 1 to CLDOK. This bit cannot be written with a zero. LDOCK can be set in DMA mode when the DMA indicates that it has completed the update of the buffered registers. <b>Note:</b> For proper initialization of the LDOCK and RUN bits, see <a href="#">Section 34.5, “Initialization/Application Information”</a> .

### 34.3.22 Fault Control Register (PWM\_FCR)

The PWM\_FCR contains four 3-bit fields. Each bit within a field controls the corresponding PWM\_FAULTn signal.

Address: 0xEC08_814C (PWM_FCR)								Access: User read/write								
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	0	FLVL			0	F AUTO			0	FSAFE			0	FIE		

**Figure 34-23. Fault Control Register (PWM\_FCR)****Table 34-27. PWM\_FCR Field Descriptions**

Field	Description
15	Reserved, must be cleared.
14–12 FLVL	Fault level. Selects the active logic level of the individual fault inputs. 0 A logic 0 on the fault input indicates a fault condition 1 A logic 1 on the fault input indicates a fault condition
11	Reserved, must be cleared.

**Table 34-27. PWM\_FCR Field Descriptions (continued)**

Field	Description
10–8 FAUTO	Automatic fault clearing. Selects automatic or manual clearing of faults. 0 Manual fault clearing. PWM outputs disabled by this fault are not enabled until the corresponding PWM_FSR[FFLAG] bit is cleared at the start of a half cycle or full cycle (depending on PWM_FSR[FFULL]). This is further controlled by the FSAFE bits. 1 Automatic fault clearing. PWM outputs disabled by this fault are enabled when PWM_FSR[FFPIN] is cleared at the start of a half cycle or full cycle (depending on PWM_FSR[FFULL]) without regard to PWM_FSR[FFLAG].
7	Reserved, must be cleared.
6–4 FSAFE	Fault safety mode. Selects the safety mode during manual fault clearing. 0 Normal mode. PWM outputs disabled by this fault are not enabled until PWM_FSR[FLAG] is cleared at the start of a half or full cycle (depending on PWM_FSR[FFULL]) without regard to PWM_FSR[FFPIN]. The PWM outputs are not re-enabled until the PWM_FAULT $n$ signal deasserts since the fault input combinationally disables the PWM outputs (as programmed in PWM_SM $n$ DISMAP). 1 Safe mode. PWM outputs disabled by this fault are not enabled until PWM_FSR[FFLAG] is cleared and PWM_FSR[FFPIN] is cleared at the start of a half or full cycle (depending on PWM_FSR[FFULL]). <b>Note:</b> PWM_FSR[FFPIN] may indicate a fault condition still exists even though PWM_FAULT $n$ pin is deasserted due to the fault filter latency.
3	Reserved, must be cleared.
2–0 FIE	Fault interrupt enables. Enables CPU interrupt requests generated by PWM_FAULT $n$ . 0 Disabled 1 Enabled <b>Note:</b> The fault protection circuit is independent of this field and is always active. If a fault is detected, the PWM outputs are disabled according to PWM_SM $n$ DISMAP.

### 34.3.23 Fault Status Register (PWM\_FSR)

Address: 0xEC08\_814E (PWM\_FSR)

Access: User read/write

R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
W				FTEST			FFPIN		0			FFULL			FFLAG		
Reset	0	0	0	0	0	U	U	U	0	0	0	0	0	0	w1c	w1c	w1c

**Figure 34-24. Fault Status Register (PWM\_FSR)****Table 34-28. PWM\_FSR Field Descriptions**

Field	Description
15–13	Reserved, must be cleared.
12 FTEST	Fault test. Simulates a fault condition. 0 Removes the simulated fault condition 1 Causes a simulated fault to be sent into the fault filters. The condition propagates to the fault flags and possibly the PWM outputs depending on PWM_SM $n$ DISMAP.
11	Reserved, must be cleared.
10–8 FFPIN	Filtered fault pins. Reflects the current state of the filtered PWM_FAULT $n$ pins converted to high polarity. A reset has no effect on FFPIN. 0 No fault 1 Fault exists on the filtered PWM_FAULT $n$ pin

**Table 34-28. PWM\_FSR Field Descriptions (continued)**

Field	Description
7	Reserved, must be cleared.
6–4 FFULL	Full cycle. Controls the timing for re-enabling the PWM outputs after a fault condition. These bits apply to both automatic and manual clearing of a fault condition. 0 PWM outputs are re-enabled at the start of a full or half cycle 1 PWM outputs are re-enabled only at the start of a full cycle
3	Reserved, must be cleared.
2–0 FFLAG	Fault flags. Set within two CPU cycles after a transition to active on PWM_FAULTn. Clear a FFLAG bit by writing a logic one to it. A reset clears FFLAG. 0 No fault on the PWM_FAULTn pin 1 Fault on the PWM_FAULTn pin

### 34.3.24 Fault Filter Register (PWM\_FFILT)

The settings in this register are shared among each of the fault input filters.

Address: 0xEC08_8150 (PWM_FFILT)																Access: User read/write							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0							
R	GSTR	0	0	0	0	0	CNT						PER										
W		0	0	0	0	0		0	0	0	0	0	0	0	0	0							

**Figure 34-25. Fault Filter Register (PWM\_FFILT)****Table 34-29. PWM\_FFILT Field Descriptions**

Field	Description
15 GSTR	Fault glitch stretch enable. Ensures that narrow fault glitches are stretched to at least two peripheral bus clock cycles wide. In some cases a narrow fault input can cause problems due to the short PWM output shutdown/re-activation time. The stretching logic ensures that when the fault filter is disabled, a glitch on the fault input is registered in the fault flags. 0 Disabled 1 Enabled
14–11	Reserved, must be cleared.
10–8 CNT	Fault filter count. Represents the number of consecutive samples that must agree prior to the input filter accepting an input transition. The value of this field affects the input latency as described in <a href="#">Section 34.3.24.1, “Input Filter Considerations”</a> . 000 3 samples 001 4 samples ... 111 10 samples
7–0 PER	Fault filter period. Represents the sampling period (in peripheral bus clock cycles) of the fault pin input filter. Each input is sampled multiple times at the rate specified by PER. The value of this field affects the input latency as described in <a href="#">Section 34.3.24.1, “Input Filter Considerations”</a> . 0x00 Input filter bypassed

### 34.3.24.1 Input Filter Considerations

Set the PER value so that the sampling period is larger than the period of the expected noise. This way a noise spike only corrupts one sample. Choose a CNT value to reduce the probability of noisy samples causing an incorrect transition to be recognized. The probability of an incorrect transition is defined as:

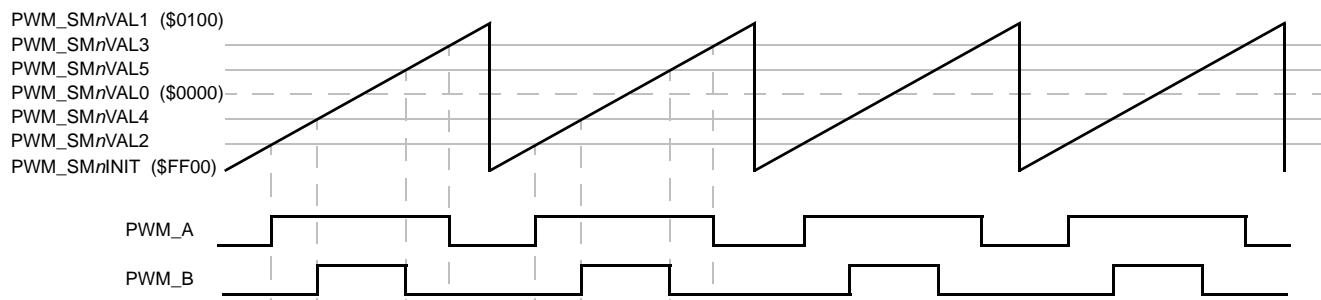
$$P(\text{incorrect transition}) = P(\text{incorrect sample})^{CNT + 3} \quad \text{Eqn. 34-1}$$

The values of PER and CNT must also be traded off against the desire for minimal latency in recognizing input transitions. Turning on the input filter (setting FILT\_PER to a non-zero value) introduces a latency of ((FILT\_CNT+4) x FILT\_PER x IPBus clock period). Note that even when the filter is enabled, there is a combinational path to disable the PWM outputs. This is to ensure rapid response to fault conditions and also to ensure fault response if the PWM module loses its clock. The latency induced by the filter will be seen in the time to set the PWM\_FSR[FFLAG, FFPIN] bits.

## 34.4 Functional Description

### 34.4.1 Center-Aligned PWMs

Each submodule has its own timer that is capable of generating PWM signals on two output pins. The edges of each of these signals are controlled independently as shown in [Figure 34-26](#).



**Figure 34-26. Center Aligned Example**

The submodule timers only count in the up direction and then reset to the PWM\_SMnINIT value. There are two values that must be specified:

- Turn-on edge
- Turn-off edge

This double-action edge generation gives you control over the pulse width and the relative alignment of the signal. As a result, there is no need to support separate PWM alignment modes since the PWM alignment mode is inherently a function of the turn on and turn off edge values.

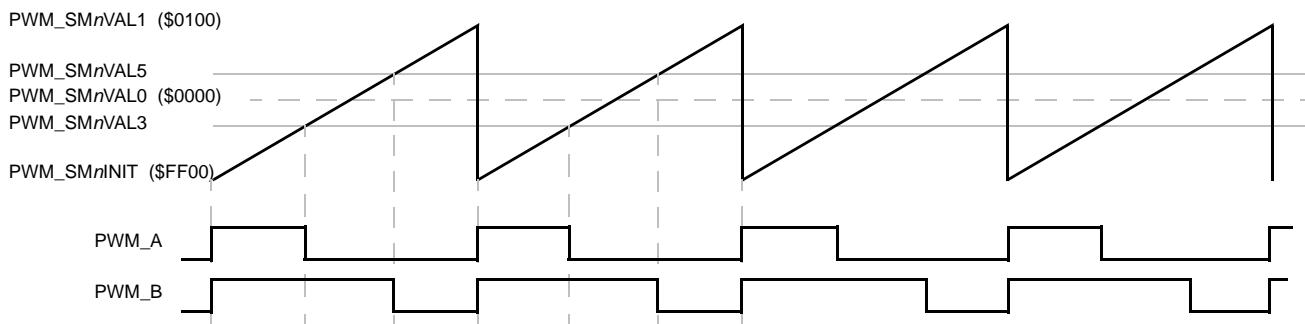
[Figure 34-26](#) also illustrates an additional enhancement to the PWM generation process. When the counter resets, it is reloaded with a user-specified value. If the value chosen is the two's complement of the modulus value, then the PWM generator operates in signed mode. This means that if each PWM's turn-on and turn-off edge values only differ in their sign, the on portion of the output signal is centered around a count value of zero. Therefore, calculate only one PWM value and load this value and its negative to the

submodule as the turn-off and turn-on edges, respectively. This technique results in a pulse width that always consists of an odd number of timer counts. If all PWM signal edge calculations follow this convention, the signals are center-aligned with respect to each other.

Center aligning the signals is not restricted to symmetry around the zero count value, as any other number also works. However, centering on zero provides the greatest range in signed mode and simplifies the calculations.

### 34.4.2 Edge-Aligned PWMs

When the turn-on edge for each pulse is the PWM\_SM<sub>n</sub>INIT value, then the PWM signals are edge-aligned, as illustrated in [Figure 34-27](#). Therefore, only the turn-off edge value must be updated to change the pulse width.



**Figure 34-27. Edge-Aligned Example  
( $\text{PWM}_{\text{SM}n\text{INIT}} = \text{PWM}_{\text{SM}n\text{VAL}2} = \text{PWM}_{\text{SM}n\text{VAL}4}$ )**

Edge-aligned PWMs benefit from signed mode as well. For example, a common way to drive an H-bridge is to use bipolar PWMs where a 50% duty cycle results in zero volts on the load. Duty cycles less than 50% result in negative load voltages and duty cycles greater than 50% generate positive load voltages. If the module is set to signed mode (PWM\_SM<sub>n</sub>INIT and PWM\_SM<sub>n</sub>VAL1 only differ in their sign), then there is a direct proportionality between the PWM turn-off edge value and the motor voltage, including the sign. Since no offset calculations are required to translate the output variable control algorithm to the voltage on an H-bridge load, signed mode simplifies user PWM code.

### 34.4.3 Phase-Shifted PWMs

If numerical biases are applied to the turn-on and turn-off edges of different PWM signals, the signals are phase shifted with respect to each other, as illustrated in [Figure 34-28](#). This results in certain advantages when applied to a power stage.

For example, when operating a multi-phase inverter at a low modulation index, all PWM switching edges from the different phases occur at nearly the same time. This can be troublesome from a noise standpoint, especially if ADC readings of the inverter must be scheduled near those times. Phase shifting the PWM signals can open timing windows between the switching edges for a signal to be sampled. However, phase shifting does not affect the duty cycle, so average load voltage is not affected.

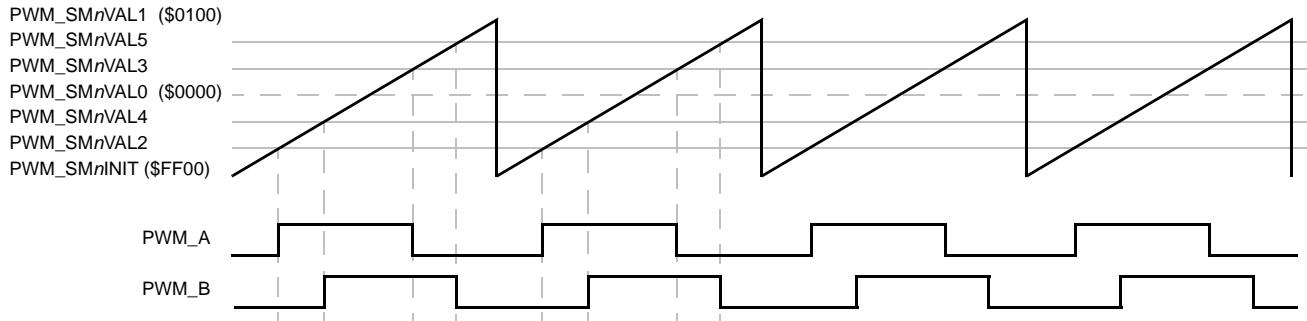


Figure 34-28. Phase-Shifted Outputs Example

Figure 34-29 illustrates an additional benefit of phase-shifted PWMs. In this case, an H-bridge circuit is driven by four PWM signals to control the voltage waveform on the transformer primary. Left- and right-side PWMs are configured to always generate a square wave with 50% duty cycle. No narrow pulse widths are generated, which reduces the high frequency switching requirements of the transistors.

Notice that the square wave on the right side of the H-bridge is phase shifted compared to the left side. As a result, the transformer primary sees the bottom waveform across its terminals. The RMS value of this waveform is controlled directly by the amount of phase shift of the square waves. Regardless of the phase shift, no DC component appears in the load voltage while the duty cycle of each square wave remains at 50% making this technique ideally suited for transformer loads. As a result, this topology is frequently used in industrial welders to adjust the amount of energy delivered to the weld arc.

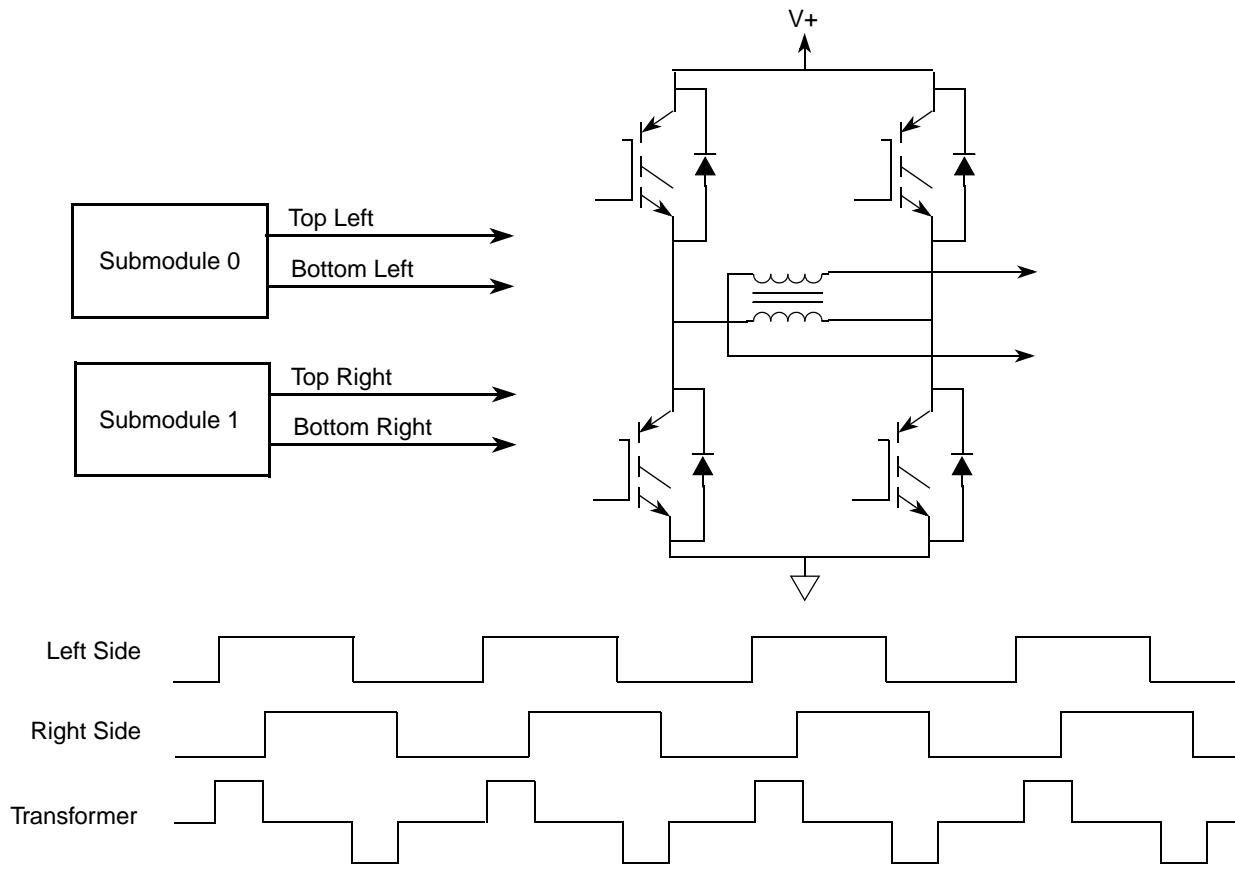


Figure 34-29. Phase-Shifted PWMs Applied to a Transformer Primary

#### 34.4.4 Double-Switching PWMs

Double-switching PWM aids in single shunt current measurement and three phase reconstruction. This method supports two independent rising edges and two independent falling edges per PWM cycle.

- PWM\_SM<sub>n</sub>VAL2 and PWM\_SM<sub>n</sub>VAL3 generate the even channel (PWM\_A in Figure 34-30)
- PWM\_SM<sub>n</sub>VAL4 and PWM\_SM<sub>n</sub>VAL5 generate the odd channel (PWM\_B in Figure 34-30)

The two channels are combined using XOR logic as shown in Figure 34-30. The DBLPWM signal can be run through the deadtime insertion logic.

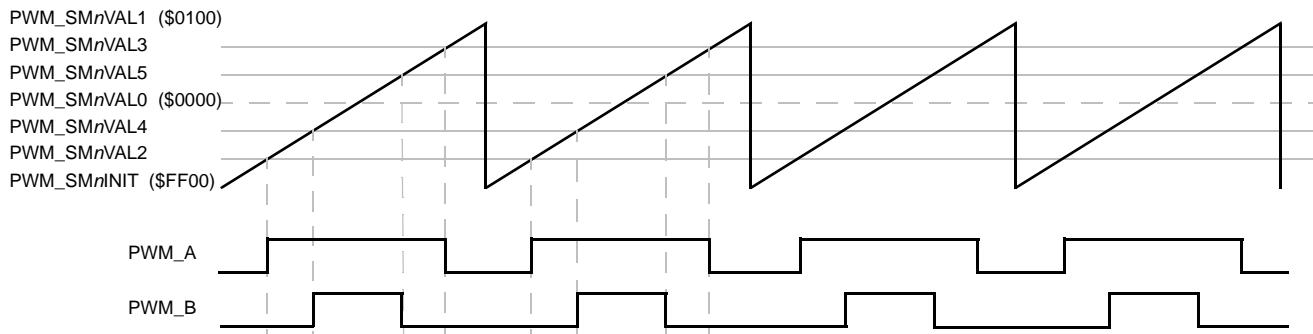


Figure 34-30. Double-Switching Output Example

### 34.4.5 ADC Triggering

When ADC trigger timing is critical, it must be scheduled as a hardware event instead of software activated. With this PWM module, multiple ADC triggers can be generated in hardware per PWM cycle without the requirement of another timer module, as shown in [Figure 34-31](#). When using complimentary mode, only two edge comparators are required to generate the output PWM signals for a given submodule. The other comparators are free to perform other functions. In this example, the software doesn't have to quickly respond after the first conversion to set up other conversions that must occur in the same PWM cycle.

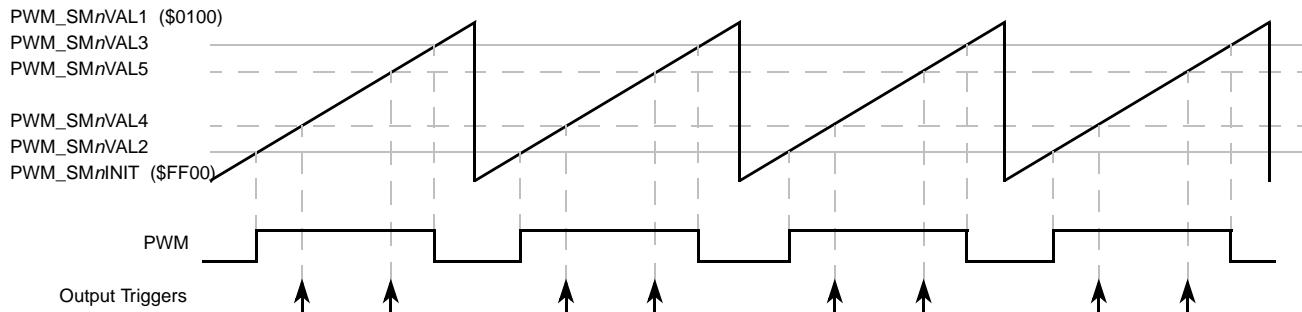


Figure 34-31. Multiple Output Trigger Generation in Hardware

Since each submodule has its own timer, it is possible for each submodule to run at a different frequency. One of possible option is to have one or more submodules running at a lower frequency, but still synchronized to the timer in submodule 0. [Figure 34-32](#) shows how this feature can schedule ADC triggers over multiple PWM cycles. A suggested use for this configuration is to use the lower frequency submodule to control the sampling frequency of the software control algorithm, where multiple ADC triggers can be scheduled over the entire sampling period. In [Figure 34-32](#), all submodule comparators are shown being used for ADC trigger generation.

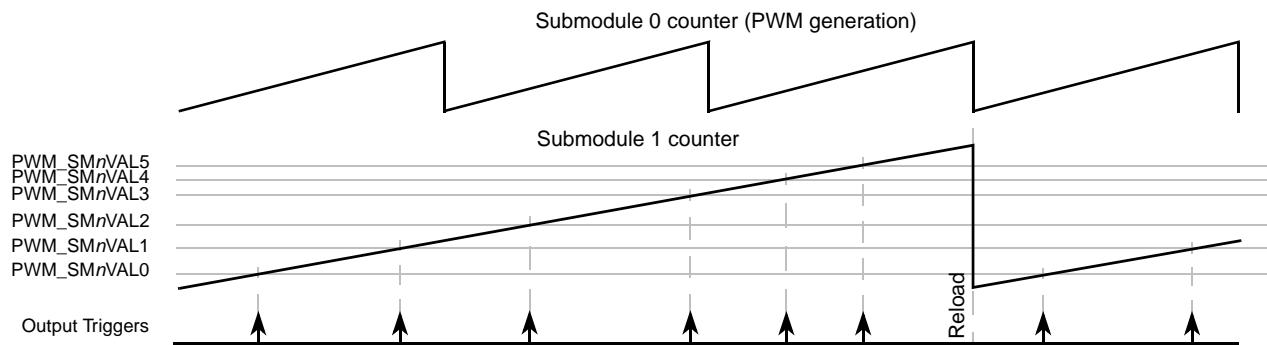


Figure 34-32. Multiple Output Triggers Over Several PWM Cycles

### 34.4.6 Enhanced Capture Capabilities (E-Capture)

When a PWM pin is not used for PWM generation, it can perform input captures. Recall that for PWM generation the edges of the PWM signal are specified by separate compare register values. When programmed for input capture, both of these registers work on the same pin to capture multiple edges, toggling from one to the other in a free-running or one-shot fashion. By simply programming the desired edge of each capture circuit, the period and pulse width of an input signal can easily be measured without the requirement to re-arm the circuit.

In addition, each edge of the input signal can clock an 8-bit counter where the counter output is compared to a user-specified value (EDGCMP). When the counter output equals EDGCMP, the value of the submodule timer is captured and the counter is automatically reset. This feature allows the module to count a specified number of edge events, and then perform a capture and interrupt. [Figure 34-33](#) illustrates some of the functionality of the E-Capture circuit.

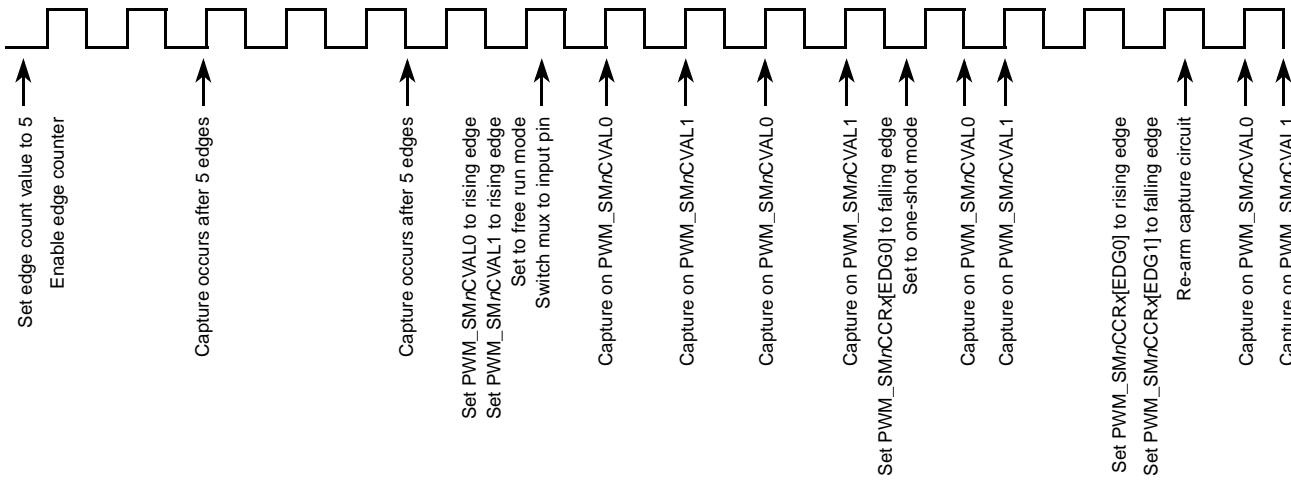


Figure 34-33. Capture Capabilities of the E-Capture Circuit

When a submodule is used for PWM generation, its timer counts up to the modulus value that specifies the PWM frequency and then is re-initialized. Therefore, using this timer for input captures on one of the other pins (e.g., PWM\_X) has limited utility since it does not count through all of the numbers and the timer reset

represents a discontinuity in the 16-bit number range. However, when measuring a signal that is synchronous to the PWM frequency, the timer modulus range is perfectly suited for the application.

For example, consider Figure 34-34. In this application the output of a PWM power stage is connected to the PWM\_X pin, which is configured for free-running input captures:

- PWM\_SMnCVAL0 capture circuitry is programmed for rising edges
- PWM\_SMnCVAL1 capture circuitry is programmed for falling edges

This results in new load pulse width data being acquired every PWM cycle. To calculate the pulse width, subtract PWM\_SMnCVAL0 from PWM\_SMnCVAL1. This measurement is extremely beneficial when performing dead-time distortion correction on a half-bridge circuit driving an inductive load. Also, these values can be directly compared to the PWM\_SMnVALm registers responsible for generating the PWM outputs to measure system propagation delays.

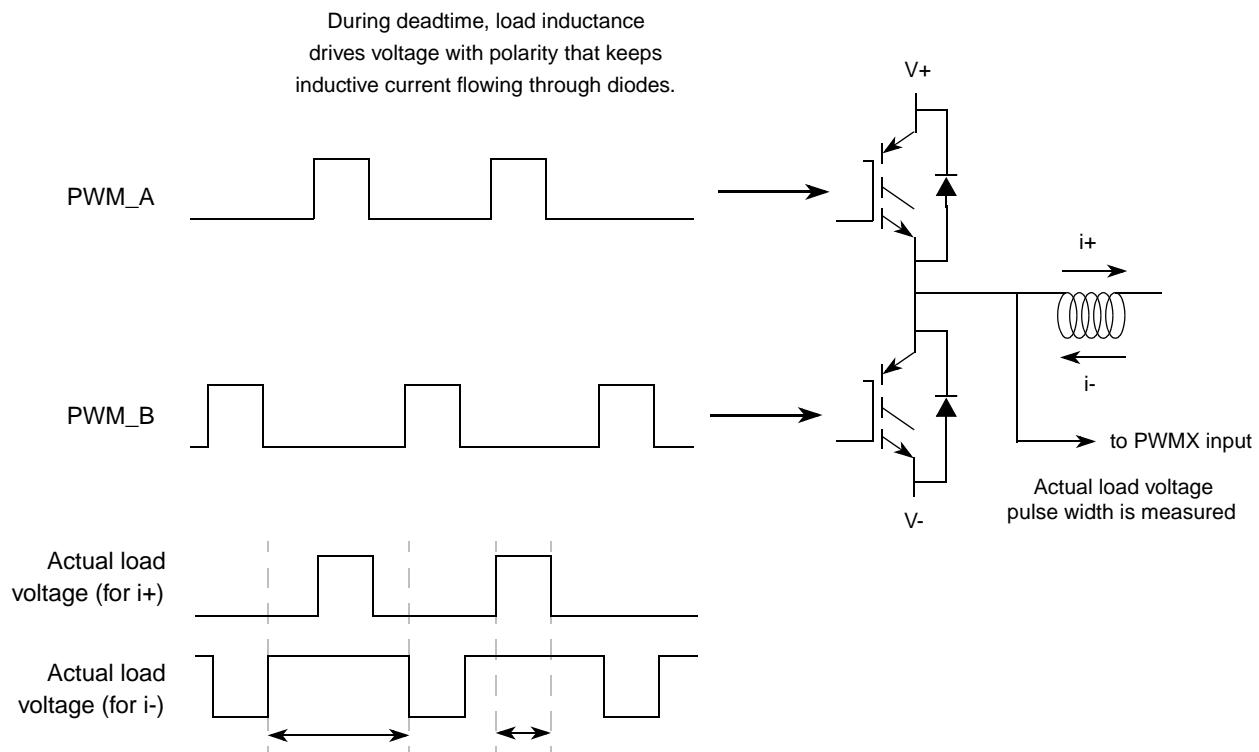


Figure 34-34. Output Pulse Width Measurement Possible with the E-Capture Circuit

#### 34.4.7 Synchronous Switching of Multiple Outputs

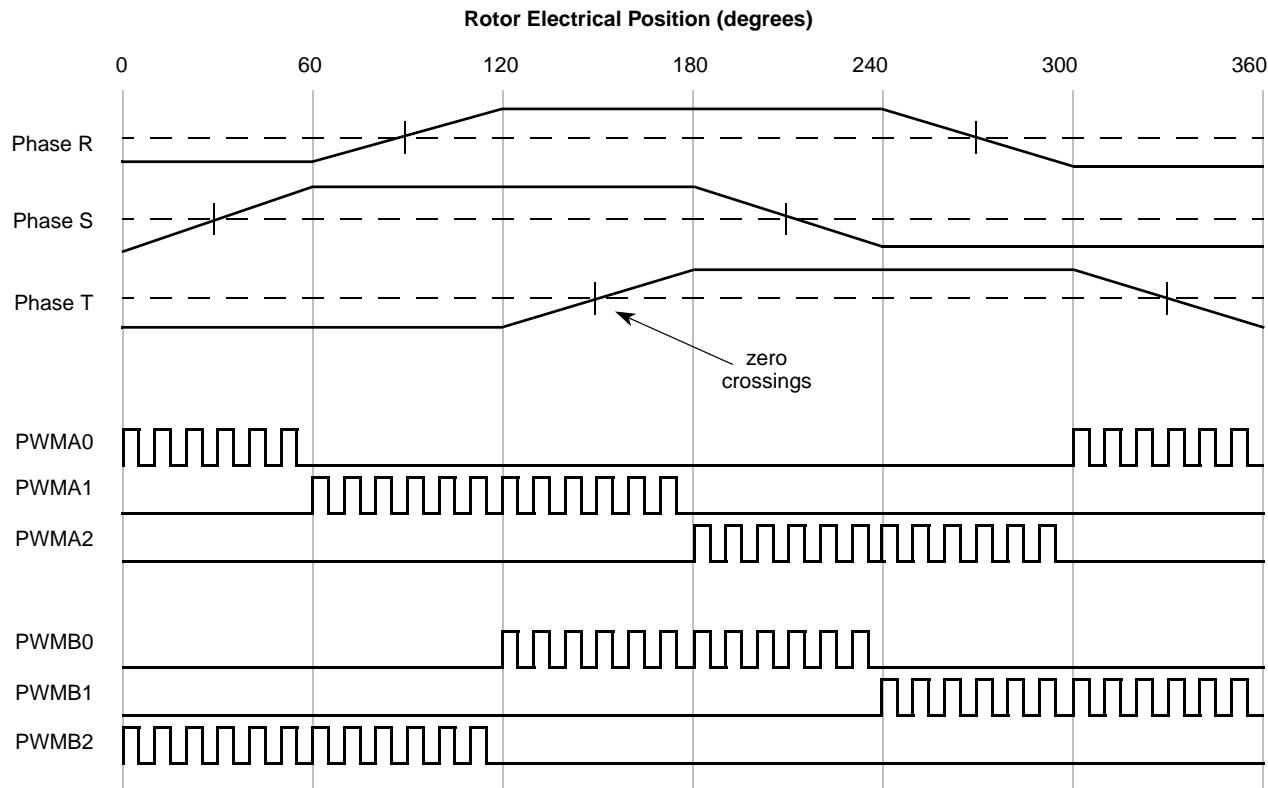
Before the PWM signals are routed to the output pins, they are processed by a hardware block that permits all submodule outputs to be switched synchronously. This feature can be extremely useful in commutated motor applications, where the next commutation state can be laid in ahead of time and immediately switched to the outputs when the appropriate condition or time is reached. The changes occur synchronously on all submodule outputs and immediately after the trigger event occurs eliminating any interrupt latency.

The synchronous output switching is accomplished via a signal called FORCE\_OUT. This signal originates from either:

- The local FORCE bit within the submodule
- Submodule 0
- External to the PWM module. In most cases, supplied from an external timer channel configured for output compare.

In a typical application, software sets the desired states of the output pins in preparation for the next FORCE\_OUT event. This selection lays dormant until the FORCE\_OUT signal transitions and then all outputs are switched simultaneously. The signal switching is performed upstream from the deadtime generator so that any abrupt changes that might occur do not violate deadtime on the power stage when in complementary mode.

**Figure 34-35** shows a popular application that can benefit from this feature. In most cases with a brushless DC motor, it is desirable to spin the motor without hall-effect sensor feedback. Instead, the back EMF of the motor phases is monitored and is used to schedule the next commutation event. The top waveforms of **Figure 34-35** are a simplistic representation of these back EMF signals. Timer compare events (represented by the vertical dashes in the diagram) are scheduled based on the zero crossings of the back-EMF waveforms. The PWM module is configured ahead of time with the next state of the PWM pins in anticipation of the compare event. When it happens, the output compare of the timer drives the FORCE\_OUT signal which immediately changes the state of the PWM pins to the next commutation state with no software latency.



**Figure 34-35. Sensorless BLDC Commutation Using the Force Out Function**

### 34.4.8 Functional Details

This section describes the implementation of various sections of the PWM in greater detail.

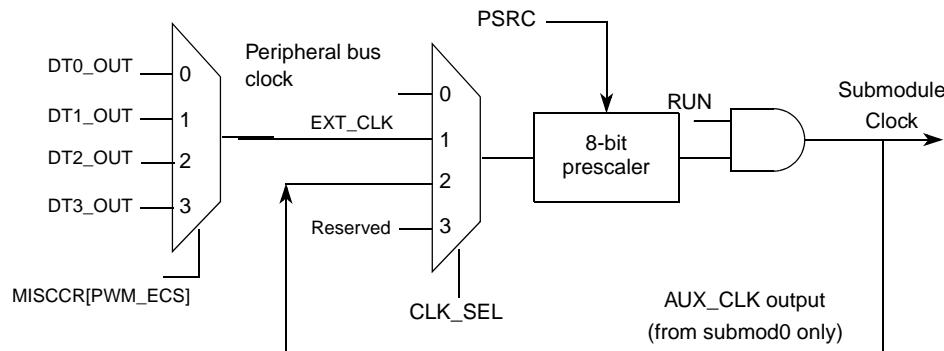
The following figure is a high-level block diagram of output PWM generation. It contains links to each subsection for more details.

**Figure 34-36. High-Level PWM Submodule Block Diagram**

#### 34.4.8.1 PWM Clocking

Figure 34-37 shows the logic used to generate the main counter clock. Each submodule can select between three clock signals:

- Peripheral bus clock
- EXT\_CLK — generated by an on-chip resource and goes to all of the submodules. For this device, EXT\_CLK is any of the DMA timer outputs as chosen by a register in the CCM module (MISCCR[PWM\_ECS]). See Chapter 10, “Chip Configuration Module (CCM)”, for more details.
- AUX\_CLK — broadcast from submodule 0 and can be selected as the clock source by other submodules. In this mode submodule 0’s 8-bit prescaler and RUN bit controls all of the submodules. The local RUN bits are ignored.



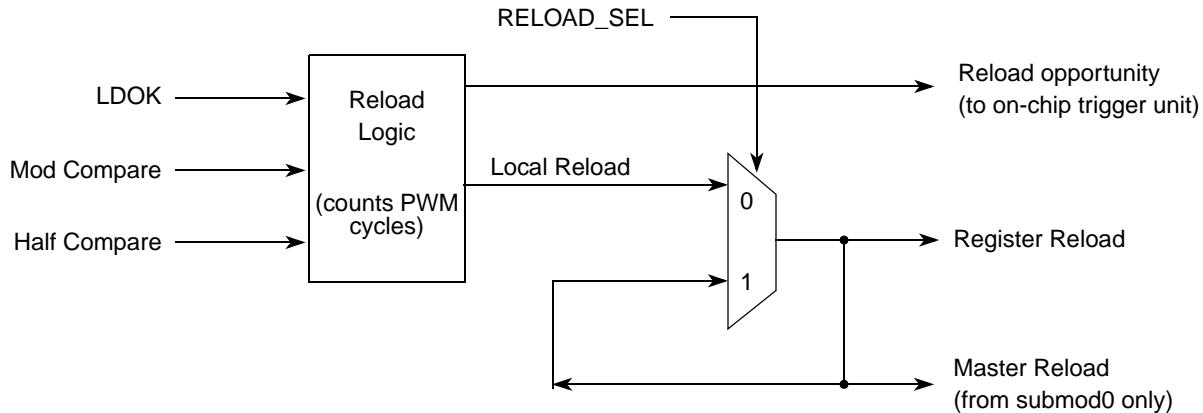
**Figure 34-37. Submodule Clocking Diagram**

To allow lower PWM frequencies, the prescaler produces the PWM clock frequency by dividing the peripheral bus clock frequency by 1–128. The prescaler bits, PWM\_SMnCR1[PRSC], select the prescaler divisor. This prescaler is buffered and is not used by the PWM generator until LDOOK is set and a new PWM reload cycle begins or PWM\_SMnCR1[LDMOD] is set.

#### 34.4.8.2 Register Reload Logic

The register reload logic determines when the outer set of registers for all double buffered register pairs are transferred to the inner set of registers. The register reload event can be scheduled to occur every  $n$  PWM cycles using the LDFQ and FULL bits. A half-cycle reload option is also supported (HALF) where the reload can take place in the middle of a PWM cycle. The half-cycle point is defined by PWM\_SMnVAL0 and does not have to be exactly in the middle of the PWM cycle.

As illustrated in [Figure 34-38](#) the reload signal from submodule 0 can be broadcast as the master reload signal to control the reload of registers in other submodules.



**Figure 34-38. Register Reload Logic**

### 34.4.8.3 Counter Synchronization

Referring to [Figure 34-39](#), the 16-bit counter is initialized to PWM\_SM $n$ INIT by one of four sources:

- Local sync
- Master reload
- Master sync
- PWM\_SYNC
- FORCE\_OUT (See [Section 34.4.8.6, “Force Out Logic”](#) for details on this signal.)

The counter counts up until its output equals PWM\_SM $n$ VAL1 which specifies the counter modulus value. The resulting compare causes a rising edge to occur on the local sync signal.

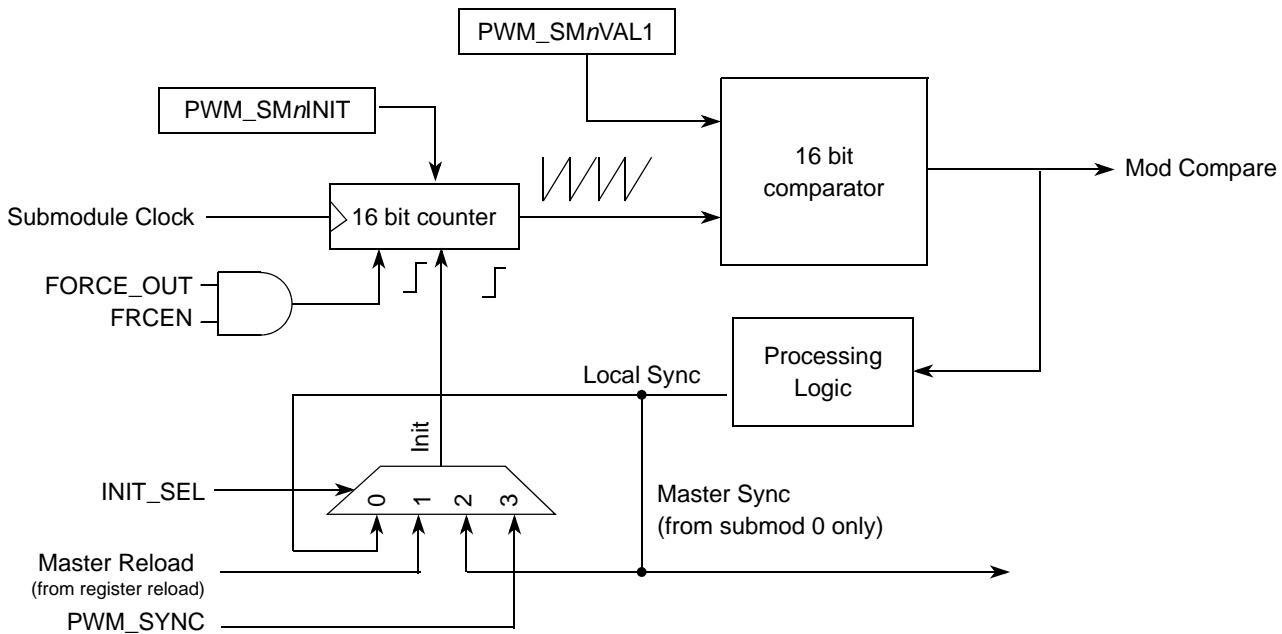


Figure 34-39. Submodule Timer Synchronization

### 34.4.8.3.1 Using the Local Sync for Counter Initialization

If local sync is selected as the initialization trigger, then PWM\_SMnVAL1 within the submodule effectively controls the timer period (and thus the PWM frequency generated by that submodule) and everything operates on a local level.

### 34.4.8.3.2 Using the Master Sync for Counter Initialization

The master sync signal originates as the local sync from submodule 0. If configured to do so, the timer period of any submodule can be locked to the period of the submodule 0 timer. The PWM\_SMnVAL1 register and associated comparator of the other submodules can then be freed up for other functions, such as:

- PWM generation
- Input captures
- Output compares
- Output triggers

### 34.4.8.3.3 Using the Master Reload for Counter Initialization

If the master reload signal is selected as the source for counter initialization, then the period of the counter is locked to the register reload frequency of submodule 0. Since the reload frequency is usually equal to the sampling frequency of the software control algorithm, the submodule counter period is equal the sampling period. As a result, this timer can be used to generate output compares or output triggers over the entire sampling period which may consist of several PWM cycles. The master reload signal only originates from submodule 0.

#### 34.4.8.3.4 Using PWM\_SYNC for Counter Initialization

If the PWM\_SYNC signal is selected as the source for counter initialization, an external source can control the period of all submodules.

#### 34.4.8.3.5 Using FORCE\_OUT for Counter Initialization

The counter can optionally initialize upon FORCE\_OUT assertion if FRCEN is set. As indicated by [Figure 34-39](#), this constitutes a second initialization input into the counter which bypasses the other counter initialization signals. The FORCE\_OUT signal is provided mainly for commutated applications.

When PWM signals are commutated on an inverter controlling a brushless DC motor, the PWM cycle must restart at the beginning of the commutation interval. This action effectively resynchronizes the PWM waveform to the commutation timing. Otherwise, the average voltage applied to a motor winding integrated over the entire commutation interval is a function of the timing between the asynchronous commutation event with respect to the PWM cycle. The effect is more critical at higher motor speeds where each commutation interval may consist of only a few PWM cycles. If the counter is not initialized at the start of each commutation interval, the result is an oscillation caused by the beating between the PWM frequency and the commutation frequency.

#### 34.4.8.4 Generation Hardware

[Figure 34-40](#) illustrates PWM generation in each submodule. Two comparators and associated PWM\_SM<sub>n</sub>VAL<sub>m</sub> registers are used for each PWM output signal:

- First comparator and PWM\_SM<sub>n</sub>VAL<sub>m</sub> register control the turn-on edge
- Second comparator and PWM\_SM<sub>n</sub>VAL<sub>m</sub> register control the turn-off edge

## Motor Control Pulse-Width Modulator (mcPWM)

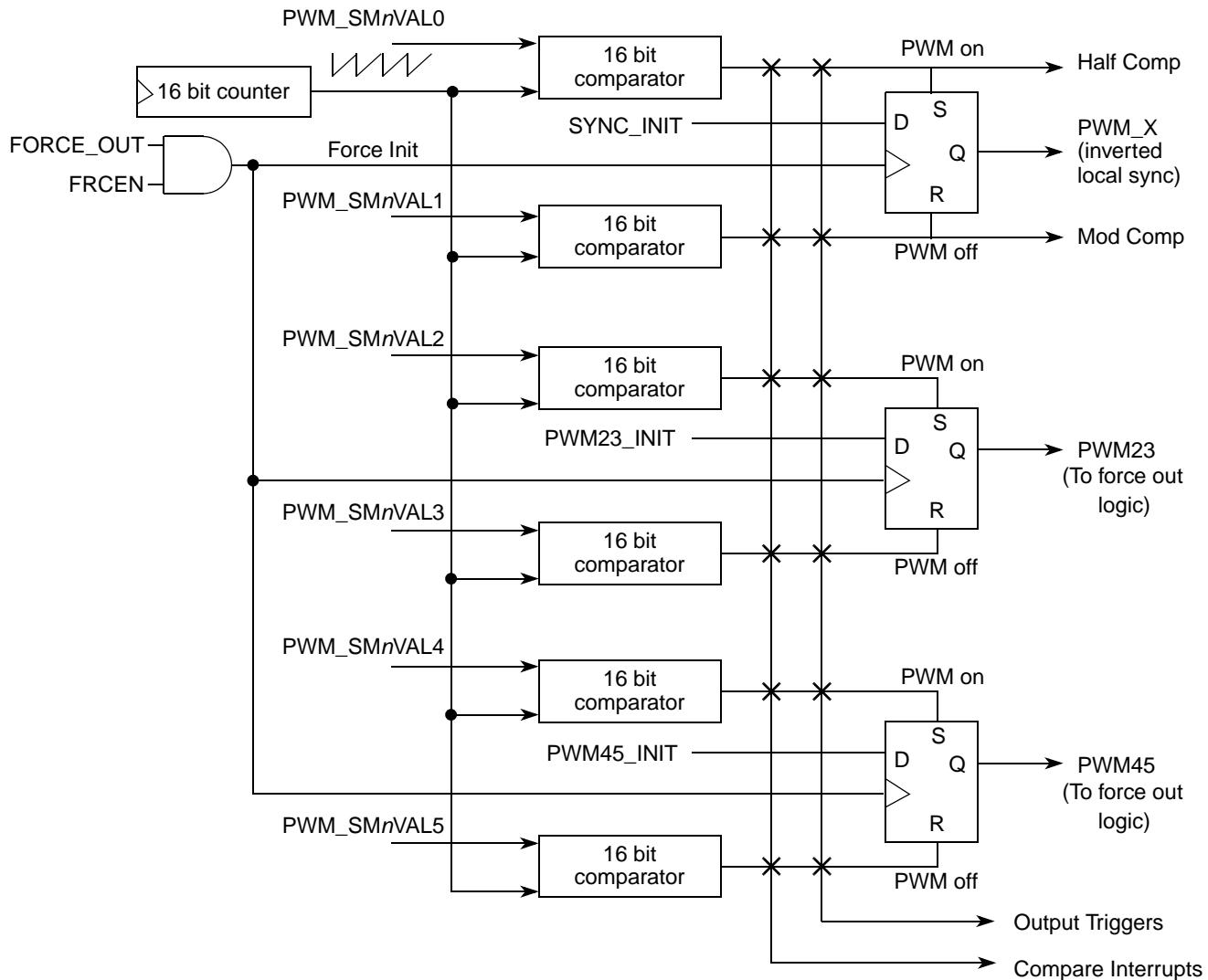


Figure 34-40. PWM Generation Hardware

The generation of the local sync signal is exactly the same as the other PWM signals in the submodule. While comparator 0 causes a rising edge of the local sync signal, comparator 1 generates a falling edge. Comparator 1 is also hardwired to the reload logic to generate the half cycle reload indicator.

If PWM\_SMnVAL1 is controlling the modulus of the counter and PWM\_SMnVAL0 is half of the PWM\_SMnVAL1 register minus the PWM\_SMnINIT value, then the half cycle reload pulse occurs exactly halfway through the timer count period and the local sync has a 50% duty cycle. Otherwise, if PWM\_SMnVAL1 and PWM\_SMnVAL0 are not required for register reloading or counter initialization, they can be used to modulate the duty cycle of the local sync signal. Effectively, this turns it into an auxiliary PWM signal (PWM\_X) assuming that the PWM\_X pin is not used for another function (such as input capture or deadtime distortion correction). Including the local sync signal, each submodule is capable of generating three PWM signals where software has complete control over each edge of each of the signals.

If the comparators and edge value registers are not required for PWM generation, they can also be used for other functions such as:

- Output compares
- Generating output triggers
- Generating interrupts at timed intervals

The 16-bit comparators shown in [Figure 34-40](#) are equal-to-or-greater-than (not just equal-to) comparators. In addition, if the set and reset of the flip-flop are asserted, then the flop output goes to zero.

#### **34.4.8.5 Output Compare Capabilities**

By using the `PWM_SMnVALm` registers in conjunction with the submodule timer and 16-bit comparators, you can perform buffered output compares with no additional hardware. Specifically, the following output compare functions are possible:

- An output compare sets the output high
- An output compare sets the output low
- An output compare generates an interrupt
- An output compare generates an output trigger

As shown in [Figure 34-40](#), an output compare is initiated by programming a `PWM_SMnVALm` register for a timer compare, which causes the output of the D flip-flop to set or reset.

For example, if an output compare is desired that asserts the `PWM_A` signal:

1. Program `PWM_SMnVAL2` with the counter value where the output compare should occur.
2. Program `PWM_SMnVAL3` to a value outside of the modulus range of the counter to prevent the D flip-flop from resetting after the compare has occurred.

Conversely, if an output compare is desired that negates the `PWM_A` signal:

1. Program `PWM_SMnVAL3` with the appropriate count value.
2. Program `PWM_SMnVAL2` with a value outside the counter modulus range.

Regardless of whether a high or low compare is programmed, a compare event can generate an interrupt or output trigger.

#### **34.4.8.6 Force Out Logic**

For each submodule software can select between seven signal sources for the `FORCE_OUT` signal:

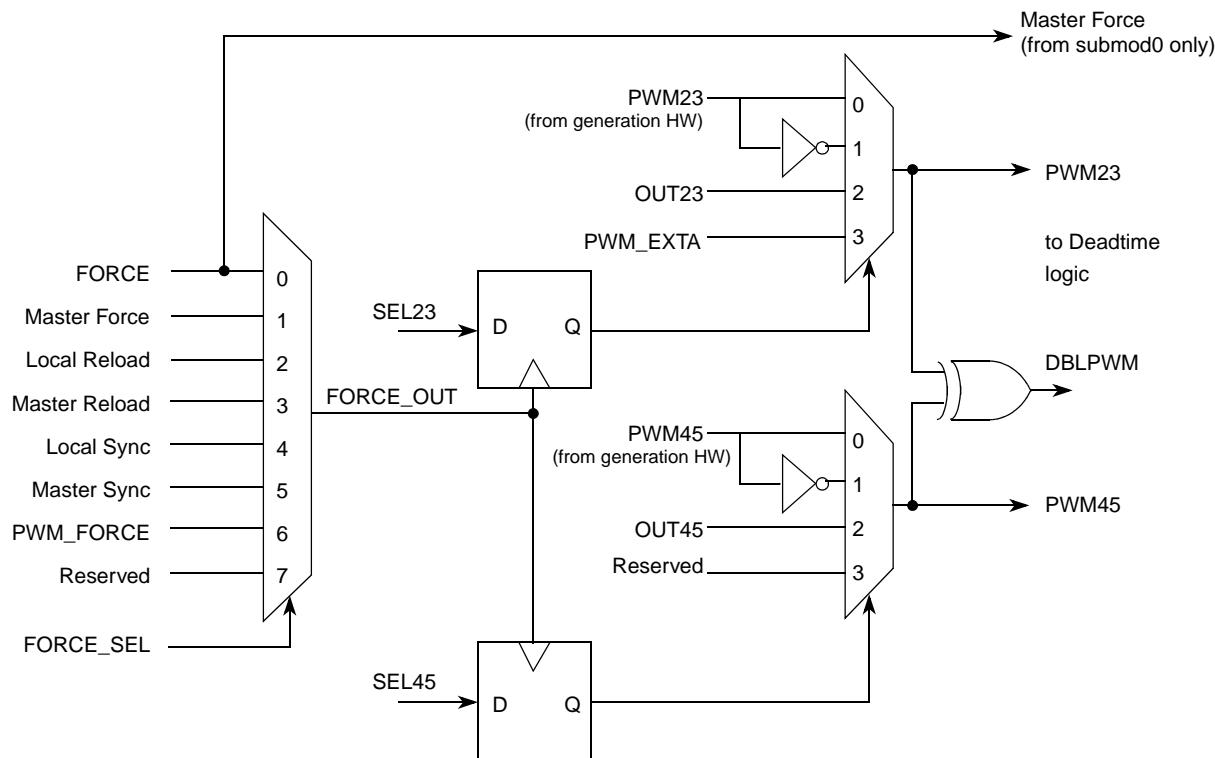
- Local FORCE bit
- Master force signal from submodule 0
- Local reload signal
- Master reload signal from submodule 0
- Local sync signal
- Master sync signal from submodule 0
- `PWM_FORCE` signal

Use the local signals when synchronization between modules is not desired. However, if all signals on all submodule outputs must change at the same time, use the master or PWM\_FORCE signals.

[Figure 34-41](#) illustrates the force out logic. The SEL23 and SEL45 fields choose from one of four signals that can be supplied to the submodule outputs:

- PWM signal
- Inverted PWM signal
- Binary level specified by software via the OUT23 and OUT45 bits
- PWM\_EXTA alternate external control signals

The selection can be determined ahead of time and, when a FORCE\_OUT event occurs, these values are presented to the signal selection mux, which immediately switches the requested signal to the output of the mux for further processing.



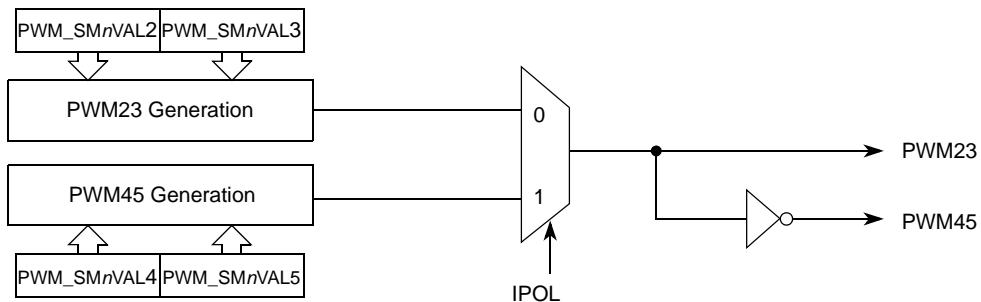
**Figure 34-41. Force Out Logic**

The local FORCE bit of submodule 0 can be broadcast as the master force signal to other submodules. This allows submodule 0's local FORCE bit to synchronously update all submodule outputs at the same time. The PWM\_FORCE signal originates from outside the PWM module from a source such as a timer or digital comparators in the analog-to-digital converter.

#### 34.4.8.7 Independent or Complementary Channel Operation

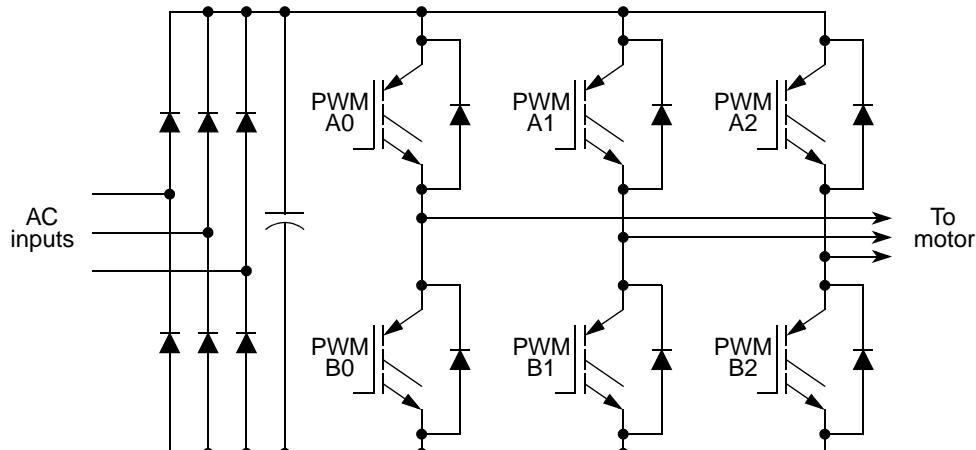
Setting PWM\_SMnCR2[INDEP] configures the pair of PWM outputs as two independent PWM channels. Each PWM output is independently controlled by its own PWM\_SMnVAL<sub>m</sub> pair.

Clearing PWM\_SM<sub>n</sub>CR2[INDEP] configures the PWM output as a pair of complementary channels. The PWM pins are paired as shown in [Figure 34-42](#). The IPOL bit determines which signal is connected to the output pin (PWM23 or PWM45).



**Figure 34-42. Complementary Channel Pair**

Use complementary channel operation for driving top and bottom transistors in a motor drive circuit, such as in [Figure 34-43](#). Complementary operation allows the use of the deadtime insertion feature.



**Figure 34-43. Typical 3-Phase AC Motor Drive**

#### 34.4.8.8 Deadtime Insertion Logic

When in complementary mode, each submodule's deadtime insertion logic creates non-overlapping complementary signals. While in this mode, a PWM pair can drive top/bottom transistors, as shown in [Figure 34-44](#). When the top PWM channel is active, the bottom PWM channel is inactive, and vice versa.

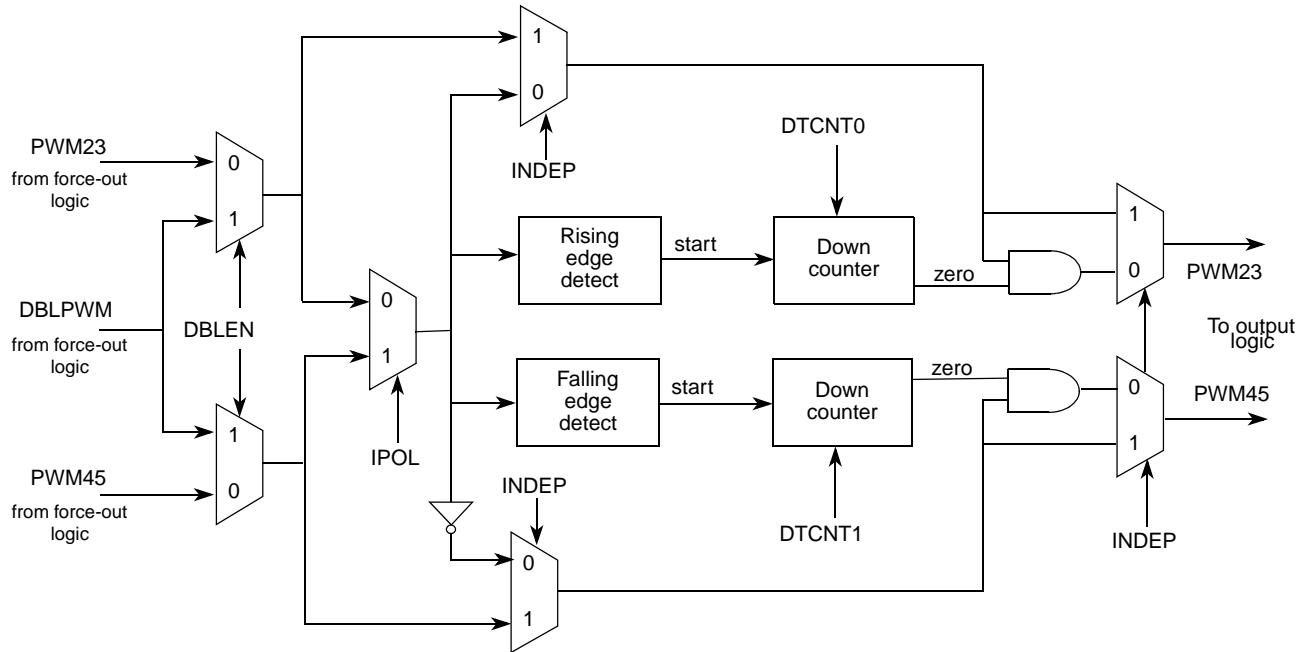


Figure 34-44. Deadtime Insertion and Fine Control Logic

**NOTE**

To avoid short-circuiting the DC bus and endangering the transistor, there must be no overlap of conducting intervals between top and bottom transistors. But the transistor's characteristics may make its switching-off time longer than its switching-on time. To avoid the conducting overlap, insert deadtime in the switching period as illustrated in [Figure 34-45](#).

The deadtime generators automatically insert software-selectable activation delays into the pair of PWM outputs. The deadtime registers (DTCNT0 and DTCNT1) specify the number of peripheral bus clock cycles to use for deadtime delay. Each time the deadtime generator inputs change state, deadtime is inserted, which forces both PWM outputs to the inactive state.

When deadtime is inserted in complementary PWM signals that are connected to an inverter driving an inductive load, the PWM waveform on the inverter output has a different duty cycle than what appears on the output pins of the PWM module. This results in a distortion in the voltage applied to the load. To correct this, add or subtract from the PWM value used, as discussed in [Section 34.4.8.8.1, “Top/Bottom Correction”](#).

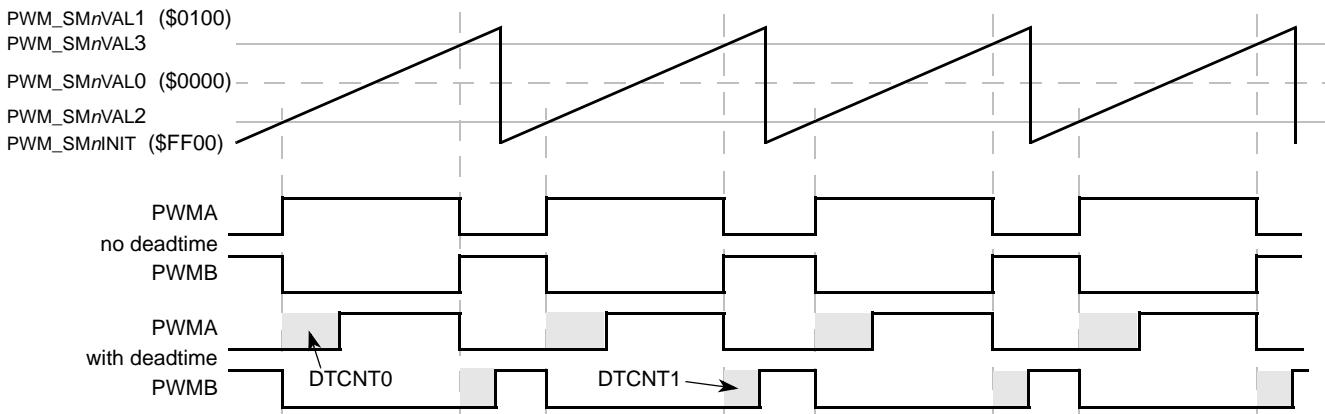


Figure 34-45. Deadtime Insertion

#### 34.4.8.8.1 Top/Bottom Correction

In complementary mode, the top or the bottom transistor controls the output voltage. However, deadtime must be inserted to avoid overlap of the conducting interval. In complementary mode both transistors are off during deadtime, allowing the output voltage to be determined by the current status of the load and to introduce distortion in the output voltage. On AC induction motors running open-loop, the distortion typically manifests itself as poor low-speed performance, such as torque ripple and rough operation.

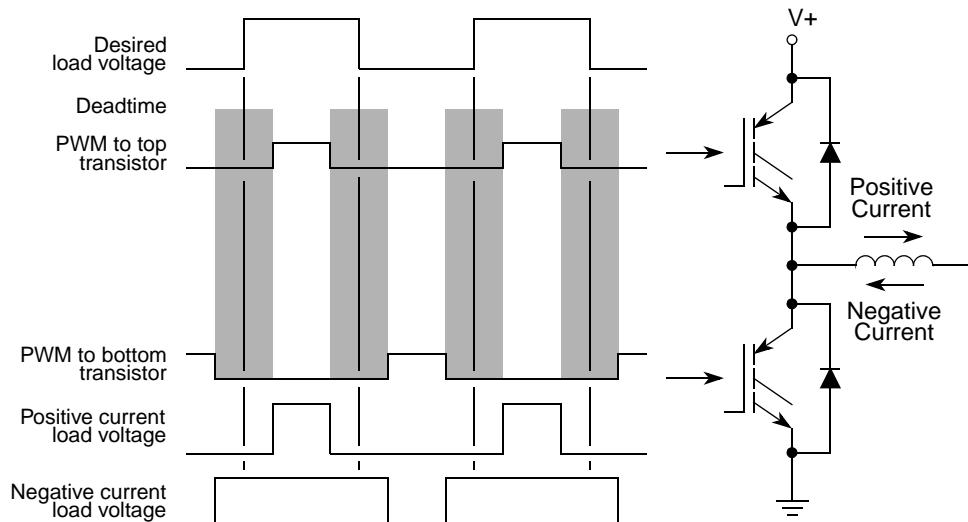


Figure 34-46. Deadtime Distortion

During deadtime, load inductance distorts output voltage by keeping current flowing through the diodes. This deadtime current flow creates a load voltage that varies with current direction.

- Positive current flow — the load voltage during deadtime is equal to the bottom supply, putting the top transistor in control.
- Negative current flow — the load voltage during deadtime is equal to the top supply, putting the bottom transistor in control.

Since the original PWM pulse widths are shortened by deadtime insertion, the averaged sinusoidal output is less than the desired value. However, when deadtime is inserted, it creates a distortion in the motor's current waveform. This distortion is aggravated by dissimilar turn-on and turn-off delays of each of the transistors. This distortion can be corrected by giving the PWM module information on which transistor is controlling at a given time.

For a typical circuit in complementary channel operation, only one of the transistors is effective in controlling the output voltage at any given time. This depends on the direction of the motor current for that pair. To correct distortion one of two factors must be added to the desired PWM value, depending on whether the top or bottom transistor is controlling the output voltage. Therefore, user software is responsible for calculating both compensated PWM values prior to placing them in the `PWM_SMnVALm` registers. Either the `PWM_SMnVAL2/3` or the `PWM_SMnVAL4/5` register pair controls the pulse width at any given time, depending on either:

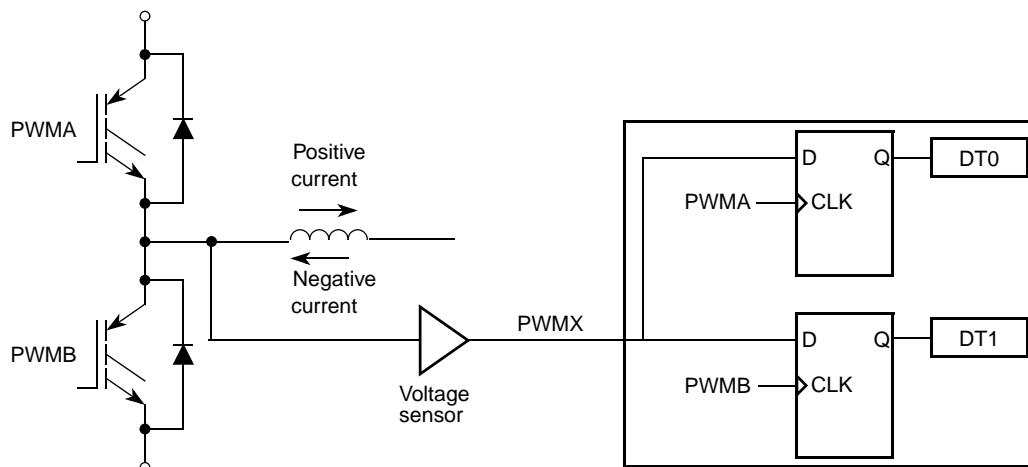
- The state of the current status pin, `PWMX`, for that driver
- The state of the odd/even correction bit, `IPOL`, for that driver

To correct deadtime distortion, decrease or increase the value in the appropriate `PWM_SMnVALm` register.

- Edge-aligned — decrease or increase `PWM_SMnVALm` by the deadtime
- Center-aligned — decrease or increase `PWM_SMnVALm` by half the deadtime

#### 34.4.8.8.2 Manual Correction

To detect the current status, the voltage on each `PWMX` pin is sampled twice in a PWM period, at the end of each deadtime. The value is stored in `PWM_SMnCR1[DT]`. The DT bits are a timing marker indicating when to toggle between PWM value registers. You can then set the `IPOL` bit to switch between `PWM_SMnVAL2/3` and `PWM_SMnVAL4/5` register pairs according to DT values.



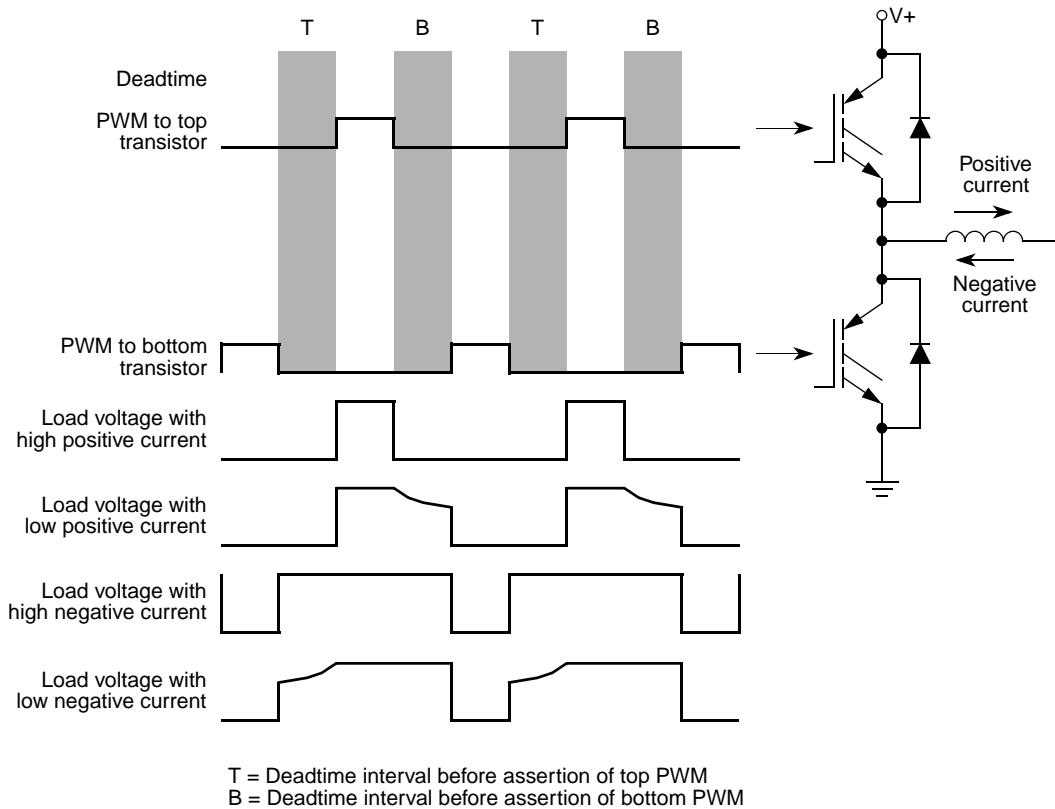
**Figure 34-47. Current-status Sense Scheme for Deadtime Correction**

In Figure 34-47, during deadtime periods:

- If current is large and flowing out of the complementary circuit, both D flip-flops latch low ( $DT0 = DT1 = 0$ ).

- If current is large and flowing into the complementary circuit, both D flip-flops latch high ( $DT0 = DT1 = 1$ ).

However, under low-current, the output voltage of the complementary circuit during deadtime is somewhere between the high and low levels. The current cannot free-wheel through the opposition anti-body diode, regardless of polarity, giving additional distortion when the current crosses zero. Sampled results are  $DT0 = 0$  and  $DT1 = 1$ . Thus, the best time to change one PWM value register to another is just before the current zero crossing.



**Figure 34-48. Output Voltage Waveforms**

#### 34.4.8.9 Output Logic

Figure 34-49 shows the output logic of each submodule, including how each PWM output has individual fault disabling, polarity control, and output enable. This allows for maximum flexibility when interfacing to the external circuitry.

The PWM23 and PWM45 signals, which are output from the deadtime logic in Figure 34-44, are positive true signals. A high level on these signals should result in the corresponding transistor in the PWM inverter being turned on. The voltage level required at the PWM output pin to turn the transistor on or off is a function of the logic between the pin and the transistor. Therefore, it is imperative to program the POLA and POLB bits before enabling the output pins. A fault condition can result in the PWM output being tri-stated, forced to a logic 1, or forced to a logic 0 depending on the values programmed into PWM\_SMnOCR[PWMxFS].

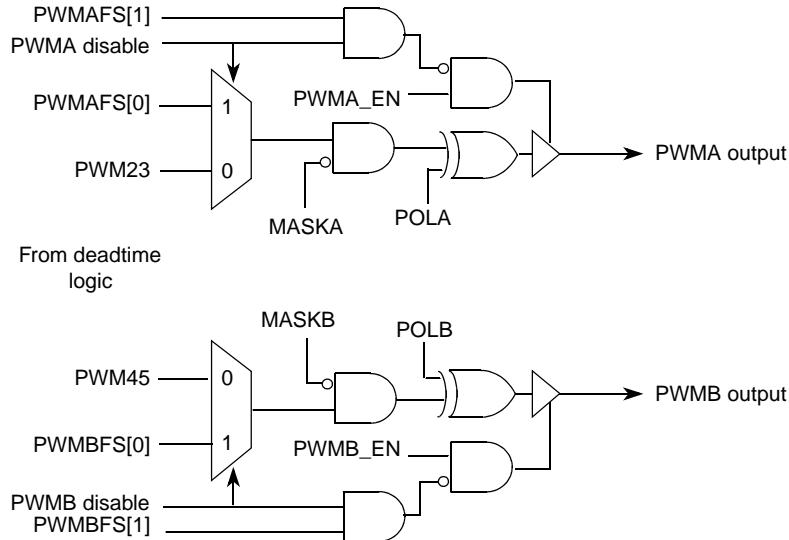


Figure 34-49. Output Logic Section

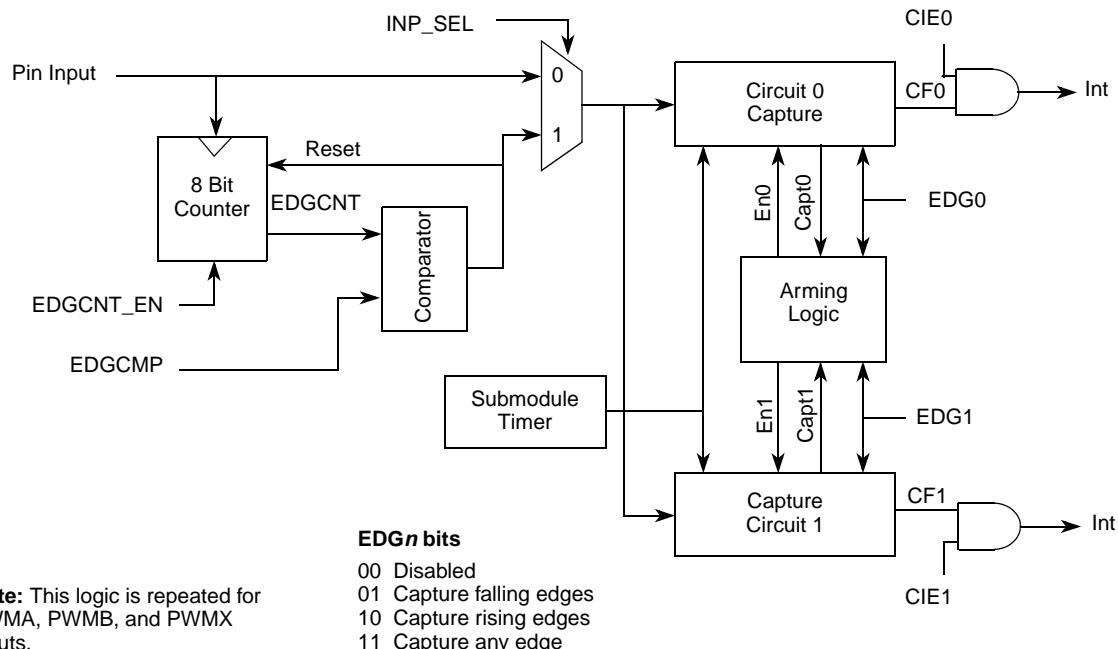
### 34.4.8.10 Enhanced Capture (E-Capture)

The enhanced capture (E-capture) logic measures both edges of an input signal. When a submodule pin is configured for input capture, the respective `PWM_SMnCVAlm` registers record the edge values.

[Figure 34-50](#) illustrates the block diagram of the E-capture circuit. Upon entering the pin input, the signal is split into two paths.

- Mux input — software can select to pass the signal directly to the capture logic for processing
- 8-bit counter — counts the rising and falling edges of the signal. The output of this 8-bit counter is compared to EDGCMP and when equal, the comparator resets the counter. A pulse is also supplied to the mux input where software can select it to be processed by the capture logic.

This feature allows the E-capture circuit to count up to 256 edge events before initiating a capture event. This feature is useful for dividing high frequency signals for capture processing, so that capture interrupts do not overwhelm the CPU. Also, this feature can simply generate an interrupt after  $n$  events have been counted.

**Figure 34-50. Enhanced Capture (E-Capture) Logic**

Based on the mode selection, the mux selects the pin input or the comparator output to be processed by the capture logic. The selected signal is routed to two separate capture circuits, which work in tandem to capture sequential edges of the signal. The type of edge to be captured by each circuit is determined by the EDG1 and EDG0 bits.

Also, the arming logic controls the operation of the capture circuits:

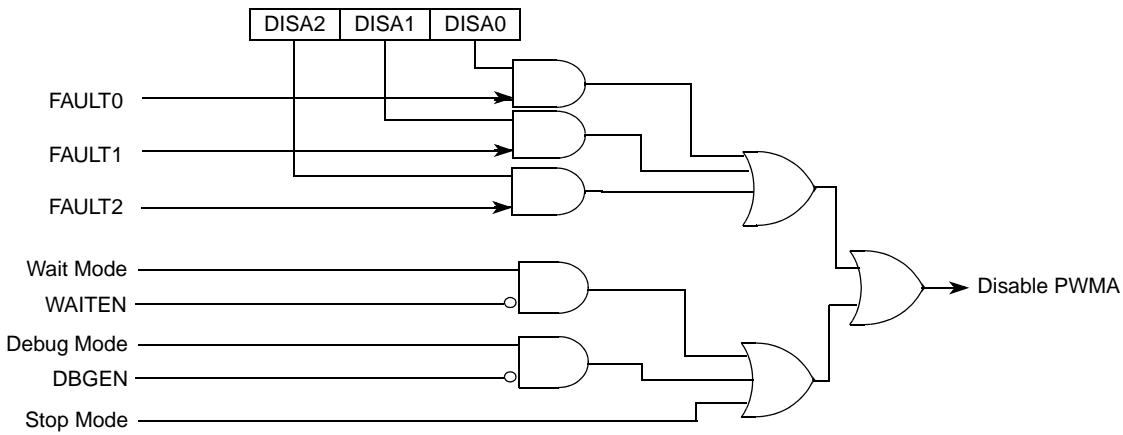
- Free running (continuous) — capture sequences are performed indefinitely. If both capture circuits are enabled, a capture event from one circuit arms the other, and vice versa.
- One shot — only one capture sequence is performed. If both capture circuits are enabled, capture circuit 0 is armed and when a capture event occurs, capture circuit 1 is armed. Once the second capture occurs, further captures are disabled until another capture sequence is initiated. Both capture circuits are also capable of generating an interrupt.

#### 34.4.8.11 Fault Protection

Fault protection can control any combination of PWM output pins. Faults are generated by assertion of any of the PWM\_FAULT $n$  pins. This polarity can be changed by PWM\_FCR[FLVL]. Each PWM\_FAULT $n$  pin can be mapped arbitrarily to any of the PWM outputs. When the fault protection hardware disables the PWM outputs, the PWM generator continues to run. Only the output pins are forced to value specified in PWM\_SM $n$ OCR[PWM $x$ FS].

The fault decoder disables PWM pins selected by the fault logic and the disable mapping register (PWM\_SM $n$ DISMAP). See [Figure 34-51](#) for an example of the fault disable logic. Each bit field in PWM\_SM $n$ DISMAP controls the mapping for a single PWM pin as shown in [Table 34-16](#).

Fault protection is enabled even when the PWM module is not enabled. Therefore, if a fault is latched it must be cleared to prevent an interrupt when the PWM is re-enabled.



**Figure 34-51. Fault Decoder for PWMA**

**Table 34-30. Fault Mapping**

PWM Pin	Controlling Register Bits
PWMA	DISA[3:0]
PWMB	DISB[3:0]
PWMX	DISX[3:0]

#### 34.4.8.11.1 Fault Pin Filter

Each fault pin has a programmable filter that can be bypassed. The sampling period of the filter can be adjusted with PWM\_FFILT[PER]. The number of consecutive samples that must agree before an input transition is recognized is configured using PWM\_FFILT[CNT]. Clearing PER disables the input filter for a given PWM\_FAULT $n$  pin.

Upon detecting a logic 0 on the filtered PWM\_FAULT $n$  pin (or a logic 1 if PWM\_FCR[FLVL] is set), the corresponding PWM\_FSR[FFPIN, FFLAG] bits are set. The FFPIN bit remains set as long as the filtered PWM\_FAULT $n$  pin is zero. Clear FFLAG by writing one to it.

If the PWM\_FAULT $n$  pin interrupt enable bit (PWM\_FCR[FIE]) is set, FFLAG generates a CPU interrupt request. The interrupt request latch remains set until:

- Software clears FFLAG by writing one to it
- Software clears FIE by writing zero to it
- A reset occurs

Even with the filter enabled, there is a combinational path from the PWM\_FAULT $n$  inputs to the PWM pins. This logic is also capable of holding a fault condition if loss-of-clock to the PWM module occurs.

### 34.4.8.11.2 Automatic Fault Clearing

Setting PWM\_FCR[FAUTO] configures faults from the PWM\_FAULT $n$  pin for automatic clearing. When this bit is set, disabled PWM pins are enabled when the PWM\_FAULT $n$  pin returns to logic one and:

- If PWM\_FSR[FFULL] is cleared, a new PWM full or half cycle begins
- If PWM\_FSR[FFULL] is set, a new PWM full cycle begins

#### NOTE

When FAUTO is set, clearing the FFLAG flag does not affect disabled PWM pins.

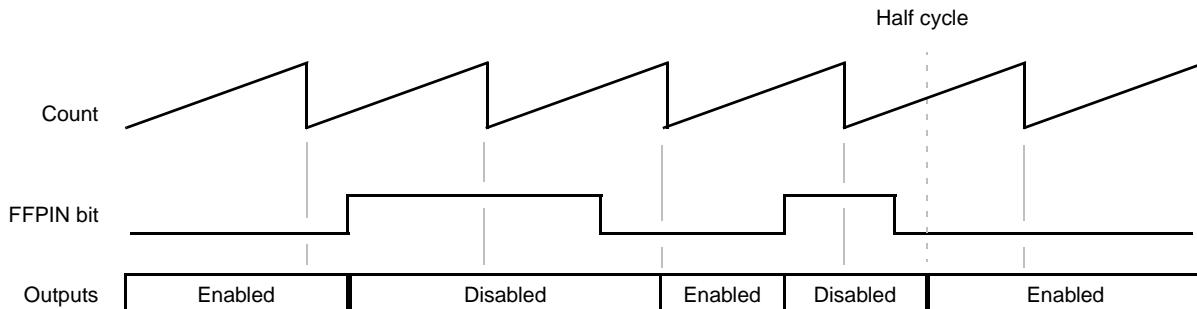


Figure 34-52. Automatic Fault Clearing

### 34.4.8.11.3 Manual Fault Clearing

Clearing PWM\_FCR[FAUTO] configures faults from the PWM\_FAULT $n$  pin for manual clearing:

- If the fault safety mode bits (PWM\_FCR[FSAFE]) are cleared, the PWM pins disabled by the PWM\_FAULT $n$  pins are enabled:
  - When software clears the corresponding FFLAG flag
  - If FFULL is cleared, when the next PWM full or half cycle begins regardless of the logic level on the PWM\_FAULT $n$  pin
  - If FFULL is set, when the next PWM full cycle begins regardless of the logic level on the PWM\_FAULT $n$  pin

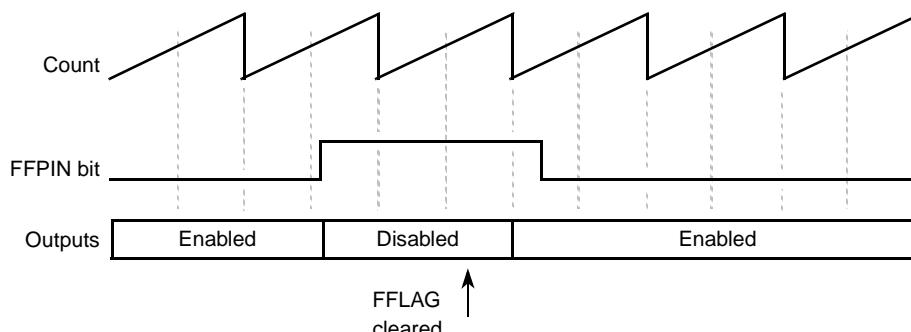
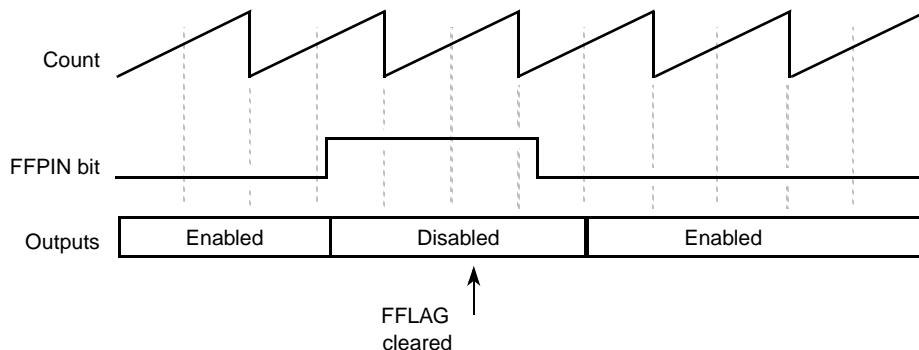


Figure 34-53. Manual Fault Clearing (FSAFE = 0)

- If the fault safety mode bits (PWM\_FCR[FSAFE]) are set, the PWM pins disabled by the PWM\_FAULT $n$  pins are enabled:
  - When software clears the corresponding FFLAG flag
  - If FFULL is cleared, when the filter detects a logic one on the PWM\_FAULT $n$  pins at the start of the next PWM full or half cycle boundary
  - If FFULL is set, when the filter detects a logic one on the PWM\_FAULT $n$  pins at the start of the next PWM full cycle boundary



**Figure 34-54. Manual Fault Clearing (FSAFE = 1)**

#### NOTE

Fault protection also applies during software output control when the SEL23 and SEL45 fields are set to select OUT23 and OUT45 bits or PWM\_EXTAn. Fault clearing still occurs at half-cycle boundaries while the PWM generator is engaged (RUN = 1). But, the OUTxx bits can control the PWM pins while the PWM generator is off (RUN = 0). Thus, fault clearing occurs at peripheral bus cycles while the PWM generator is off and at the start of PWM cycles when the generator is engaged.

#### 34.4.8.11.4 Fault Testing

Use the FTEST bit to simulate a fault condition on each of the fault inputs.

### 34.4.9 PWM Generator Loading

#### 34.4.9.1 Load Enable

The PWM\_MCR[LDOCK] bit enables loading of the following PWM generator parameters:

- Prescaler divisor — PWM\_SMnCR1[PRSC]
- PWM period and pulse width — PWM\_SMnINIT and PWM\_SMnVAL $m$

LDOCK allows software to finish calculating these PWM parameters so they can be synchronously updated.

- If PWM\_SMnCR1[LDMOD] is cleared, setting LDOCK transfers these values to the PWM generator at the beginning of the next PWM reload cycle
- If PWM\_SMnCR1[LDMOD] is set, setting LDOCK transfers these values immediately

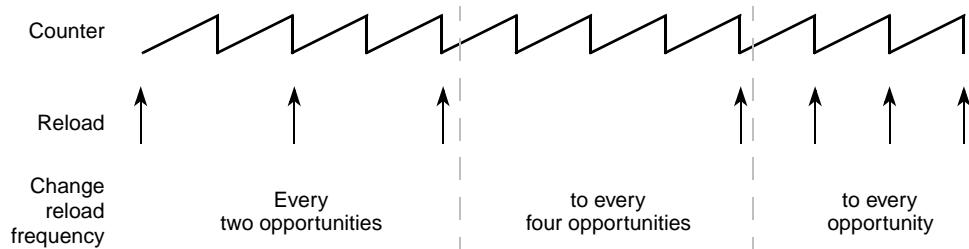
Set LDOK by reading it when it is cleared, and then writing a one to it. After loading the values, LDOK is automatically cleared.

### 34.4.9.2 Load Frequency

The PWM\_SMnCR1[LDFQ] bits select an integral loading frequency of one to 16 PWM reload opportunities. The LDFQ bits take effect every PWM reload opportunity, regardless the state of LDOK.

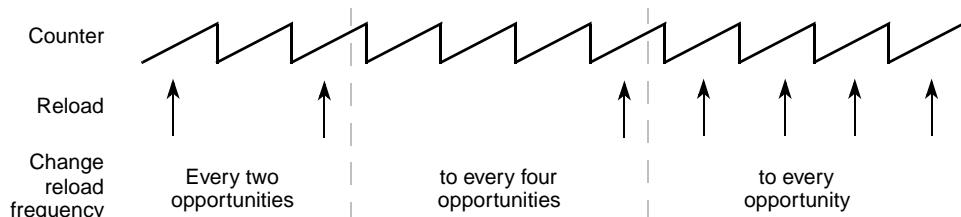
The PWM\_SMnCR1[HALF, FUL] bits control reload timing:

- If FULL is set, a reload opportunity occurs at the end of every PWM cycle when the count equals PWM\_SMnVAL1.



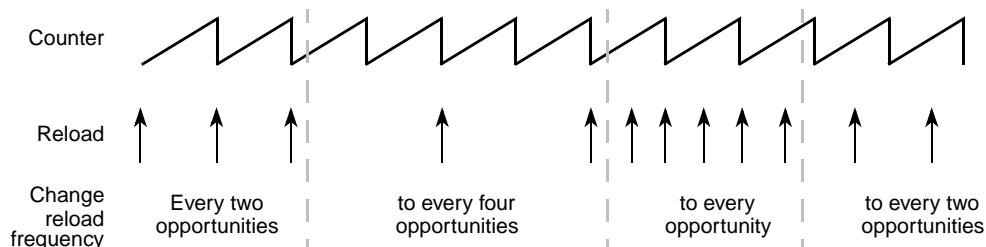
**Figure 34-55. Full Cycle Reload Frequency Change**

- If HALF is set, a reload opportunity occurs at the half cycle when the count equals PWM\_SMnVAL0.



**Figure 34-56. Half Cycle Reload Frequency Change**

- If both are set, a reload opportunity occurs twice per PWM cycle when the count equals PWM\_SMnVAL1 and PWM\_SMnVAL0.



**Figure 34-57. Full and Half Cycle Reload Frequency Change**

### 34.4.9.3 Reload Flag

At every reload opportunity the reload flag (PWM\_SMnSR[RF]) is set. Setting RF happens even if an actual reload is prevented by LDOOK.

- If PWM\_SMnIER[RIE] is set, RF generates interrupt requests allowing software to calculate new PWM parameters in real-time.
- If PWM\_SMnIER[RIE] is cleared, reloads still occur at the selected reload rate without generating interrupt requests.

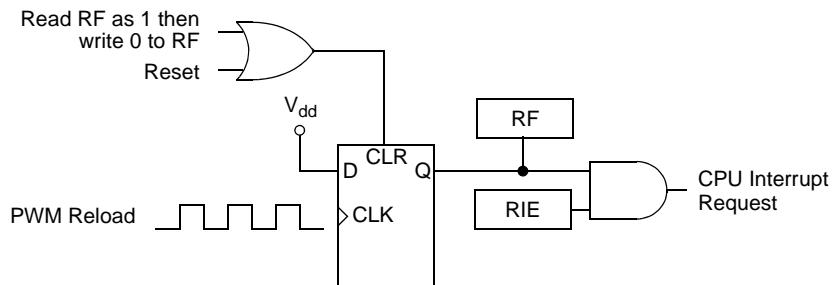


Figure 34-58. PWMF Reload Interrupt Request

### 34.4.9.4 Reload Errors

When one of the PWM\_SMnVAL $m$ , or PSRC registers is updated, the PWM\_SMnSR[RUF] flag is set to indicate that the data is not coherent. RUF is cleared by a successful reload, which consists of the reload signal while LDOOK is set.

- If RUF is set and LDOOK is cleared when the reload signal asserts, a reload error has taken place and REF is set
- If RUF is cleared when a reload signal asserts, then the data is coherent and no error is flagged

## 34.5 Initialization/Application Information

Initialize all registers and set the PWM\_MCR[LDOOK] bit before setting the PWM\_MCR[RUN] bit.

### NOTE

Even if LDOOK is not set, setting RUN also sets the PWM\_SMnSR[RF] flag.

To prevent an interrupt request, clear PWM\_SMnIER[RIE] before setting RUN.

While LDOOK is cleared, if RUN is cleared and then set, the PWM generator uses the last values loaded.

When the RUN bit is cleared:

- RF and pending CPU interrupt requests are not cleared
- All fault circuitry remains active
- Software/external output control remains active
- Deadtime insertion continues during software/external output control

### 34.5.1 Interrupt Requests

Each of the submodules within the mcPWM module can generate an interrupt from several sources. The fault logic can also generate interrupts. The interrupt service routine (ISR) must check the related interrupt enables and interrupt flags to determine the actual cause of the interrupt. See [Chapter 17, “Interrupt Controller Modules”](#), for more details on the interrupt controller.

**Table 34-31. Interrupt Summary**

INTC2 Source Number	Interrupt Flag	Interrupt Enable	Description
1	PWM_SM0SR [CFxn]	PWM_SM0IER [CxnlE]	Submodule 0 input capture interrupt
2	PWM_SM1SR [CFxn]	PWM_SM1IER [CxnlE]	Submodule 1 input capture interrupt
3	PWM_SM2SR [CFxn]	PWM_SM2IER [CxnlE]	Submodule 2 input capture interrupt
4	PWM_SM3SR [CFxn]	PWM_SM3IER [CxnlE]	Submodule 3 input capture interrupt
5	PWM_SM0SR [RF]	PWM_SM0IER [RIE]	Submodule 0 reload interrupt
6	PWM_SM1SR [RF]	PWM_SM1IER [RIE]	Submodule 1 reload interrupt
7	PWM_SM2SR [RF]	PWM_SM2IER [RIE]	Submodule 2 reload interrupt
8	PWM_SM3SR [RF]	PWM_SM3IER [RIE]	Submodule 3 reload interrupt
9	PWM_FSR [FFLAG]	PWM_FCR [FIE]	Fault input interrupt
10	PWM_SM0SR [REF]	PWM_SM0IER [REIE]	Submodule 0 reload error interrupt
	PWM_SM1SR [REF]	PWM_SM1IER [REIE]	Submodule 1 reload error interrupt
	PWM_SM2SR [REF]	PWM_SM2IER [REIE]	Submodule 2 reload error interrupt
	PWM_SM3SR [REF]	PWM_SM3IER [REIE]	Submodule 3 reload error interrupt
19	PWM_SM0SR [CMPF]	PWM_SM0IER [CMPIE]	Submodule 0 compare interrupt
20	PWM_SM1SR [CMPF]	PWM_SM1IER [CMPIE]	Submodule 1 compare interrupt
21	PWM_SM2SR [CMPF]	PWM_SM2IER [CMPIE]	Submodule 2 compare interrupt
22	PWM_SM3SR [CMPF]	PWM_SM3IER [CMPIE]	Submodule 3 compare interrupt

### 34.5.2 DMA Requests

Each submodule can request a DMA read access for its capture FIFOs and a DMA write request for its double-buffered PWM\_SM<sub>n</sub>VAL<sub>m</sub> registers. Due to the limited number of DMA channels available on this device, the DMA requests are OR'd into two read and write requests. See [Chapter 19, “Enhanced Direct Memory Access \(eDMA\)”,](#) for details on the DMA controller.

**Table 34-32. DMA Summary**

DMA Request	DMA Enable	Name	Description
56	PWM_SM <sub>n</sub> DMAEN [Cx <sub>n</sub> DE]	Capture FIFO read request	Read PWM_SM <sub>n</sub> SR to determine which PWM_SM <sub>n</sub> CVAL <sub>m</sub> contains a value to read.
57	PWM_SM <sub>n</sub> DMAEN [VALDE]	PWM_SM <sub>n</sub> VAL <sub>m</sub> write request	Read PWM_SM <sub>n</sub> SR[RF] to determine which PWM_SM <sub>n</sub> VAL <sub>m</sub> registers updating

# Chapter 35

## Synchronous Serial Interface (SSI)

### 35.1 Introduction

This section presents the synchronous serial interface (SSI), and discusses the architecture, the programming model, the operating modes, and initialization of the SSI module. This device contains two identical SSI modules.

The SSI module, as shown in [Figure 35-1](#), consists of separate transmit and receive circuits with FIFO registers and separate serial clock and frame sync generation for the transmit and receive sections. The second set of Tx and Rx FIFOs replicates the logic used for the first set of FIFOs.

#### NOTE

This device contains SSI bits to control the clock rate and the SSI DMA request sources within the chip configuration module (CCM). See [Chapter 10, “Chip Configuration Module \(CCM\),”](#) for detailed information on these bit fields.

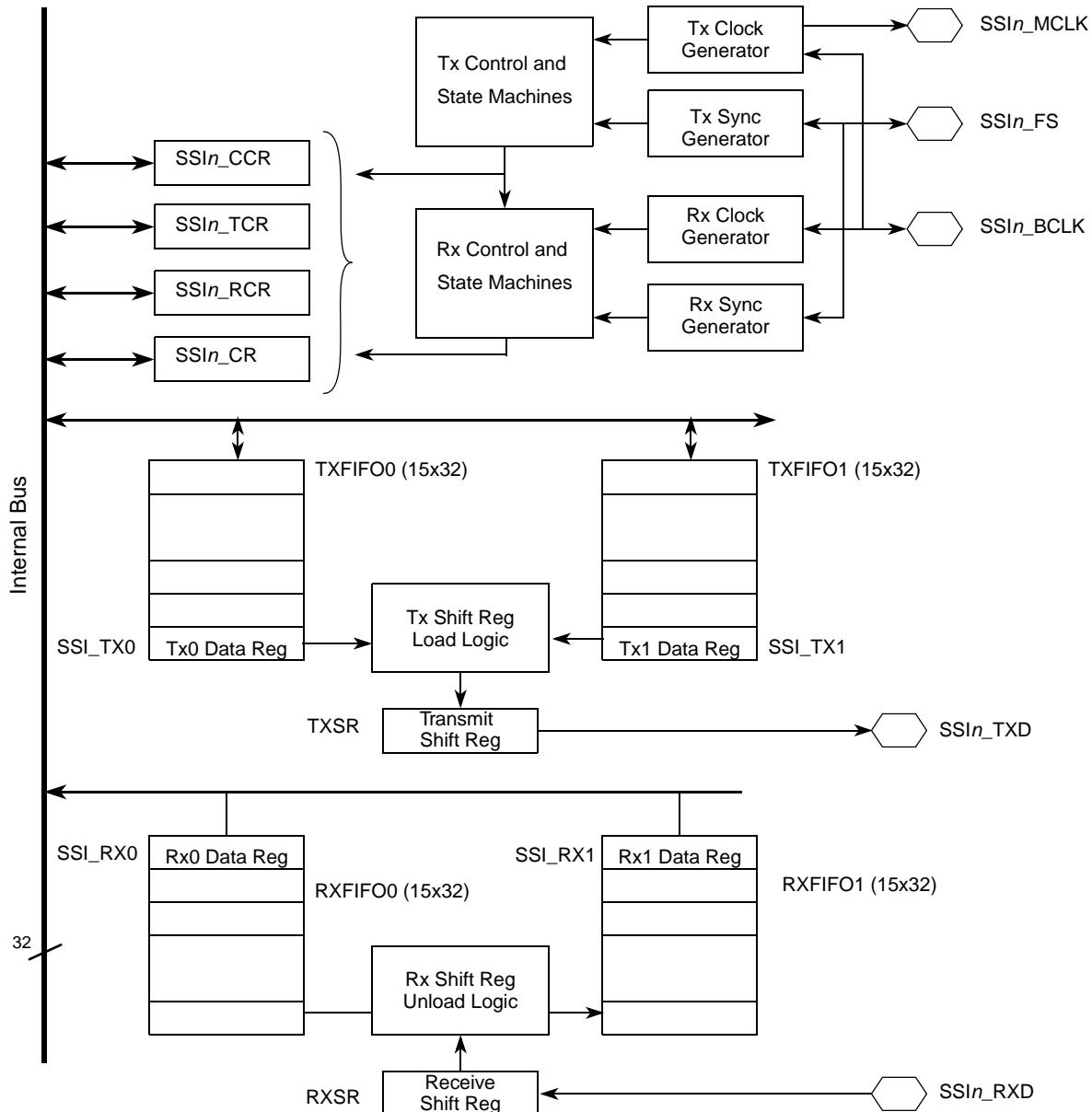


Figure 35-1. SSI Block Diagram

### 35.1.1 Overview

The SSI is a full-duplex serial port that allows the processor to communicate with a variety of serial devices. Such serial devices are:

- Standard codecs
- Digital signal processors (DSPs)
- Microprocessors
- Peripherals
- Audio codecs that implement the inter-IC sound bus (I<sup>2</sup>S) and the Intel® AC97 standards

The SSI module typically transfers samples in a periodic manner. The SSI consists of independent transmitter and receiver sections with shared clock generation and frame synchronization.

#### NOTE

The pin multiplexing and control module must be configured to enable the peripheral function of the appropriate pins (refer to [Chapter 15, “Pin-Multiplexing and Control”](#)) prior to configuring the SSI.

### 35.1.2 Features

The SSI includes the following features:

- Synchronous transmit and receive sections with shared internal/external clocks and frame syncs, operating in master or slave mode.
- Normal mode operation using frame sync
- Network mode operation allowing multiple devices to share the port with up to 32 time slots
- Gated clock mode operation requiring no frame sync
- Two sets of transmit and receive FIFOs. Each FIFO is 15x32 bits, which can be used in network mode to provide two independent channels for transmission and reception
- Programmable data interface modes such as I<sup>2</sup>S, lsb, msb aligned
- Programmable word length (8, 10, 12, 16, 18, 20, 22 or 24 bits)
- Program options for frame sync and clock generation
- Programmable I<sup>2</sup>S modes (master or slave). Oversampling clock available as output from SSIn\_MCLK in I<sup>2</sup>S master mode
- AC97 support
- Completely separate clock and frame sync selections. In the AC97 standard, the clock is taken from an external source and frame sync is generated internally.
- Programmable oversampling clock (SSIn\_MCLK) of the sampling frequency available as output in master mode
- Programmable internal clock divider
- Transmit and receive time slot mask registers for reduced CPU overhead
- SSI power-down feature

### 35.1.3 Modes of Operation

SSI has the following basic synchronous operating modes.

- Normal mode
- Network mode
- Gated clock mode

These modes can be programmed via the SSI control registers. [Table 35-1](#) lists these operating modes and some of the typical applications in which they can be used:

**Table 35-1. SSI Operating Modes**

TX, RX Sections	Serial Clock	Mode	Typical Application
Synchronous	Continuous	Normal	Multiple synchronous codecs
Synchronous	Continuous	Network	TDM codec or DSP network
Synchronous	Gated	Normal	SPI-type devices; DSP to MCU

The transmit and receive sections of the SSI are only available in synchronous mode. In this mode, the transmitter and the receiver use a common clock and frame synchronization signal. The  $\text{SSI}_{in\_RCR}[RXBIT0, RSHFD]$  bits can continue affecting shifting-in of received data in synchronous mode. Continuous or gated clock mode can be selected. In continuous mode, the clock runs continuously. In gated clock mode, the clock is only functioning during transmission.

Normal or network mode can also be selected. In normal mode, the SSI functions with one data word of I/O per frame. In network mode, any number from two to thirty-two data words of I/O per frame can be used. Network mode is typically used in star or ring time-division-multiplex networks with other processors or codecs, allowing interface to time division multiplexed networks without additional logic. Use of the gated clock is not allowed in network mode. These distinctions result in the basic operating modes that allow the SSI to communicate with a wide variety of devices.

Typically, normal and network modes are used in a periodic manner, where data transfers at regular intervals, such as at the sampling rate of an external codec. Both modes use the concept of a frame. The beginning of the frame is marked with a frame sync when programmed with continuous clock. The  $\text{SSI}_{in\_CCR}[DC]$  bits determine length of the frame, depending on whether data is being transmitted or received.

The number of words transferred per frame depends on the mode of the SSI. In normal mode, one data word transfers per frame. In network mode, the frame divides into two to 32 time slots. In each time slot, one data word is optionally transferred.

Apart from the above basic modes of operation, SSI supports the following modes that require some specific programming:

- I<sup>2</sup>S mode
- AC97 mode
  - AC97 fixed mode
  - AC97 variable mode

In non-I<sup>2</sup>S slave modes (external frame sync), the SSI's programmed word length setting should be equal to the word length setting of the master. In I<sup>2</sup>S slave mode, the SSI's programmed word length setting can be lesser than or equal to the word length setting of the I<sup>2</sup>S master (external codec).

In slave modes, the SSI's programmed frame length setting (DC bits) can be lesser than or equal to the frame length setting of the master (external codec).

See [Section 35.4.1, “Detailed Operating Mode Descriptions,”](#) for more details on the above modes.

## 35.2 External Signal Description

The five SSI signals are explained below.

**Table 35-2. Signal Properties**

Name	Function	Direction	Reset State	Pull up
SSI_CLKIN	SSI Clock Input	I	I	Passive
SSIn_BCLK	Serial Bit Clock	I/O	0	Passive
SSIn_MCLK	Serial Master Clock	O	0	Passive
SSIn_FS	Serial Frame Sync	I/O	0	Passive
SSIn_RXD	Serial Receive Data	I	—	—
SSIn_TXD	Serial Transmit Data	O	0	Passive

### 35.2.1 SSI\_CLKIN — SSI Clock Input

The SSI module can be clocked by the internal core frequency derived from the PLL or this input clock. The source is selected by the MISCCR[SSISRC] bit in the CCM. See [Chapter 10, “Chip Configuration Module \(CCM\),”](#) and [Figure 35-40](#).

### 35.2.2 SSIn\_BCLK — Serial Bit Clock

This input or output signal is used by the transmitter and receiver and can be continuous or gated. During gated clock mode, data on the SSIn\_BCLK port is valid only during the transmission of data; otherwise, it is pulled to the programmed inactive state.

### 35.2.3 SSIn\_MCLK — Serial Master Clock

This clock signal is output from the device when it is the master. When in I<sup>2</sup>S master mode, this signal is referred to as the oversampling clock. The frequency of SSIn\_MCLK is a multiple of the frame clock.

### 35.2.4 SSIn\_FS — Serial Frame Sync

The input or output frame sync is used by the transmitter and receiver to synchronize the transfer of data. The frame sync signal can be one bit or one word in length and can occur one bit before the transfer of data or right at the transfer of data. In gated clock mode, the frame sync signal is not used. If SSIn\_FS is configured as an input, the external device should drive SSIn\_FS during rising edge of SSIn\_BCLK.

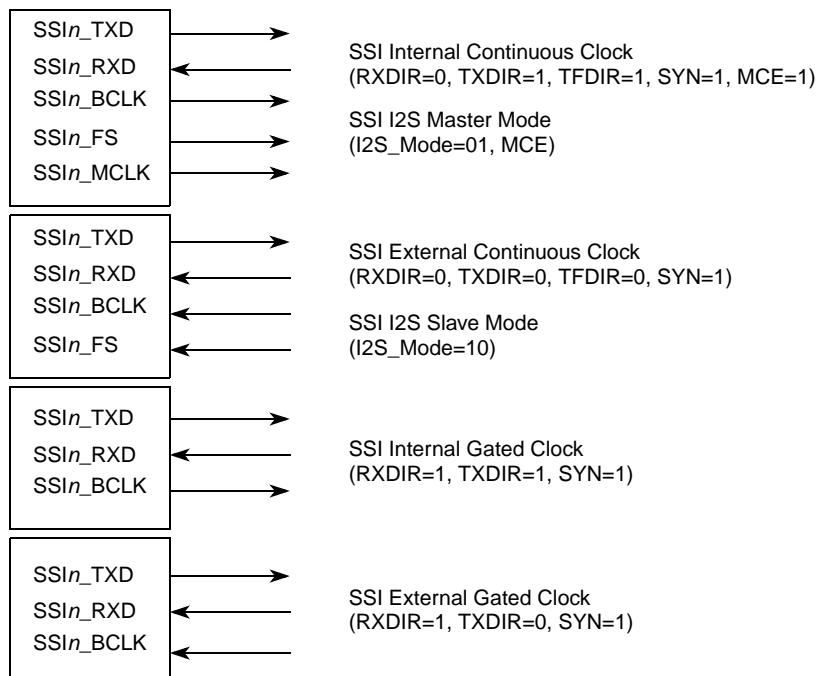
### 35.2.5 SSIn\_RXD — Serial Receive Data

The SSIn\_RXD port is an input and brings serial data into the receive data shift register.

### 35.2.6 SSIn\_TXD — Serial Transmit Data

The SSIn\_TXD port is an output and transmits data from the serial transmit shift register. The SSIn\_TXD port is an output port when data is transmitted and disabled between data word transmissions and on the trailing edge of the bit clock after the last bit of a word is transmitted.

[Figure 35-2](#) shows the main SSI configurations. These ports support all transmit and receive functions with continuous or gated clock as shown. Gated clock implementations do not require the use of the frame sync port (SSIn\_FS).



**Figure 35-2. Synchronous SSI Configurations—Continuous and Gated Clock**

[Figure 35-3](#) shows an example of the port signals for an 8-bit data transfer. Continuous and gated clock signals are shown, as well as the bit-length frame sync signal and the word-length frame sync signal. The shift direction can be defined as msb first or lsb first, and there are other options on the clock and frame sync.

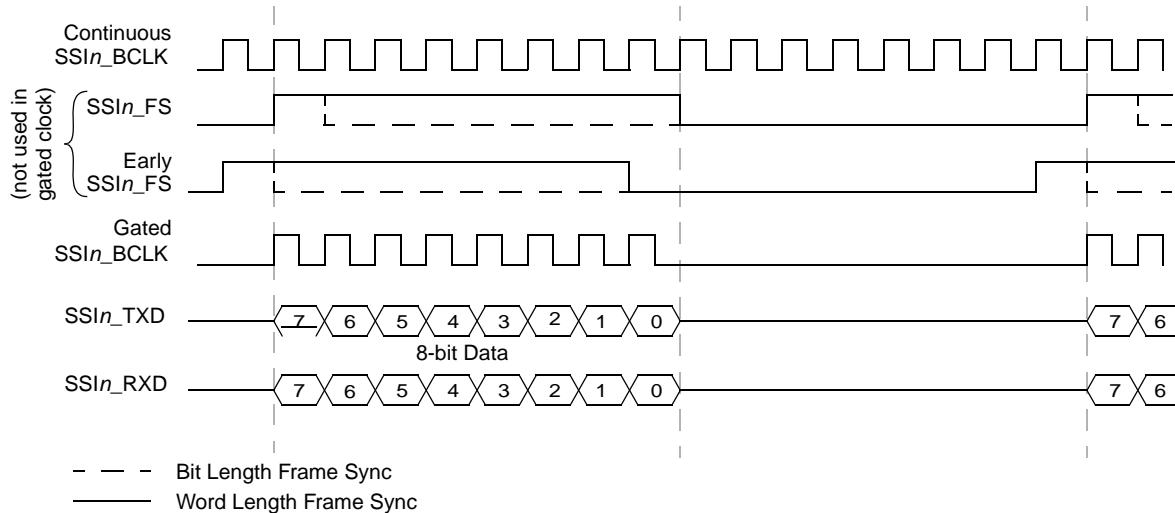


Figure 35-3. Serial Clock and Frame Sync Timing

Table 35-3. Clock and Frame Sync Pin Configuration

SSIn_CR [SYN]	SSIn_RCR [RXDIR]	SSIn_TCR		SSIn_BCLK	SSIn_FS
		TXDIR	TFDIR		
Synchronous Mode					
1	0	0	0	Bit clock in	FS in
1	0	0	1	Bit clock in	FS out
1	0	1	0	Bit clock out	FS in
1	0	1	1	Bit clock out	FS out
1	1	0	x	Gated clock in	—
1	1	1	x	Gated clock out	—

### 35.3 Memory Map/Register Definition

This section consists of register descriptions in address order. Each description includes a standard register diagram with an associated figure number. Details of register bit and field function follow the register diagrams, in bit order.

Table 35-4. SSI Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
SSIO SSI1					
0xFC0B_C000 0xFC0C_8000	SSIn Transmit Data Register 0 (SSIn_TX0)	32	R/W	0x0000_0000	<a href="#">35.3.1/35-8</a>
0xFC0B_C004 0xFC0C_8004	SSIn Transmit Data Register 1 (SSIn_TX1)	32	R/W	0x0000_0000	<a href="#">35.3.1/35-8</a>
0xFC0B_C008 0xFC0C_8008	SSIn Receive Data Register 0 (SSIn_RX0)	32	R	0x0000_0000	<a href="#">35.3.4/35-11</a>

Table 35-4. SSI Memory Map (continued)

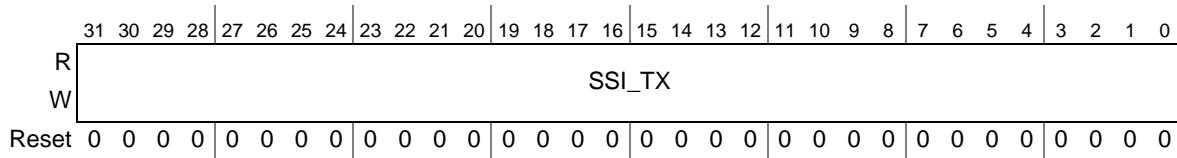
Address	Register	Width (bits)	Access	Reset Value	Section/Page
SSI0 SSI1					
0xFC0B_C00C 0xFC0C_800C	SSIn Receive Data Register 1 (SSIn_RX1)	32	R	0x0000_0000	<a href="#">35.3.4/35-11</a>
0xFC0B_C010 0xFC0C_8010	SSIn Control Register (SSIn_CR)	32	R/W	0x0000_0000	<a href="#">35.3.7/35-13</a>
0xFC0B_C014 0xFC0C_8014	SSIn Interrupt Status Register (SSIn_ISR)	32	R/W	0x0000_3003	<a href="#">35.3.8/35-15</a>
0xFC0B_C018 0xFC0C_8018	SSIn Interrupt Enable Register (SSIn_IER)	32	R/W	0x0000_3003	<a href="#">35.3.9/35-21</a>
0xFC0B_C01C 0xFC0C_801C	SSIn Transmit Configuration Register (SSIn_TCR)	32	R/W	0x0000_0200	<a href="#">35.3.10/35-23</a>
0xFC0B_C020 0xFC0C_8020	SSIn Receive Configuration Register (SSIn_RCR)	32	R/W	0x0000_0200	<a href="#">35.3.11/35-24</a>
0xFC0B_C024 0xFC0C_8024	SSIn Clock Control Register (SSIn_CCR)	32	R/W	0x0004_0000	<a href="#">35.3.12/35-25</a>
0xFC0B_C02C 0xFC0C_802C	SSI FIFO Control/Status Register (SSIn_FCSR)	32	R/W	0x0081_0081	<a href="#">35.3.13/35-27</a>
0xFC0B_C038 0xFC0C_8038	SSIn AC97 Control Register (SSIn_ACR)	32	R/W	0x0000_0000	<a href="#">35.3.14/35-33</a>
0xFC0B_C03C 0xFC0C_803C	SSIn AC97 Command Address Register (SSIn_ACADD)	32	R/W	0x0000_0000	<a href="#">35.3.15/35-34</a>
0xFC0B_C040 0xFC0C_8040	SSIn AC97 Command Data Register (SSIn_ACDAT)	32	R/W	0x0000_0000	<a href="#">35.3.16/35-34</a>
0xFC0B_C044 0xFC0C_8044	SSIn AC97 Tag Register (SSIn_ATAG)	32	R/W	0x0000_0000	<a href="#">35.3.17/35-35</a>
0xFC0B_C048 0xFC0C_8048	SSIn Transmit Time Slot Mask Register (SSIn_TMASK)	32	R/W	0x0000_0000	<a href="#">35.3.18/35-35</a>
0xFC0B_C04C 0xFC0C_8048	SSIn Receive Time Slot Mask Register (SSIn_RMASK)	32	R/W	0x0000_0000	<a href="#">35.3.19/35-36</a>
0xFC0B_C050 0xFC0C_8050	SSIn AC97 Channel Status Register (SSIn_ACCSR)	32	R	0x0000_0000	<a href="#">35.3.20/35-36</a>
0xFC0B_C054 0xFC0C_8054	SSIn AC97 Channel Enable Register (SSIn_ACCEN)	32	R	0x0000_0000	<a href="#">35.3.21/35-37</a>
0xFC0B_C058 0xFC0C_8058	SSIn AC97 Channel Disable Register (SSIn_ACCDIS)	32	R	0x0000_0000	<a href="#">35.3.22/35-37</a>

### 35.3.1 SSI Transmit Data Registers 0 and 1 (SSIn\_TX0/1)

The SSIn\_TX0/1 registers store the data to be transmitted by the SSI. For details on data alignment see [Section 35.4.4, “Supported Data Alignment Formats.”](#)

Address: 0xFC0B\_C000 (SSI0\_TX0)  
0xFC0B\_C004 (SSI0\_TX1)  
0xFC0C\_8000 (SSI1\_TX0)  
0xFC0C\_8004 (SSI1\_TX1)

Access: User read/write



**Figure 35-4. SSI Transmit Data Registers (SSI<sub>n</sub>\_TX0, SSI<sub>n</sub>\_TX1)**

**Table 35-5. SSI<sub>n</sub> TX0/1 Field Descriptions**

Field	Description
31–0 SSI_TX	<p>SSI transmit data. The <math>SSI_n\_TX0/1</math> registers are implemented as the first word of their respective Tx FIFOs. Data written to these registers transfers to the transmit shift register (TXSR), when shifting of the previous data is complete. If both FIFOs are in use, data alternately transfers from <math>SSI\_TX0</math> and <math>SSI\_TX1</math> to TXSR. <math>SSI\_TX1</math> can only be used in two-channel mode.</p> <p>Multiple writes to the <math>SSI_n\_TX</math> registers do not result in the previous data being over-written by the subsequent data. Instead, they are ignored. Protection from over-writing is present irrespective of whether the transmitter is enabled or not.</p> <p>Example: If Tx FIFO0 is in use and you write Data1 – 16 to <math>SSI\_TX0</math>, Data16 does not overwrite Data1. Data1 – 15 are stored in the FIFO while Data16 is discarded.</p> <p>Example: If Tx FIFO0 is not in use and you write Data1, Data2 to <math>SSI\_TX0</math>, Data2 does not overwrite Data1 and is discarded.</p> <p><b>Note:</b> Enable SSI (<math>SSI_n\_CR[SSI\_EN] = 1</math>) before writing to the SSI transmit data registers</p>

### **35.3.2 SSI Transmit FIFO 0 and 1 Registers**

The SSI transmit FIFO registers are 15x32-bit registers. These registers are not directly accessible. The transmit shift register (TXSR) receives its values from these FIFO registers. When the transmit interrupt enable ( $\text{SSI}_{In\_IER}[TIE]$ ) bit and either of the transmit FIFO empty ( $\text{SSI}_{In\_ISR}[TFE0, TFE1]$ ) bits are set, an interrupt is generated when the data level in of the SSI transmit FIFOs falls below the selected threshold.

### 35.3.3 SSI Transmit Shift Register (TXSR)

TXSR is a 24-bit shift register that contains the data transmitted and is not directly accessible. When a continuous clock is used, the selected bit clock shifts data out to the SSIn\_TXD pin when the associated frame sync is asserted. When a gated clock is used, the selected gated clock shifts data out to the SSIn\_TXD port.

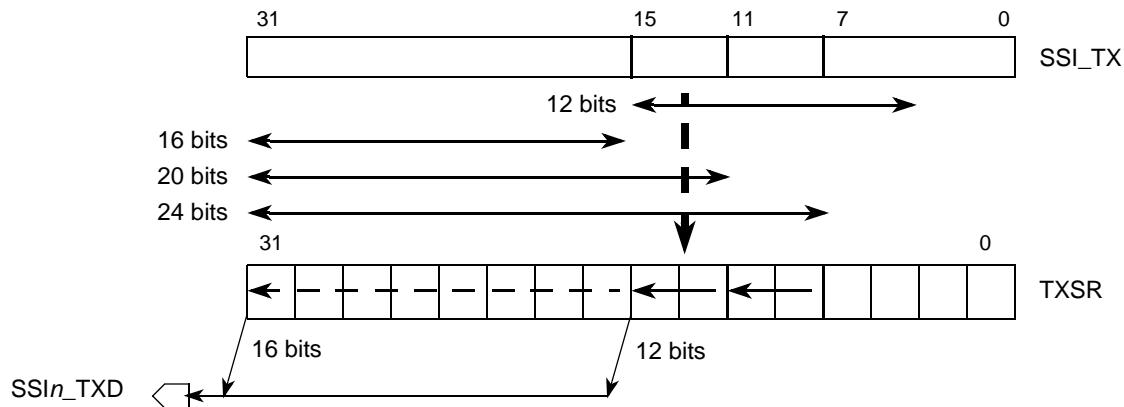
The word length control bits ( $\text{SSI}_n\text{-CCR}[WL]$ ) determine the number of bits to shift out of the TXSR before it is considered empty and can be written to again. The data to be transmitted occupies the most significant portion of the shift register if  $\text{SSI}_n\text{-TCR}[TXBIT0]$  is cleared. Otherwise, it occupies the least significant portion. The unused portion of the register is ignored.

## **NOTE**

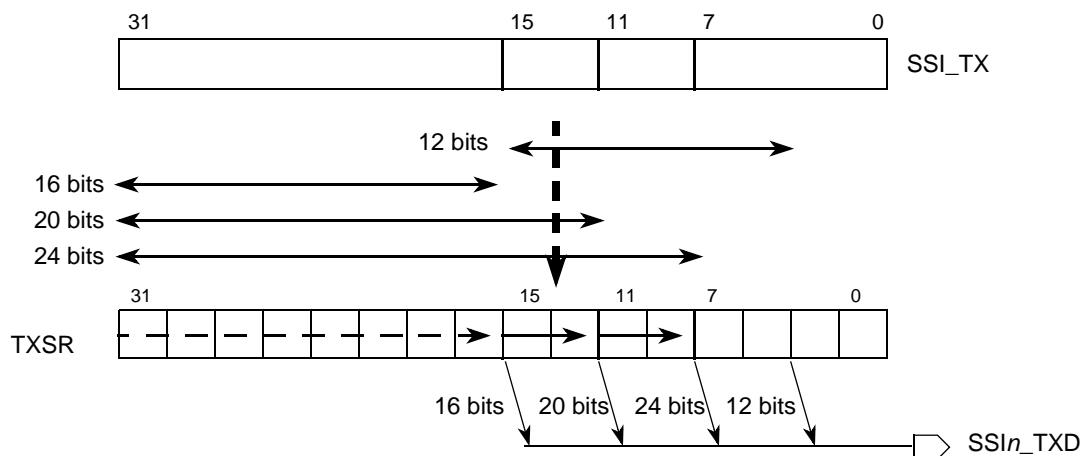
If TXBIT0 is cleared and the word length is less than 16 bits, data occupies the most significant portion of the lower 16 bits of the transmit register.

## Synchronous Serial Interface (SSI)

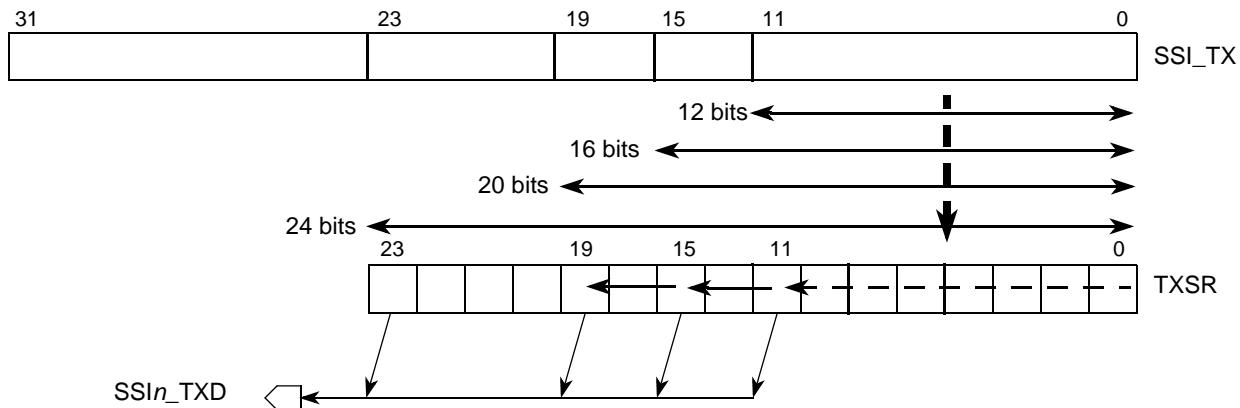
When  $\text{SSI}_{in\_TCR}[\text{SHFD}]$  is cleared, data is shifted out of this register with the most significant bit (msb) first. If this bit is set, the least significant bit (lsb) is shifted out first. The following figures show the transmitter loading and shifting operation. They illustrate some possible values for WL, which can be extended for the other values.



**Figure 35-5. Transmit Data Path ( $\text{TXBIT0}=0$ ,  $\text{TSHFD}=0$ ) (msb Alignment)**



**Figure 35-6. Transmit Data Path ( $\text{TXBIT0}=0$ ,  $\text{TSHFD}=1$ ) (msb Alignment)**



**Figure 35-7. Transmit Data Path ( $\text{TXBIT0}=1$ ,  $\text{TSHFD}=0$ ) (lsb Alignment)**

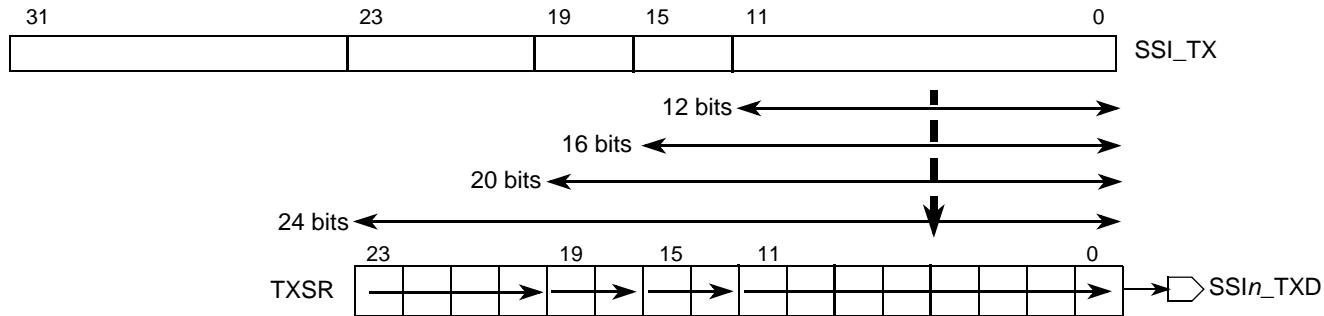


Figure 35-8. Transmit Data Path (TXBIT0=1, TSHFD=1) (lsb Alignment)

### 35.3.4 SSI Receive Data Registers 0 and 1 (SSIn\_RX0/1)

The SSIn\_RX0/1 registers store the data received by the SSI. For details on data alignment see Section 35.3.6, “SSI Receive Shift Register (RXSR).”

Address:	0xFC0B_C008 (SSI0_RX0) 0xFC0B_C00C (SSI0_RX1) 0xFC0C_8008 (SSI1_RX0) 0xFC0C_800C (SSI1_RX1)	Access:	User read/write
R	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 SSI_RX		
W			
Reset	0 0		

Figure 35-9. SSI Receive Data Registers (SSIn\_RX0, SSIn\_RX1)

Table 35-6. SSIn\_RX0/1 Field Descriptions

Field	Description
31–0 SSIn_RX	SSI receive data. SSIn_RX0/1 are implemented as the first word of their respective Rx FIFOs. These bits receive data from RXSR depending on the mode of operation. If both FIFOs are in use, data is transferred to each data register alternately. SSIn_RX1 is only used in two-channel mode.

### 35.3.5 SSI Receive FIFO 0 and 1 Registers

The SSI receive FIFO registers are 15x32-bit registers and are not directly accessible. They always accept data from the receive shift register (RXSR). If the associated interrupt is enabled, an interrupt is generated when the data level in either of the SSI receive FIFOs reaches the selected threshold.

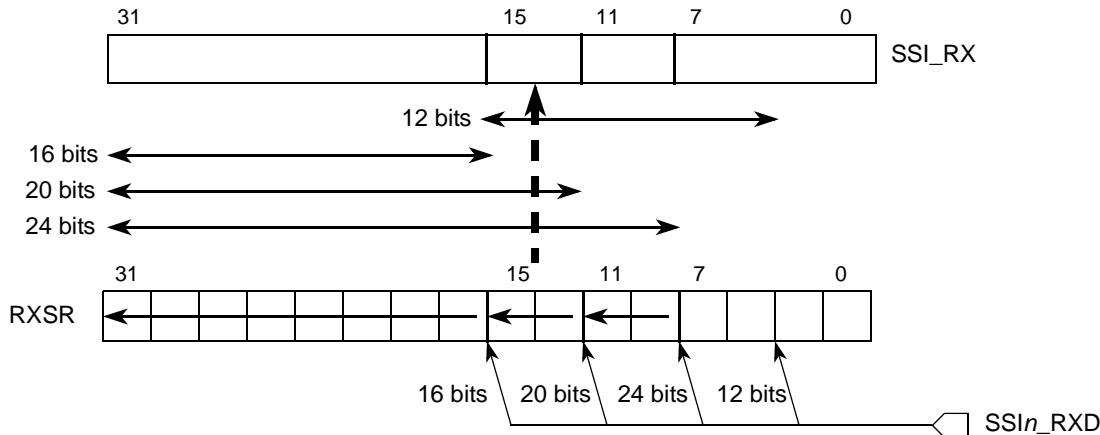
### 35.3.6 SSI Receive Shift Register (RXSR)

RXSR is a 24-bit shift register receiving incoming data from the SSIn\_RXD pin. This register is not directly accessible. When a continuous clock is used, data is shifted in by the bit clock when the associated frame sync is asserted. When a gated clock is used, data is shifted in by the gated clock. Data is assumed to be received msb first if SSIn\_RCR[SHFD] is cleared. If this bit is set, the data is received lsb first. Data is transferred to the appropriate SSI receive data register or receive FIFOs (if the receive FIFO is enabled

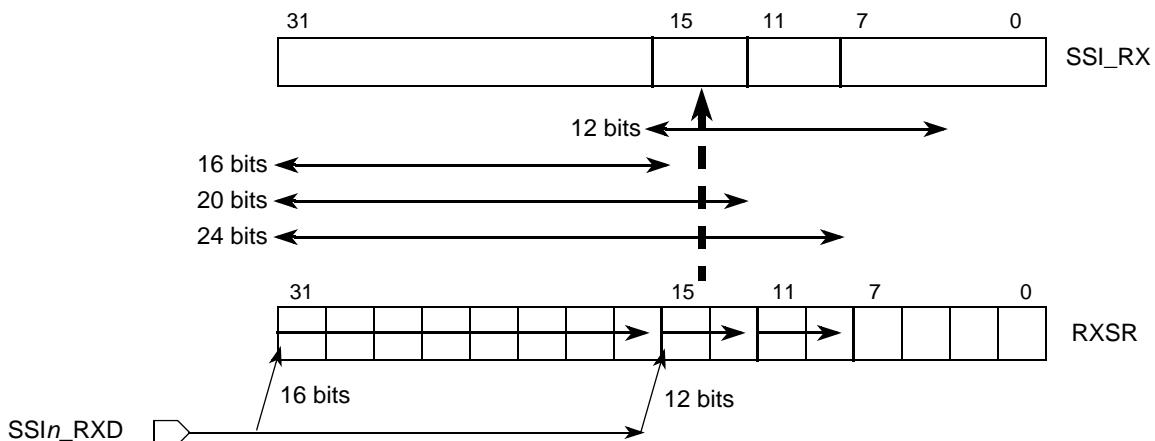
## Synchronous Serial Interface (SSI)

and the corresponding  $\text{SSI}_{n\_RX}$  is full) after a word has been shifted in. For receiving less than 24 bits of data, the lsb bits are appended with 0.

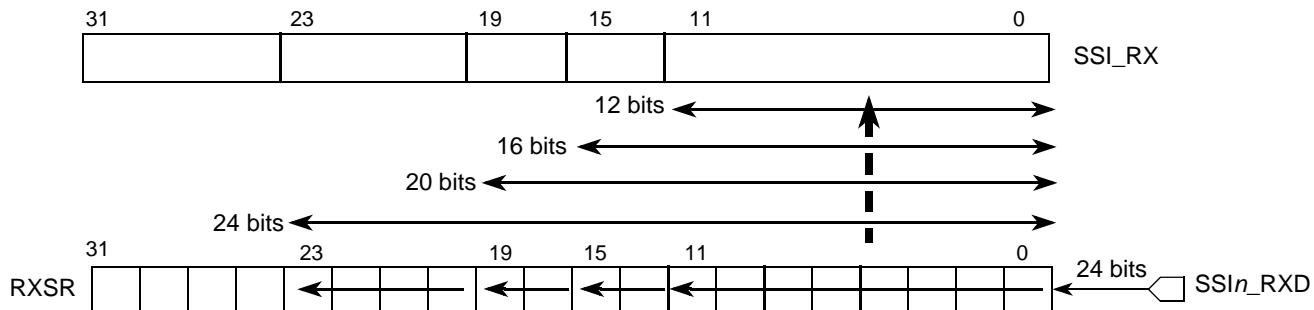
The following figures show the receiver loading and shifting operation. They illustrate some possible values for WL, which can be extended for the other values.



**Figure 35-10. Receive Data Path (RXBIT0=0, RSHFD=0) (msb Alignment)**



**Figure 35-11. Receive Data Path (RXBIT0=0, RSHFD=1) (msb Alignment)**



**Figure 35-12. Receive Data Path (RXBIT0=1, RSHFD=0) (lsb Alignment)**

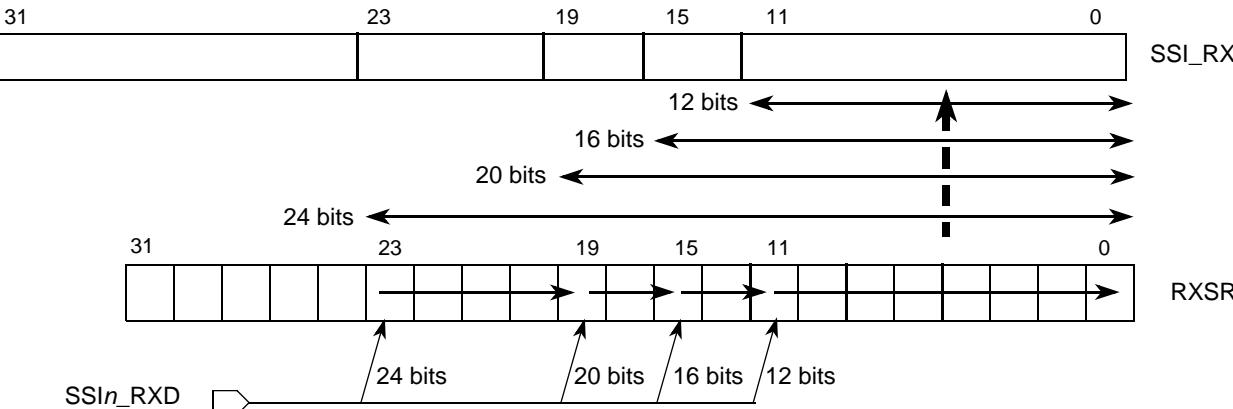


Figure 35-13. Receive Data Path (RXBIT0=1, RSHFD=1) (lsb Alignment)

### 35.3.7 SSI Control Register (SSIn\_CR)

The SSI control register sets up the SSI modules. SSI operating modes are selected in this register (except AC97 mode, which is selected in SSIn\_ACR register).

SSIn_CR																Access: User read/write			
R				W				R				W				R			
31	30	29	28	0	0	0	0	27	26	25	24	0	0	0	0	19	18	17	16
R	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
15	14	13	12	0	0	0	0	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	RCD	TCD	CIS	TCH	MCE	I2S	SYN	NET	RE	TE	SSI_EN				
W					0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 35-14. SSI Control Register (SSIn\_CR)

Table 35-7. SSIn\_CR Field Descriptions

Field	Description
31–12	Reserved, must be cleared.
11 RCD	Receive frame clock disable. Disables the frame sync and clock after the current receive frame when the receiver is disabled (RE is cleared). Writing to this bit only has affect when RE is cleared. 0 Continue frame sync and clock generation after the current frame if RE is cleared. Use this setting when the frame sync and clocks are required, even when no data is received. 1 Stop frame sync and clock generation at the next frame boundary if RE is cleared.
10 TCD	Transmit frame clock disable. Disables the frame sync and clock after the current transmit frame when the transmitter is disabled (TE is cleared). Writing to this bit only has affect when TE is cleared. 0 Continue frame sync and clock generation after the current frame if TE is cleared. Use this setting when the frame sync and clocks are required, even when no data is transmitted. 1 Stop frame sync and clock generation at the next frame boundary if TE is cleared.

**Table 35-7. SSIn\_CR Field Descriptions (continued)**

Field	Description
9 CIS	Clock idle state. Controls the idle state of the transmit clock port (SSIn_BCLK and SSIn_MCLK) during internal gated clock mode. 0 Clock idle state is 1 1 Clock idle state is 0
8 TCH	Two channel operation enable. In this mode, two time slots are used out of the possible 32. Any two time slots (0 – 31) can be selected by the mask registers. The data in the two time slots is alternately handled by the two data registers (0 and 1). While receiving, RXSR transfers data to SSIn_RX0 and SSIn_RX1 alternately, and while transmitting, data is alternately transferred from SSIn_TX0 and SSIn_TX1 to TXSR. Two channel operation can be enabled for an even number of slots larger than two to optimize usage of both FIFOs. However, TCH should be cleared for an odd number of time slots. 0 Two channel mode disabled 1 Two channel mode enabled
7 MCE	Master clock enable. Allows the SSI to output the master clock at the SSIn_MCLK port, if network mode and transmit internal clock mode are set. The DIV2, PSR, and PM bits determine the relationship between the bit clock (SSIn_BCLK) and SSIn_MCLK. In I <sup>2</sup> S master mode, this bit is used to output the oversampling clock on SSIn_MCLK. 0 Master clock not output on the SSIn_MCLK pin 1 Master clock output on the SSIn_MCLK pin
6–5 I <sup>2</sup> S	I <sup>2</sup> S mode select. Selects normal, I <sup>2</sup> S master, or I <sup>2</sup> S slave mode. Refer to <a href="#">Section 35.4.1.4, “I<sup>2</sup>S Mode,”</a> for a detailed description of I <sup>2</sup> S mode. 00 Normal mode 01 I <sup>2</sup> S master mode 10 I <sup>2</sup> S slave mode 11 Normal mode
4 SYN	Synchronous mode enable. In synchronous mode, transmit and receive sections of SSI share a common clock port (SSIn_BCLK) and frame sync port (SSIn_FS). 0 Reserved. 1 Synchronous mode selected.
3 NET	Network mode enable. 0 Network mode not selected 1 Network mode selected
2 RE	Receiver enable. When this bit is set, data reception starts with the arrival of the next frame sync. If data is received when this bit is cleared, data reception continues with the end of the current frame and then stops. If this bit is set again before the second to last bit of the last time slot in the current frame, reception continues without interruption. 0 Receiver disabled 1 Receiver enabled

**Table 35-7. SSIn\_CR Field Descriptions (continued)**

Field	Description
1 TE	<p>Transmitter. Enables the transfer of the contents of the SSIn_TX registers to the TXSR, and also enables the internal transmit clock. The transmit section is enabled when this bit is set and a frame boundary is detected. When this bit is cleared, the transmitter continues to send data until the end of the current frame and then stops. Data can be written to the SSIn_TX registers with the TE bit cleared (the corresponding TDE bit is cleared). If the TE bit is cleared and set again before the second to last bit of the last time slot in the current frame, data transmission continues without interruption.</p> <p>The normal transmit enable sequence is to:</p> <ol style="list-style-type: none"> <li>1. Write data to the SSIn_TX register(s)</li> <li>2. Set the TE bit</li> </ol> <p>The normal transmit disable sequence is to:</p> <ol style="list-style-type: none"> <li>1. Wait for TDE to set</li> <li>2. Clear the TE and TIE bits</li> </ol> <p>In gated clock mode, clearing the TE bit results in the clock stopping after the data currently in TXSR has shifted out. When the TE bit is set, the clock starts immediately in internal gated clock mode.</p> <p>0 Transmitter disabled 1 Transmitter enabled</p>
0 SSI_EN	<p>SSI enable. When disabled, all SSI status bits are reset to the same state produced by the power-on reset, all control bits are unaffected, and the contents of Tx and Rx FIFOs are cleared. When SSI is disabled, all internal clocks are disabled (except the register access clock).</p> <p>0 SSI module is disabled 1 SSI module is enabled</p>

### 35.3.8 SSI Interrupt Status Register (SSIn\_ISR)

The SSI interrupt status register monitors the SSI. This register is used by the processor to interrogate the status of the SSI module. All receiver-related interrupts are generated only if the receiver is enabled (SSIn\_CR[RE] = 1). Likewise, all transmitter-related interrupts are generated only if the transmitter is enabled (SSIn\_CR[TE] = 1).

#### NOTE

Refer to [Section 35.4.5, “Receive Interrupt Enable Bit Description,”](#) and [Section 35.4.6, “Transmit Interrupt Enable Bit Description,”](#) for more details on SSI interrupt generation.

All flags in the SSIn\_ISR are updated after the first bit of the next SSI word has completed transmission or reception. Some status bits (ROE0/1 and TUE0/1) are cleared by writing one to the corresponding bit in the SSIn\_ISR.

## Synchronous Serial Interface (SSI)

Address: 0xFC0B\_C014 (SSI0\_ISR)  
0xFC0C\_8014 (SSI1\_ISR)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	RFRC	TRFC	0	0	0	0	CMDAU	CMDDU	RXT
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	RDR1	RDR0	TDE1	TDE0	ROE1	ROE0	TUE1	TUE0	TFS	RFS	TLS	RLS	RFF1	RFF0	TFE1	TFE0
W					w1c	w1c	w1c	w1c								
Reset	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1

Figure 35-15. SSI Interrupt Status Register (SSI<sub>n</sub>\_ISR)

Table 35-8. SSI<sub>n</sub>\_ISR Field Descriptions

Field	Description
31–25	Reserved, must be cleared.
24 RFRC	Receive frame complete. Indicates the end of the frame when the receiver is disabled. If the receive frame and clock are not disabled in the same frame, this bit is also set at the end of the frame that the receive frame and clock are disabled. See the description of SSI <sub>n</sub> _CR[RCD] for more details on enabling/disabling the receive frame and clock after the receiver is disabled. 0 End of frame not reached 1 End of frame reached after clearing SSI <sub>n</sub> _CR[RE], or setting SSI <sub>n</sub> _CR[RCD] when RE is already cleared.
23 TFRC	Transmit frame complete. Indicates the end of the frame when the transmitter is disabled. If the transmit frame and clock are not disabled in the same frame, this bit is also set at the end of the frame that the transmit frame and clock are disabled. See the description of SSI <sub>n</sub> _CR[TCD] for more details on enabling/disabling the transmit frame and clock after the transmitter is disabled. 0 End of frame not reached 1 End of frame reached after clearing SSI <sub>n</sub> _CR[TE], or setting SSI <sub>n</sub> _CR[RCD] when TE is already cleared.
22–19	Reserved, must be cleared.
18 CMDAU	AC97 command address register updated. This bit causes the command address updated interrupt when the SSI <sub>n</sub> _IER[CMDAU] bit is set. This status bit is set each time there is a difference in the previous and current value of the received command address. This bit is cleared upon reading the SSI <sub>n</sub> _ACADD register. 0 No change in SSI <sub>n</sub> _ACADD register 1 SSI <sub>n</sub> _ACADD register updated with different value
17 CMDDU	AC97 command data register updated. This bit causes the command data updated interrupt when the SSI <sub>n</sub> _IER[CMDDU] bit is set. This status bit is set each time there is a difference in the previous and current value of the received command data. This bit is cleared upon reading the SSI <sub>n</sub> _ACDAT register. 0 No change in SSI <sub>n</sub> _ACDAT register 1 SSI <sub>n</sub> _ACDAT register updated with different value
16 RXT	AC97 receive tag updated. This status bit is set each time there is a difference in the previous and current value of the received tag. It causes the receive tag interrupt if the SSI <sub>n</sub> _IER[RXT] bit is set. This bit is cleared upon reading the SSI <sub>n</sub> _ATAG register. 0 No change in SSI <sub>n</sub> _ATAG register 1 SSI <sub>n</sub> _ATAG register updated with different value

**Table 35-8. SSIn\_ISR Field Descriptions (continued)**

Field	Description																				
15 RDR1	Receive data ready 1. Only valid in two-channel mode. Indicates new data is available for the processor to read.																				
	<table border="1"> <thead> <tr> <th colspan="2">Rx FIFO1</th><th>Receive data 1 interrupt</th></tr> <tr> <th colspan="2">Required conditions</th><th>Trigger</th></tr> </thead> <tbody> <tr> <td colspan="2">Enabled</td><td> <ul style="list-style-type: none"> <li>SSIn_IER[RIE] set</li> <li>SSIn_IER[RFF1] set</li> </ul> </td></tr> <tr> <td colspan="2">Disabled</td><td> <ul style="list-style-type: none"> <li>SSIn_IER[RIE] set</li> <li>SSIn_IER[RDR1] set</li> </ul> </td></tr> </tbody> </table>			Rx FIFO1		Receive data 1 interrupt	Required conditions		Trigger	Enabled		<ul style="list-style-type: none"> <li>SSIn_IER[RIE] set</li> <li>SSIn_IER[RFF1] set</li> </ul>	Disabled		<ul style="list-style-type: none"> <li>SSIn_IER[RIE] set</li> <li>SSIn_IER[RDR1] set</li> </ul>						
Rx FIFO1		Receive data 1 interrupt																			
Required conditions		Trigger																			
Enabled		<ul style="list-style-type: none"> <li>SSIn_IER[RIE] set</li> <li>SSIn_IER[RFF1] set</li> </ul>																			
Disabled		<ul style="list-style-type: none"> <li>SSIn_IER[RIE] set</li> <li>SSIn_IER[RDR1] set</li> </ul>																			
	<table border="1"> <thead> <tr> <th colspan="2">Rx FIFO1</th><th>RDR1 is set when</th></tr> <tr> <th colspan="2">RDR1 is cleared during any of the following</th><th></th></tr> </thead> <tbody> <tr> <td colspan="2">Enabled</td><td> <ul style="list-style-type: none"> <li>Rx FIFO1 loaded with new value</li> </ul> </td></tr> <tr> <td colspan="2">Disabled</td><td> <ul style="list-style-type: none"> <li>SSIn_RX1 loaded with new value</li> </ul> </td></tr> <tr> <td colspan="2"></td><td> <ul style="list-style-type: none"> <li>Rx FIFO1 is empty</li> <li>SSI reset</li> <li>POR reset</li> </ul> </td></tr> <tr> <td colspan="2"></td><td> <ul style="list-style-type: none"> <li>SSIn_RX1 is read</li> <li>SSI reset</li> <li>POR reset</li> </ul> </td></tr> </tbody> </table>			Rx FIFO1		RDR1 is set when	RDR1 is cleared during any of the following			Enabled		<ul style="list-style-type: none"> <li>Rx FIFO1 loaded with new value</li> </ul>	Disabled		<ul style="list-style-type: none"> <li>SSIn_RX1 loaded with new value</li> </ul>			<ul style="list-style-type: none"> <li>Rx FIFO1 is empty</li> <li>SSI reset</li> <li>POR reset</li> </ul>			<ul style="list-style-type: none"> <li>SSIn_RX1 is read</li> <li>SSI reset</li> <li>POR reset</li> </ul>
Rx FIFO1		RDR1 is set when																			
RDR1 is cleared during any of the following																					
Enabled		<ul style="list-style-type: none"> <li>Rx FIFO1 loaded with new value</li> </ul>																			
Disabled		<ul style="list-style-type: none"> <li>SSIn_RX1 loaded with new value</li> </ul>																			
		<ul style="list-style-type: none"> <li>Rx FIFO1 is empty</li> <li>SSI reset</li> <li>POR reset</li> </ul>																			
		<ul style="list-style-type: none"> <li>SSIn_RX1 is read</li> <li>SSI reset</li> <li>POR reset</li> </ul>																			
14 RDR0	Receive data ready 0. Similar description as RDR1 but pertains to Rx FIFO 0 and it is not necessary to be in two-channel mode for this bit to be set. 0 No new data for core to read 1 New data for core to read																				
13 TDE1	Transmit data register empty 1. Only valid in two-channel mode. Indicates that data needs to be written to the SSI.																				
	<table border="1"> <thead> <tr> <th colspan="2">Tx FIFO1</th><th>Transmit data 1 interrupt</th></tr> <tr> <th colspan="2">Required conditions</th><th>Trigger</th></tr> </thead> <tbody> <tr> <td colspan="2">Enabled</td><td> <ul style="list-style-type: none"> <li>SSIn_IER[TIE] set</li> <li>SSIn_IER[TDE1] set</li> </ul> </td></tr> <tr> <td colspan="2">Disabled</td><td> <ul style="list-style-type: none"> <li>SSIn_IER[TIE] set</li> <li>SSIn_IER[TDE1] set</li> </ul> </td></tr> </tbody> </table>			Tx FIFO1		Transmit data 1 interrupt	Required conditions		Trigger	Enabled		<ul style="list-style-type: none"> <li>SSIn_IER[TIE] set</li> <li>SSIn_IER[TDE1] set</li> </ul>	Disabled		<ul style="list-style-type: none"> <li>SSIn_IER[TIE] set</li> <li>SSIn_IER[TDE1] set</li> </ul>						
Tx FIFO1		Transmit data 1 interrupt																			
Required conditions		Trigger																			
Enabled		<ul style="list-style-type: none"> <li>SSIn_IER[TIE] set</li> <li>SSIn_IER[TDE1] set</li> </ul>																			
Disabled		<ul style="list-style-type: none"> <li>SSIn_IER[TIE] set</li> <li>SSIn_IER[TDE1] set</li> </ul>																			
	<table border="1"> <thead> <tr> <th colspan="2">Tx FIFO1</th><th>TDE1 is set when</th></tr> <tr> <th colspan="2">TDE1 is cleared when any of the following occur</th><th></th></tr> </thead> <tbody> <tr> <td colspan="2">Enabled</td><td> <ul style="list-style-type: none"> <li>At least one empty slot in Tx FIFO1</li> </ul> </td></tr> <tr> <td colspan="2">Disabled</td><td> <ul style="list-style-type: none"> <li>SSIn_TX1 data transferred to TXSR</li> </ul> </td></tr> <tr> <td colspan="2"></td><td> <ul style="list-style-type: none"> <li>Tx FIFO1 is full</li> <li>SSI reset</li> <li>POR reset</li> </ul> </td></tr> <tr> <td colspan="2"></td><td> <ul style="list-style-type: none"> <li>SSIn_TX1 is written</li> <li>SSI reset</li> <li>POR reset</li> </ul> </td></tr> </tbody> </table>			Tx FIFO1		TDE1 is set when	TDE1 is cleared when any of the following occur			Enabled		<ul style="list-style-type: none"> <li>At least one empty slot in Tx FIFO1</li> </ul>	Disabled		<ul style="list-style-type: none"> <li>SSIn_TX1 data transferred to TXSR</li> </ul>			<ul style="list-style-type: none"> <li>Tx FIFO1 is full</li> <li>SSI reset</li> <li>POR reset</li> </ul>			<ul style="list-style-type: none"> <li>SSIn_TX1 is written</li> <li>SSI reset</li> <li>POR reset</li> </ul>
Tx FIFO1		TDE1 is set when																			
TDE1 is cleared when any of the following occur																					
Enabled		<ul style="list-style-type: none"> <li>At least one empty slot in Tx FIFO1</li> </ul>																			
Disabled		<ul style="list-style-type: none"> <li>SSIn_TX1 data transferred to TXSR</li> </ul>																			
		<ul style="list-style-type: none"> <li>Tx FIFO1 is full</li> <li>SSI reset</li> <li>POR reset</li> </ul>																			
		<ul style="list-style-type: none"> <li>SSIn_TX1 is written</li> <li>SSI reset</li> <li>POR reset</li> </ul>																			

**Table 35-8. SSIn\_ISR Field Descriptions (continued)**

Field	Description																							
12 TDE0	<p>Transmit data register empty 0. Similar description as TE1 but pertains to Tx FIFO 0 and it is not necessary to be in two-channel mode for this bit to be set.</p> <p>0 Data available for transmission 1 Data needs to be written by the core for transmission</p>																							
11 ROE1	<p>Receiver overrun error 1. Only valid in two-channel mode. Indicates an overrun error has occurred.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="3">Receiver overrun error 1 interrupt</th> </tr> <tr> <th>Rx FIFO1</th> <th>Required conditions</th> <th>Trigger</th> </tr> </thead> <tbody> <tr> <td>Enabled</td> <td> <ul style="list-style-type: none"> <li>SSIn_IER[RIE] set</li> <li>SSIn_IER[ROE1] set</li> </ul> </td> <td> <ul style="list-style-type: none"> <li>SSIn_ISR[ROE1] sets</li> </ul> </td></tr> <tr> <td>Disabled</td> <td></td> <td></td></tr> </tbody> </table> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Rx FIFO1</th> <th>ROE1 is set when all of the following occur</th> <th>ROE1 is cleared when any of the following occur</th> </tr> </thead> <tbody> <tr> <td>Enabled</td> <td> <ul style="list-style-type: none"> <li>RXSR is full</li> <li>Rx FIFO1 is full</li> </ul> </td> <td> <ul style="list-style-type: none"> <li>Writing a 1 to ROE1</li> <li>SSI reset</li> <li>POR reset</li> </ul> </td></tr> <tr> <td>Disabled</td> <td> <ul style="list-style-type: none"> <li>RXSR is full</li> <li>SSIn_RX1 is full</li> </ul> </td> <td></td></tr> </tbody> </table>			Receiver overrun error 1 interrupt			Rx FIFO1	Required conditions	Trigger	Enabled	<ul style="list-style-type: none"> <li>SSIn_IER[RIE] set</li> <li>SSIn_IER[ROE1] set</li> </ul>	<ul style="list-style-type: none"> <li>SSIn_ISR[ROE1] sets</li> </ul>	Disabled			Rx FIFO1	ROE1 is set when all of the following occur	ROE1 is cleared when any of the following occur	Enabled	<ul style="list-style-type: none"> <li>RXSR is full</li> <li>Rx FIFO1 is full</li> </ul>	<ul style="list-style-type: none"> <li>Writing a 1 to ROE1</li> <li>SSI reset</li> <li>POR reset</li> </ul>	Disabled	<ul style="list-style-type: none"> <li>RXSR is full</li> <li>SSIn_RX1 is full</li> </ul>	
Receiver overrun error 1 interrupt																								
Rx FIFO1	Required conditions	Trigger																						
Enabled	<ul style="list-style-type: none"> <li>SSIn_IER[RIE] set</li> <li>SSIn_IER[ROE1] set</li> </ul>	<ul style="list-style-type: none"> <li>SSIn_ISR[ROE1] sets</li> </ul>																						
Disabled																								
Rx FIFO1	ROE1 is set when all of the following occur	ROE1 is cleared when any of the following occur																						
Enabled	<ul style="list-style-type: none"> <li>RXSR is full</li> <li>Rx FIFO1 is full</li> </ul>	<ul style="list-style-type: none"> <li>Writing a 1 to ROE1</li> <li>SSI reset</li> <li>POR reset</li> </ul>																						
Disabled	<ul style="list-style-type: none"> <li>RXSR is full</li> <li>SSIn_RX1 is full</li> </ul>																							
	<p><b>Note:</b> If Rx FIFO 1 is enabled, the RFF1 flag indicates the FIFO is full. If Rx FIFO 1 is disabled, the RDR1 flag indicates the SSIn_RX1 register is full.</p>																							
10 ROE0	<p>Receiver overrun error 0. Similar description as ROE1 but pertains to Rx FIFO 0 and it is not necessary to be in two-channel mode for this bit to be set.</p> <p>0 No overrun detected 1 Receiver 0 overrun error occurred</p>																							
9 TUE1	<p>Transmitter underrun error 1. Only valid in two-channel mode. When a transmit underrun error occurs, the previous data is retransmitted. In network mode, each time slot requires data transmission (unless masked through the SSIn_TMASK register), when the transmitter is enabled.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="3">Transmit underrun error 1 interrupt</th> </tr> <tr> <th>Tx FIFO1</th> <th>Required conditions</th> <th>Trigger</th> </tr> </thead> <tbody> <tr> <td>Enabled</td> <td> <ul style="list-style-type: none"> <li>SSI_IER[TIE] set</li> <li>SSI_IER[TUE1] set</li> </ul> </td> <td> <ul style="list-style-type: none"> <li>SSI_ISR[TUE1] sets</li> </ul> </td></tr> <tr> <td>Disabled</td> <td></td> <td></td></tr> </tbody> </table> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>Tx FIFO1</th> <th>TUE1 is set when all of the following occur</th> <th>TUE1 is cleared when any of the following occur</th> </tr> </thead> <tbody> <tr> <td>Enabled</td> <td> <ul style="list-style-type: none"> <li>TXSR is empty</li> <li>SSIn_ISR[TDE1] set</li> <li>Transmit time slot occurs</li> </ul> </td> <td> <ul style="list-style-type: none"> <li>Writing a 1 to TUE1</li> <li>SSI reset</li> <li>POR reset</li> </ul> </td></tr> <tr> <td>Disabled</td> <td></td> <td></td></tr> </tbody> </table>			Transmit underrun error 1 interrupt			Tx FIFO1	Required conditions	Trigger	Enabled	<ul style="list-style-type: none"> <li>SSI_IER[TIE] set</li> <li>SSI_IER[TUE1] set</li> </ul>	<ul style="list-style-type: none"> <li>SSI_ISR[TUE1] sets</li> </ul>	Disabled			Tx FIFO1	TUE1 is set when all of the following occur	TUE1 is cleared when any of the following occur	Enabled	<ul style="list-style-type: none"> <li>TXSR is empty</li> <li>SSIn_ISR[TDE1] set</li> <li>Transmit time slot occurs</li> </ul>	<ul style="list-style-type: none"> <li>Writing a 1 to TUE1</li> <li>SSI reset</li> <li>POR reset</li> </ul>	Disabled		
Transmit underrun error 1 interrupt																								
Tx FIFO1	Required conditions	Trigger																						
Enabled	<ul style="list-style-type: none"> <li>SSI_IER[TIE] set</li> <li>SSI_IER[TUE1] set</li> </ul>	<ul style="list-style-type: none"> <li>SSI_ISR[TUE1] sets</li> </ul>																						
Disabled																								
Tx FIFO1	TUE1 is set when all of the following occur	TUE1 is cleared when any of the following occur																						
Enabled	<ul style="list-style-type: none"> <li>TXSR is empty</li> <li>SSIn_ISR[TDE1] set</li> <li>Transmit time slot occurs</li> </ul>	<ul style="list-style-type: none"> <li>Writing a 1 to TUE1</li> <li>SSI reset</li> <li>POR reset</li> </ul>																						
Disabled																								

**Table 35-8. SSIn\_ISR Field Descriptions (continued)**

Field	Description																						
8 TUE0	Transmitter underrun error 0. Similar description as TUE1 but pertains to TDE0 and it is not necessary to be in two-channel mode for this bit to be set. 0 No underrun detected 1 Transmitter 0 underrun error occurred																						
7 TFS	Transmit frame sync. Indicates occurrence of a transmit frame sync during transmission of the last word written to the SSIn_TX registers																						
		<table border="1"> <thead> <tr> <th>SSI Mode</th> <th colspan="2">Transmit frame sync interrupt</th> </tr> <tr> <th></th> <th>Required conditions</th> <th>Trigger</th> </tr> </thead> <tbody> <tr> <td>Normal</td> <td> <ul style="list-style-type: none"> <li>SSIn_IER[TIE] set</li> <li>SSIn_IER[TFS] set</li> </ul> </td> <td> <ul style="list-style-type: none"> <li>SSIn_ISR[TFS] sets</li> </ul> </td> </tr> <tr> <td>Network</td> <td></td> <td></td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th>SSI Mode</th> <th>TFS is set when</th> <th>TFS is cleared when any of the following occur</th> </tr> </thead> <tbody> <tr> <td>Normal</td> <td> <ul style="list-style-type: none"> <li>TFS is always set</li> </ul> </td> <td> <ul style="list-style-type: none"> <li>SSI reset</li> <li>POR reset</li> </ul> </td> </tr> <tr> <td>Network</td> <td> <ul style="list-style-type: none"> <li>First time slot transmission</li> </ul> </td> <td> <ul style="list-style-type: none"> <li>Starts transmitting next time slot</li> <li>SSI reset</li> <li>POR reset</li> </ul> </td> </tr> </tbody> </table>	SSI Mode	Transmit frame sync interrupt			Required conditions	Trigger	Normal	<ul style="list-style-type: none"> <li>SSIn_IER[TIE] set</li> <li>SSIn_IER[TFS] set</li> </ul>	<ul style="list-style-type: none"> <li>SSIn_ISR[TFS] sets</li> </ul>	Network			SSI Mode	TFS is set when	TFS is cleared when any of the following occur	Normal	<ul style="list-style-type: none"> <li>TFS is always set</li> </ul>	<ul style="list-style-type: none"> <li>SSI reset</li> <li>POR reset</li> </ul>	Network	<ul style="list-style-type: none"> <li>First time slot transmission</li> </ul>	<ul style="list-style-type: none"> <li>Starts transmitting next time slot</li> <li>SSI reset</li> <li>POR reset</li> </ul>
SSI Mode	Transmit frame sync interrupt																						
	Required conditions	Trigger																					
Normal	<ul style="list-style-type: none"> <li>SSIn_IER[TIE] set</li> <li>SSIn_IER[TFS] set</li> </ul>	<ul style="list-style-type: none"> <li>SSIn_ISR[TFS] sets</li> </ul>																					
Network																							
SSI Mode	TFS is set when	TFS is cleared when any of the following occur																					
Normal	<ul style="list-style-type: none"> <li>TFS is always set</li> </ul>	<ul style="list-style-type: none"> <li>SSI reset</li> <li>POR reset</li> </ul>																					
Network	<ul style="list-style-type: none"> <li>First time slot transmission</li> </ul>	<ul style="list-style-type: none"> <li>Starts transmitting next time slot</li> <li>SSI reset</li> <li>POR reset</li> </ul>																					
<b>Note:</b> Data written to the SSIn_TX registers during the time slot when the TFS flag is set is sent during the second time slot (in network mode) or in the next first time slot (in normal mode).																							
6 RFS	Receive frame sync. Indicates occurrence of a receive frame sync during reception of the next word in SSIn_RX registers.																						
		<table border="1"> <thead> <tr> <th>SSI Mode</th> <th colspan="2">Receive frame sync interrupt</th> </tr> <tr> <th></th> <th>Required conditions</th> <th>Trigger</th> </tr> </thead> <tbody> <tr> <td>Normal</td> <td> <ul style="list-style-type: none"> <li>SSIn_IER[RIE] set</li> <li>SSIn_IER[RFS] set</li> </ul> </td> <td> <ul style="list-style-type: none"> <li>SSIn_ISR[RFS] sets</li> </ul> </td> </tr> <tr> <td>Network</td> <td></td> <td></td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th>SSI Mode</th> <th>RFS is set when</th> <th>RFS is cleared when any of the following occur</th> </tr> </thead> <tbody> <tr> <td>Normal</td> <td> <ul style="list-style-type: none"> <li>RFS is always set</li> </ul> </td> <td> <ul style="list-style-type: none"> <li>SSI reset</li> <li>POR reset</li> </ul> </td> </tr> <tr> <td>Network</td> <td> <ul style="list-style-type: none"> <li>First time slot received</li> </ul> </td> <td> <ul style="list-style-type: none"> <li>Starts receiving next time slot</li> <li>SSI reset</li> <li>POR reset</li> </ul> </td> </tr> </tbody> </table>	SSI Mode	Receive frame sync interrupt			Required conditions	Trigger	Normal	<ul style="list-style-type: none"> <li>SSIn_IER[RIE] set</li> <li>SSIn_IER[RFS] set</li> </ul>	<ul style="list-style-type: none"> <li>SSIn_ISR[RFS] sets</li> </ul>	Network			SSI Mode	RFS is set when	RFS is cleared when any of the following occur	Normal	<ul style="list-style-type: none"> <li>RFS is always set</li> </ul>	<ul style="list-style-type: none"> <li>SSI reset</li> <li>POR reset</li> </ul>	Network	<ul style="list-style-type: none"> <li>First time slot received</li> </ul>	<ul style="list-style-type: none"> <li>Starts receiving next time slot</li> <li>SSI reset</li> <li>POR reset</li> </ul>
SSI Mode	Receive frame sync interrupt																						
	Required conditions	Trigger																					
Normal	<ul style="list-style-type: none"> <li>SSIn_IER[RIE] set</li> <li>SSIn_IER[RFS] set</li> </ul>	<ul style="list-style-type: none"> <li>SSIn_ISR[RFS] sets</li> </ul>																					
Network																							
SSI Mode	RFS is set when	RFS is cleared when any of the following occur																					
Normal	<ul style="list-style-type: none"> <li>RFS is always set</li> </ul>	<ul style="list-style-type: none"> <li>SSI reset</li> <li>POR reset</li> </ul>																					
Network	<ul style="list-style-type: none"> <li>First time slot received</li> </ul>	<ul style="list-style-type: none"> <li>Starts receiving next time slot</li> <li>SSI reset</li> <li>POR reset</li> </ul>																					

**Table 35-8. SSIn\_ISR Field Descriptions (continued)**

Field	Description														
5 TLS 4 RLS	Transmit/receive last time slot. Indicates the current time slot is the last time slot of the frame.														
	<table border="1"> <thead> <tr> <th colspan="3">Last time slot interrupts</th> </tr> <tr> <th></th> <th>Required conditions</th> <th>Trigger</th> </tr> </thead> <tbody> <tr> <td>TLS</td><td> <ul style="list-style-type: none"> <li>SSIn_IER[TIE] set</li> <li>SSIn_IER[TLS] set</li> </ul> </td><td> <ul style="list-style-type: none"> <li>SSIn_ISR[TLS] sets</li> </ul> </td></tr> <tr> <td>RLS</td><td> <ul style="list-style-type: none"> <li>SSIn_IER[RIE] set</li> <li>SSIn_IER[RLS] set</li> </ul> </td><td> <ul style="list-style-type: none"> <li>SSIn_ISR[RLS] sets</li> </ul> </td></tr> </tbody> </table>			Last time slot interrupts				Required conditions	Trigger	TLS	<ul style="list-style-type: none"> <li>SSIn_IER[TIE] set</li> <li>SSIn_IER[TLS] set</li> </ul>	<ul style="list-style-type: none"> <li>SSIn_ISR[TLS] sets</li> </ul>	RLS	<ul style="list-style-type: none"> <li>SSIn_IER[RIE] set</li> <li>SSIn_IER[RLS] set</li> </ul>	<ul style="list-style-type: none"> <li>SSIn_ISR[RLS] sets</li> </ul>
Last time slot interrupts															
	Required conditions	Trigger													
TLS	<ul style="list-style-type: none"> <li>SSIn_IER[TIE] set</li> <li>SSIn_IER[TLS] set</li> </ul>	<ul style="list-style-type: none"> <li>SSIn_ISR[TLS] sets</li> </ul>													
RLS	<ul style="list-style-type: none"> <li>SSIn_IER[RIE] set</li> <li>SSIn_IER[RLS] set</li> </ul>	<ul style="list-style-type: none"> <li>SSIn_ISR[RLS] sets</li> </ul>													
	<table border="1"> <thead> <tr> <th></th> <th>Is set when</th> <th>Is cleared when any of the following occur</th> </tr> </thead> <tbody> <tr> <td>TLS</td><td> <ul style="list-style-type: none"> <li>Start of last transmit time slot</li> </ul> </td><td> <ul style="list-style-type: none"> <li>SSIn_ISR is read with TLS set</li> <li>SSI reset</li> <li>POR reset</li> </ul> </td></tr> <tr> <td>RLS</td><td> <ul style="list-style-type: none"> <li>End of last receive time slot</li> </ul> </td><td> <ul style="list-style-type: none"> <li>SSIn_ISR is read with RLS set</li> <li>SSI reset</li> <li>POR reset</li> </ul> </td></tr> </tbody> </table>				Is set when	Is cleared when any of the following occur	TLS	<ul style="list-style-type: none"> <li>Start of last transmit time slot</li> </ul>	<ul style="list-style-type: none"> <li>SSIn_ISR is read with TLS set</li> <li>SSI reset</li> <li>POR reset</li> </ul>	RLS	<ul style="list-style-type: none"> <li>End of last receive time slot</li> </ul>	<ul style="list-style-type: none"> <li>SSIn_ISR is read with RLS set</li> <li>SSI reset</li> <li>POR reset</li> </ul>			
	Is set when	Is cleared when any of the following occur													
TLS	<ul style="list-style-type: none"> <li>Start of last transmit time slot</li> </ul>	<ul style="list-style-type: none"> <li>SSIn_ISR is read with TLS set</li> <li>SSI reset</li> <li>POR reset</li> </ul>													
RLS	<ul style="list-style-type: none"> <li>End of last receive time slot</li> </ul>	<ul style="list-style-type: none"> <li>SSIn_ISR is read with RLS set</li> <li>SSI reset</li> <li>POR reset</li> </ul>													
3 RFF1	Receive FIFO full 1. Only valid in two-channel mode and if Rx FIFO 1 is enabled. When Rx FIFO1 is full, all further data received (for storage in this FIFO) is ignored until the FIFO contents are read.														
	<table border="1"> <thead> <tr> <th colspan="3">Receive FIFO full 1 interrupt</th> </tr> <tr> <th></th> <th>Required conditions</th> <th>Trigger</th> </tr> </thead> <tbody> <tr> <td>Rx FIFO1</td><td></td><td></td></tr> <tr> <td>Enabled</td><td> <ul style="list-style-type: none"> <li>SSIn_IER[RIE] set</li> <li>SSIn_IER[RFF1] set</li> </ul> </td><td> <ul style="list-style-type: none"> <li>SSIn_ISR[RFF1] sets</li> </ul> </td></tr> </tbody> </table>			Receive FIFO full 1 interrupt				Required conditions	Trigger	Rx FIFO1			Enabled	<ul style="list-style-type: none"> <li>SSIn_IER[RIE] set</li> <li>SSIn_IER[RFF1] set</li> </ul>	<ul style="list-style-type: none"> <li>SSIn_ISR[RFF1] sets</li> </ul>
Receive FIFO full 1 interrupt															
	Required conditions	Trigger													
Rx FIFO1															
Enabled	<ul style="list-style-type: none"> <li>SSIn_IER[RIE] set</li> <li>SSIn_IER[RFF1] set</li> </ul>	<ul style="list-style-type: none"> <li>SSIn_ISR[RFF1] sets</li> </ul>													
	<table border="1"> <thead> <tr> <th>Rx FIFO1</th> <th>RFF1 is set when</th> <th>RFF1 is cleared when any of the following occur</th> </tr> </thead> <tbody> <tr> <td>Enabled</td><td> <ul style="list-style-type: none"> <li>Rx FIFO 1 level reaches Rx FIFO watermark 1 (RFWM1) threshold</li> </ul> </td><td> <ul style="list-style-type: none"> <li>Rx FIFO 1 level falls below RFWM1 level</li> <li>SSI reset</li> <li>POR reset</li> </ul> </td></tr> </tbody> </table>			Rx FIFO1	RFF1 is set when	RFF1 is cleared when any of the following occur	Enabled	<ul style="list-style-type: none"> <li>Rx FIFO 1 level reaches Rx FIFO watermark 1 (RFWM1) threshold</li> </ul>	<ul style="list-style-type: none"> <li>Rx FIFO 1 level falls below RFWM1 level</li> <li>SSI reset</li> <li>POR reset</li> </ul>						
Rx FIFO1	RFF1 is set when	RFF1 is cleared when any of the following occur													
Enabled	<ul style="list-style-type: none"> <li>Rx FIFO 1 level reaches Rx FIFO watermark 1 (RFWM1) threshold</li> </ul>	<ul style="list-style-type: none"> <li>Rx FIFO 1 level falls below RFWM1 level</li> <li>SSI reset</li> <li>POR reset</li> </ul>													
2 RFF0	Receive FIFO full 0. Similar to description of RFF1, but pertains to Rx FIFO 0 and is not necessary to be in two-channel mode for this bit to be set. 0 Space available in receive FIFO 0 1 Receive FIFO 0 is full														

**Table 35-8. SSIn\_ISR Field Descriptions (continued)**

Field	Description	
1 TFE1	Transmit FIFO empty 1. Only valid when in two-channel mode and Tx FIFO 1 is enabled.	
	<b>Tx FIFO1</b>	<b>Transmit FIFO full 1 interrupt</b>
	Required conditions	Trigger
	Enabled	<ul style="list-style-type: none"> <li>SSIn_IER[RIE] set</li> <li>SSIn_IER[TFE1] set</li> </ul>
	<b>TFE1 is set when any of the following occur</b>	
	Enabled	<ul style="list-style-type: none"> <li>Tx FIFO 1 level falls below Tx FIFO watermark 1 (TFWM1) threshold</li> <li>SSI reset</li> <li>POR reset</li> </ul>
	<b>TFE1 is cleared when any of the following occur</b>	
0 TFE0	Transmit FIFO empty 0. Similar to description of TFE1 but pertains to TX FIFO 0 and it is not necessary to be in two-channel mode for this bit to be set. 0 Transmit FIFO 0 has data for transmission 1 Transmit FIFO 0 is empty	

### 35.3.9 SSI Interrupt Enable Register (SSIn\_IER)

The SSIn\_IER register sets up the SSI interrupts and DMA requests.

Address: 0xFC0B\_C018 (SSI0\_IER)  
0xFC0C\_8018 (SSI1\_IER)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	0	0	0	0	0	0	0	RFRC	TFRC	RDMAE	RIE	TDMAE	TIE	CMD AU	CMDU	RXT
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	RDR1	RDR0	TDE1	TDE0	ROE1	ROE0	TUE1	TUE0	TFS	RFS	TLS	RLS	RFF1	RFF0	TFE1	TFE0
W																
Reset	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1

**Figure 35-16. SSI Interrupt Enable Register (SSIn\_IER)**

**Table 35-9. SSIn\_IER Field Descriptions**

<b>Field</b>	<b>Description</b>
31–25	Reserved, must be cleared.
24–23 RFRC TFRC	Controls if the corresponding status bit in SSIn_ISR can issue an interrupt to the processor. See <a href="#">Section 35.3.8, “SSI Interrupt Status Register (SSIn_ISR),”</a> for details on the individual bits. 0 Status bit cannot issue interrupt. 1 Status bit can issue interrupt.
22 RDMAE	Receive DMA enable. <ul style="list-style-type: none"> <li>• If the Rx FIFO is enabled, a DMA request generates when either of the SSIn_ISR[RFF0/1] bits is set.</li> <li>• If the Rx FIFO is disabled, a DMA request generates when either of the SSIn_ISR[RDR0/1] bits is set.</li> </ul> 0 SSI receiver DMA requests disabled. 1 SSI receiver DMA requests enabled.
21 RIE	Receive interrupt enable. Allows the SSI to issue receiver related interrupts to the processor. Refer to <a href="#">Section 35.4.5, “Receive Interrupt Enable Bit Description,”</a> for a detailed description of this bit. 0 SSI receiver interrupt requests disabled. 1 SSI receiver interrupt requests enabled.
20 TDMAE	Transmit DMA enable. <ul style="list-style-type: none"> <li>• If the Tx FIFO is enabled, a DMA request generates when either of the SSIn_ISR[TFE0/1] bits is set.</li> <li>• If the Tx FIFO is disabled, a DMA request generates when either of the SSIn_ISR[TDE0/1] bits is set.</li> </ul> 0 SSI transmitter DMA requests disabled. 1 SSI transmitter DMA requests enabled.
19 TIE	Transmit interrupt enable. Allows the SSI to issue transmitter data related interrupts to the core. Refer to <a href="#">Section 35.4.6, “Transmit Interrupt Enable Bit Description,”</a> for a detailed description of this bit. 0 SSI transmitter interrupt requests disabled. 1 SSI transmitter interrupt requests enabled.
18–0	Controls if the corresponding status bit in SSIn_ISR can issue an interrupt to the processor. See <a href="#">Section 35.3.8, “SSI Interrupt Status Register (SSIn_ISR),”</a> for details on the individual bits. 0 Status bit cannot issue interrupt. 1 Status bit can issue interrupt.

### 35.3.10 SSI Transmit Configuration Register (SSIn\_TCR)

The SSI transmit configuration register directs the transmit operation of the SSI. A power-on reset clears all SSIn\_TCR bits. However, an SSI reset does not affect the SSIn\_TCR bits.

Address: 0xFC0B_C01C (SSI0_TCR) 0xFC0C_801C (SSI1_TCR)																Access: User read/write			
<b>R</b>																			
<b>W</b>																			
Reset																			
<b>R</b>																			
<b>W</b>																			
Reset																			

Figure 35-17. SSI Transmit Configuration Register (SSIn\_TCR)

Table 35-10. SSIn\_TCR Field Descriptions

Field	Description
31–10	Reserved, must be cleared.
9 TXBIT0	Transmit bit 0 (Alignment). Allows SSI to transmit data word from bit position 0 or 15/31 in the transmit shift register. The shifting data direction can be msb or lsb first, controlled by the TSHFD bit. 0 msb-aligned. Shift with respect to bit 31 (if the word length is 16, 18, 20, 22 or 24) or bit 15 (if the word length is 8, 10 or 12) of the transmit shift register 1 lsb-aligned. Shift with respect to bit 0 of the transmit shift register
8 TFEN1	Transmit FIFO enable 1. <ul style="list-style-type: none"><li>When enabled, the FIFO allows eight samples to be transmitted by the SSI (per channel) (a ninth sample can be shifting out) before SSIn_ISR[TDE1] is set.</li><li>When the FIFO is disabled, SSIn_ISR[TDE1] is set when a single sample is transferred to the transmit shift register. This issues an interrupt if the interrupt is enabled.</li></ul> 0 Transmit FIFO 1 disabled 1 Transmit FIFO 1 enabled
7 TFEN0	Transmit FIFO enable 0. Similar description as TFEN1, but pertains to Tx FIFO 0. 0 Transmit FIFO 0 disabled 1 Transmit FIFO 0 enabled
6 TFDIR	Frame sync direction. Controls the direction and source of the frame sync signal on the SSIn_FS pin. 0 Frame sync is external 1 Frame sync generated internally
5 TXDIR	Clock direction. Controls the direction and source of the clock signal on the SSIn_BCLK pin. Refer to <a href="#">Table 35-3</a> for details of clock port configuration. 0 Clock is external 1 Clock generated internally
4 TSHFD	Transmit shift direction. Controls whether the msb or lsb is transmitted first in a sample. 0 Data transmitted msb first 1 Data transmitted lsb first

**Table 35-10. SSIn\_TCR Field Descriptions (continued)**

Field	Description
3 TSCKP	Transmit clock polarity. Controls which bit clock edge is used to clock out data for the transmit section. 0 Data clocked out on rising edge of bit clock 1 Data clocked out on falling edge of bit clock
2 TFSI	Transmit frame sync invert. Controls the active state of the frame sync I/O signal for the transmit section of SSI. 0 Transmit frame sync is active high 1 Transmit frame sync is active low
1 TFSL	Transmit frame sync length. Controls the length of the frame sync signal generated or recognized for the transmit section. The length of a word-long frame sync is the same as the length of the data word selected by SSIn_CCR[WL]. 0 Transmit frame sync is one-word long 1 Transmit frame sync is one-bit-clock-period long
0 TEFS	Transmit early frame sync. Controls when the frame sync is initiated for the transmit section. The frame sync signal is deasserted after one bit for a bit length frame sync (TFSL = 1) and after one word for word length frame sync (TFSL = 0). The frame sync can also be initiated upon receiving the first bit of data. 0 Transmit frame sync initiated as first bit of data transmits 1 Transmit frame sync is initiated one bit before the data transmits

### 35.3.11 SSI Receive Configuration Register (SSIn\_RCR)

The SSIn\_RCR directs the receive operation of the SSI. A power-on reset clears all SSIn\_RCR bits. However, an SSI reset does not affect the SSIn\_RCR bits.

Address: 0xFC0B\_C020 (SSI0\_RCR)  
0xFC0C\_8020 (SSI1\_RCR)

Access: User read/write

R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	0	0	0	0	0	RX EXT	RX BIT0	RFEN1	RFENO	0	RXDIR	RSHFD	RSCKP	RFSI	RFSL	REFS
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0

**Figure 35-18. SSI Receive Configuration Register (SSIn\_RCR)****Table 35-11. SSIn\_RCR Field Descriptions**

Field	Description
31–11	Reserved, must be cleared.
10 RXEXT	Receive data extension. Allows the SSI to store the received data word in sign-extended form. This bit affects data storage only if the received data is lsb-aligned (RXBIT0 = 1) 0 Sign extension disabled 1 Sign extension enabled

**Table 35-11. SSIn\_RCR Field Descriptions (continued)**

Field	Description
9 RXBIT0	Receive bit 0 (Alignment). Allows SSI to receive the data word at bit position 0 or 15/31 in the receive shift register. The shifting data direction can be msb or lsb first, controlled by the RSHFD bit. 0 msb aligned. Shifting with respect to bit 31 (if word length equals 16, 18, 20, 22 or 24) or bit 15 (if word length equals 8, 10 or 12) of the receive shift register 1 lsb aligned. Shifting with respect to bit 0 of the receive shift register.
8 RFEN1	Receive FIFO enable 1. <ul style="list-style-type: none"> <li>• When the FIFO is enabled, the FIFO allows eight samples to be received by the SSI (per channel) (a ninth sample can be shifting in) before the SSIn_ISR[RDR1] bit is set.</li> <li>• When the FIFO is disabled, SSIn_ISR[RDR1] is set when a single sample is received by the SSI.</li> </ul> 0 Receive FIFO 1 disabled 1 Receive FIFO 1 enabled
7 RFENO	Receive FIFO enable 0. Similar description as RFEN1 but pertains to Rx FIFO 0. 0 Receive FIFO 0 disabled 1 Receive FIFO 0 enabled
6	Reserved, must be cleared.
5 RXDIR	Gated clock enable. In synchronous mode, this bit enables gated clock mode. 0 Gated clock mode disabled 1 Gated clock mode enabled
4 RSHFD	Receive shift direction. Controls whether the msb or lsb is received first in a sample. 0 Data received msb first 1 Data received lsb first
3 RSCKP	Receive clock polarity. Controls which bit clock edge latches in data for the receive section. 0 Data latched on falling edge of bit clock 1 Data latched on rising edge of bit clock
2 RFSI	Receive frame sync invert. Controls the active state of the frame sync signal for the receive section of SSI. 0 Receive frame sync is active high 1 Receive frame sync is active low
1 RFSL	Receive frame sync length. Controls the length of the frame sync signal generated or recognized for the receive section. The length of a word-long frame sync is the same as the length of the data word selected by SSIn_CCR[WL]. 0 Receive frame sync is one word long. 1 Receive frame sync is one bit-clock-period long.
0 REFS	Receive early frame sync. Controls when the frame sync is initiated for the receive section. The frame sync is disabled after one bit for bit length frame sync and after one word for word length frame sync. 0 Receive frame sync initiated as the first bit of data is received. 1 Receive frame sync is initiated one bit before the data is received.

### 35.3.12 SSI Clock Control Register (SSIn\_CCR)

The SSI clock control register controls the SSI clock generator, bit and frame sync rates, word length, and number of words per frame for the serial data. The SSIn\_CCR register controls the receive and transmit sections. Power-on reset clears all SSIn\_CCR bits, while an SSI reset does not affect these bits.

## Synchronous Serial Interface (SSI)

Address: 0xFC0B\_C024 (SSI0\_CCR)  
0xFC0C\_8024 (SSI1\_CCR)

Access: User 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	0	0	0	0	0	0	0	0	0	0	0	0	0	DIV2	PSR	WL	DC	PM														
W														0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 35-19. SSI Clock Control Register (SSI<sub>n</sub>\_CCR)

Table 35-12. SSI<sub>n</sub>\_CCR Field Descriptions

Field	Description																																																							
31–19	Reserved, must be cleared.																																																							
18 DIV2	Divide-by-2. Controls the divide-by-two divider in series with the rest of the prescalers. 0 Divider bypassed 1 Divider enabled to divide clock by 2																																																							
17 PSR	Prescaler range. Controls a fixed divide-by-eight prescaler in series with the variable prescaler. It extends the range of the prescaler for those cases where a slower bit clock is required. 0 Prescaler bypassed 1 Prescaler enabled to divide the clock by 8																																																							
16–13 WL	Word length. Controls: <ul style="list-style-type: none"> <li>the length of the data words transferred by the SSI</li> <li>the word length divider in the clock generator</li> <li>the frame sync pulse length when the FSL bit is cleared</li> </ul> In I <sup>2</sup> S master mode, the SSI works with a fixed word length of 32, and the WL bits control the amount of valid data in those 32 bits. Bits per word equal $2 \times (WL + 1)$ . Refer to the below table for details of data word lengths supported by the SSI module. <b>Note:</b> In AC97 mode, if WL is set to any value other than 16 bits, a word length of 20 bits is used.																																																							
	<table border="1"> <thead> <tr> <th>WL</th> <th>Bits/word</th> <th>Supported?</th> <th>WL</th> <th>Bits/word</th> <th>Supported?</th> </tr> </thead> <tbody> <tr> <td>0000</td> <td>2</td> <td>No</td> <td>1000</td> <td>18</td> <td>Yes</td> </tr> <tr> <td>0001</td> <td>4</td> <td>No</td> <td>1001</td> <td>20</td> <td>Yes</td> </tr> <tr> <td>0010</td> <td>6</td> <td>No</td> <td>1010</td> <td>22</td> <td>Yes</td> </tr> <tr> <td>0011</td> <td>8</td> <td>Yes</td> <td>1011</td> <td>24</td> <td>Yes</td> </tr> <tr> <td>0100</td> <td>10</td> <td>Yes</td> <td>1100</td> <td>26</td> <td>No</td> </tr> <tr> <td>0101</td> <td>12</td> <td>Yes</td> <td>1101</td> <td>28</td> <td>No</td> </tr> <tr> <td>0110</td> <td>14</td> <td>No</td> <td>1110</td> <td>30</td> <td>No</td> </tr> <tr> <td>0111</td> <td>16</td> <td>Yes</td> <td>1111</td> <td>32</td> <td>No</td> </tr> </tbody> </table>		WL	Bits/word	Supported?	WL	Bits/word	Supported?	0000	2	No	1000	18	Yes	0001	4	No	1001	20	Yes	0010	6	No	1010	22	Yes	0011	8	Yes	1011	24	Yes	0100	10	Yes	1100	26	No	0101	12	Yes	1101	28	No	0110	14	No	1110	30	No	0111	16	Yes	1111	32	No
WL	Bits/word	Supported?	WL	Bits/word	Supported?																																																			
0000	2	No	1000	18	Yes																																																			
0001	4	No	1001	20	Yes																																																			
0010	6	No	1010	22	Yes																																																			
0011	8	Yes	1011	24	Yes																																																			
0100	10	Yes	1100	26	No																																																			
0101	12	Yes	1101	28	No																																																			
0110	14	No	1110	30	No																																																			
0111	16	Yes	1111	32	No																																																			

**Table 35-12. SSIn\_CCR Field Descriptions (continued)**

Field	Description
12–8 DC	<p>Frame rate divider control. Controls the divide ratio for the programmable frame rate dividers. The divide ratio works on the word clock.</p> <ul style="list-style-type: none"> <li>In normal mode, the ratio determines the word transfer rate. Ranges from 1 to 32.</li> <li>In network mode, this field sets the number of words per frame. Ranges from 2 to 32.</li> </ul> <p>In normal mode, a divide ratio of 1 (DC = 00000) provides continuous periodic data word transfer. A bit-length frame sync must be used in this case; otherwise, in word-length mode the frame sync is always asserted.</p>
7–0 PM	<p>Prescaler modulus select. Controls the prescale divider in the clock generator. This prescaler is used only in internal clock mode to divide the SSI clock. The bit clock output is available at the SSI_BCLK clock pin.</p> <p>A divide ratio from 1 to 256 (PM = 0x00 to 0xFF) can be selected. Refer to <a href="#">Section 35.4.2.2, “DIV2, PSR and PM Bit Description,”</a> for details regarding settings.</p>

### 35.3.13 SSI FIFO Control/Status Register (SSIn\_FCSR)

See [Figure 35-20](#) for illustration of valid bits in SSI FIFO Control/Status Register and [Table 35-13](#) for description of the bit fields in the register.

0xBASE_2C (SFCSR)																Access: User read/write			
R																			
W																			
RESET																			
15 14 13 12																			
R																			
W																			
RESET																			

**Figure 35-20. SSI FIFO Control/Status Register**

**Table 35-13. SSI FIFO Control/Status Register Field Descriptions**

<b>Field</b>	<b>Description</b>																																		
31–28 RFCNT1[3:0]	<p>Receive FIFO Counter1. These bits indicate the number of data words in Receive FIFO 1. Refer to <a href="#">Table 35-14</a> for details regarding settings for receive FIFO counter bits.</p> <p style="text-align: center;"><b>Table 35-14. Receive FIFO Counter Bit Description</b></p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;"><b>Bits</b></th><th style="text-align: center;"><b>Description</b></th></tr> </thead> <tbody> <tr><td style="text-align: center;">0000</td><td>0 data word in receive FIFO</td></tr> <tr><td style="text-align: center;">0001</td><td>1 data word in receive FIFO</td></tr> <tr><td style="text-align: center;">0010</td><td>2 data word in receive FIFO</td></tr> <tr><td style="text-align: center;">0011</td><td>3 data word in receive FIFO</td></tr> <tr><td style="text-align: center;">0100</td><td>4 data word in receive FIFO</td></tr> <tr><td style="text-align: center;">0101</td><td>5 data word in receive FIFO</td></tr> <tr><td style="text-align: center;">0110</td><td>6 data word in receive FIFO</td></tr> <tr><td style="text-align: center;">0111</td><td>7 data word in receive FIFO</td></tr> <tr><td style="text-align: center;">1000</td><td>8 data word in receive FIFO</td></tr> <tr><td style="text-align: center;">1001</td><td>9 data word in receive FIFO</td></tr> <tr><td style="text-align: center;">1010</td><td>10 data word in receive FIFO</td></tr> <tr><td style="text-align: center;">1011</td><td>11 data word in receive FIFO</td></tr> <tr><td style="text-align: center;">1100</td><td>12 data word in receive FIFO</td></tr> <tr><td style="text-align: center;">1101</td><td>13 data word in receive FIFO</td></tr> <tr><td style="text-align: center;">1110</td><td>14 data word in receive FIFO</td></tr> <tr><td style="text-align: center;">1111</td><td>15 data word in receive FIFO</td></tr> </tbody> </table>	<b>Bits</b>	<b>Description</b>	0000	0 data word in receive FIFO	0001	1 data word in receive FIFO	0010	2 data word in receive FIFO	0011	3 data word in receive FIFO	0100	4 data word in receive FIFO	0101	5 data word in receive FIFO	0110	6 data word in receive FIFO	0111	7 data word in receive FIFO	1000	8 data word in receive FIFO	1001	9 data word in receive FIFO	1010	10 data word in receive FIFO	1011	11 data word in receive FIFO	1100	12 data word in receive FIFO	1101	13 data word in receive FIFO	1110	14 data word in receive FIFO	1111	15 data word in receive FIFO
<b>Bits</b>	<b>Description</b>																																		
0000	0 data word in receive FIFO																																		
0001	1 data word in receive FIFO																																		
0010	2 data word in receive FIFO																																		
0011	3 data word in receive FIFO																																		
0100	4 data word in receive FIFO																																		
0101	5 data word in receive FIFO																																		
0110	6 data word in receive FIFO																																		
0111	7 data word in receive FIFO																																		
1000	8 data word in receive FIFO																																		
1001	9 data word in receive FIFO																																		
1010	10 data word in receive FIFO																																		
1011	11 data word in receive FIFO																																		
1100	12 data word in receive FIFO																																		
1101	13 data word in receive FIFO																																		
1110	14 data word in receive FIFO																																		
1111	15 data word in receive FIFO																																		

**Table 35-13. SSI FIFO Control/Status Register Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>																																		
27–24 TFCNT1[3:0]	<p>Transmit FIFO Counter1. These bits indicate the number of data words in Transmit FIFO. Refer to <a href="#">Table 35-15</a> for details regarding settings for transmit FIFO counter bits.</p> <p style="text-align: center;"><b>Table 35-15. Transmit FIFO Counter Bit Description</b></p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;"><b>Bits</b></th><th style="text-align: center;"><b>Description</b></th></tr> </thead> <tbody> <tr><td style="text-align: center;">0000</td><td>0 data word in transmit FIFO</td></tr> <tr><td style="text-align: center;">0001</td><td>1 data word in transmit FIFO</td></tr> <tr><td style="text-align: center;">0010</td><td>2 data word in transmit FIFO</td></tr> <tr><td style="text-align: center;">0011</td><td>3 data word in transmit FIFO</td></tr> <tr><td style="text-align: center;">0100</td><td>4 data word in transmit FIFO</td></tr> <tr><td style="text-align: center;">0101</td><td>5 data word in transmit FIFO</td></tr> <tr><td style="text-align: center;">0110</td><td>6 data word in transmit FIFO</td></tr> <tr><td style="text-align: center;">0111</td><td>7 data word in transmit FIFO</td></tr> <tr><td style="text-align: center;">1000</td><td>8 data word in transmit FIFO</td></tr> <tr><td style="text-align: center;">1001</td><td>9 data word in transmit FIFO</td></tr> <tr><td style="text-align: center;">1010</td><td>10 data word in transmit FIFO</td></tr> <tr><td style="text-align: center;">1011</td><td>11 data word in transmit FIFO</td></tr> <tr><td style="text-align: center;">1100</td><td>12 data word in transmit FIFO</td></tr> <tr><td style="text-align: center;">1101</td><td>13 data word in transmit FIFO</td></tr> <tr><td style="text-align: center;">1110</td><td>14 data word in transmit FIFO</td></tr> <tr><td style="text-align: center;">1111</td><td>15 data word in transmit FIFO</td></tr> </tbody> </table>	<b>Bits</b>	<b>Description</b>	0000	0 data word in transmit FIFO	0001	1 data word in transmit FIFO	0010	2 data word in transmit FIFO	0011	3 data word in transmit FIFO	0100	4 data word in transmit FIFO	0101	5 data word in transmit FIFO	0110	6 data word in transmit FIFO	0111	7 data word in transmit FIFO	1000	8 data word in transmit FIFO	1001	9 data word in transmit FIFO	1010	10 data word in transmit FIFO	1011	11 data word in transmit FIFO	1100	12 data word in transmit FIFO	1101	13 data word in transmit FIFO	1110	14 data word in transmit FIFO	1111	15 data word in transmit FIFO
<b>Bits</b>	<b>Description</b>																																		
0000	0 data word in transmit FIFO																																		
0001	1 data word in transmit FIFO																																		
0010	2 data word in transmit FIFO																																		
0011	3 data word in transmit FIFO																																		
0100	4 data word in transmit FIFO																																		
0101	5 data word in transmit FIFO																																		
0110	6 data word in transmit FIFO																																		
0111	7 data word in transmit FIFO																																		
1000	8 data word in transmit FIFO																																		
1001	9 data word in transmit FIFO																																		
1010	10 data word in transmit FIFO																																		
1011	11 data word in transmit FIFO																																		
1100	12 data word in transmit FIFO																																		
1101	13 data word in transmit FIFO																																		
1110	14 data word in transmit FIFO																																		
1111	15 data word in transmit FIFO																																		

**Table 35-13. SSI FIFO Control/Status Register Field Descriptions (continued)**

Field	Description																																		
23–20 RFWM1[3:0]	<p>Receive FIFO Full WaterMark 1. These bits control the threshold at which the RFF1 flag will be set. The RFF1 flag is set whenever the data level in Rx FIFO 1 reaches the selected threshold. Refer to <a href="#">Table 35-16</a> for details regarding settings for receive FIFO watermark bits.</p> <p style="text-align: center;"><b>Table 35-16. Receive FIFO WaterMark Bit Description</b></p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Bits</th><th style="text-align: center;">Description</th></tr> </thead> <tbody> <tr> <td style="text-align: center;">0000</td><td>Reserved</td></tr> <tr> <td style="text-align: center;">0001</td><td>RFF set when at least one data word have been written to the Receive FIFO Set when RxFIFO = 1,2.....15 data words</td></tr> <tr> <td style="text-align: center;">0010</td><td>RFF set when more than or equal to 2 data word have been written to the Receive FIFO. Set when RxFIFO = 2,3.....15 data words</td></tr> <tr> <td style="text-align: center;">0011</td><td>RFF set when more than or equal to 3 data word have been written to the Receive FIFO. Set when RxFIFO = 3,4.....15 data words</td></tr> <tr> <td style="text-align: center;">0100</td><td>RFF set when more than or equal to 4 data word have been written to the Receive FIFO. Set when RxFIFO = 4,5.....15 data words</td></tr> <tr> <td style="text-align: center;">0101</td><td>RFF set when more than or equal to 5 data word have been written to the Receive FIFO. Set when RxFIFO = 5,6.....15 data words</td></tr> <tr> <td style="text-align: center;">0110</td><td>RFF set when more than or equal to 6 data word have been written to the Receive. Set when RxFIFO = 6,7.....15 data words</td></tr> <tr> <td style="text-align: center;">0111</td><td>RFF set when more than or equal to 7 data word have been written to the Receive FIFO. Set when RxFIFO = 7,8.....15 data words</td></tr> <tr> <td style="text-align: center;">1000</td><td>RFF set when more than or equal to 8 data word have been written to the Receive FIFO. Set when RxFIFO = 8,9.....15 data words</td></tr> <tr> <td style="text-align: center;">1001</td><td>RFF set when more than or equal to 9 data word have been written to the Receive FIFO. Set when RxFIFO = 9,10.....15 data words</td></tr> <tr> <td style="text-align: center;">1010</td><td>RFF set when more than or equal to 10 data word have been written to the Receive FIFO. Set when RxFIFO = 10,11.....15 data words</td></tr> <tr> <td style="text-align: center;">1011</td><td>RFF set when more than or equal to 11 data word have been written to the Receive FIFO. Set when RxFIFO = 11,12.....15 data words</td></tr> <tr> <td style="text-align: center;">1100</td><td>RFF set when more than or equal to 12 data word have been written to the Receive FIFO. Set when RxFIFO = 12,13.....15 data words</td></tr> <tr> <td style="text-align: center;">1101</td><td>RFF set when more than or equal to 13 data word have been written to the Receive FIFO. Set when RxFIFO = 13,14,15 data words</td></tr> <tr> <td style="text-align: center;">1110</td><td>RFF set when more than or equal to 14 data word have been written to the Receive FIFO. Set when RxFIFO = 14,15 data words</td></tr> <tr> <td style="text-align: center;">1111</td><td>RFF set when 15 data word have been written to the Receive FIFO (default). Set when RxFIFO = 15 data words</td></tr> </tbody> </table>	Bits	Description	0000	Reserved	0001	RFF set when at least one data word have been written to the Receive FIFO Set when RxFIFO = 1,2.....15 data words	0010	RFF set when more than or equal to 2 data word have been written to the Receive FIFO. Set when RxFIFO = 2,3.....15 data words	0011	RFF set when more than or equal to 3 data word have been written to the Receive FIFO. Set when RxFIFO = 3,4.....15 data words	0100	RFF set when more than or equal to 4 data word have been written to the Receive FIFO. Set when RxFIFO = 4,5.....15 data words	0101	RFF set when more than or equal to 5 data word have been written to the Receive FIFO. Set when RxFIFO = 5,6.....15 data words	0110	RFF set when more than or equal to 6 data word have been written to the Receive. Set when RxFIFO = 6,7.....15 data words	0111	RFF set when more than or equal to 7 data word have been written to the Receive FIFO. Set when RxFIFO = 7,8.....15 data words	1000	RFF set when more than or equal to 8 data word have been written to the Receive FIFO. Set when RxFIFO = 8,9.....15 data words	1001	RFF set when more than or equal to 9 data word have been written to the Receive FIFO. Set when RxFIFO = 9,10.....15 data words	1010	RFF set when more than or equal to 10 data word have been written to the Receive FIFO. Set when RxFIFO = 10,11.....15 data words	1011	RFF set when more than or equal to 11 data word have been written to the Receive FIFO. Set when RxFIFO = 11,12.....15 data words	1100	RFF set when more than or equal to 12 data word have been written to the Receive FIFO. Set when RxFIFO = 12,13.....15 data words	1101	RFF set when more than or equal to 13 data word have been written to the Receive FIFO. Set when RxFIFO = 13,14,15 data words	1110	RFF set when more than or equal to 14 data word have been written to the Receive FIFO. Set when RxFIFO = 14,15 data words	1111	RFF set when 15 data word have been written to the Receive FIFO (default). Set when RxFIFO = 15 data words
Bits	Description																																		
0000	Reserved																																		
0001	RFF set when at least one data word have been written to the Receive FIFO Set when RxFIFO = 1,2.....15 data words																																		
0010	RFF set when more than or equal to 2 data word have been written to the Receive FIFO. Set when RxFIFO = 2,3.....15 data words																																		
0011	RFF set when more than or equal to 3 data word have been written to the Receive FIFO. Set when RxFIFO = 3,4.....15 data words																																		
0100	RFF set when more than or equal to 4 data word have been written to the Receive FIFO. Set when RxFIFO = 4,5.....15 data words																																		
0101	RFF set when more than or equal to 5 data word have been written to the Receive FIFO. Set when RxFIFO = 5,6.....15 data words																																		
0110	RFF set when more than or equal to 6 data word have been written to the Receive. Set when RxFIFO = 6,7.....15 data words																																		
0111	RFF set when more than or equal to 7 data word have been written to the Receive FIFO. Set when RxFIFO = 7,8.....15 data words																																		
1000	RFF set when more than or equal to 8 data word have been written to the Receive FIFO. Set when RxFIFO = 8,9.....15 data words																																		
1001	RFF set when more than or equal to 9 data word have been written to the Receive FIFO. Set when RxFIFO = 9,10.....15 data words																																		
1010	RFF set when more than or equal to 10 data word have been written to the Receive FIFO. Set when RxFIFO = 10,11.....15 data words																																		
1011	RFF set when more than or equal to 11 data word have been written to the Receive FIFO. Set when RxFIFO = 11,12.....15 data words																																		
1100	RFF set when more than or equal to 12 data word have been written to the Receive FIFO. Set when RxFIFO = 12,13.....15 data words																																		
1101	RFF set when more than or equal to 13 data word have been written to the Receive FIFO. Set when RxFIFO = 13,14,15 data words																																		
1110	RFF set when more than or equal to 14 data word have been written to the Receive FIFO. Set when RxFIFO = 14,15 data words																																		
1111	RFF set when 15 data word have been written to the Receive FIFO (default). Set when RxFIFO = 15 data words																																		

**Table 35-13. SSI FIFO Control/Status Register Field Descriptions (continued)**

Field	Description																																		
19–16 TFWM1[3:0]	<p>Transmit FIFO Empty WaterMark 1. These bits control the threshold at which the TFE1 flag will be set. The TFE1 flag is set whenever the empty slots in Tx FIFO exceed or are equal to the selected threshold. Refer to <a href="#">Table 35-17</a> for details regarding settings for transmit FIFO watermark bits.</p> <p style="text-align: center;"><b>Table 35-17. Transmit FIFO WaterMark Bit Description</b></p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: center;">Bits</th><th style="text-align: center;">Description</th></tr> </thead> <tbody> <tr><td style="text-align: center;">0000</td><td>Reserved</td></tr> <tr><td style="text-align: center;">0001</td><td>TFE set when there are more than or equal to 1 empty slots in Transmit FIFO. (default) Transmit FIFO empty is set when TxFIFO &lt;= 14 data.</td></tr> <tr><td style="text-align: center;">0010</td><td>TFE set when there are more than or equal to 2 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO &lt;= 13 data.</td></tr> <tr><td style="text-align: center;">0011</td><td>TFE set when there are more than or equal to 3 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO &lt;= 12 data.</td></tr> <tr><td style="text-align: center;">0100</td><td>TFE set when there are more than or equal to 4 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO &lt;= 11 data.</td></tr> <tr><td style="text-align: center;">0101</td><td>TFE set when there are more than or equal to 5 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO &lt;= 10 data.</td></tr> <tr><td style="text-align: center;">0110</td><td>TFE set when there are more than or equal to 6 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO &lt;= 9 data.</td></tr> <tr><td style="text-align: center;">0111</td><td>TFE set when there are more than or equal to 7 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO &lt;= 8 data.</td></tr> <tr><td style="text-align: center;">1000</td><td>TFE set when there are more than or equal to 8 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO &lt;= 7 data.</td></tr> <tr><td style="text-align: center;">1001</td><td>TFE set when there are more than or equal to 9 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO &lt;= 6 data.</td></tr> <tr><td style="text-align: center;">1010</td><td>TFE set when there are more than or equal to 10 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO &lt;= 5 data.</td></tr> <tr><td style="text-align: center;">1011</td><td>TFE set when there are more than or equal to 11 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO &lt;= 4 data.</td></tr> <tr><td style="text-align: center;">1100</td><td>TFE set when there are more than or equal to 12 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO &lt;= 3 data.</td></tr> <tr><td style="text-align: center;">1101</td><td>TFE set when there are more than or equal to 13 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO &lt;= 2 data.</td></tr> <tr><td style="text-align: center;">1110</td><td>TFE set when there are more than or equal to 14 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO &lt;= 1 data.</td></tr> <tr><td style="text-align: center;">1111</td><td>TFE set when there are 15 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO = 0 data.</td></tr> </tbody> </table>	Bits	Description	0000	Reserved	0001	TFE set when there are more than or equal to 1 empty slots in Transmit FIFO. (default) Transmit FIFO empty is set when TxFIFO <= 14 data.	0010	TFE set when there are more than or equal to 2 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 13 data.	0011	TFE set when there are more than or equal to 3 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 12 data.	0100	TFE set when there are more than or equal to 4 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 11 data.	0101	TFE set when there are more than or equal to 5 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 10 data.	0110	TFE set when there are more than or equal to 6 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 9 data.	0111	TFE set when there are more than or equal to 7 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 8 data.	1000	TFE set when there are more than or equal to 8 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 7 data.	1001	TFE set when there are more than or equal to 9 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 6 data.	1010	TFE set when there are more than or equal to 10 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 5 data.	1011	TFE set when there are more than or equal to 11 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 4 data.	1100	TFE set when there are more than or equal to 12 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 3 data.	1101	TFE set when there are more than or equal to 13 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 2 data.	1110	TFE set when there are more than or equal to 14 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 1 data.	1111	TFE set when there are 15 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO = 0 data.
Bits	Description																																		
0000	Reserved																																		
0001	TFE set when there are more than or equal to 1 empty slots in Transmit FIFO. (default) Transmit FIFO empty is set when TxFIFO <= 14 data.																																		
0010	TFE set when there are more than or equal to 2 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 13 data.																																		
0011	TFE set when there are more than or equal to 3 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 12 data.																																		
0100	TFE set when there are more than or equal to 4 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 11 data.																																		
0101	TFE set when there are more than or equal to 5 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 10 data.																																		
0110	TFE set when there are more than or equal to 6 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 9 data.																																		
0111	TFE set when there are more than or equal to 7 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 8 data.																																		
1000	TFE set when there are more than or equal to 8 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 7 data.																																		
1001	TFE set when there are more than or equal to 9 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 6 data.																																		
1010	TFE set when there are more than or equal to 10 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 5 data.																																		
1011	TFE set when there are more than or equal to 11 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 4 data.																																		
1100	TFE set when there are more than or equal to 12 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 3 data.																																		
1101	TFE set when there are more than or equal to 13 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 2 data.																																		
1110	TFE set when there are more than or equal to 14 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO <= 1 data.																																		
1111	TFE set when there are 15 empty slots in Transmit FIFO. Transmit FIFO empty is set when TxFIFO = 0 data.																																		

**Table 35-13. SSI FIFO Control/Status Register Field Descriptions (continued)**

Field	Description
15–12 RFCNT0[3:0]	Receive FIFO Counter 0. These bits indicate the number of data words in Receive FIFO 0. Refer to <a href="#">Table 35-14</a> for details regarding settings for receive FIFO counter bits.
11–8 TFCNT0[3:0]	Transmit FIFO Counter 0. These bits indicate the number of data words in Transmit FIFO 0. Refer to <a href="#">Table 35-15</a> for details regarding settings for transmit FIFO counter bits.
7–4 RFWM0[3:0]	Receive FIFO Full WaterMark 0. These bits control the threshold at which the RFF0 flag will be set. The RFF0 flag is set whenever the data level in Rx FIFO 0 reaches the selected threshold. Refer to <a href="#">Table 35-16</a> for details regarding settings for receive FIFO watermark bits.
3–0 TFWM0[3:0]	Transmit FIFO Empty WaterMark 0. These bits control the threshold at which the TFE0 flag will be set. The TFE0 flag is set whenever the empty slots in Tx FIFO exceed or are equal to the selected threshold. Refer to <a href="#">Table 35-17</a> for details regarding settings for transmit FIFO watermark bits.

[Table 35-18](#) indicates the status of the Transmit FIFO Empty flag, with different settings of the Transmit FIFO WaterMark bits and varying amounts of data in the Tx FIFO.

**Table 35-18. Status of Transmit FIFO Empty Flag**

Transmit FIFO Watermark (TFWM)	Number of data in Tx-Fifo														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
3	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
4	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
5	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
6	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0
7	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
8	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
9	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
10	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
11	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
12	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
13	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
14	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

### 35.3.14 SSI AC97 Control Register (SSIn\_ACR)

**SSI<sub>n</sub>\_ACR** controls various features of the SSI operating in AC97 mode.

Address: 0xFC0B\_C038 (SSI0\_ACR)  
0xFC0C 8038 (SSI1 ACR)

Access: User read/write

**Figure 35-21. SSI AC97 Control Register (SSI<sub>n</sub>\_ACR)**

**Table 35-19. SSIn ACR Field Descriptions**

Field	Description
31–11	Reserved, must be cleared.
10–5 FRDIV	Frame rate divider. Controls the frequency of AC97 data transmission/reception. This field is programmed with the number of frames for which the SSI should be idle after operating in one frame. Through these bits, the AC97 frequency of operation, from 48 KHz (000000) to 1 KHz (101111) can be achieved. E.g: 001010 (10 Decimal) equals SSI operates once every 11 frames.
4 WR	Write command. Specifies whether the next frame carries an AC97 write command or not. When this bit is set, the corresponding tag bits (corresponding to command address and command data slots of the next transmit frame) are automatically set. The SSI automatically clears this bit after completing transmission of a frame. 0 Next frame does not have a write command 1 Next frame does have a write command <b>Note:</b> Do not set WR and RD at the same time.
3 RD	Read command. Specifies whether the next frame carries an AC97 read command or not. When this bit is set, the corresponding tag bit (corresponding to command address slot of the next transmit frame) is automatically set. The SSI automatically clears this bit after completing transmission of a frame. 0 Next frame does not have a read command 1 Next frame does have a read command <b>Note:</b> Do not set WR and RD at the same time.
2 TIF	Tag in FIFO. Controls the destination of the information received in the AC97 tag slot (slot #0). 0 Tag information stored in SSIn_ATAG register 1 Tag information stored in Rx FIFO 0
1 FV	Fixed/variable operation. 0 AC97 fixed mode 1 AC97 variable mode
0 AC97EN	AC97 mode enable. Refer to <a href="#">Section 35.4.1.5, “AC97 Mode,”</a> for details of AC97 operation. 0 AC97 mode disabled 1 AC97 mode enabled

### 35.3.15 SSI AC97 Command Address Register (SSIn\_ACADD)

SSIn\_ACADD contains the command address slot information.

Address:	0xFC0B_C03C (SSI0_ACADD) 0xFC0C_803C (SSI1_ACADD)	Access:	User read/write
ACADD			
R	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 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		
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 35-22. SSI AC97 Command Address Register (SSIn\_ACADD)

Table 35-20. SSIn\_ACADD Field Descriptions

Field	Description
31–19	Reserved, must be cleared.
18–0 ACADD	AC97 command address. Stores the command address slot information (bit 19 of the slot is sent in accordance with the SSIn_ACR[WR and RD] bits). A direct write from the core or the information received in the incoming command address slot can update these bits. If contents of these bits change due to an update, the SSIn_ISR[CMDAU] bit is set.

### 35.3.16 SSI AC97 Command Data Register (SSIn\_ACDAT)

SSIn\_ACDAT contains the outgoing command data slot.

Address:	0xFC0B_C040 (SSI0_ACDAT) 0xFC0C_8040 (SSI1_ACDAT)	Access:	User read/write
ACDAT			
R	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 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		
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 35-23. SSI AC97 Command Data Register (SSIn\_ACDAT)

Table 35-21. SSIn\_ACDAT Field Descriptions

Field	Description
31–20	Reserved, must be cleared.
19–0 ACDAT	AC97 command data. The outgoing command data slot carries the information contained in these bits. A direct write from the core or the information received in the incoming command data slot can update these bits. If the contents of these bits change due to an update, the SSIn_ISR[CMDDU] bit is set. During an AC97 read command, 0x0_0000 in time slot #2.

### 35.3.17 SSI AC97 Tag Register (SSIn\_ATAG)

**Figure 35-24. SSI AC97 Tag Register (SSIn\_ATAG)**

**Table 35-22. SSIn\_ATAG Field Descriptions**

Field	Description
31–16	Reserved, must be cleared.
15–0 ATAG	AC97 tag. Writing to this register sets the value of the Tx tag (in AC97 fixed mode). On a read, the processor gets the last Rx tag value received. It is updated at the start of each received frame. The contents of this register also generate the transmit tag in AC97 variable mode. When the received tag value changes, the SSIn_ISR[RXT] bit is set, if enabled. If the SSIn_ACR[TIF] bit is set, the TAG value is also stored in Rx FIFO. <b>Note:</b> Bits 1–0 convey the codec-ID. Because only primary codecs are supported, these bits must be cleared.

### 35.3.18 SSI Transmit Time Slot Mask Register (SSI<sub>n</sub>\_TMASK)

This register controls the time slots that the SSI transmits data in network mode.

Address: 0xFC0B_C048 (SSI0_TMASK)																Access: User read/write															
0xFC0C_8048 (SSI1_TMASK)																															

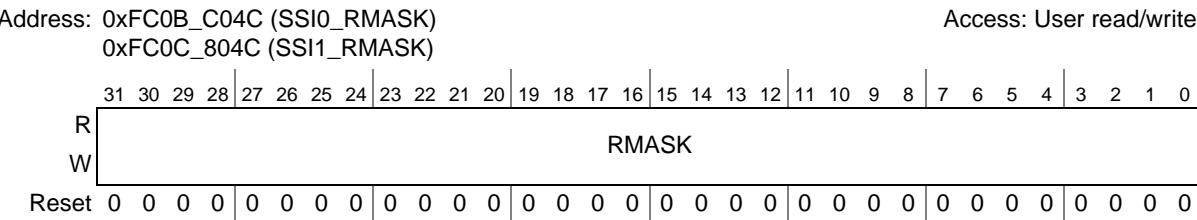
**Figure 35-25. SSI Transmit Time Slot Mask Register (SSI<sub>n</sub>\_TMASK)**

**Table 35-23. SSI<sub>n</sub> TMASK Field Descriptions**

Field	Description				
31–0 TMASK	<p>Transmit mask. Indicates which transmit time slot has been masked in the current frame. Each bit corresponds to the respective time slot in the frame. If a change is made to the register contents, the transmission pattern is updated from the next time slot. Transmit mask bits should not be used in I<sup>2</sup>S slave mode.</p> <table border="0" data-bbox="321 1408 1356 1421"> <tr> <td data-bbox="321 1408 362 1415">0</td> <td data-bbox="362 1408 566 1415">Valid time slot</td> </tr> <tr> <td data-bbox="321 1415 362 1421">1</td> <td data-bbox="362 1415 566 1421">Time slot masked (no data transmitted in this time slot)</td> </tr> </table>	0	Valid time slot	1	Time slot masked (no data transmitted in this time slot)
0	Valid time slot				
1	Time slot masked (no data transmitted in this time slot)				

### 35.3.19 SSI Receive Time Slot Mask Register (SSI<sub>n</sub>\_RMASK)

This register controls the time slots that the SSI receives data in network mode.



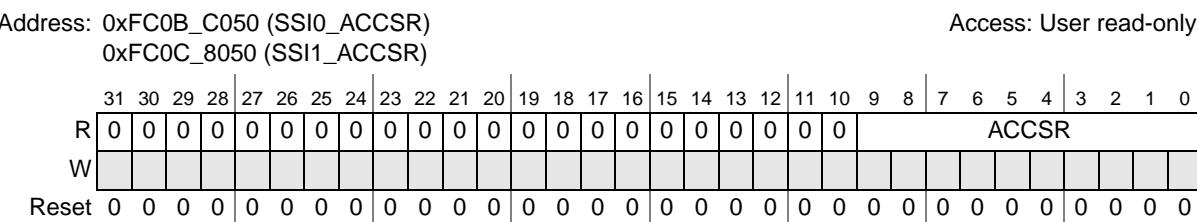
**Figure 35-26. SSI Receive Time Slot Mask Register (SSI<sub>n</sub>\_RMASK)**

**Table 35-24. SSI<sub>n</sub> RMASK Field Descriptions**

Field	Description
31–0 RMASK	<p>Receive mask. Indicates which received time slot has been masked in the current frame. Each bit corresponds to the respective time slot in the frame. If a change is made to the register contents, the reception pattern is updated from the next time slot. Receive mask bits should not be used in I<sup>2</sup>S slave mode.</p> <ul style="list-style-type: none"> <li>0 Valid time slot</li> <li>1 Time slot masked (no data received in this time slot)</li> </ul>

### 35.3.20 SSI AC97 Channel Status Register (SSI<sub>n</sub>\_ACCSR)

**SSIn\_ACCSR** indicates which data slot is enabled in AC97 variable mode operation.



**Figure 35-27. SSI AC97 Channel Status Register (SSIn ACCSR)**

**Table 35-25. SSI<sub>n</sub> ACCSR Field Descriptions**

Field	Description
31–10	Reserved, must be cleared.
9–0 ACCSR	AC97 channel status. Indicates which data slot is enabled in AC97 variable mode operation. This register is updated if the core enables or disables a channel through a write to SSIn_ACCEEN or SSIn_ACCDIS or the external codec enables a channel by sending a 1 in the corresponding SLOTREQ bit. Bit 0 corresponds to the first data slot in an AC97 frame (slot #3) and bit 9 corresponds to the tenth data slot (slot #12). Writes to this register result in an error response. 0 Data channel disabled 1 Data channel enabled

### 35.3.21 SSI AC97 Channel Enable Register (SSIn\_ACCEN)

**SSI<sub>n</sub>\_ACCEN** enables data slots in AC97 variable mode operation.

**Figure 35-28. SSI AC97 Channel Enable Register (SSIn\_ACCEN)**

**Table 35-26. SSIn\_ACCEN Field Descriptions**

Field	Description
31–10	Reserved, must be cleared.
9–0 ACCEN	AC97 channel enable. Enables a data channel in AC97 variable mode. Writing a zero has no effect. Bit 0 corresponds to the first data slot in an AC97 frame (slot #3) and bit 9 corresponds to the tenth data slot (slot #12). These bits always read as zero. 0 No effect 1 Enables the corresponding data channel

### 35.3.22 SSI AC97 Channel Disable Register (SSI<sub>n</sub>\_ACCDIS)

**SSIIn\_ACCDIS** disables data slots in AC97 variable mode operation.

**Figure 35-29. SSI AC97 Channel Disable Register (SSIn\_ACCDIS)**

**Table 35-27. SSIn\_ACCDIS Field Descriptions**

Field	Description
31–10	Reserved, must be cleared.
9–0 ACCDIS	AC97 channel disable. Disables a data channel in AC97 variable mode. Writing a zero has no effect. Bit 0 corresponds to the first data slot in an AC97 frame (slot #3) and bit 9 corresponds to the tenth data slot (slot #12). These bits always read as zero. 0 No effect 1 Disables the corresponding data channel

## 35.4 Functional Description

### 35.4.1 Detailed Operating Mode Descriptions

The following sections describe in detail the main operating modes of the SSI module: normal, network, gated clock, I<sup>2</sup>S, and AC97.

#### 35.4.1.1 Normal Mode

Normal mode is the simplest mode of the SSI. It transfers data in one time slot per frame. A time slot is a unit of data and the WL bits define the number of bits in a time slot. In continuous clock mode, a frame sync occurs at the beginning of each frame. The following factors determine the length of the frame:

- Period of the serial bit clock (DIV2, PSR, PM bits for internal clock or the frequency of the external clock on the SSIn\_BCLK port)
- Number of bits per time slot (WL bits)
- Number of time slots per frame (DC bits)

If normal mode is configured with more than one time slot per frame, data transfers only in the first time slot of the frame. No data transfers in subsequent time slots. In normal mode, DC values corresponding to more than a single time slot in a frame only result in lengthening the frame.

##### 35.4.1.1.1 Normal Mode Transmit

Conditions for data transmission from the SSI in normal mode are:

1. SSI enabled (SSIn\_CR[SSI\_EN] = 1)
2. Enable FIFO and configure transmit and receive watermark if the FIFO is used.
3. Write data to transmit data register (SSIn\_TX0)
4. Transmitter enabled (TE = 1)
5. Frame sync active (for continuous clock case)
6. Bit clock begins (for gated clock case)

When the above conditions occur in normal mode, the next data word transfers into the transmit shift register (TXSR) from the transmit data register 0 (SSIn\_TX0) or from the transmit FIFO 0 register, if enabled. The new data word transmits immediately.

If transmit FIFO 0 is not enabled and the transmit data register empty (TDE0) bit is set, a transmit interrupt 0 occurs if the TIE and SSIn\_IER[TDE0] bits are set.

If transmit FIFO 0 is enabled and the transmit FIFO empty (TFE0) bit is set, transmit interrupt 0 occurs if the TIE and SSIn\_IER[TFE0] bits are set. If transmit FIFO 0 is enabled and filled with data, eight data words can be transferred before the core must write new data to the SSIn\_TX0 register.

The SSIn\_TXD port is disabled except during the data transmission period. For a continuous clock, the optional frame sync output and clock outputs are not disabled, even if receiver and transmitter are disabled.

### 35.4.1.1.2 Normal Mode Receive

Conditions for data reception from the SSI are:

1. SSI enabled (*SSIn\_CR[SSI\_EN]* = 1)
2. Enable receive FIFO (optional)
3. Receiver enabled (*RE* = 1)
4. Frame sync active (for continuous clock case)
5. Bit clock begins (for gated clock case)

With the above conditions in normal mode with a continuous clock, each time the frame sync signal is generated (or detected), a data word is clocked in. With the above conditions and a gated clock, each time the clock begins, a data word is clocked in.

If receive FIFO 0 is not enabled, the received data word is transferred from the receive shift register (RXSR) to the receive data register 0 (*SSIn\_RX0*), and the RDR0 flag is set. Receive interrupt 0 occurs if the RIE and *SSIn\_IER[RDR0]* bits are set.

If receive FIFO 0 is enabled, the received data word is transferred to the receive FIFO 0. The RFF0 flag is set if the receive data register (*SSIn\_RX0*) is full and receive FIFO 0 reaches the selected threshold. Receive interrupt 0 occurs if RIE and *SSIn\_IER[RFF0]* bits are set.

The core has to read the data from the *SSIn\_RX0* register before a new data word is transferred from the RXSR; otherwise, receive overrun error 0 (ROE0) bit is set. If receive FIFO 0 is enabled, the ROE0 bit is set when the receive FIFO 0 data level reaches the selected threshold and a new data word is ready to transfer to the receive FIFO 0.

**Figure 35-30** shows transmitter and receiver timing for an 8-bit word with two words per time slot in normal mode and continuous clock with a late word length frame sync. The Tx data register is loaded with the data to be transmitted. On arrival of the frame sync, this data is transferred to the transmit shift register and transmitted on the *SSIn\_TXD* output. Simultaneously, the receive shift register shifts in the received data available on the *SSIn\_RXD* input. At the end of the time slot, this data is transferred to the Rx data register.

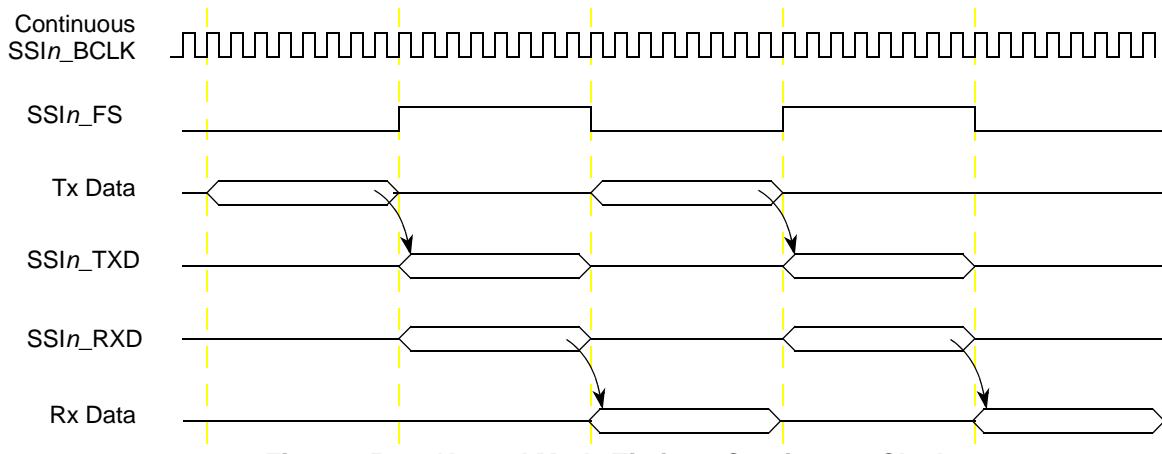


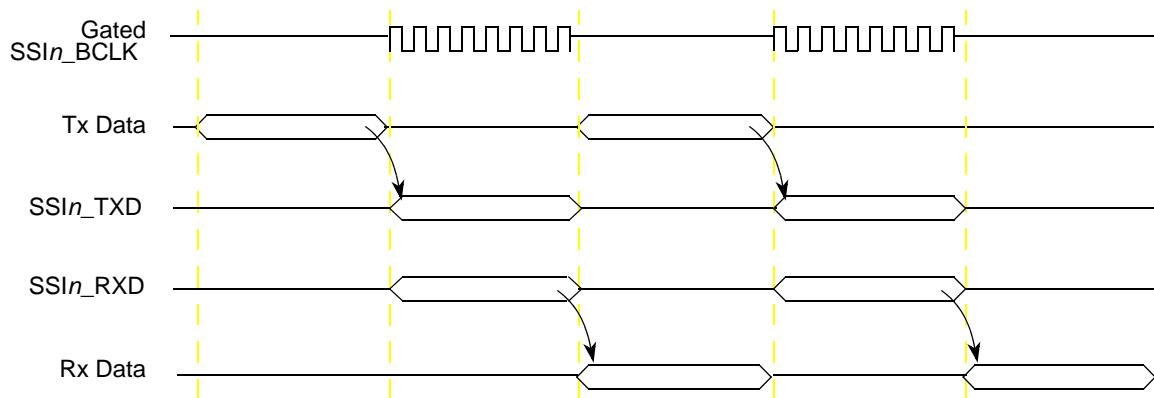
Figure 35-30. Normal Mode Timing - Continuous Clock

Figure 35-31 shows a similar case for internal (SSI generates clock) gated clock mode, and Figure 35-32 shows a case for external (SSI receives clock) gated clock mode.

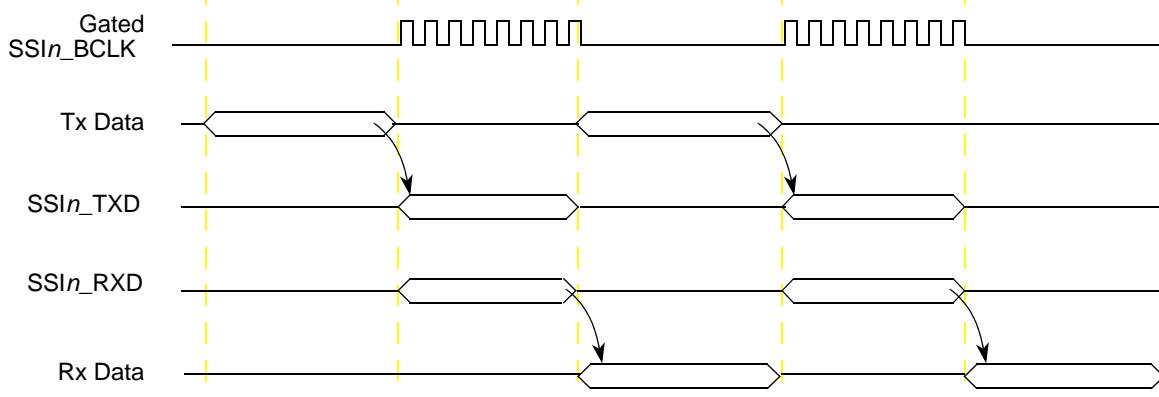
#### NOTE

A pull-down resistor is required in gated clock mode, because the clock port is disabled between transmissions.

The Tx data register is loaded with the data to be transmitted. On arrival of the clock, this data transfers to the transmit shift register and transmits on the `SSIn_TXD` output. Simultaneously, the receive shift register shifts in the received data available on the `SSIn_RXD` input, and at the end of the time slot, this data transfers to the Rx data register. In internal gated clock mode, the Tx data line and clock output port are tri-stated at the end of transmission of the last bit (at the completion of the complete clock cycle). Whereas, in external gated clock mode, the Tx data line is tri-stated at the last inactive edge of the incoming bit clock (during the last bit in a data word).



**Figure 35-31. Normal Mode Timing - Internal Gated Clock**



**Figure 35-32. Normal Mode Timing - External Gated Clock**

#### 35.4.1.2 Network Mode

Network mode creates a time division multiplexed (TDM) network, such as a TDM codec network or a network of DSPs. In continuous clock mode, a frame sync occurs at the beginning of each frame. In this mode, the frame is divided into more than one time slot. During each time slot, one data word can be transferred (rather than in the frame sync time slot as in normal mode). Each time slot is then assigned to

an appropriate codec or DSP on the network. The processor can be a master device that controls its own private network or a slave device connected to an existing TDM network and occupies a few time slots.

The frame rate dividers, controlled by the DC bits, select two to thirty-two time slots per frame. The length of the frame is determined by:

- The period of the serial bit clock (PSR, PM bits for internal clock, or the frequency of the external clock on the SSIn\_BCLK pin)
- The number of bits per sample (WL bits)
- The number of time slots per frame (DC bits)

In network mode, data can be transmitted in any time slot. The distinction of network mode is each time slot is identified with respect to the frame sync (data word time). This time slot identification allows the option of transmitting data during the time slot by writing to the SSIn\_TX registers or ignoring the time slot as determined by the SSIn\_TMASK register bits. The receiver is treated in the same manner and received data is only transferred to the receive data register/FIFO if the corresponding time slot is enabled through SSIn\_RMASK.

By using the SSIn\_TMASK and SSIn\_RMASK registers, software only has to service the SSI during valid time slots. This eliminates any overhead associated with unused time slots. Refer to [Section 35.3.18, “SSI Transmit Time Slot Mask Register \(SSIn\\_TMASK\),”](#) and [Section 35.3.19, “SSI Receive Time Slot Mask Register \(SSIn\\_RMASK\),”](#) for more information on the SSIn\_TMASK and SSIn\_RMASK registers.

In two channel mode (SSIn\_CR[TCH] = 1), the second set of transmit and receive FIFOs and data registers create two separate channels (for example, left and right channels for a stereo codec). These channels are completely independent with their own set of interrupts and DMA requests identical to the ones available for the default channel. In this mode, data is transmitted/received in enabled time slots alternately from/to FIFO 0 and FIFO 1, starting from FIFO 0. The first data word is taken from FIFO 0 and transmitted in the first enabled time slot and subsequently, data is loaded from FIFO 1 and FIFO 0 alternately and transmitted. Similarly, the first received data is sent to FIFO 0 and subsequent data is sent to FIFO 1 and FIFO 0 alternately. Time slots are selected through the transmit and receive time slot mask registers (SSIn\_TMASK and SSIn\_RMASK).

### 35.4.1.2.1 Network Mode Transmit

The transmit portion of SSI is enabled when the SSIn\_CR[SSI\_EN and TE] bits are set. However, for continuous clock when the TE bit is set, the transmitter is enabled only after detection of a new frame sync (transmission starts from the next frame boundary).

Normal start-up sequence for transmission:

- Write the data to be transmitted to the SSIn\_TX register. This clears the TDE flag.
- Set the SSIn\_CR[TE] bit to enable the transmitter on the next word boundary (for continuous clock).
- Enable transmit interrupts.

Alternately, the user may decide not to transmit in a time slot by writing to the SSIn\_TMASK. The TDE flag is not cleared, but the SSIn\_TXD port remains disabled during the time slot. When the frame sync is detected or generated (continuous clock), the first enabled data word is transferred from the SSIn\_TX

register to the TXSR and is shifted out (transmitted). When the *SSIn\_TX* register is empty, the TDE bit is set, which causes a transmitter interrupt (if the FIFO is disabled) to be sent if the TIE bit is set. Software can poll the TDE bit or use interrupts to reload the *SSIn\_TX* register with new data for the next time slot. Failing to reload the *SSIn\_TX* register before the TXSR is finished shifting (empty) causes a transmitter underrun error (the TUE bit is set). If the FIFO is enabled, the TFE flag is set in accordance with the watermark setting and this flag causes a transmitter interrupt to occur.

Clearing the TE bit disables the transmitter after completion of transmission of the current frame. Setting the TE bit enables transmission from the next frame. During that time the *SSIn\_TXD* port is disabled. The TE bit should be cleared after the TDE bit is set to ensure that all pending data is transmitted.

To summarize, the network mode transmitter generates interrupts every enabled time slot and requires the processor to respond to each enabled time slot. These responses may be:

- Write data in data register to enable transmission in the next time slot.
- Configure the time slot register to disable transmission in the next time slot (unless the time slot is already masked by the *SSIn\_TMASK* register bit).
- Do nothing—transmit underrun occurs at the beginning of the next time slot and the previous data is re-transmitted.

In two channel operation, both channels (data registers, FIFOs, interrupts, and DMA requests) operate in the same manner, as described above. The only difference is interrupts related to the second channel are generated only if this mode of operation is selected (TDE1 is low by default).

#### **35.4.1.2.2 Network Mode Receive**

The receiver portion of the SSI is enabled when both the *SSIn\_CR[SSI\_EN* and RE] bits are set. However, the receive enable only takes place during that time slot if RE is enabled before the second to last bit of the word. If the RE bit is cleared, the receiver is disabled at the end of the current frame. The SSI module is capable of finding the start of the next frame automatically. When the word is completely received, it is transferred to the *SSIn\_RX* register, which sets the RDR bit. This causes a receive interrupt to occur if the the RIE bit is set. The second data word (second time slot in the frame) begins shifting in immediately after the transfer of the first data word to the *SSIn\_RX* register. The processor has to read the data from the receive data register (which clears RDR) before the second data word is completely received (ready to transfer to RX data register) or a receive overrun error occurs (the ROE bit is set).

An interrupt can occur after the reception of each enabled data word or the user can poll the RDR flag. The processor response can be:

- Read RX and use the data.
- Read RX and ignore the data.
- Do nothing—the receiver overrun exception occurs at the end of the current time slot.

**NOTE**

For a continuous clock, the optional frame sync output and clock output signals are not affected, even if transmitter or receiver is disabled. TE and RE do not disable the bit clock or the frame sync generation. To disable the bit clock and the frame sync generation, the *SSIn\_CR[SSI\_EN]* bit can be cleared or the port control logic external to the SSI (e.g. GPIO) can be reconfigured.

In two channel operation, both the channels (data registers, FIFOs, interrupts, and DMA requests) operate in the same manner as described above. The only difference is second channel interrupts are generated only in this mode of operation.

[Figure 35-33](#) shows the transmitter and receiver timing for an 8-bit word with continuous clock, FIFO disabled, three words per frame sync in network mode.

**NOTE**

The transmitter repeats the value 0x5E because of an underrun condition.

For the transmit section, the *SSIn\_TMASK* value is updated in the last time slot of frame 1 to mask the first two time slots (0x3). This value takes effect at the next time slot and, consequently, the next frame transmits data in the third time slot only.

For the receive section, data received on the *SSIn\_RXD* pin is transferred to the *SSIn\_RX* register at the end of each time slot. If the FIFO is disabled, RDR flag sets and causes a receiver interrupt if the RE, RIE, and *SSIn\_IER[RDR]* bits are set. If the FIFO is enabled, the RFF flag generates interrupts (this flag is set in accordance with the watermark settings). In this example all time slots are enabled. The receive data ready flag is set after reception of the first data (0x55). Because the flag is not cleared (Rx data register is not read), the receive overrun error (ROE) flag is set on reception of the next data (0x5E). The ROE flag is cleared by writing one to the corresponding flag in the *SSIn\_ISR*.

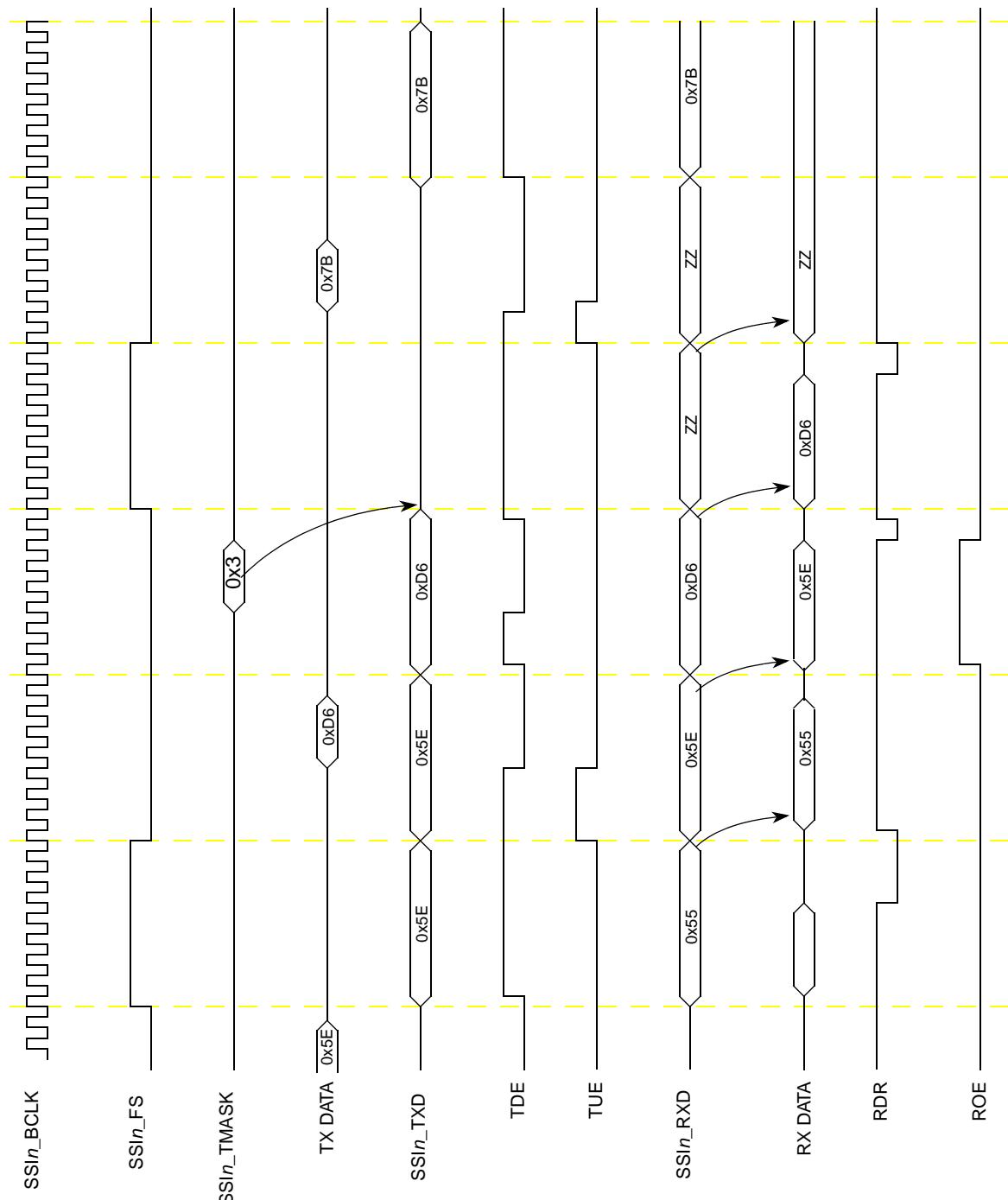


Figure 35-33. Network Mode Timing - Continuous Clock

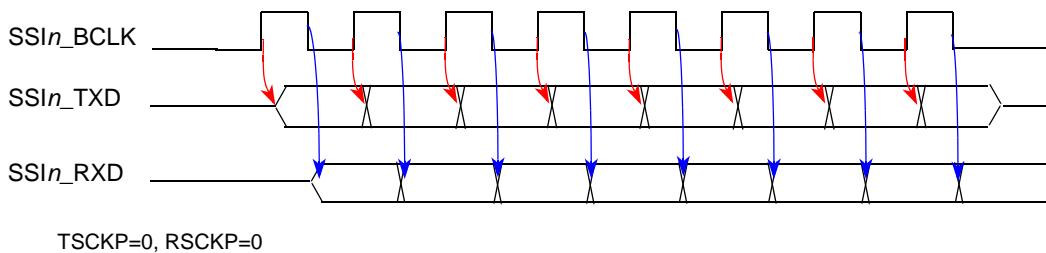
### 35.4.1.3 Gated Clock Mode

Gated clock mode often connects to SPI-type interfaces on microcontroller units (MCUs) or external peripheral devices. In gated clock mode, presence of the clock indicates that valid data is on the SSIn\_TXD or SSIn\_RXD signals. For this reason, no frame sync is needed in this mode. After transmission of data completes, the clock is pulled to the inactive state. Gated clocks are allowed for the transmit and receive

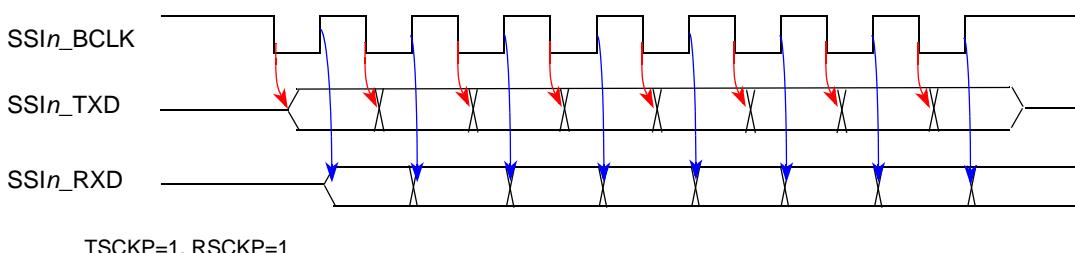
sections with internal or external clock and in normal mode. Gated clocks are not allowed in network mode. Refer to [Table 35-3](#) for SSI configuration for gated mode operation.

The clock operates when the TE bit and/or the RE bit are appropriately enabled. For an internally generated clock, all internal bit clocks, word clocks, and frame clocks continue to operate. When a valid time slot occurs (such as the first time slot in normal mode), the internal bit clock is enabled onto the clock port. This allows data to be transferred out in periodic intervals in gated clock mode. With an external clock, the SSI module waits for a clock signal to be received. After the clock begins, valid data is shifted in. Care should be taken to clear all DC bits when the module is used in gated mode.

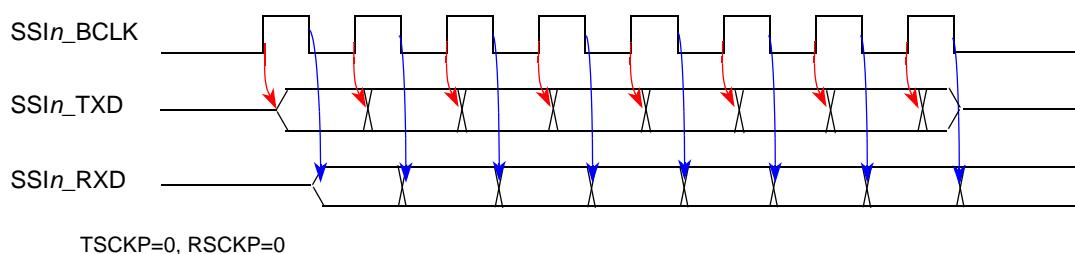
For gated clock operated in external clock mode, proper clock signalling must apply to SSIn\_BCLK for it to function properly. If the SSI uses rising edge transition to clock data (TSCKP = 0) and falling edge transition to latch data (RSCKP = 0), the clock must be in an active low state when idle. If the SSI uses falling edge transition to clock data (TSCKP = 1) and rising edge transition to latch data (RSCKP = 1), the clock must be in an active high state when idle. The following diagrams illustrate the different edge clocking/latching.



**Figure 35-34. Internal Gated Mode Timing - Rising Edge Clocking/Falling Edge Latching**



**Figure 35-35. Internal Gated Mode Timing - Falling Edge Clocking/Rising Edge Latching**



**Figure 35-36. External Gated Mode Timing - Rising Edge Clocking/Falling Edge Latching**

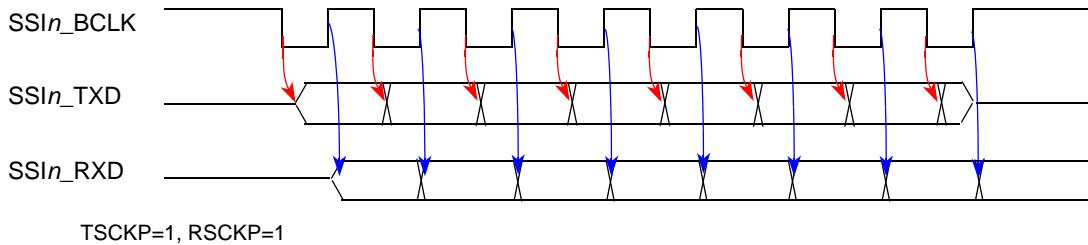


Figure 35-37. External Gated Mode Timing - Falling Edge Clocking/Rising Edge Latching

**NOTE**

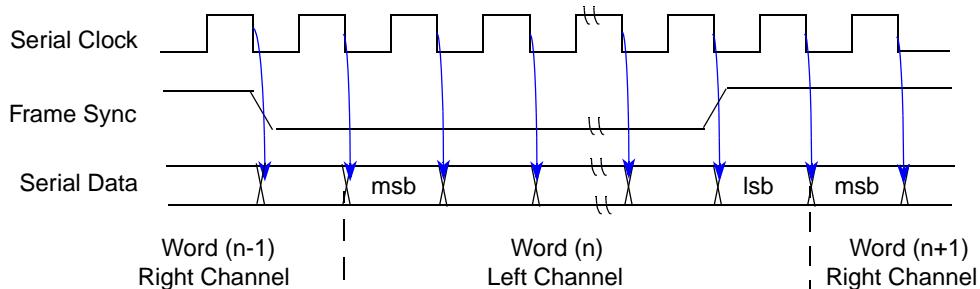
The bit clock signals must not have timing glitches. If a single glitch occurs, all ensuing transfers are out of synchronization.

**NOTE**

In external gated mode, even though the transmit data line is tri-stated at the last non-active edge of the bit clock, the round trip delay should sufficiently take care of hold time requirements at the external receiver.

### 35.4.1.4 I<sup>2</sup>S Mode

The SSI is compliant to I<sup>2</sup>S bus specification from Philips Semiconductors (February 1986, Revised June 5, 1996). [Figure 35-38](#) depicts basic I<sup>2</sup>S protocol timing.

Figure 35-38. I<sup>2</sup>S Mode Timing - Serial Clock, Frame Sync and Serial Data

I<sup>2</sup>S mode can be selected by the SSIn\_CR[I2S] bits as follows:

Table 35-28. I<sup>2</sup>S Mode Selection

SSIn_CR[I2S]	Mode
00	Normal mode
01	I <sup>2</sup> S master mode
10	I <sup>2</sup> S slave mode
11	Normal mode

In normal (non-I<sup>2</sup>S) mode operation, no register bits are forced to any particular state internally, and the user can program the SSI to work in any operating condition.

When I<sup>2</sup>S modes are entered ( $SSIn\_CR[I2S] = 01$  or  $10$ ), these settings are recommended:

- Synchronous mode ( $SSIn\_CR[SYN] = 1$ )
- Tx shift direction: msb transmitted first ( $SSIn\_TCR[TSHFD] = 0$ )
- Rx shift direction: msb received first ( $SSIn\_RCR[RSHFD] = 0$ )
- Tx data clocked at falling edge of the clock ( $SSIn\_TCR[TSCKP] = 1$ )
- Rx data latched at rising edge of the clock ( $SSIn\_RCR[RSCKP] = 1$ )
- Tx frame sync active low ( $SSIn\_TCR[TFSI] = 1$ )
- Rx frame sync active low ( $SSIn\_RCR[RFSI] = 1$ )
- Tx frame sync initiated one bit before data is transmitted ( $SSIn\_TCR[TEFS] = 1$ )
- Rx frame sync initiated one bit before data is received ( $SSIn\_RCR[REFS] = 1$ )

#### **35.4.1.4.1 I<sup>2</sup>S Master Mode**

In I<sup>2</sup>S master mode ( $SSIn\_CR[I2S] = 01$ ), these additional settings are recommended:

- Internal generated bit clock ( $SSIn\_TCR[TXDIR] = 1$ )
- Internal generated frame sync ( $SSIn\_TCR[TFDIR] = 1$ )

The processor automatically performs these settings when in I<sup>2</sup>S master mode:

- Network mode is selected ( $SSIn\_CR[NET] = 1$ )
- Tx frame sync length set to one-word-long-frame ( $SSIn\_TCR[TFSL] = 0$ )
- Rx frame sync length set to one-word-long-frame ( $SSIn\_RCR[RFSL] = 0$ )
- Tx shifting w.r.t. bit 0 of TXSR ( $SSIn\_TCR[TXBIT0] = 1$ )
- Rx shifting w.r.t. bit 0 of RXSR ( $SSIn\_RCR[RXBIT0] = 1$ )

Set the  $SSIn\_CCR[PM, PSR, DIV2, WL, DC]$  control bits to configure the bit clock and frame sync.

The word length is fixed to 32 in I<sup>2</sup>S master mode, and the WL bits determine the number of bits that contain valid data (out of the 32 transmitted/received bits in each channel). The fixing of word duration as 32 simplifies the relation between oversampling clock ( $SSIn\_MCLK$ ) and the frame sync ( $SSIn\_MCLK$  becomes an integer multiple of frame sync). The period of the oversampling clock must be at least 4x the internal bus clock period.

#### **35.4.1.4.2 I<sup>2</sup>S Slave Mode**

In I<sup>2</sup>S slave mode ( $SSIn\_CR[I2S] = 10$ ), the following additional settings are recommended:

- External generated bit clock ( $SSIn\_TCR[TXDIR] = 0$ )
- External generated frame sync ( $SSIn\_TCR[TFDIR] = 0$ )

The following settings are done automatically by the processor when in I<sup>2</sup>S slave mode:

- Normal mode is selected ( $\text{SSI}_n\text{-CR[NET]} = 0$ )
- Tx frame sync length set to one-bit-long-frame ( $\text{SSI}_n\text{-TCR[TFSL]} = 1$ )
- Rx frame sync length set to one-bit-long-frame ( $\text{SSI}_n\text{-RCR[RFSL]} = 1$ )
- Tx shifting w.r.t. bit 0 of TXSR ( $\text{SSI}_n\text{-TCR[TXBIT0]} = 1$ )
- Rx shifting w.r.t. bit 0 of RXSR ( $\text{SSI}_n\text{-RCR[RXBIT0]} = 1$ )

Set the  $\text{SSI}_n\text{-CCR[WL, DC]}$  bits to configure the data transmission.

The word length is variable in I<sup>2</sup>S slave mode and the WL bits determine the number of bits that contain valid data. The actual word length is determined by the external codec. The external I<sup>2</sup>S master sends a frame sync according to the I<sup>2</sup>S protocol (early, word wide, and active low). The SSI internally operates so each frame sync transition is the start of a new frame (the WL bits determine the number of bits to be transmitted/received). After one data word has been transferred, the SSI waits for the next frame sync transition to start operation in the next time slot. Transmit and receive mask bits should not be used in I<sup>2</sup>S slave mode.

### 35.4.1.5 AC97 Mode

In AC97 mode, SSI transmits a 16-bit tag slot at the start of a frame and the rest of the slots (in that frame) are all 20-bits wide. The same sequence is followed while receiving data. Refer to the AC97 specification for details regarding transmit and receive sequences and data formats.

#### NOTE

Since the SSI has only one RxDATA pin, only one codec is supported.  
Secondary codecs are not supported.

When AC97 mode is enabled, the hardware internally overrides the following settings. The programmed register values are not changed by entering AC97 mode, but they no longer apply to the module's operation. Writing to the programmed register fields updates their values. These updates can be seen by reading back the register fields. However, these settings do not take effect until AC97 mode is turned off.

The register bits within the bracket are equivalent settings.

- Synchronous mode is entered ( $\text{SSI}_n\text{-CR[SYN]} = 1$ )
- Network mode is selected ( $\text{SSI}_n\text{-CR[NET]} = 1$ )
- Tx shift direction is msb transmitted first ( $\text{SSI}_n\text{-TCR[TSHFD]} = 0$ )
- Rx shift direction is msb received first ( $\text{SSI}_n\text{-RCR[RSHFD]} = 0$ )
- Tx data is clocked at rising edge of the clock ( $\text{SSI}_n\text{-TCR[TSCKP]} = 0$ )
- Rx data is latched at falling edge of the clock ( $\text{SSI}_n\text{-RCR[RSCKP]} = 0$ )
- Tx frame sync is active high ( $\text{SSI}_n\text{-TCR[TFSI]} = 0$ )
- Rx frame sync is active high ( $\text{SSI}_n\text{-RCR[RFSI]} = 0$ )
- Tx frame sync length is one-word-long-frame ( $\text{SSI}_n\text{-TCR[TFSL]} = 0$ )
- Rx frame sync length is one-word-long-frame ( $\text{SSI}_n\text{-RCR[RFSL]} = 0$ )
- Tx frame sync initiated one bit before data is transmitted ( $\text{SSI}_n\text{-TCR[TEFS]} = 1$ )

- Rx frame sync initiated one bit before data is received ( $\text{SSI}_{In\_RCR}[\text{REFS}] = 1$ )
- Tx shifting w.r.t. bit 0 of TXSR ( $\text{SSI}_{In\_TCR}[\text{TXBIT0}] = 1$ )
- Rx shifting w.r.t. bit 0 of RXSR ( $\text{SSI}_{In\_RCR}[\text{RXBIT0}] = 1$ )
- Tx FIFO is enabled ( $\text{SSI}_{In\_TCR}[\text{TFEN0}] = 1$ )
- Rx FIFO is enabled ( $\text{SSI}_{In\_RCR}[\text{RFEN0}] = 1$ )
- Internally-generated frame sync ( $\text{SSI}_{In\_TCR}[\text{TFDIR}] = 1$ )
- Externally-generated bit clock ( $\text{SSI}_{In\_TCR}[\text{TXDIR}] = 0$ )

Any alteration of these bits does not affect the operational conditions of the SSI unless AC97 mode is deselected. Hence, the only control bits that need to be set to configure the data transmission/reception are the  $\text{SSI}_{In\_CCR}[\text{WL}, \text{DC}]$  bits. In AC97 mode, the WL bits can only legally take the values corresponding to 16-bit (truncated data) or 20-bit time slots. If the WL bits are set to select 16-bit time slots, while receiving, the SSI pads the data (four least significant bits) with 0s, and while receiving, the SSI stores only the 16 most significant bits in the Rx FIFO.

The following sequence should be followed for programming the SSI to work in AC97 mode:

1. Program the  $\text{SSI}_{In\_CCR}[\text{WL}]$  bits to a value corresponding to 16 or 20 bits. The WL bit setting is only for the data portion of the AC97 frame (slots #3 through #12). The tag slot (slot #0) is always 16-bits wide and the command address and command data slots (slots #1 and #2) are always 20 bits wide.
2. Select the number of time slots through the  $\text{SSI}_{In\_CCR}[\text{DC}]$  bits. For AC97 operation, the DC bits should be set to a value of 0xC, resulting in 13 time slots per frame.
3. Write data to be transmitted in Tx FIFO 0 (through Tx data register 0)
4. Program the  $\text{SSI}_{In\_ACR}[\text{FV}, \text{TIF}, \text{RD}, \text{WR}$  and  $\text{FRDIV}]$  bits
5. Update the contents of  $\text{SSI}_{In\_ACADD}$ ,  $\text{SSI}_{In\_ACDAT}$  and  $\text{SSI}_{In\_ATAG}$  (for fixed mode only) registers
6. Enable AC97 mode ( $\text{SSI}_{In\_ACR}[\text{AC97EN}]$  bit)

After the SSI starts transmitting and receiving data after being configured in AC97 mode, the processor needs to service the interrupts when they are raised (updates to command address/data or tag registers, reading of received data, and writing more data for transmission). Further details regarding fixed and variable mode implementation appear in the following sections.

While using AC97 in two-channel mode ( $\text{TCH} = 1$ ), it is recommended that the received tag is not stored in the Rx FIFO ( $\text{TIF} = 0$ ). If you need to update the  $\text{SSI}_{In\_ATAG}$  register and also issue a RD/WR command (in a single frame), it is recommended that the  $\text{SSI}_{In\_ATAG}$  register is updated prior to issuing a RD/WR command.

### 35.4.1.5.1 AC97 Fixed Mode ( $\text{SSI}_{In\_ACR}[\text{FV}] = 0$ )

In fixed mode of operation, SSI transmits in accordance with the frame rate divider bits that decide the number of frames for which the SSI should be idle, after operating for one frame. The following shows the slot assignments in a valid transmit frame:

- Slot 0: The tag value (written by the user program)

- Slot 1: If RD/WR command, command address
- Slot 2: If WR command, command data
- Slot 3–12: Transmit FIFO data, depending on the valid slots indicated by the TAG value

While receiving, bit 15 of the received tag slot (slot 0) is checked to see if the codec is ready. If this bit is set, the frame is received. The received tag provides the information about slots containing valid data. If the corresponding tag bit is valid, the command address (slot 1) and command data (slot 2) values are stored in the corresponding registers. The received data (slot 3–12) is then stored in the receive FIFO (for valid slots).

#### 35.4.1.5.2 AC97 Variable Mode ( $\text{SSI}_{In\text{--}}\text{ACR[FV]=1}$ )

In variable mode, the transmit slots that should contain data in the current frame are determined by the SLOTREQ bits received in slot 1 of the previous frame. While receiving, if the codec is ready, the frame is received and the SLOTREQ bits are stored for scheduling transmission in the next frame.

The SACCST, SACCENT and SACCDIS registers help determine which transmit slots are active. This information is used to ensure that SSI does not transmit data for powered-down/inactive channels.

### 35.4.2 SSI Clocking

The SSI uses the following clocks:

- **SSI\_CLOCK** — This is the internal clock that drives the SSI's clock generation logic, which can be a fraction of the internal core clock ( $f_{sys}$ ) or the clock input on the SSI\_CLKIN pin. The CCM's MISCCR register can select either of these sources. Having this choice allows the user to operate the SSI module at frequencies that would not be achievable if standard internal core clock frequencies are used. This is also the output master clock ( $\text{SSI}_{In\text{--}}\text{MCLK}$ ) when in master mode.
- Bit clock — Serially clocks the data bits in and out of the SSI port. This clock is generated internally or taken from external clock source (through  $\text{SSI}_{In\text{--}}\text{BCLK}$ ).
- Word clock — Counts the number of data bits per word (8, 10, 12, 16, 18, 20, 22 or 24 bits). This clock is generated internally from the bit clock.
- Frame clock (frame sync) — Counts the number of words in a frame. This signal can be generated internally from the bit clock or taken from external source (from  $\text{SSI}_{In\text{--}}\text{FS}$ ).
- Master clock — In master mode, this is an integer multiple of frame clock. It is used in cases when SSI has to provide a clock to the connected devices.

Take care to ensure that the bit clock frequency (internally generated or sourced from an external device) is never greater than 1/5 of the internal bus frequency ( $f_{sys/2}$ ).

In normal mode, the bit clock, used to serially clock the data, is visible on the serial clock ( $\text{SSI}_{In\text{--}}\text{BCLK}$ ) port. The word clock is an internal clock that determines when transmission of an 8, 10, 12, 16, 18, 20, 22, or 24-bit word has completed. The word clock then clocks the frame clock, which counts the number of words in the frame. The frame clock can be viewed on the  $\text{SSI}_{In\text{--}}\text{FS}$  frame sync port because a frame sync generates after the correct number of words in the frame have passed. In master mode, the  $\text{SSI}_{In\text{--}}\text{MCLK}$  signal is the serial master clock if enabled by the  $\text{SSI}_{In\text{--}}\text{CR[MCE]}$  bit. This serial master clock is an oversampling clock of the frame sync clock ( $\text{SSI}_{In\text{--}}\text{FS}$ ). In this mode, the word length (WL), prescaler

range (PSR), prescaler modulus (PM), and frame rate (DC) selects the ratio of  $\text{SSIn\_MCLK}$  to sampling clock,  $\text{SSIn\_FS}$ . In I<sup>2</sup>S mode, the oversampling clock is available on this port if the  $\text{SSIn\_CR[MCE]}$  bit is set.

Figure 35-39 shows the relationship between the clocks and the dividers. The bit clock can be received from an SSI clock port or generated from the internal clock ( $\text{SSI\_CLOCK}$ ) through a divider, as shown in Figure 35-40.

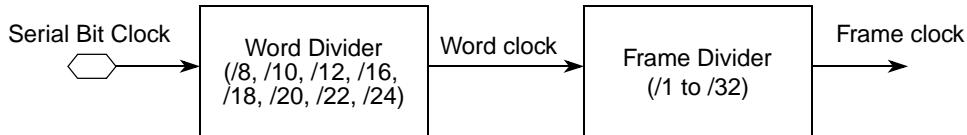


Figure 35-39. SSI Clocking

### 35.4.2.1 SSI Clock and Frame Sync Generation

Data clock and frame sync signals can be generated internally or obtained from external sources. If internally generated, the SSI clock generator derives bit clock and frame sync signals from the  $\text{SSI\_CLOCK}$ . The SSI clock generator consists of a selectable, fixed prescaler and a programmable prescaler for bit rate clock generation. A programmable frame rate sync signal generation.

Figure 35-40 shows a block diagram of the clock generator for the transmit section. The serial bit clock can be internal or external, depending on the transmit direction ( $\text{SSIn\_TCR[TXDIR]}$ ) bit.

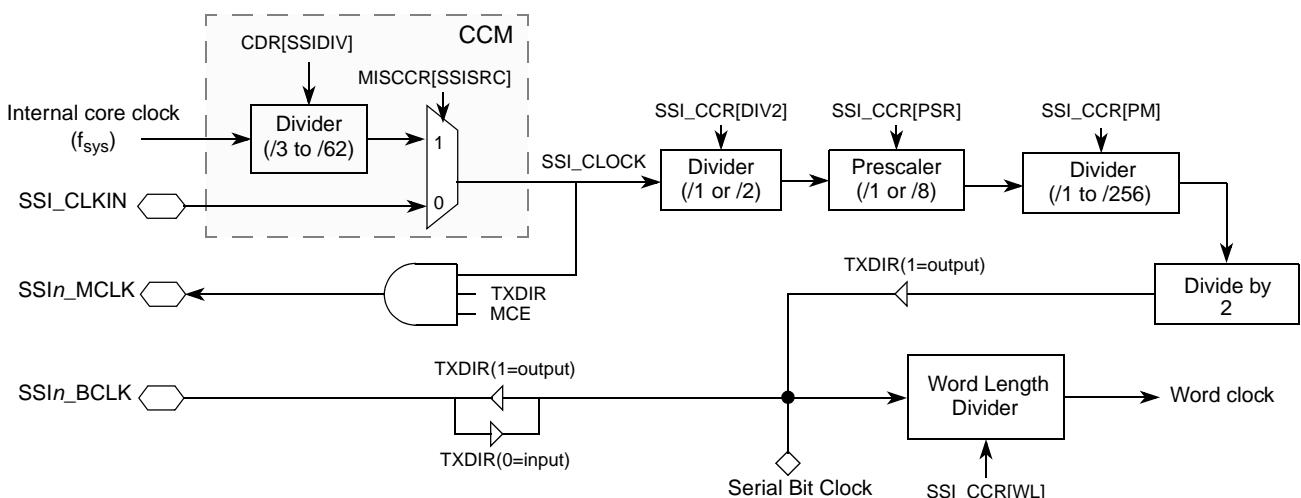


Figure 35-40. SSI Transmit Clock Generator Block Diagram

Figure 35-41 shows the frame sync generator block for the transmit section. When internally generated, receive and transmit frame sync generate from the word clock and are defined by the frame rate divider (DC) bits and the word length (WL) bits of the  $\text{SSIn\_CCR}$ .

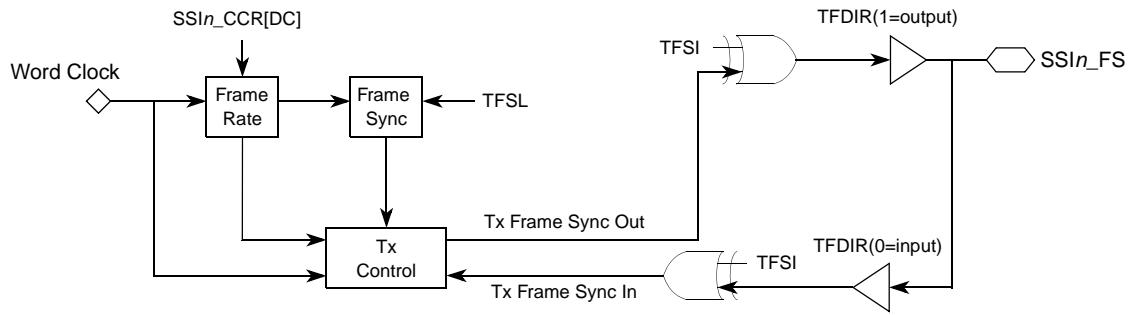


Figure 35-41. SSI Transmit Frame Sync Generator Block Diagram

### 35.4.2.2 DIV2, PSR and PM Bit Description

The bit clock frequency can be calculated from the SSI serial system clock (SSIn\_CLOCK), using [Equation 35-1](#).

#### NOTE

You must ensure that the bit-clock frequency is at most one-fifth the internal bus frequency ( $f_{sys/2}$ ). The oversampling clock frequency can go up to internal bus frequency. Bits DIV2, PSR, and PM must not be cleared at the same time.

$$f_{INT\_BIT\_CLK} = \frac{\text{SSI serial system clock}}{(DIV2 + 1) \times (7 \times PSR + 1) \times (PM + 1) \times 2}$$

**Eqn. 35-1**

From this, the frame clock frequency can be calculated:

$$f_{FS\_CLK} = \frac{f_{INT\_BIT\_CLK}}{(DC + 1) \times (2 \times (WL + 1))}$$

**Eqn. 35-2**

For example, if the SSI working clock is 19.2 MHz, in 8-bit word normal mode with DC = 1, PM = 0x4A (74), PSR = 0, DIV2 = 1, a bit clock rate of 64 kHz is generated. Because the 8-bit word rate equals two, sampling rate (or frame sync rate) would then be  $64/(2 \times 8) = 4$  kHz.

In the next example, SSIn\_CLOCK is 12 MHz. A 16-bit word network mode with DC = 1, PM = 1, the PSR = 0, DIV2 = 1, a bit clock rate of  $12/[14 \times 2] = 1.5$  MHz is generated. Because the 16-bit word rate equals two, sampling rate (or frame sync rate) would be  $1.5/(2 \times 16) = 46.875$  kHz.

[Table 35-29](#) shows the example of programming PSR and PM bits to generate different bit clock (SSIn\_BCLK) frequencies. The SSIn\_CLKIN signal is used in this example (MISCCR[SSISRC] = 0) because when operating the processor at the typical 240 MHz frequency, the SSI module is not able to accurately produce standard bit and sample rates.

Table 35-29. SSI Bit Clock and Frame Rate as a Function of PSR, PM, and DIV2

SSIn_CLKIN freq (MHz) (SSIn_MCLK)	SSIn_CCR					Bit Clk (kHz) SSIn_BCLK	Frame rate (kHz)
	DIV2	PSR	PM	WL	DC		
12.288	0	0	23	3	3	256	8
12.288	0	0	11	3	3	512	16

**Table 35-29. SSI Bit Clock and Frame Rate as a Function of PSR, PM, and DIV2 (continued)**

SSI_CLKIN freq (MHz) (SSIn_MCLK)	SSIn_CCR					Bit Clk (kHz) SSIn_BCLK	Frame rate (kHz)
	DIV2	PSR	PM	WL	DC		
12.288	0	0	5	3	3	1024	32
12.288	0	0	3	3	3	1536	48
12.288	0	0	23	7	3	256	4
12.288	0	0	11	7	3	512	8
12.288	0	0	5	7	3	1024	16
12.288	0	0	3	7	3	1536	24

**Table 35-30** shows the example of programming clock controller divider ratio to generate the SSIn\_MCLK and SSIn\_BCLK frequencies close to the ideal sampling rates. In these examples, setting the SSI to I<sup>2</sup>S master mode (SSIn\_CR[I2S] = 01) or individually programming the SSI into network, transmit internal clock mode selects the master mode. (The table specifically illustrates the I<sup>2</sup>S mode frequencies/sample rates.)

I<sup>2</sup>S master mode requires a 32-bit word length, regardless of the actual data type. Consequently, the fixed I<sup>2</sup>S frame rate of 64 bits per frame (word length (WL) can be any value) and DC = 1 are assumed.

**Table 35-30. SSI Sys Clock, Bit Clock, Frame Clock in Master Mode**

Sampling /Frame rate (kHz)	Over- sampling rate	SSI_CLKIN freq (MHz) (SSIn_MCLK)	SSIn_CCR			Bit Clk (kHz) SSIn_BCLK
			DIV2	PSR	PM	
44.10	384	16.934	0	0	2	2822.33
22.05	384	16.934	0	0	5	1411.17
11.025	384	16.934	0	0	11	705.58
48.00	256	12.288	0	0	1	3072

### 35.4.3 External Frame and Clock Operation

When applying external frame sync and clock signals to the SSI module, at least four bit clock cycles should exist between the enabling of the transmit or receive section and the rising edge of the corresponding frame sync signal. The transition of SSIn\_FS should be synchronized with the rising edge of external clock signal, SSIn\_BCLK.

### 35.4.4 Supported Data Alignment Formats

The SSI supports three data formats to provide flexibility with managing data. These formats dictate how data is written to and read from the data registers. Therefore, data can appear in different places in SSIn\_TX0/1 and SSIn\_RX0/1 based on the data format and the number of bits per word. Independent data formats are supported for the transmitter and receiver (i.e. the transmitter and receiver can use different data formats).

## Synchronous Serial Interface (SSI)

The supported data formats are:

- msb alignment
- lsb alignment
  - Zero-extended (receive data only)
  - Sign-extended (receive data only)

With msb alignment, the most significant byte is bits 31–24 of the data register if the word length is larger than, or equal to, 16 bits. If the word length is less than 16 bits and msb alignment is chosen, the most significant byte is bits 15–8. With lsb alignment, the least significant byte is bits 7–0. The `SSIn_TCR[TXBIT0]` and the `SSIn_RCR[RXBIT0]` bits control data alignment. [Table 35-31](#) shows the bit assignment for all the data formats supported by the SSI module.

**Table 35-31. Data Alignment**

Format	Bit Number																																						
	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							
8-bit lsb Aligned																											7	6	5	4	3	2	1	0					
8-bit msb Aligned																				7	6	5	4	3	2	1	0												
10-bit lsb Aligned																											9	8	7	6	5	4	3	2	1	0			
10-bit msb Aligned																				9	8	7	6	5	4	3	2	1	0										
12-bit lsb Aligned																											11	10	9	8	7	6	5	4	3	2	1	0	
12-bit msb Aligned																				11	10	9	8	7	6	5	4	3	2	1	0								
16-bit lsb Aligned																				15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
16-bit msb Aligned	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																							
18-bit lsb Aligned																			17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
18-bit msb Aligned	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																					
20-bit lsb Aligned																		19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
20-bit msb Aligned	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																			
22-bit lsb Aligned																		21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
22-bit msb Aligned	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0																	
24-bit lsb Aligned															23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
24-bit msb Aligned	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0															

In addition, if lsb alignment is selected, the receive data can be zero-extended or sign-extended.

- In zero-extension, all bits above the most significant bit are 0s. This format is useful when data is stored in a pure integer format.
- In sign-extension, all bits above the most significant bit are equal to the most significant bit. This format is useful when data is stored in a fixed-point integer format (which implies fractional values).

The  $\text{SSI}_n\text{\_RCR}[RXEXT]$  bit controls receive data extension. Transmit data used with lsb alignment has no concept of sign/zero-extension. Unused bits above the most significant bit are simply ignored.

When configured in I<sup>2</sup>S or AC97 mode, the SSI forces the lsb alignment. However, the  $\text{SSI}_n\text{\_RCR}[RXEXT]$  bit chooses zero-extension or sign-extension.

Refer to [Section 35.3.10, “SSI Transmit Configuration Register \(SSIn\\_TCR\),”](#) and [Section 35.3.11, “SSI Receive Configuration Register \(SSIn\\_RCR\),”](#) for more detail on the relevant bits in the SSIn\_TCR and SSIn\_RCR registers.

### 35.4.5 Receive Interrupt Enable Bit Description

If the receive FIFO is not enabled, an interrupt occurs when the corresponding SSI receive data ready ( $\text{SSI}_n\text{\_ISR}[RDR0/1]$ ) bit is set. If the receive FIFO is enabled and the RIE and RE bit are set, the processor is interrupted when either of the SSI receives FIFO full ( $\text{SSI}_n\text{\_ISR}[RFF0/1]$ ) bits is set. When the receive FIFO is enabled, a maximum of eight values are available to be read (eight values per channel in two-channel mode). If not enabled, one value can be read from the SSIn\_RX register (one each in two-channel mode).

If the RIE bit is cleared, these interrupts are disabled. However, the RFF0/1 and RDR0/1 bits indicate the receive data register full condition. Reading the SSIn\_RX registers clears the RDR bits, thus clearing the pending interrupt. Two receive data interrupts (two per channel in two-channel mode) are available: receive data with exception status and receive data without exception. [Table 35-32](#) shows the conditions these interrupts are generated.

**Table 35-32. SSI Receive Data Interrupts**

Interrupt	RIE	ROEn	RFFn/RDRn
<b>Receive Data 0 Interrupts (n = 0)</b>			
Receive Data 0 (with exception status)	1	1	1
Receive Data 0 (without exception)	1	0	1
<b>Receive Data 1 Interrupts (n = 1)</b>			
Receive Data 1 (with exception status)	1	1	1
Receive Data 1 (without exception)	1	0	1

### 35.4.6 Transmit Interrupt Enable Bit Description

The SSI transmit interrupt enable (TIE) bit controls interrupts for the SSI transmitter. If the transmit FIFO is enabled and the TIE and TE bits are set, the processor is interrupted when either of the SSI transmit FIFO empty ( $\text{SSI}_n\text{\_ISR}[TFE0/1]$ ) flags is set. If the corresponding transmit FIFO is not enabled, an interrupt is generated when the corresponding  $\text{SSI}_n\text{\_ISR}[TDE0/1]$  flag is set and transmit enable (TE) bit is set.

When transmit FIFO 0 is enabled, a maximum of eight values can be written to the SSI (eight per channel in two-channel mode using Tx FIFO 1). If not enabled, then one value can be written to the SSIn\_TX0 register (one per channel in two-channel mode using SSIn\_TX1). When the TIE bit is cleared, all transmit interrupts are disabled. However, the TDE0/1 bits always indicate the corresponding SSIn\_TX register

empty condition, even when the transmitter is disabled by the transmit enable ( $SSIn\_CR[TE]$ ) bit. Writing data to the  $SSIn\_TX$  clears the corresponding TDE bit, thus clearing the interrupt.

Two transmit data interrupts are available (two per channel in two-Channel mode): transmit data with exception status and transmit data without exceptions. [Table 35-33](#) shows the conditions under which these interrupts are generated.

**Table 35-33. SSI Transmit Data Interrupts**

Interrupt	TIE	TUE $n$	TFE $n$ /TDE $n$
<b>Transmit Data 0 Interrupts (<math>n = 0</math>)</b>			
Transmit Data 1 (with exception status)	1	1	1
Transmit Data 1 (without exception)	1	0	1
<b>Transmit Data 1 Interrupts (<math>n = 1</math>)</b>			
Transmit Data 0 (with exception status)	1	1	1
Transmit Data 0 (without exception)	1	0	1

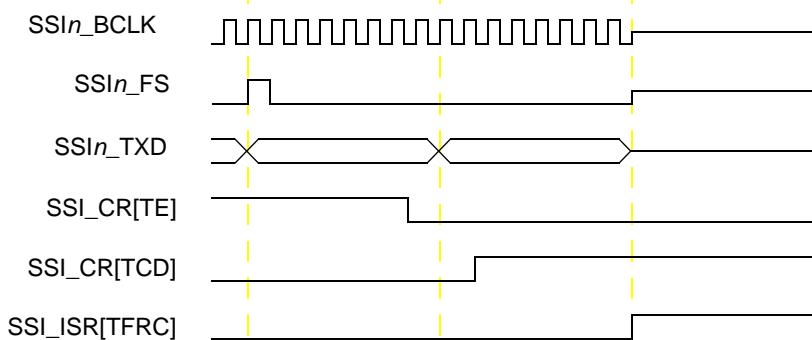
### 35.4.7 Internal Frame and Clock Shutdown

The frame sync and clock operation is determined by the  $SSIn\_CR[TCD, RCD]$  and  $SSIn\_CR[TE, RE]$  bits.

During transmit/receive operation, clearing TE/RE stops data transmission/reception when the current frame ends. If the  $SSIn\_CR[TCD]$  or  $RCD$  bit is set in the current or previous frames, the SSI stops driving the frame sync and clock signals when the current frame ends. After this, the  $SSIn\_ISR[TFRC, RFRC]$  status bits indicate the frame completion state.

[Figure 35-42](#) illustrates a transmission case where:

- TXDIR and TFDIR are set
- TE is cleared
- TCD bit is set during the current frame



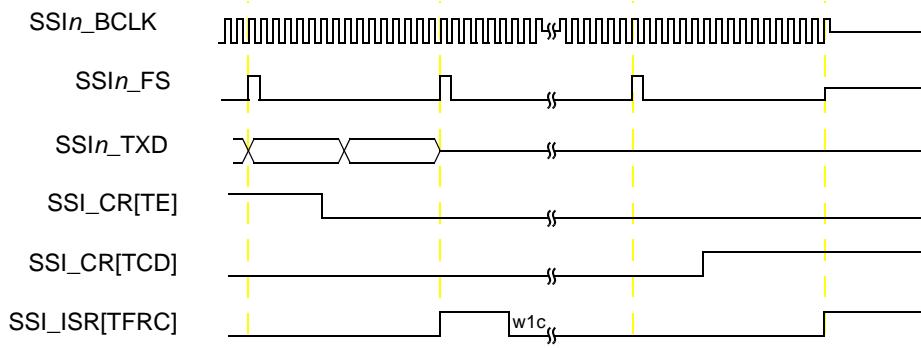
**Figure 35-42. SSIn<sub>n</sub>\_CR[TCD] Assertion in Same Frame as TE is Disabled**

If  $SSIn\_CR[TCD]$  or  $RCD$  is not set while  $SSIn\_CR[TE]$  or  $RE$  is cleared, the SSI continues generating the frame sync and clock signals. Upon setting  $SSIn\_CR[TCD]$  or  $RCD$ , the SSI stops driving these signals

at the end of the current frame. Following this, the TFRC/RFRC status bits are set to indicate the frame completion state.

**Figure 35-43** illustrates a transmission case where:

- TXDIR and TFDIR are set
- SSIn\_CR[TCD] is set a few frames after clearing TE
- SSIn\_ISR[TFRC] is set at the frame boundary after TE is cleared. Once software services this interrupt and later sets SSIn\_CR[TCD], TFRC is again set at the following frame boundary.



**Figure 35-43. SSIn\_CR[TCD] Assertion in Frame After Disabling TE**

## 35.5 Initialization/Application Information

The following types of reset affected the SSI:

- Power-on reset—Asserting the **RESET** signal generates the power-on reset. This reset clears the SSIn\_CR[SSI\_EN] bit, which disables the SSI. All other status and control bits in the SSI are affected as described in [Table 35-4](#)
- SSI reset—The SSI reset is generated when the SSIn\_CR[SSI\_EN] bit is cleared. The SSI status bits are reset to the same state produced by the power-on reset. The SSI control bits, including those in SSIn\_CR, are unaffected. The SSI reset is useful for selective reset of the SSI, without changing the present SSI control bits and without affecting the other peripherals.

The correct sequence to initialize the SSI is:

1. Issue a power-on or SSI reset (SSIn\_CR[SSI\_EN] = 0).
2. Set all control bits for configuring the SSI (refer to [Table 35-34](#)).
3. Enable appropriate interrupts/DMA requests through SSIn\_IER.
4. Set the SSIn\_CR[SSI\_EN] bit to enable the SSI.
5. For AC97 mode, set the SSIn\_ACR[AC97EN] bit after programming the SSIn\_ATAG register (if needed, for AC97 fixed mode).
6. Set SSIn\_CR[TE/RE] bits.

To ensure proper operation of the SSI, use the power-on or SSI reset before changing any of the control bits listed in [Table 35-34](#).

**NOTE**

These control bits should not be changed when the SSI module is enabled.

**Table 35-34. SSI Control Bits Requiring SSI to be Disabled Before Change**

Control Register	Bit
SSIn_CR	[9]=CIS [8]=TCH [7]=MCE [6:5]=I2S [4]=SYN [3]=NET
SSIn_IER	[22]=RDMAE [20]=TDMAE
SSIn_RCR SSIn_TCR	[9]=RXBIT0 and TXBIT0 [8]=RFEN1 and TFEN1 [7]=RFEN0 and TFEN0 [6]=TFDIR [5]=RXDIR and TXDIR [4]=RSHFD and TSHFD [3]=RSCKP and TSCKP [2]=RFSI and TFSI [1]=RFSL and TFSL [0]=REFS and TEFS
SSIn_CCR	[16:13]=WL
SSIn_ACR	[1]=FV [10:5]=FRDIV

# Chapter 36

## 1-Wire Module

### 36.1 Overview

The 1-Wire® module provides the communication link to a generic 1-Kbit add-only memory. The module sends or receives one bit at a time with an option for software to manage the data using bytes. The required protocol for accessing the generic 1-Wire device is defined by Maxim, and is fully described at [www.maxim-ic.com/products/1-wire/](http://www.maxim-ic.com/products/1-wire/).

#### 36.1.1 Block Diagram

Figure 36-1 shows a block diagram of the 1-Wire module.

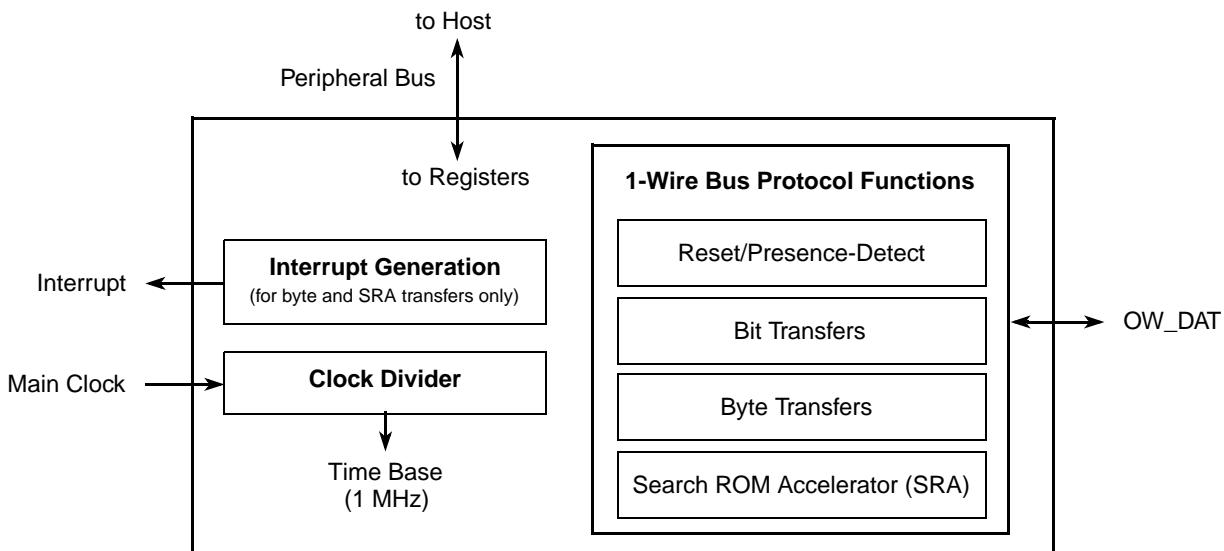


Figure 36-1. 1-Wire Module Block Diagram

#### 36.1.2 Features

The 1-Wire module includes the following features:

- Performs the 1-Wire bus protocol to communicate with an external 1-Wire device
- Clock divider to generate the 1-Wire bus reference clock from the peripheral bus clock
- Supports byte transfers with optional interrupts or DMA for more efficient programming
- Provides a search ROM accelerator mode to speed the search ROM protocol

### 36.1.3 Modes of Operation

The 1-Wire module supports the following operations:

- Normal operating modes
  - Bit or byte Transfers
  - Reset/presence-detect pulse
  - Search ROM accelerator mode
- Low power mode

## 36.2 External Signals

[Table 36-1](#) shows the signal that interfaces with a generic 1-Wire device.

**Table 36-1. 1-Wire Module Signal**

Signal	I/O	Function
OW_DAT	I/O	One-Wire bus Requires an external pull-up resistor. The recommended resistor value is specified by the generic 1-Wire device used in a given system.

## 36.3 Memory Map/Register Definition

**Table 36-2. 1-Wire Memory Map**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xEC00_8000	Control register (OW_CR)	8	R/W	0x00	<a href="#">36.3.1/36-3</a>
0xEC00_8004	Time divider register (OW_DIV)	8	R/W	0x00	<a href="#">36.3.2/36-4</a>
0xEC00_8008	Reset register (OW_RST)	8	R/W	0x00	<a href="#">36.3.3/36-4</a>
0xEC00_800C	Command register (OW_CMD)	8	R/W	0x00	<a href="#">36.3.4/36-5</a>
0xEC00_8010	Transmit/receive register (OW_TXRX)	8	R/W	0x00	<a href="#">36.3.5/36-5</a>
0xEC00_8014	Interrupt status register (OW_ISR)	8	R	0x0E	<a href="#">36.3.6/36-6</a>
0xEC00_8018	Interrupt enable register (OW_IER)	8	R/W	0x00	<a href="#">36.3.7/36-7</a>

### 36.3.1 Control Register (OW\_CR)

The control register is used to initiate the reset/presence-detect sequence and bit transfers. The register also provides the presence-detect status and bit-read status and DMA enable.

Address: 0xEC00_8000 (OW_CR)								Access: User read/write	
	7	6	5	4	3	2	1	0	
R	RPP	PST	WR0	WR1	RDST	0	0	DMA EN	
W									
Reset	0	0	0	0	0	0	0	0	

Figure 36-2. 1-Wire Control Register (OW\_CR)

Table 36-3. OW\_CR Field Descriptions

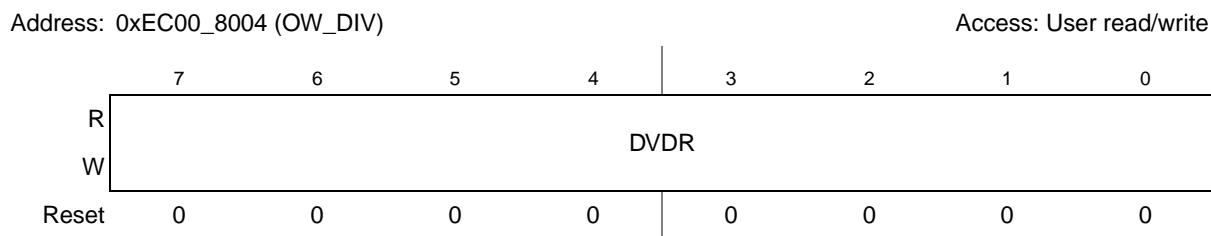
Field	Description
7 RPP	Reset/presence-detect pulse. This bit self-clears after the presence is determined. When writing... 0 Do nothing 1 Generate reset pulse and sample the bus for the presence pulse from the external device  When reading... 0 Reset pulse complete 1 Sequence not complete
6 PST	Presence status. This bit is valid after the RPP bit self-clears. 0 Device is not present 1 Device is present
5 WR0	Write 0. This bit self-clears when the write of the bit is complete. When writing... 0 Do nothing 1 Write a 0 bit to the interface  When reading... 0 Write sequence complete 1 Sequence not complete
4 WR1	Write-1/read. This bit self-clears when the write sequence completes. When writing... 0 Do nothing 1 Write a 1 bit to the interface and sample the bus  When reading... 0 Sequence complete 1 Sequence not complete
3 RDST	Read status. This bit is valid after the WR1 bit self-clears. 0 A 0 is sampled 1 A 1 is sampled

**Table 36-3. OW\_CR Field Descriptions (continued)**

Field	Description
2–1	Reserved, must be cleared.
0 DMAEN	DMA request enable. Enables the 1-Wire DMA request if OW_IER[ERBF] is also set. When a single-byte transmit or receive transfer completes, a request is sent to the DMA controller. 0 Disable DMA request 1 If RBF interrupt is enabled, enable DMA request

### 36.3.2 Time Divider Register (OW\_DIV)

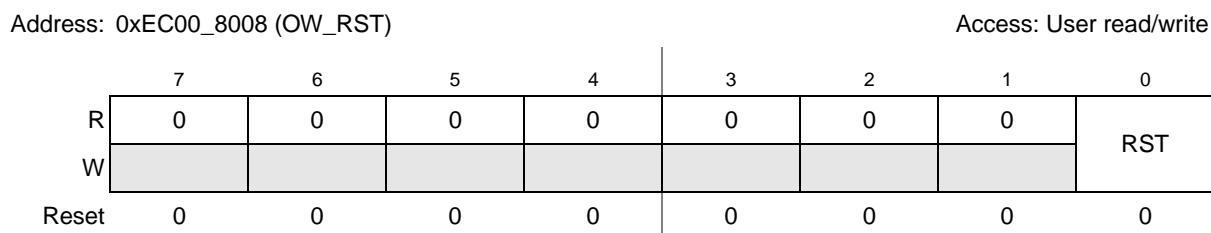
The time divider register divides the main clock input down to 1 MHz.

**Figure 36-3. 1-Wire Time Divider Register (OW\_DIV)****Table 36-4. OW\_DIV Field Descriptions**

Field	Description
7–0 DVDR	Divider factor. The internal clock divider uses this field to generate the required time base for the module. 0x00 1 0x01 2 ... 0xFF 256

### 36.3.3 Reset Register (OW\_RST)

The reset register performs a software reset of the 1-Wire module.

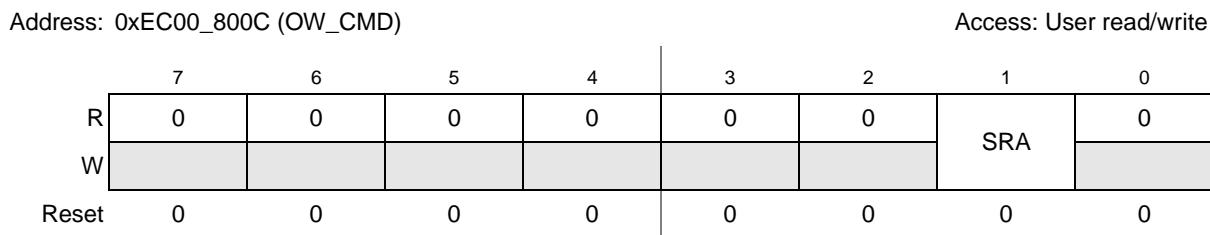
**Figure 36-4. 1-Wire Reset Register (OW\_RST)**

**Table 36-5. OW\_RST Field Descriptions**

Field	Description
7–1	Reserved, must be cleared.
0 RST	Software reset. 0 Do not perform a software reset 1 Initiate a software reset and hold the module in the software-reset state

### 36.3.4 Command Register (OW\_CMD)

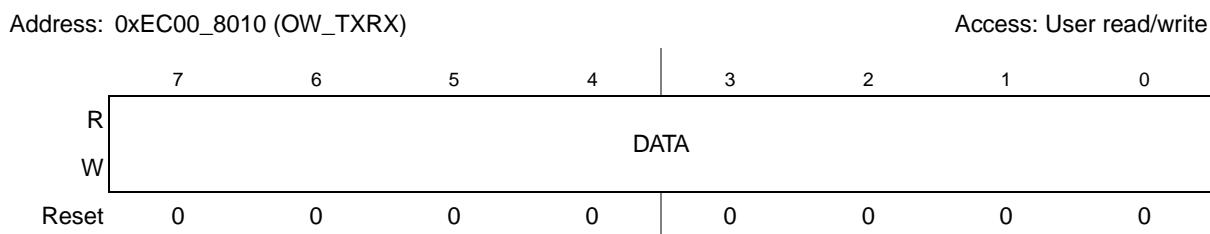
The 1-Wire module can be configured to run in search ROM accelerator mode using the command register.

**Figure 36-5. 1-Wire Command Register (OW\_CMD)****Table 36-6. OW\_CMD Field Descriptions**

Field	Description
7–2	Reserved, must be cleared.
1 SRA	Search ROM Accelerator. This bit is cleared when the reset-presence-pulse bit OW_CR[RPP] is set. 0 Deactivate the search ROM accelerator. 1 Switch to search ROM accelerator mode.
0	Reserved, must be cleared.

### 36.3.5 Transmit/Receive Register (OW\_TXRX)

Data sent and received from the 1-Wire module passes through the transmit/receive (OW\_TXRX) register. The 1-Wire module is double-buffered with separate transmit and receive buffers connected to the OW\_TXRX register.

**Figure 36-6. 1-Wire Transmit/Receive Register (OW\_TXRX)**

**Table 36-7. OW\_TXRX Field Descriptions**

Field	Description
7–0 DATA	Data byte. When writing... The data byte is written to the transmit buffer.  When reading... A data byte is read from the receive buffer. The data is valid only when OW_ISR[RBF] is set.

### 36.3.6 Interrupt Register (OW\_ISR)

OW\_ISR contains flags for the reset/presence-detect sequence and byte transfer operations. These flags can generate an interrupt if the corresponding enable bit is set in OW\_IER.

If interrupts are enabled, reading OW\_ISR deactivates the interrupt even if all current flags are not cleared; therefore, the interrupt service routine should clear all pending flags during each routine call.

#### NOTE

When a byte is written to OW\_TXRX, software then waits for a transmit shift register empty (TSRE) interrupt to occur. When TSRE sets, the receive buffer full (RBF) flag is also set. The RBF flag does not trigger an interrupt, assuming it is disabled. However, software should read the OW\_TXRX to clear the RBF flag to give a proper status of the pending interrupts.

Address: 0xEC00_8014 (OW_ISR)								Access: User read-only										
		7	6	5	4			3	2	1	0			7	6	5	4	
		R	0	0	RSRF	RBF			TSRE	TBE	PDR			R	0	0	RSRF	RBF
		W												W				
Reset	0	0	0	0	0	0		1	1	1	0			0	0	0	0	

**Figure 36-7. 1-Wire Interrupt Status Register (OW\_ISR)****Table 36-8. OW\_ISR Field Descriptions**

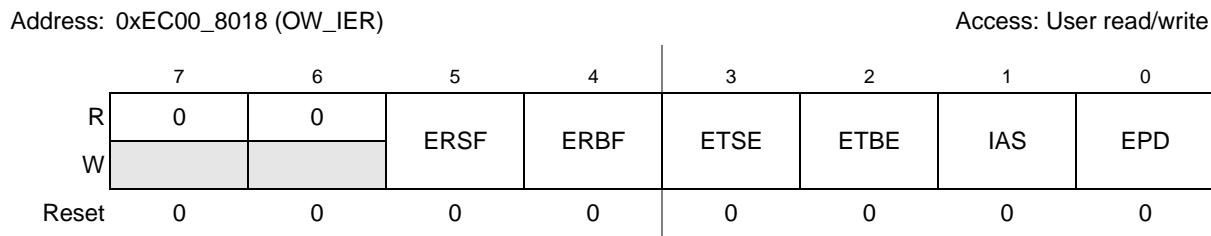
Field	Description
7–6	Reserved, must be cleared.
5 RSRF	Receive shift register full. Automatically clears when data in the receive shift register is transferred to the receive buffer. 0 The receive shift register is empty or currently receiving data 1 A byte is waiting in the receive shift register to be transferred to the receive buffer
4 RBF	Receive buffer full. Clears when software reads the byte from the OW_TXRX register. This flag prevents new data from being shifted into the receive buffer from the receive shift register. 0 No new data 1 A byte is waiting to be read from the OW_TXRX register

**Table 36-8. OW\_ISR Field Descriptions (continued)**

Field	Description
3 TSRE	Transmit shift register empty. Automatically clears when data in the transmit buffer is transferred to the transmit shift register. 0 Sending data 1 The transmit shift register is empty and is ready to receive the next byte from the transmit buffer
2 TBE	Transmit buffer empty. Clears when software writes a byte to the OW_TXRX register. 0 The transmit buffer is currently sending data to the transmit shift register 1 Nothing to transmit
1 PDR	Presence detect result. When a presence-detect (PD) interrupt occurs, this bit reflects the result of the presence-detect sequence. <b>Note:</b> This bit does not generate an interrupt. 0 Device found 1 Device not found
0 PD	Presence detect. After a 1-Wire reset is issued, this flag is set after the appropriate amount of time for a presence-detect pulse to have occurred. This flag is cleared when OW_ISR is read. 0 Reset/presence-detect sequence has not been issued 1 Reset/presence-detect sequence has completed and the result is indicated by the PDR bit.

### 36.3.7 Interrupt Enable Register (OW\_IER)

The interrupt enable register allows you to specify the source of interrupts. During a reset (hardware or software), all bits in this register are cleared, disabling all interrupt sources.

**Figure 36-8. 1-Wire Interrupt Enable Register (OW\_IER)****Table 36-9. OW\_IER Field Descriptions**

Field	Description
7–6	Reserved, must be cleared.
5 ERSF	Enable receive shift register full interrupt. 0 Disable 1 Enable
4 ERBF	Enable receive buffer full interrupt. 0 Disable 1 Enable
3 ETSE	Enable transmit shift register empty interrupt. 0 Disable 1 Enable

**Table 36-9. OW\_IER Field Descriptions (continued)**

Field	Description
2 ETBE	Enable transmit buffer empty interrupt. 0 Disable 1 Enable
1 IAS	Interrupt trigger active state. Determines the polarity for the 1-Wire interrupts. <b>Note:</b> This bit is not an interrupt-enable bit. 0 Active high 1 Active low
0 EPD	Enable presence detect. 0 Disable 1 Enable

## 36.4 Functional Description

The 1-Wire module interfaces with a generic 1-Kbit add-only memory through a simple 1-bit bus. Use the 1-Wire bus to program and read this memory.

The protocol involves first issuing one of four ROM function commands before the EEPROM is accessible:

- Read ROM
- Match ROM
- Search ROM
- Skip ROM

Through the 1-Wire bus, the host (master) software interfaces with the generic 1-Wire device (slave) and issues the commands to control the device's EEPROM. The *1kb Add-Only Memory Data Sheet* (DS2502) from Maxim Integrated Products, Inc. describes the generic 1-Wire device's operation procedures. It can be found at:

- <http://www.maxim-ic.com/products/1-wire/>

### 36.4.1 Normal Operating Modes

The 1-Wire module supports the following 1-Wire bus protocol functions:

- Reset/presence-detect pulse using the OW\_CR register
- Bit transfers using the OW\_CR register
- Byte transfers using the OW\_TXRX register
- Search ROM accelerator mode using the command and OW\_TXRX registers

#### 36.4.1.1 Reset/Presence-Detect Pulse

The 1-Wire module supports an automated initialization sequence for the 1-Wire bus, which is initiated by setting OW\_CR[RPP]. The automated initialization sequence is as follows:

1. Generate a reset pulse.

2. Listen for a response from an external device by sampling for the 1-Wire device presence bit.
3. After an amount of time determined by the 1-Wire standard, latch the presence bit (true or false) in OW\_CR[PST].

If an external device is detected (PST = 1), software can begin communication on the 1-Wire bus.

### 36.4.1.2 Bit Transfers

After the initialization sequence described in [Section 36.4.1.1, “Reset/Presence-Detect Pulse](#), software can write and read one bit at a time using OW\_CR.

#### 36.4.1.2.1 Write-0 Sequence

The write-0 sequence writes a zero bit to the 1-Wire device. Set OW\_CR[WR0] to initiate the write-0 pulse sequence. When the write completes, WR0 automatically clears.

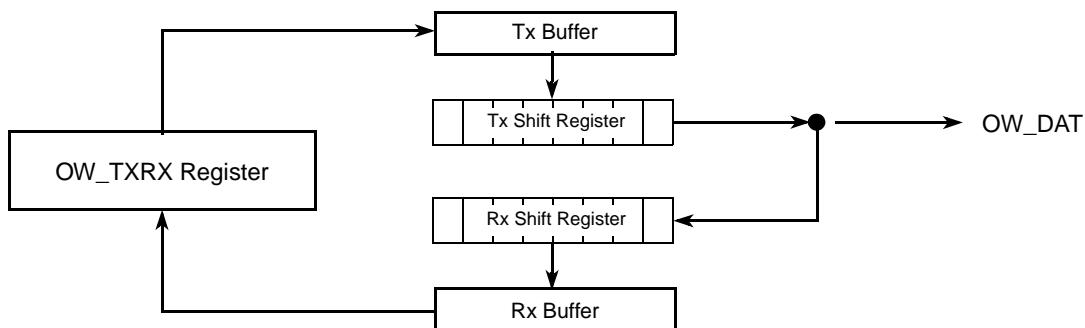
#### 36.4.1.2.2 Write-1 / Read Sequence

The write-1 sequence writes a one to the generic 1-Wire device. Set OW\_CR[WR1] to initiate the write-1 pulse sequence. When the write completes, WR1 automatically clears.

Because the write-1 and read timings are identical, this sequence also reads a bit from the bus. The sampled value is stored in OW\_CR[RDST] and is valid after WR1 self-clears.

### 36.4.1.3 Byte Transfers

After the initialization sequence described in [Section 36.4.1.1, “Reset/Presence-Detect Pulse](#), byte transfers can be performed using the OW\_TXRX register. Writing to the register connects to the transmit buffer; reading to the register connects to the receive buffer. See [Figure 36-9](#).



**Figure 36-9. Byte Transfers**

The transmit buffer connects to an internal transmit shift register where data is shifted serially onto the bus lsb first. Similarly, the receive buffer connects to an internal receive shift register where data is sampled serially from the bus.

You can read a byte from the generic 1-Wire device as follows:

1. Write 0xFF to OW\_TXRX (connected to the transmit buffer).

2. Wait for the receive-buffer-full DMA request or interrupt (or poll OW\_ISR[RBF] directly if the interrupt is disabled). During this time, the hardware is writing ones on the bus while sampling the wired-AND of the data from the device. The read data is shifted into the receive shift register. When a byte is collected in the receive shift register, the data is transferred to the receive buffer, and the RBF flag sets.
3. Read from OW\_TXRX (connected to the receive buffer) upon receiving the RBF interrupt or DMA request.

If the receive buffer is full, new data is not shifted from the receive shift register until the current data is read. To prevent loss of data, software must read OW\_TXRX to clear OW\_ISR[RBF]. This allows the receive shift register to shift new data into the receive buffer.

#### 36.4.1.4 Search ROM Accelerator Mode

In search ROM accelerator mode the 1-Wire module relieves you from having to perform single-bit operations on the bus and helps determine if more than one generic 1-Wire device exists.

Set OW\_CMD[SRA] to enter search ROM accelerator mode. This protocol specifies that the bus master reads two bits (a bit and its complement), then writes a bit to specify which devices should remain on the bus for further processing.

##### NOTE

This mode requires that a reset, followed by the search ROM command (0xF0) has already been issued on the 1-Wire bus.

The 1-Wire module automatically exits search ROM accelerator mode if the 1-Wire bus is re-initialized.

#### 36.4.2 Low Power Mode

The 1-Wire module automatically goes into low power mode when it is not communicating with a generic 1-Wire device. The main clock is gated off in low power mode. When software writes to any register, the module exits low power mode.

#### 36.4.3 Clocks

The 1-Wire module passes the peripheral bus clock through a clock divider. You must program the divider to generate a 1-MHz clock, which is the module's time base, as given by:

$$\text{time\_base} = \frac{f_{\text{sys}/2}}{\text{OW\_DIV}[DVDR] + 1} \quad \text{Eqn. 36-1}$$

For example, if the main clock frequency is 30 MHz, write 29 to OW\_DIV. If the main clock input frequency is not an integer, ensure the time base frequency is within the range given by [Equation 36-2](#).

$$0.98 \text{ MHz} \leq \text{time\_base} \leq 1.02 \text{ MHz}$$

[Eqn. 36-2](#)

##### NOTE

A main clock frequency below 10 MHz causes improper function of the module.

### 36.4.3.1 Hardware Reset

When a device reset occurs, a hard reset is performed on the 1-Wire module. This clears all values written to the registers.

### 36.4.3.2 Software Reset

Set OW\_RST[RST] to initiate a software reset. This clears all data written to the registers except for OW\_CMD and OW\_ISR.

#### NOTE

The reset register is not cleared during a software reset. You must clear OW\_RST[RST] to release the software reset.

## 36.4.4 Interrupts

The 1-Wire generates the following interrupts through the programming of OW\_IER:

- Receive shift register or buffer full
- Transmit shift register or buffer empty
- Presence detect

When any of these conditions are met, OW\_ISR sets the corresponding bit and generates an interrupt if enabled in OW\_IER. OW\_IER[IAS] determines if the interrupt generated is active low or active high. By default all interrupts are active high, and you should not modify IAS.



# Chapter 37

## Robust Real Time Clock

### 37.1 Introduction

The robust real time clock provides the functionality of a basic RTC, such as time keeping and calendaring. It also contains many advanced features, including:

- Protection against spurious memory/register updates
- Automatic switching to battery operation
- Compensation of the 1 Hz clock against variations in the 32 kHz clock oscillator due to crystal or temperature
- A standby RAM to store any 16-bit data that must be retained when in battery operation mode

### 37.2 Overview

The robust RTC block uses second, minute, hour, date, day-of-week, month and year counters, with automatic adjustment for leap year and daylight saving. Reading these counters indicates the current date and time and writing to these registers sets the date and time. This can be done in binary-coded decimal (BCD) or binary format.

When the time counters match the alarm hour, minute, and second setting, an alarm flag is set and an interrupt is generated (if enabled). The alarm can additionally be configured to match the day, month, and year to generate the alarm interrupt. The alarm interrupt can also wake the processor from various low power modes.

A countdown timer with minute resolution is also provided for time keeping applications. An interrupt is generated at the expiration of this counter. The RTC module also provides seven sampling timer interrupts apart from normal interrupts for alarm and countdown timeout.

A frequency compensation block is integrated into the RTC to correct any error in the 1 Hz clock due to variations in the 32 kHz clock caused by crystal inaccuracy, board variations, or temperature change. The compensation value is set by software and the correction is accomplished in hardware.

A protection mechanism is built in the RTC to protect against spurious writes to the RTC by any run-away code. You must write a specific sequence of codes to the write enable bits to allow write access to the registers. When finished updating the registers write any value to these bits to enable the write protection. After unlocking the registers, the CPU has a window of two seconds for updating the register space. On power on reset a window of 15 seconds is available. After this time the registers are locked automatically. Any further updates would require the CPU to unlock the registers.

The robust RTC battery supply maintains normal RTC functionality when the CPU power is removed. The battery supply allows RTC to keep functioning if the CPU is completely turned off. The RTC block is reset only when the battery supply and CPU power are removed and either is powered up.

The RTC is also equipped with a RAM that is powered by the battery supply if the main supply is removed. The processor can use this RAM to store any data it wants to preserve in case of power failure. This RAM loses its contents when the CPU and battery power are removed.

The RTC contains a 32-bit up-counter register that increments on writes, and reads of this register return the latest count value.

### 37.2.1 Block Diagram

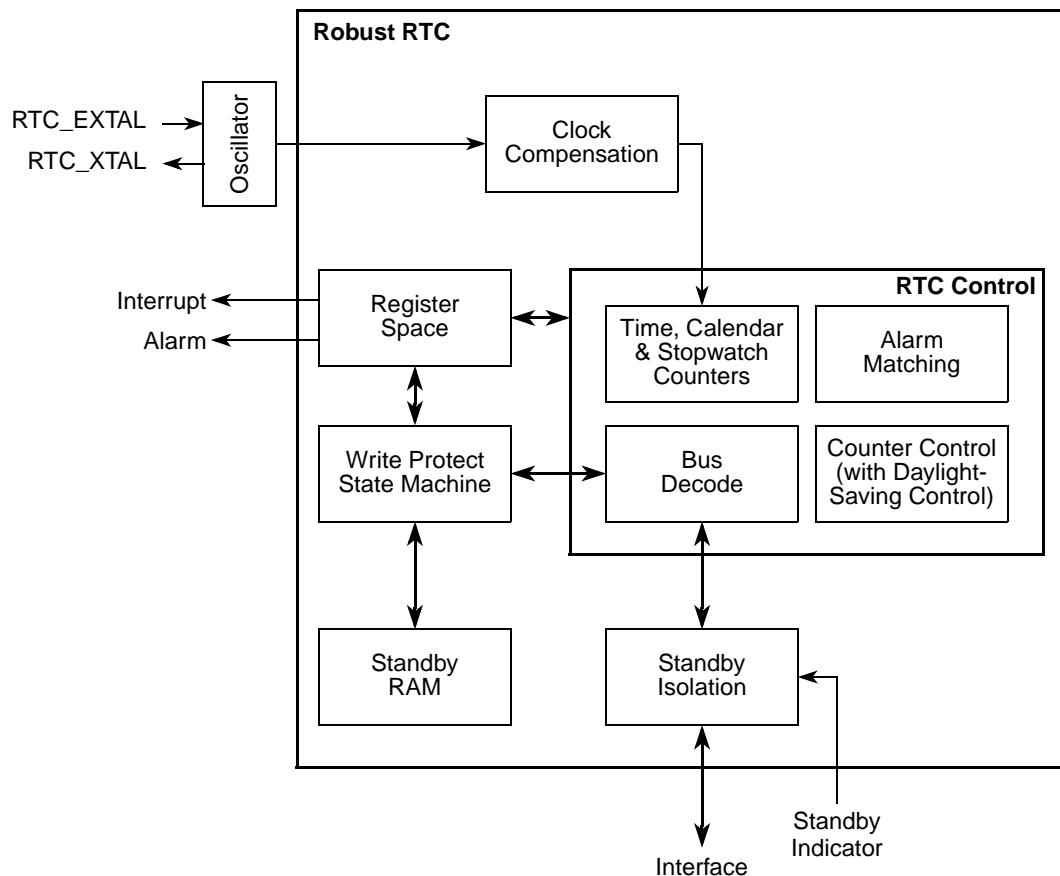


Figure 37-1. Robust RTC Block Diagram

### 37.2.2 Features

The robust RTC supports the following features:

- Full clock – Hour, minutes, and seconds with option for storing values in binary-coded decimal (BCD) or binary format.
- Calendaring – Day, month, year, and day of the week with option for storing values in BCD or binary format.

- Auto-adjustment for daylight saving time with user-defined parameters
- Automatic month and leap year adjustment
- RTC utilizes local time which implicitly contains the time zone offset
- Programmable alarm with interrupt
- Seven periodic interrupts
- Minute countdown timer with one minute resolution
- 32.768 kHz input clock
- Hardware compensation to compensate the 1 Hz clock to the counters against frequency variations in the oscillator clock due to temperature or crystal characteristics.
- Reset to the RTC block is generated only when both the battery supply and CPU power are removed and either is powered up.
- Battery operation (standby mode) ensures seamless RTC operation when CPU power is removed
- 2KB standby RAM

### 37.3 External Signal Description

The below table describes the RTC external signals.

**Table 37-1. RTC Signal Summary**

Signal Name	Abbreviation	Function	I/O
RTC external clock In	RTC_EXTAL	Crystal input clock	I
RTC crystal	RTC_XTAL	Oscillator output to crystal	O
RTC standby voltage	VSTBY_RTC	Standby voltage supply	—

### 37.4 Memory Map/Register Definition

The RTC contains 16-bit aligned registers and a standby RAM. Both can be accessed by 8-bit and 16-bit read/write cycles.

#### NOTE

All registers except RTC\_CR[WE] are protected from spurious updates by any run-away code.

The RTC module does not check for correctness of values programmed into its registers. Programming illogical time and date entries results in undefined operation and normal functionality is not guaranteed.

**Table 37-2. Real Time Clock Memory Map**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_8000	Month and year counter register (RTC_YEARMON)	16	R/W	0x0001	<a href="#">37.4.1/37-4</a>
0xFC0A_8002	Day and day-of-week counter register (RTC_DAYS)	16	R/W	0x0001	<a href="#">37.4.2/37-5</a>

**Table 37-2. Real Time Clock Memory Map (continued)**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_8004	Hour and minute counter register (RTC_HOURMIN)	16	R/W	0x0000	<a href="#">37.4.3/37-6</a>
0xFC0A_8006	Second counter register (RTC_SECONDS)	16	R/W	0x0000	<a href="#">37.4.4/37-7</a>
0xFC0A_8008	Year and month alarm register (RTC_ALM_YRMON)	16	R/W	0x0000	<a href="#">37.4.5/37-8</a>
0xFC0A_800A	Day alarm register (RTC_ALM_DAYS)	16	R/W	0x0000	<a href="#">37.4.6/37-9</a>
0xFC0A_800C	Hour and minute alarm register (RTC_ALM_HM)	16	R/W	0x0000	<a href="#">37.4.7/37-10</a>
0xFC0A_800E	Second alarm register (RTC_ALM_SEC)	16	R/W	0x0000	<a href="#">37.4.8/37-11</a>
0xFC0A_8010	Control register (RTC_CR)	16	R/W	0x0000	<a href="#">37.4.9/37-12</a>
0xFC0A_8012	Status register (RTC_SR)	16	R	0x0000	<a href="#">37.4.10/37-14</a>
0xFC0A_8014	Interrupt status register (RTC_ISR)	16	R/W	0x0001	<a href="#">37.4.11/37-15</a>
0xFC0A_8016	Interrupt enable register (RTC_IER)	16	R/W	0x0001	<a href="#">37.4.12/37-16</a>
0xFC0A_8018	Countdown timer register (RTC_COUNT_DN)	16	R/W	0x0000	<a href="#">37.4.13/37-17</a>
0xFC0A_8020	Configuration data register (RTC_CFG_DATA)	16	R/W	0x0000	<a href="#">37.4.14/37-18</a>
0xFC0A_8022	Daylight saving time hour register (RTC_DST_HOUR)	16	R/W	0x0000	<a href="#">37.4.15/37-19</a>
0xFC0A_8024	Daylight saving time month register (RTC_DST_MON)	16	R/W	0x0000	<a href="#">37.4.16/37-20</a>
0xFC0A_8026	Daylight saving time day register (RTC_DST_DAY)	16	R/W	0x0000	<a href="#">37.4.17/37-20</a>
0xFC0A_8028	Compensation register (RTC_COMPEN)	16	R/W	0x0000	<a href="#">37.4.18/37-21</a>
0xFC0A_8032	Count up high register (RTC_UP_CNTRH)	16	R	0x0000	<a href="#">37.4.19/37-21</a>
0xFC0A_8034	Count up low register (RTC_UP_CNTRL)	16	R	0x0000	<a href="#">37.4.20/37-22</a>
0xFC0A_8040 – 0xFC0A_883E	Standby RAM	2KB	R/W	Undefined	—

### 37.4.1 RTC Year & Month Counter Register (RTC\_YEARMON)

This register stores the value of the month and year counters. The year bits do not store the year value but calculate the increment in years. This is a signed value and ranges from –128 to +127. Software programs the offset from the base year of 2112 into this field. For example, if the current year is 2007, then write –105 or 0x97 to this field. The range of years supported is 1984 (2112 – 128) to 2239 (2112 + 127).

The month field stores the count value of the month. Writing to this field loads the month counter with this new value. The valid values are mentioned in table below. Writing any other value does not guarantee correct operation of RTC. Both month and year are unaffected on software reset.

In binary-coded decimal (BCD) mode, the value of month counter is represented in BCD format. Since the year value is a two's complement value this is not converted to BCD, even in BCD mode.

Prior to reading or writing to RTC\_YEARMON, read the RTC\_SR[INVAL] bit to determine that the counters are stable and they can be changed. The INVAL bit ensures that no operation is done at the boundary of a second when counters change value.

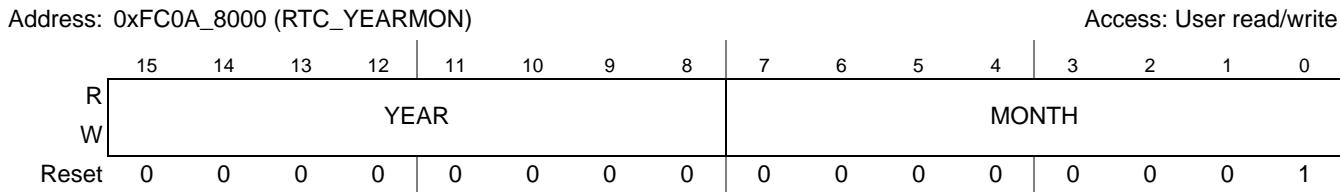


Figure 37-2. RTC Year and Month Counter Register (RTC\_YEARMON)

Table 37-3. RTC\_YEARMON Field Descriptions

Field	Description																										
15–8 YEAR	<p>Year count value. Two's complement value that indicates the offset in years from the base year, which is hard coded as 2112.</p> <p>0x00 2112 + 0 = 2112      ...      0x7F 2112 + 127 = 2239      0x80 2112 - 128 = 1984      ...      0xFF 2112 - 1 = 2111</p> <p><b>Note:</b> This field does not show the actual year value.</p>																										
7–0 MONTH	<p>Month count value. Indicates the month, depending on the value of RTC_CR[BCDEN].</p> <table border="1"> <thead> <tr> <th colspan="2">MONTH</th> <th rowspan="2">Month</th> </tr> <tr> <th>Binary</th> <th>BCD</th> </tr> </thead> <tbody> <tr> <td>0x00</td> <td>0x00</td> <td>Reserved</td> </tr> <tr> <td>0x01</td> <td>0x01</td> <td>January</td> </tr> <tr> <td>...</td> <td>...</td> <td>...</td> </tr> <tr> <td>0x0A</td> <td>0x10</td> <td>October</td> </tr> <tr> <td>0x0B</td> <td>0x11</td> <td>November</td> </tr> <tr> <td>0x0C</td> <td>0x12</td> <td>December</td> </tr> <tr> <td>0x0D–0xFF</td> <td>0x13–0xFF</td> <td>Reserved</td> </tr> </tbody> </table>	MONTH		Month	Binary	BCD	0x00	0x00	Reserved	0x01	0x01	January	...	...	...	0x0A	0x10	October	0x0B	0x11	November	0x0C	0x12	December	0x0D–0xFF	0x13–0xFF	Reserved
MONTH		Month																									
Binary	BCD																										
0x00	0x00	Reserved																									
0x01	0x01	January																									
...	...	...																									
0x0A	0x10	October																									
0x0B	0x11	November																									
0x0C	0x12	December																									
0x0D–0xFF	0x13–0xFF	Reserved																									

### 37.4.2 RTC Day & Day-of-Week Counters Register (RTC\_DAYS)

This read/write register contains the current value of the day-of-week and day counters. This register can be read at any time without affecting the counter count values. Writing to this register loads the value to the day-of-week and day counters and they continue to count from this new value. This register is unaffected by a software reset.

In BCD mode, the days counter value is converted to BCD format, while the day-of-week counter remains unchanged.

Prior to reading or writing to RTC\_DAYS, read the RTC\_SR[INVAL] bit to determine that the counters are stable and they can be changed. The INVAL bit ensures that no operation is done at the boundary of a second when counters change value.

Address: 0xFC0A_8002 (RTC_DAYS)																Access: User read/write								
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0								
R	0	0	0	0	0	DAYWEEK				DAY														
W						0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1		
Reset	0	0	0	0	0																			

Figure 37-3. RTC Day-of-Week and Day Counter Register (RTC\_DAYS)

Table 37-4. RTC\_DAYS Field Descriptions

Field	Description
15–11	Reserved, must be cleared.
10–8 DAYWEEK	Day of the week count. 000 Sunday 001 Monday ... 110 Saturday 111 Reserved
7–0 DAY	Day of the month count. Indicates the day, depending on the value of RTC_CR[BCDEN].

DAY		Day of Month
Binary	BCD	
0x00	0x00	Reserved
0x01	0x01	1
...	...	...
0x0A	0x10	10
...	...	...
0x1F	0x31	31
0x20–0xFF	0x32–0xFF	Reserved

### 37.4.3 RTC Hour and Minute Counter Register (RTC\_HOURMIN)

This read/write register contains the current value of the hour and minute counters. It can be read any time to get the current value of the counters. Only power-on reset resets this register. The hours counter can be set from 0 to 23. The minutes counter can be set from 0 to 59. This register is unaffected by software reset.

Prior to reading or writing to RTC\_HOURMIN, read the RTC\_SR[INVAL] bit to determine that the counters are stable and they can be changed. The INVAL bit ensures that no operation is done at the boundary of a second when counters change value.

Address: 0xFC0A\_8004 (RTC\_HOURMIN)

Access: User read/write

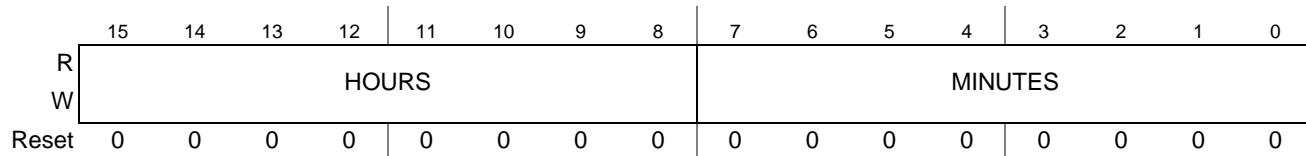


Figure 37-4. RTC Hour and Minute Counter Register (RTC\_HOURMIN)

Table 37-5. RTC\_HOURMIN Field Descriptions

Field	Description																												
15–8 HOURS	<p>Hour of day counter. Indicates the hour, depending on the value of RTC_CR[BCDEN].</p> <table border="1"> <thead> <tr> <th colspan="2">HOURS</th> <th rowspan="2">Hour of day</th> </tr> <tr> <th>Binary</th> <th>BCD</th> </tr> </thead> <tbody> <tr> <td>0x00</td> <td>0x00</td> <td>12:00 AM</td> </tr> <tr> <td>0x01</td> <td>0x01</td> <td>1:00 AM</td> </tr> <tr> <td>...</td> <td>...</td> <td>...</td> </tr> <tr> <td>0x0A</td> <td>0x10</td> <td>10:00 AM</td> </tr> <tr> <td>...</td> <td>...</td> <td>...</td> </tr> <tr> <td>0x17</td> <td>0x23</td> <td>11:00 PM</td> </tr> <tr> <td>0x18–0xFF</td> <td>0x24–0xFF</td> <td>Reserved</td> </tr> </tbody> </table>			HOURS		Hour of day	Binary	BCD	0x00	0x00	12:00 AM	0x01	0x01	1:00 AM	...	...	...	0x0A	0x10	10:00 AM	...	...	...	0x17	0x23	11:00 PM	0x18–0xFF	0x24–0xFF	Reserved
HOURS		Hour of day																											
Binary	BCD																												
0x00	0x00	12:00 AM																											
0x01	0x01	1:00 AM																											
...	...	...																											
0x0A	0x10	10:00 AM																											
...	...	...																											
0x17	0x23	11:00 PM																											
0x18–0xFF	0x24–0xFF	Reserved																											
7–0 MINUTES	<p>Minute of hour counter. Indicates the minute, depending on the value of RTC_CR[BCDEN].</p> <table border="1"> <thead> <tr> <th colspan="2">MINUTES</th> <th rowspan="2">Minute of hour</th> </tr> <tr> <th>Binary</th> <th>BCD</th> </tr> </thead> <tbody> <tr> <td>0x00</td> <td>0x00</td> <td>0</td> </tr> <tr> <td>0x01</td> <td>0x01</td> <td>1</td> </tr> <tr> <td>...</td> <td>...</td> <td>...</td> </tr> <tr> <td>0x0A</td> <td>0x10</td> <td>10</td> </tr> <tr> <td>...</td> <td>...</td> <td>...</td> </tr> <tr> <td>0x3B</td> <td>0x59</td> <td>59</td> </tr> <tr> <td>0x3C–0xFF</td> <td>0x5A–0xFF</td> <td>Reserved</td> </tr> </tbody> </table>			MINUTES		Minute of hour	Binary	BCD	0x00	0x00	0	0x01	0x01	1	...	...	...	0x0A	0x10	10	...	...	...	0x3B	0x59	59	0x3C–0xFF	0x5A–0xFF	Reserved
MINUTES		Minute of hour																											
Binary	BCD																												
0x00	0x00	0																											
0x01	0x01	1																											
...	...	...																											
0x0A	0x10	10																											
...	...	...																											
0x3B	0x59	59																											
0x3C–0xFF	0x5A–0xFF	Reserved																											

### 37.4.4 RTC Second Counter Register (RTC\_SECONDS)

This read/write register contains the current value of the second counter. It can be read any time to get the current value of the counter. Only power-on reset can reset this register. The second counter can be set from 0 to 59. This register is unaffected by software reset.

Prior to reading or writing to RTC\_SECONDS, read the RTC\_SR[INVAL] bit to determine that the counters are stable and they can be changed. The INVAL bit ensures that no operation is done at the boundary of a second when counters change value.

Address: 0xFC0A_8006 (RTC_SECONDS)																Access: User read/write							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0							
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
W																0							

Figure 37-5. RTC Second Counter Register (RTC\_SECONDS)

Table 37-6. RTC\_SECONDS Field Descriptions

Field1	Description
15–8	Reserved, must be cleared.
15–8 SECONDS	Second of minute counter. Indicates the second, depending on the value of RTC_CR[BCDEN].

SECONDS		Second of minute
Binary	BCD	
0x00	0x00	0
0x01	0x01	1
...	...	...
0x0A	0x10	10
...	...	...
0x3B	0x59	59
0x3C–0xFF	0x5A–0xFF	Reserved

### 37.4.5 RTC Year & Month Alarm Register (RTC\_ALM\_YRMON)

RTC\_ALM\_YRMON configures the month and year setting of the alarm. The alarm setting can be read or written anytime. This register is reset to its default state on software reset. The alarm interrupt bit (RTC\_ISR[ALM]) is set when all values of the alarm seconds, minutes, hours, days, month, and year match their respective counter values.

Address: 0xFC0A_8008 (RTC_ALM_YRMON)																Access: User read/write							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0							
R					YEAR						MONTH												
W					0	0	0	0	0	0	0	0	0	0	0	0							

Figure 37-6. RTC Year &amp; Month Alarm Register (RTC\_ALM\_YRMON)

**Table 37-7. RTC\_ALM\_YRMON Field Descriptions**

Field1	Description
15–8 YEAR	Year alarm value. Two's complement value that indicates the offset in years from the base year, which is hard coded as 2112. This signed value does not show the actual year value. 0x00 2112 + 0 = 2112 ... 0x7F 2112 + 127 = 2239 0x80 2112 - 128 = 1984 ... 0xFF 2112 - 1 = 2111
7–0 MONTH	Month alarm value. Indicates the month, depending on the value of RTC_CR[BCDEN].

MONTH		Month
Binary	BCD	
0x00	0x00	Reserved
0x01	0x01	January
...	...	...
0x0A	0x10	October
0x0B	0x11	November
0x0C	0x12	December
0x0D–0xFF	0x13–0xFF	Reserved

### 37.4.6 RTC Days Alarm Register (RTC\_ALM\_DAYS)

RTC\_ALM\_DAYS configures the day setting of the alarm. The alarm setting can be read or written anytime. This register is reset to its default state on software reset. Alarm interrupt bit (RTC\_ISR[ALM]) is set when all values of alarm seconds, minutes, hours, days, month, and year match their respective counter values.

Address: 0xFC0A_800A (RTC_ALM_DAYS)												Access: User read/write							
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

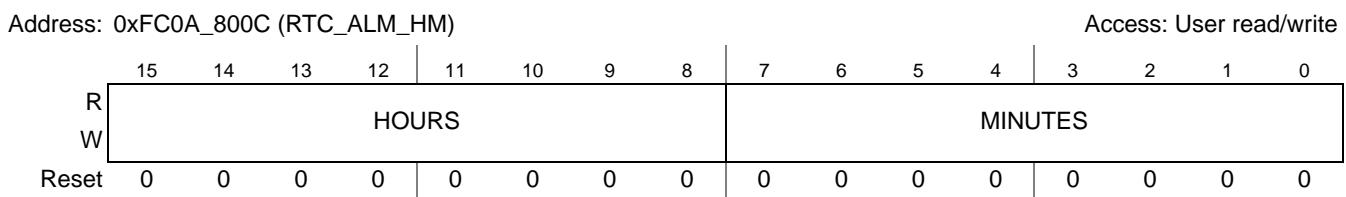
Figure 37-7. RTC Day-of-Week and Day Alarm Register (RTC\_ALM\_DAYS)

**Table 37-8. RTC\_ALM\_DAYS Field Descriptions**

Field	Description																												
15–8	Reserved, must be cleared.																												
7–0 DAY	Day of the month alarm. Indicates the day, depending on the value of RTC_CR[BCDEN].																												
	<table border="1"> <thead> <tr> <th colspan="2">DAY</th> <th rowspan="2">Day of Month</th> </tr> <tr> <th>Binary</th> <th>BCD</th> </tr> </thead> <tbody> <tr> <td>0x00</td> <td>0x00</td> <td>Reserved</td> </tr> <tr> <td>0x01</td> <td>0x01</td> <td>1</td> </tr> <tr> <td>...</td> <td>...</td> <td>...</td> </tr> <tr> <td>0x0A</td> <td>0x10</td> <td>10</td> </tr> <tr> <td>...</td> <td>...</td> <td>...</td> </tr> <tr> <td>0x1F</td> <td>0x31</td> <td>31</td> </tr> <tr> <td>0x20–0xFF</td> <td>0x32–0xFF</td> <td>Reserved</td> </tr> </tbody> </table>			DAY		Day of Month	Binary	BCD	0x00	0x00	Reserved	0x01	0x01	1	...	...	...	0x0A	0x10	10	...	...	...	0x1F	0x31	31	0x20–0xFF	0x32–0xFF	Reserved
DAY		Day of Month																											
Binary	BCD																												
0x00	0x00	Reserved																											
0x01	0x01	1																											
...	...	...																											
0x0A	0x10	10																											
...	...	...																											
0x1F	0x31	31																											
0x20–0xFF	0x32–0xFF	Reserved																											

### 37.4.7 RTC Hours and Minutes Alarm Register (RTC\_ALM\_HM)

RTC\_ALM\_HM configures the hour and minute setting of the alarm. The alarm setting can be read or written anytime. This register is reset to default state on software reset. In BCD mode, all eight bits of each field indicate the time.

**Figure 37-8. RTC Hours and Minutes Alarm Register (RTC\_ALM\_HM)**

**Table 37-9. RTC\_ALM\_HM Field Descriptions**

Field	Description																												
15–8 HOURS	Hour of day alarm. Indicates the hour alarm, depending on the value of RTC_CR[BCDEN]. <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2">HOURS</th> <th rowspan="2">Hour of day</th> </tr> <tr> <th>Binary</th> <th>BCD</th> </tr> </thead> <tbody> <tr> <td>0x00</td> <td>0x00</td> <td>12:00 AM</td> </tr> <tr> <td>0x01</td> <td>0x01</td> <td>1:00 AM</td> </tr> <tr> <td>...</td> <td>...</td> <td>...</td> </tr> <tr> <td>0x0A</td> <td>0x10</td> <td>10:00 AM</td> </tr> <tr> <td>...</td> <td>...</td> <td>...</td> </tr> <tr> <td>0x17</td> <td>0x23</td> <td>11:00 PM</td> </tr> <tr> <td>0x18–0xFF</td> <td>0x24–0xFF</td> <td>Reserved</td> </tr> </tbody> </table>			HOURS		Hour of day	Binary	BCD	0x00	0x00	12:00 AM	0x01	0x01	1:00 AM	...	...	...	0x0A	0x10	10:00 AM	...	...	...	0x17	0x23	11:00 PM	0x18–0xFF	0x24–0xFF	Reserved
HOURS		Hour of day																											
Binary	BCD																												
0x00	0x00	12:00 AM																											
0x01	0x01	1:00 AM																											
...	...	...																											
0x0A	0x10	10:00 AM																											
...	...	...																											
0x17	0x23	11:00 PM																											
0x18–0xFF	0x24–0xFF	Reserved																											
7–0 MINUTES	Minute of hour alarm. Indicates the minute alarm, depending on the value of RTC_CR[BCDEN]. <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th colspan="2">MINUTES</th> <th rowspan="2">Minute of hour</th> </tr> <tr> <th>Binary</th> <th>BCD</th> </tr> </thead> <tbody> <tr> <td>0x00</td> <td>0x00</td> <td>0</td> </tr> <tr> <td>0x01</td> <td>0x01</td> <td>1</td> </tr> <tr> <td>...</td> <td>...</td> <td>...</td> </tr> <tr> <td>0x0A</td> <td>0x10</td> <td>10</td> </tr> <tr> <td>...</td> <td>...</td> <td>...</td> </tr> <tr> <td>0x3B</td> <td>0x59</td> <td>59</td> </tr> <tr> <td>0x3C–0xFF</td> <td>0x5A–0xFF</td> <td>Reserved</td> </tr> </tbody> </table>			MINUTES		Minute of hour	Binary	BCD	0x00	0x00	0	0x01	0x01	1	...	...	...	0x0A	0x10	10	...	...	...	0x3B	0x59	59	0x3C–0xFF	0x5A–0xFF	Reserved
MINUTES		Minute of hour																											
Binary	BCD																												
0x00	0x00	0																											
0x01	0x01	1																											
...	...	...																											
0x0A	0x10	10																											
...	...	...																											
0x3B	0x59	59																											
0x3C–0xFF	0x5A–0xFF	Reserved																											

### 37.4.8 RTC Seconds Alarm Register (RTC\_ALM\_SEC)

RTC\_ALM\_SEC configures the seconds setting of the alarm. The alarm setting can be read or written anytime. This register is reset to default value on software reset. In BCD mode, all eight bits indicate the time.

The INC and DEC bits provide the option for the MCU to perform a correction on seconds counter to compensate for leap seconds. Write to these bits adds or subtracts one from the seconds counter and read returns zeros.

Prior to reading or writing to INC or DEC, read the RTC\_SR[INVAL] bit to determine that the counters are stable and they can be incremented or decremented. The INVAL bit ensures that no operation is done at the boundary of a second when counters change value.

Address: 0xFC0A_800E (RTC_ALM_SEC)										Access: User read/write							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
W							INC	DEC									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 37-9. RTC Seconds Alarm Register (RTC\_ALM\_SEC)

Table 37-10. RTC\_ALM\_SEC Field Descriptions

Field1	Description																												
15–10	Reserved, must be cleared.																												
9 INC	Increment one second. Increments the seconds counter to compensate for leap seconds or to perform fine trimming when needed. 0 No effect 1 Increments the seconds counter. This bit self-clears on next positive clock edge.																												
8 DEC	Decrement one second. Decrements the seconds counter to compensate for leap seconds or to perform fine trimming when needed. 0 No effect 1 Decrements the seconds counter. This bit self-clears on next positive clock edge.																												
7–0 SECONDS	Second of minute alarm. Indicates the second alarm, depending on the value of RTC_CR[BCDEN].																												
	<table border="1"> <thead> <tr> <th colspan="2">SECONDS</th> <th rowspan="2">Second of minute</th> </tr> <tr> <th>Binary</th> <th>BCD</th> </tr> </thead> <tbody> <tr> <td>0x00</td> <td>0x00</td> <td>0</td> </tr> <tr> <td>0x01</td> <td>0x01</td> <td>1</td> </tr> <tr> <td>...</td> <td>...</td> <td>...</td> </tr> <tr> <td>0x0A</td> <td>0x10</td> <td>10</td> </tr> <tr> <td>...</td> <td>...</td> <td>...</td> </tr> <tr> <td>0x3B</td> <td>0x59</td> <td>59</td> </tr> <tr> <td>0x3C–0xFF</td> <td>0x5A–0xFF</td> <td>Reserved</td> </tr> </tbody> </table>			SECONDS		Second of minute	Binary	BCD	0x00	0x00	0	0x01	0x01	1	...	...	...	0x0A	0x10	10	...	...	...	0x3B	0x59	59	0x3C–0xFF	0x5A–0xFF	Reserved
SECONDS		Second of minute																											
Binary	BCD																												
0x00	0x00	0																											
0x01	0x01	1																											
...	...	...																											
0x0A	0x10	10																											
...	...	...																											
0x3B	0x59	59																											
0x3C–0xFF	0x5A–0xFF	Reserved																											

### 37.4.9 RTC Control Register (RTC\_CR)

The control register governs all operations inside the RTC. This register specifies software reset, compensation, and write protection.

The write protect bits are the only bits that are freely writeable. The rest of the bits, registers, and RAM are protected with a write protect mechanism. The write protect bits (RTC\_CR[WE]) are self-clearing and always return zeros on reads.

Address: 0xFC0A\_8010 (RTC\_CR)

Access: User read/write

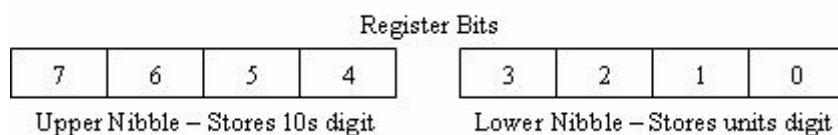
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	BCD EN	DST EN	0	0	AM	0	0	0
W								SWR						0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 37-10. RTC Control Register (RTC\_CR)

Table 37-11. RTC\_CR Field Descriptions

Field1	Description
15–13	Reserved, must be cleared.
8 SWR	Software reset. This self-clearing bit is automatically cleared on the next clock after a write. 0 No software reset 1 Software reset. Clears the contents of the alarm, interrupt (status & enable) registers and has no effect on DST, standby RAM, up counter, time, and calendaring registers.
7 BCDEN	BCD format enable. Controls whether the read/write value of the time and date registers is in binary or BCD format. 0 Binary format 1 Binary-coded decimal (BCD) format
6 DSTEN	Daylight saving time enable. Enables automatic adjustment of time during daylight saving time. 0 Disabled 1 Enabled
5–4	Reserved, must be cleared.
3–2 AM	Alarm match. Defines which time and calendar counters are used for matching and generating an alarm. 00 Seconds, minutes, and hours matched 01 Seconds, minutes, hours, and days matched 10 Seconds, minutes, hours, days, and months matched 11 Seconds, minutes, hours, days, months, and year matched
1–0 WE	Write enable bits. Controls the entry and exit into/from the register/memory write protection mode. This field is self clearing. To enable write protection: Write 10 to this field. To disable write protection: Write 00 to this field, then Write 01 to this field, then Write 11 to this field, then Write 10 to this field. <b>Note:</b> When the registers are unlocked, they remain unlocked for two seconds and are then automatically locked. Also, immediately following power-on-reset, the registers are unlocked. After 15 seconds, they are automatically locked.

Depending on the BCD format enable bit, the width of time (seconds, minutes and hours) and day related registers changes and the values in the register becomes as follows:



The table below shows how data gets changed on reads. The reverse happens for writes. Data input is in BCD and converted to binary before storing in appropriate registers.

Table 37-12. Register Changes for BCD Format

Register Name	RTC_CR[BCDEN] = 0	RTC_CR[BCDEN] = 1
	Values stored in Binary	Values stored in BCD
RTC_HOURMIN	Bits 12:8 – Hour Bits 5:0 – Minutes	Bits 15:8 – Hour Bits 7:0 – Minutes
RTC_SECONDS	Bits 5:0 – Seconds	Bits 7:0 – Seconds
RTC_DAYS	Bits 4:0 – Days	Bits 7:0 – Days
RTC_MONYEAR	Bits 3:0 – Months	Bits 7:0 – Months
RTC_ALM_HOURMIN	Bits 12:8 – Hour Bits 5:0 – Minutes	Bits 15:8 – Hour Bits 7:0 – Minutes
RTC_ALM_SECONDS	Bits 5:0 – Seconds	Bits 7:0 – Seconds
RTC_ALM_DAYS	Bits 4:0 – Days	Bits 7:0 – Days
RTC_ALM_MONYR	Bits 3:0 – Months	Bits 7:0 – Months
RTC_DST_HOUR	Bits 12:8 – Hour Start Bits 4:0 – Hour End	Bits 15:8 – Hour Start Bits 7:0 – Hour End
RTC_DST_DAY	Bits 12:8 – Day Start Bits 4:0 – Day End	Bits 15:8 – Day Start Bits 7:0 – Day End
RTC_DST_MONTH	Bits 11:8 – Months Start Bits 3:0 – Months End	Bits 11:8 – Months Start Bits 3:0 – Months End
RTC_TTSR_HM	Bits 12:8 – Hour Bits 5:0 – Minutes	Bits 15:8 – Hour Bits 7:0 – Minutes
RTC_TTSR_SEC	Bits 5:0 – Seconds	Bits 7:0 – Seconds
RTC_COUNT_DN	Bits 5:0 – Minutes	Bits 7:0 – Minutes

### 37.4.10 RTC Status Register (RTC\_SR)

This register indicates the status of various processes inside the RTC. Status of crystal or temperature compensation can be obtained. This register also helps the MCU to read time or date register when their values are stable and not changing. A software reset clears this register. The compensation done bit is cleared on reads.

Address: 0xFC0A\_8012 (RTC\_SR)

Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0	0	PORB	0	WPE	0	BERR	CDON	INVAL
W													w1c	w1c		
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 37-11. RTC Status Register (RTC\_SR)

**Table 37-13. RTC\_SR Field Descriptions**

<b>Field1</b>	<b>Description</b>
15–7	Reserved, must be cleared.
6 PORB	Boot source. Indicates if the device is booting after a power-on reset or standby mode exit. 0 Booting after standby mode exit 1 Booting after POR
5	Reserved, must be cleared.
4 WPE	Write protect enable. Indicates the registers are in locked mode and writes to the registers are disabled. Any write access made to the register space when write protection is enabled generates a transfer error. 0 Write protection disabled 1 Write protection enabled
3	Reserved, must be cleared.
2 BERR	Bus error. Indicates that a read or write cycle was started by the MCU while INVAL was set. Write access to time/date registers is nullified (terminate normally) and no register value is changed. Reads while INVAL is set returns 0xFFFF and no transfer error is generated.
1 CDON	Compensation done. Read-only bit that is cleared when one is written to it. 0 Compensation busy or not enabled 1 Compensation complete <b>Note:</b> This bit is set a few oscillator cycles before the actual compensation interval completes, so that back-to-back compensation can be enabled.
0 INVAL	Invalid time bit. Indicates the time is invalid or changing and should not be read. This bit is set an oscillator clock cycle before and after the 1 Hz (seconds) boundary. Write access to time/date registers is nullified (terminate normally) and no register value is changed. Read during INVAL bit asserted returns 0xFFFF and no transfer error is generated. 0 Time is valid and can be read 1 Time/date is invalid and counter values are changing

### 37.4.11 RTC Interrupt Status Register (RTC\_ISR)

The RTC\_ISR register indicates the status of the various real-time clock interrupts. When an event of the types included in this register occurs then the bit is set regardless of its corresponding RTC\_IER bit. The status bits are cleared by writing one to them, which also clears the interrupt. Interrupts may occur while the system clock is idle or in standby mode. When the system enters the active power mode, an interrupt is generated to the CPU.

Address: 0xFC0A_8014 (RTC_ISR)																Access: User read/write			
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
R	SAM7	SAM6	SAM5	SAM4	SAM3	SAM2	SAM1	SAM0	2HZ	1HZ	MIN	HR	DAY	ALM	STW	1			
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    1

**Figure 37-12. RTC Interrupt Status Register (RTC\_ISR)**

**Table 37-14. RTC\_ISR Field Descriptions**

<b>Field1</b>	<b>Description</b>																																														
15–8 SAMn	Sampling timer 7–0 interrupt flags. Indicates an interrupt has occurred at the corresponding sampling rate, equal to $2^{n+2}$ Hz. 0 No SAM7–0 interrupt has occurred 1 A SAM7–0 interrupt has occurred																																														
7 2HZ	2 Hz flag. Indicates an interrupt has occurred at a 2 Hz sampling rate.																																														
6 1HZ	1 Hz flag. Indicates that the seconds counter has incremented.																																														
5 MIN	Minutes flag. Indicates that the minutes counter has incremented.																																														
4 HR	Hour flag. Indicates that the hour counter has incremented.																																														
3 DAY	Day flag. Indicates that the day counter has incremented.																																														
2 ALM	Alarm flag. Indicates that the alarm value programmed matches the counter values. Depending on the setting of RTC_CR[AM], this interrupt flag is set only when the seconds, minutes, hours, days, month, and year counters match their respective alarm register values.																																														
	<table border="1"> <thead> <tr> <th rowspan="2">RTC_CR[AM]</th> <th colspan="6">Counters Matched</th> <th rowspan="2">Alarm Type</th> </tr> <tr> <th>SEC</th> <th>MIN</th> <th>HOUR</th> <th>DAY</th> <th>MONTH</th> <th>YEAR</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>—</td> <td>—</td> <td>—</td> <td>Daily</td> </tr> <tr> <td>01</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>—</td> <td>—</td> <td>Monthly</td> </tr> <tr> <td>10</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>—</td> <td>Yearly</td> </tr> <tr> <td>11</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>✓</td> <td>One-Time</td> </tr> </tbody> </table>	RTC_CR[AM]	Counters Matched						Alarm Type	SEC	MIN	HOUR	DAY	MONTH	YEAR	00	✓	✓	✓	—	—	—	Daily	01	✓	✓	✓	✓	—	—	Monthly	10	✓	✓	✓	✓	✓	—	Yearly	11	✓	✓	✓	✓	✓	✓	One-Time
RTC_CR[AM]	Counters Matched						Alarm Type																																								
	SEC	MIN	HOUR	DAY	MONTH	YEAR																																									
00	✓	✓	✓	—	—	—	Daily																																								
01	✓	✓	✓	✓	—	—	Monthly																																								
10	✓	✓	✓	✓	✓	—	Yearly																																								
11	✓	✓	✓	✓	✓	✓	One-Time																																								
1 STW	Countdown timer expiration flag. Indicates that the programmed count in the countdown timer has expired.																																														
0	Reserved, must be set.																																														

### 37.4.12 RTC Interrupt Enable Register (RTC\_IER)

The real-time clock interrupt enable register (RTC\_IER) enables/disables the various real-time clock interrupts. Disabling an interrupt bit has no effect on its corresponding status bit.

A single interrupt is output from this module which is an OR'd version of all interrupts. Read RTC\_ISR in the interrupt status routine to determine which interrupt has occurred.

Address: 0xFC0A\_8016 (RTC\_IER)

Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R W	SAM7	SAM6	SAM5	SAM4	SAM3	SAM2	SAM1	SAM0	2HZ	1HZ	MIN	HR	DAY	ALM	STW	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 37-13. RTC Interrupt Enable Register (RTC\_IER)

Table 37-15. RTC\_IER Field Descriptions

Field1	Description
15–8 SAM <sub>n</sub>	Sampling timer 7–0 interrupt enable. 0 Disable 1 Enable
7 2Hz	2 Hz frequency interrupt enable. 0 Disable 1 Enable
6 1Hz	1 Hz frequency interrupt enable. 0 Disable 1 Enable
5 MIN	Minutes interrupt enable. 0 Disable 1 Enable
4 HR	Hour interrupt enable. 0 Disable 1 Enable
3 DAY	Day interrupt enable. 0 Disable 1 Enable
2 ALM	Alarm interrupt enable. 0 Disable 1 Enable
1 STW	Countdown timer expiration interrupt enable. 0 Disable 1 Enable
0	Reserved, must be set.

### 37.4.13 RTC Countdown Timer Register (RTC\_COUNT\_DN)

This counter can generate an interrupt on a minute boundary to, for example, turn off the LCD controller after five minutes of inactivity. The countdown timer is decremented by the minute tick output from the real-time clock; so, the average tolerance of the count is 0.5 minutes. For better accuracy, enable the countdown timer by waiting for RTC\_ISR[MIN] to set. The actual delay includes the seconds from setting the countdown to the next minute tick. An interrupt is generated when the timer reaches 0. The value in the register shows the current value of the counter at all times.

Loading the countdown timer counter with 0 has no effect. A software reset clears the countdown timer count.

In BCD mode, the maximum countdown value is 99, since a value bigger than 99 cannot be displayed in eight bits. Programming a value bigger than 99 does not ensure correct reading in BCD mode. However the binary value can still be read.

Address: 0xFC0A_8018 (RTC_COUNT_DN)																Access: User read/write								
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0								
R	0	0	0	0	0	0	0	0	COUNT															
W									0	0	0	0	0	0	0	0								
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0								

Figure 37-14. RTC Countdown Timer Register (RTC\_COUNT\_DN)

Table 37-16. RTC\_COUNT\_DN Field Descriptions

Field1	Description																									
15–8	Reserved, must be cleared.																									
7–0 COUNT	Countdown counter value in minutes. <b>Note:</b> In BCD mode, the valid values are 0–99. In binary mode, the valid values are 0x00–0x7F as shown below.																									
	<table border="1"> <thead> <tr> <th>RTC_CR [BCDEN]</th> <th>COUNT</th> <th>Countdown counter value</th> </tr> </thead> <tbody> <tr> <td rowspan="7">1 (BCD mode)</td> <td>0x00–0x09</td> <td>0–9 minutes</td> </tr> <tr> <td>0x0A–0x0F</td> <td>Reserved</td> </tr> <tr> <td>0x10–0x19</td> <td>10–19 minutes</td> </tr> <tr> <td>0x1A–0x1F</td> <td>Reserved</td> </tr> <tr> <td>...</td> <td>...</td> </tr> <tr> <td>0x90–0x99</td> <td>90–99 minutes</td> </tr> <tr> <td>0x9A–0xFF</td> <td>Reserved</td> </tr> <tr> <td rowspan="2">0 (binary mode)</td> <td>0x00–0x7F</td> <td>0–127 minutes</td> </tr> <tr> <td>0x80–FF</td> <td>Reserved</td> </tr> </tbody> </table>	RTC_CR [BCDEN]	COUNT	Countdown counter value	1 (BCD mode)	0x00–0x09	0–9 minutes	0x0A–0x0F	Reserved	0x10–0x19	10–19 minutes	0x1A–0x1F	Reserved	...	...	0x90–0x99	90–99 minutes	0x9A–0xFF	Reserved	0 (binary mode)	0x00–0x7F	0–127 minutes	0x80–FF	Reserved		
RTC_CR [BCDEN]	COUNT	Countdown counter value																								
1 (BCD mode)	0x00–0x09	0–9 minutes																								
	0x0A–0x0F	Reserved																								
	0x10–0x19	10–19 minutes																								
	0x1A–0x1F	Reserved																								
	...	...																								
	0x90–0x99	90–99 minutes																								
	0x9A–0xFF	Reserved																								
0 (binary mode)	0x00–0x7F	0–127 minutes																								
	0x80–FF	Reserved																								

### 37.4.14 RTC Configuration Data Register (RTC\_CFG\_DATA)

This register controls the 32-kHz oscillator. Software reset clears the register to its default state. This oscillator requires an external RTC crystal.

Address: 0xFC0A\_8020 (RTC\_CFG\_DATA)

Access: User read/write

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

Figure 37-15. RTC Configuration Data Register (RTC\_CFG\_DATA)

Table 37-17. RTC\_CFG\_DATA Field Descriptions

Field1	Description
15–5	Reserved, must be cleared.
4 OSCBYP <sup>1</sup>	Oscillator bypass. 0 On-chip 32-kHz oscillator not bypassed 1 On-chip 32-kHz oscillator bypassed
3 OSCEN	Oscillator enable 0 On-chip 32-kHz oscillator disabled 1 On-chip 32-kHz oscillator enabled
2–0	Reserved, must be cleared.

<sup>1</sup> Even in oscillator bypass mode, the on-chip oscillator must be enabled by setting the OSCEN bit.

### 37.4.15 RTC Daylight Saving Time Hour Register (RTC\_DST\_HOUR)

RTC\_DST\_HOUR stores the time in hours when the daylight savings is to be applied and reversed. Program the correct hour value (0–23) for your regional settings.

For example, if daylight saving time starts at 2:00 AM on March 25 and ends at 2:00 AM on October 28, then time advances or falls back at 1:59 AM. Therefore, write 0x0101 to this register, not 0x0202. The 59 minute count is automatically checked inside the RTC and is not required to be programmed. This register has no effect from a software reset.

In BCD mode, all eight bits indicate the value of hours in both fields.

Address: 0xFC0A\_8022 (RTC\_DST\_HOUR)

Access: User read/write

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

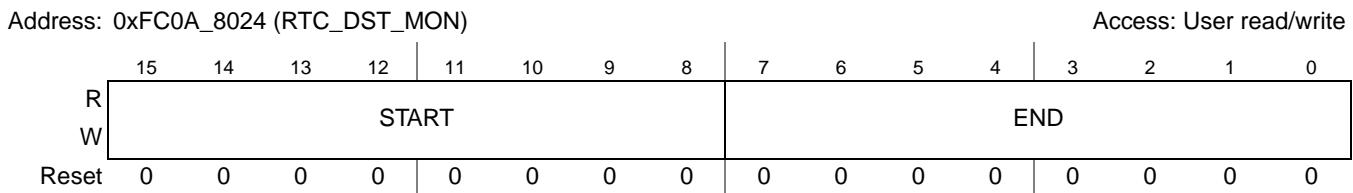
Figure 37-16. RTC Daylight Saving Time Hour Register (RTC\_DST\_HOUR)

**Table 37-18. RTC\_DST\_HOUR Field Descriptions**

Field1	Description
15–8 START	Hour value when daylight saving time is to start. <b>Note:</b> In BCD mode, valid values are 0–23. In binary mode, valid values are 0x00–0x17
7–0 END	Hour value when daylight saving time is to end. <b>Note:</b> In BCD mode, valid values are 0–23. In binary mode, valid values are 0x00–0x17

### 37.4.16 RTC Daylight Saving Time Month Register (RTC\_DST\_MON)

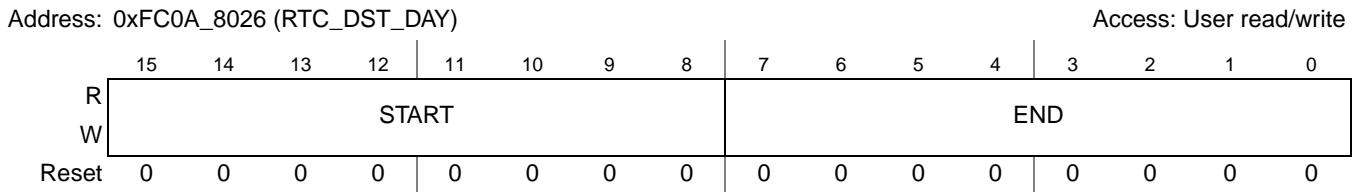
This register stores the month when daylight saving time starts and stops. Program the correct month value (1–12) for the regional settings. For example, if daylight saving time starts on March 25 and ends on October 28, write 0x030A to this register (in binary format mode). Software reset does not affect this register.

**Figure 37-17. RTC Daylight Saving Time Month Register (RTC\_DST\_MON)****Table 37-19. RTC\_DST\_MON Field Descriptions**

Field1	Description
15–8 START	Month value when daylight saving time is to start. <b>Note:</b> In BCD mode, valid values are 1–12. In binary mode, valid values are 0x01–0x0C
7–0 END	Month value when daylight saving time is to end. <b>Note:</b> In BCD mode, valid values are 1–12. In binary mode, valid values are 0x01–0x0C

### 37.4.17 RTC Daylight Saving Time Day Register (RTC\_DST\_DAY)

This register stores the day when daylight saving time starts and stops. Program the correct day value (1–31) for the regional settings. For example, if daylight saving time starts on March 25 and ends on October 28, write 0x191C in this register (in binary format mode). Software reset does not affect this register.

**Figure 37-18. RTC Daylight Saving Time Day Register (RTC\_DST\_DAY)**

**Table 37-20. RTC\_DST\_DAY Field Descriptions**

Field1	Description
15–8 START	Day of month value when Daylight saving time is to start. <b>Note:</b> In BCD mode, valid values are 1–31. In binary mode, valid values are 0x01–0x1F
7–0 END	Day of month value when Daylight saving time is to end. <b>Note:</b> In BCD mode, valid values are 0–31. In binary mode, valid values are 0x01–0x1F

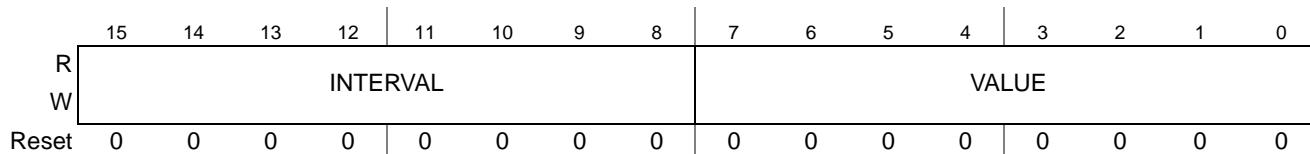
### 37.4.18 RTC Compensation Register (RTC\_COMPEN)

RTC\_COMPEN stores the compensation value to correct the 1 Hz clock. It indicates the number of oscillator clock cycles to be added or removed. The value that is stored is in two's complement format with a range of -128 to +127. RTC\_SR[CDON] is set when the compensation cycle is complete.

The RTC continues to compensate with this value until it is disabled by clearing the INTERVAL bit. When programmed a new value, it takes affect when the current compensation cycle completes.

Address: 0xFC0A\_8028 (RTC\_COMPEN)

Access: User read/write

**Figure 37-19. RTC Compensation Register (RTC\_COMPEN)****Table 37-21. RTC\_COMPEN Field Descriptions**

Field1	Description
15–8 INTERVAL	Compensation interval. Indicates the window in seconds when the compensation must be carried out. A non-zero value starts the compensation logic. This register is cleared when RTC_SR[CDON] is set. 0x00 Compensation logic disabled 0x01 1 second ... 0xFF 255 seconds
7–0 VALUE	Compensation value. Two's complement number which indicates the number of oscillator clock cycles the RTC requires to compensate for the specified compensation interval. 0x00 No compensation needed 0x01 +1 oscillator clock ... 0x7F +127 oscillator clocks 0x80 -128 oscillator clocks ... 0xFF -1 oscillator clocks

### 37.4.19 RTC Up-Counter High Register (RTC\_UP\_CNTRH)

This register tracks the number of user-specified events. For example, the energy units consumed over a period. This register increments upon rollover from RTC\_UP\_CTRL. Software reset has no effect on this counter.

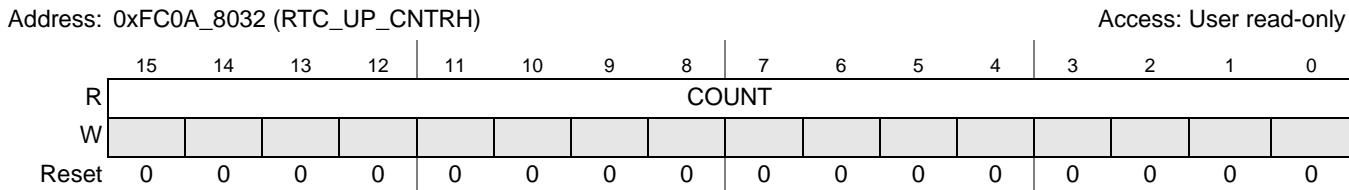


Figure 37-20. RTC Up-Counter High Register (RTC\_UP\_CNTRH)

Table 37-22. RTC\_UP\_CNTRH Field Descriptions

Field1	Description
15-0 COUNT	Up-counter register value. Upper 16 bits of the 32-bit counter. Value increments on rollover from RTC_UP_CNTRL.

### 37.4.20 RTC Up-Counter Low Register (RTC\_UP\_CNTRL)

This register tracks the number of user-specified events. For example, the energy units consumed over a period. Writing to the lower eight bits increments the value in this register. Overflows are carried into the RTC\_UP\_CNTRH register. Software reset has no effect on this counter.

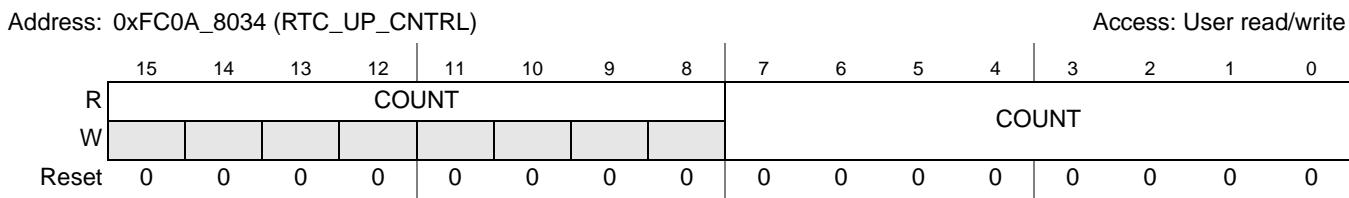


Figure 37-21. RTC Up-Counter Low Register (RTC\_UP\_CNTRL)

Table 37-23. RTC\_UP\_CNTRL Field Descriptions

Field1	Description
15-0 COUNT	Up-counter register value. Lower 16 bits of the up counter. Only the lower byte is writeable and a carry generated from this byte overflows and increments the upper byte and RTC_UP_CNTRH.

## 37.5 Functional Description

### 37.5.1 Basic Data Flow

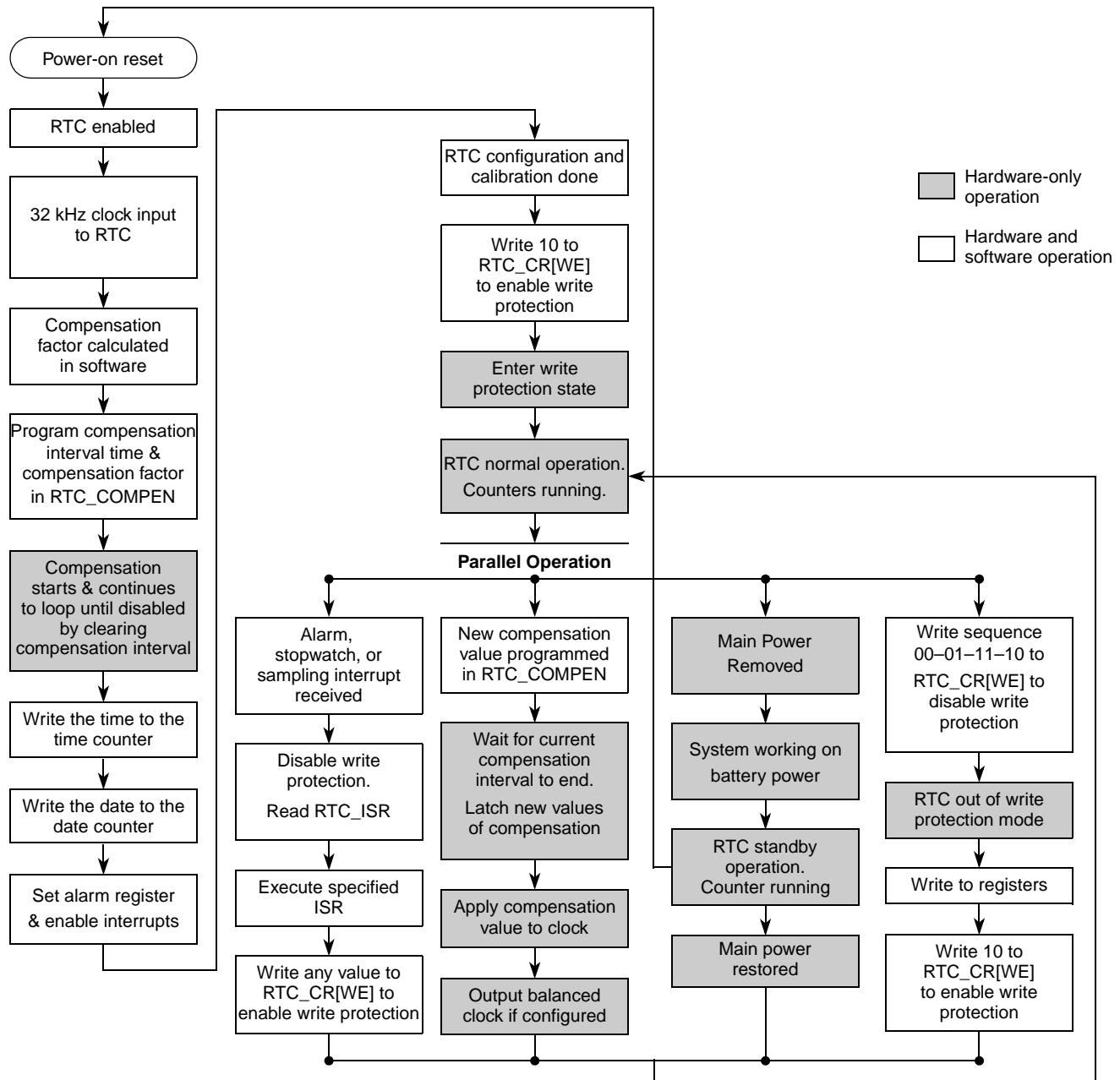


Figure 37-22. Top Level Data Flow

#### 37.5.1.1 Configuration

The initial flow that must be completed after each power-on reset:

- Set the date and time in the RTC registers

- Set other control information needed for the RTC to function
- Enable the various interrupts and alarm time as required
- Store any critical data needed in the event of main voltage loss in the standby RAM. This can also be done during normal operation.
- The RTC has configuration bits that can be used for application dependent protocol and these bits need to retain state during main voltage removal. These are programmed by the CPU.
- After the RTC is configured, place it in the write protect state. If not, this is automatically done after 15 seconds of power on.

### 37.5.1.2 Normal Operation

During normal operation of the RTC:

- All write accesses to registers are blocked. You may gain write access to modify the contents on the registers by following the RTC\_CR[WE] write sequence described in [Section 37.4.9, “RTC Control Register \(RTC\\_CR\)”](#). This allows you to:
  - Service any RTC interrupts
  - Update the standby RAM
  - Change control information
- Interrupts can occur when:
  - The counters match the alarm registers
  - A sampling timer expires

Disable write-protect mode, service these interrupts, and place the RTC back in write-protect mode. If not, write protect is enabled two seconds after they are unlocked.

### 37.5.1.3 Standby Operation

Standby operation occurs when the RTC is on battery voltage only:

- When the main voltage falls below a certain threshold, the RTC switches to battery power and functions normally.
- When the MCU voltage is restored, the RTC switches back to this supply.
- If the battery drains and power is supplied again to the RTC (by a new battery or MCU power), a power on reset is sent to the RTC and all registers are reset. So, your application needs recalibrate the RTC.

### 37.5.1.4 Calibration

- Perform calculations external to the RTC to determine the correction factor in the 1 Hz clock to remove any variations that might creep into the 32 kHz oscillator clock due to temperature or the crystal used.
- Program this correction factor into the RTC registers. Hardware within the RTC compensates the clock.
- The compensation is performed on a user-defined interval

## 37.6 Initialization/Application Information

### 37.6.1 Compensation

The compensation circuit provides an accurate and wide compensation range, which is suitable for many crystals, and can correct errors as high as 3906 ppm and as low as 0.119 ppm. The same hardware logic supports temperature and frequency compensation.

To perform temperature compensation:

1. Maintain a look-up-table which lists the change in frequency for each degree change in temperature.
2. Measure the external temperature periodically via a temperature sensor connected to the A/D converter.
3. Use the look-up-table to determine the compensation factor and write the value to be compensated to the RTC\_COMPEN register. Based on the value written the hardware add/remove pulses accordingly to adjust the 1 Hz frequency due to variation on temperature.

To perform crystal compensation:

1. Firmware calculates the correction using crystal characteristics.
2. Set the correction factor in two's complement format in the RTC\_COMPEN register. Based on the values written in the RTC\_COMPEN register, the circuit compensates by adding/skipping pulses in the oscillator clock signal.

There are two important components in the compensation algorithm. These are defined as:

- Compensation/correction value — Two's complement value that the RTC oscillator clock is modified by adding or removing pulses from it.
- Compensation interval — Duration the correction value is applied. This is the time the RTC adds or removes pulses, ensuring that the compensation interval is close to the interval obtained with an ideal 1 Hz clock.

Vital statistics:

- Compensation range: -128 to +127 RTC oscillator clocks
- Compensation interval range: 1 to 255 seconds (0 disables compensation)
- Selection criteria: Compensation is enabled when RTC\_COMPEN[INTERVAL] is non-zero. Clear this field to disable compensation.

#### 37.6.1.1 Compensation Flow

The operation starts in an idle state waiting for the firmware to enable compensation. Since the same hardware logic is used for temperature and crystal compensation, the firmware provides a value that accounts for correction for both temperature and crystal. When enabled, the compensation cycles are added or removed until the compensation interval expires. When the compensation interval completes, the RTC\_SR[CDON] bit is set and if compensation is still enabled, the next compensation cycle starts. The compensation state machine returns to the idle state when you clear the compensation interval. A newly programmed value is picked only when the current compensation cycle has completed.

Figure below shows the flow chart for the logical compensation flow of the state machine which has been explained in subsequent pages.

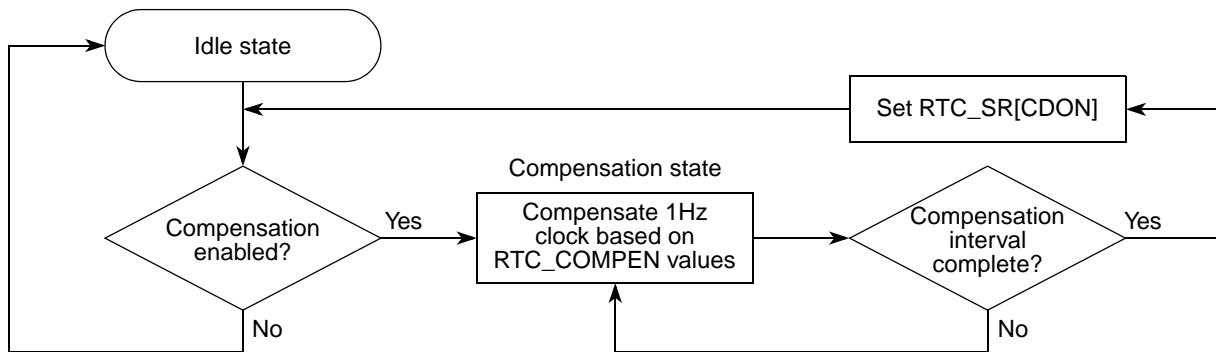


Figure 37-23. Compensation Control Flow

### 37.6.1.2 Compensation Logic Hardware:

The compensation logic hardware comprises of a simple counter which divides the 32 kHz clock down to 1 Hz by counting up to 32,767. To add or remove pulses the start point of the counter is shifted and the counter still counts up to 32,767 to generate a balanced 1 Hz clock. The state machine for this block controls the loading of correction value into the counter and ensures that each compensation window is always aligned to the seconds boundary. Switching to newly programmed compensation values is done when the compensation interval of current run is complete and CPU has not disabled the compensation logic. If no new value is programmed, the state machine continues to perform compensation with the previously programmed values until compensation is disabled by writing zero to the compensation interval. The figure below shows the block diagram for the compensation block.

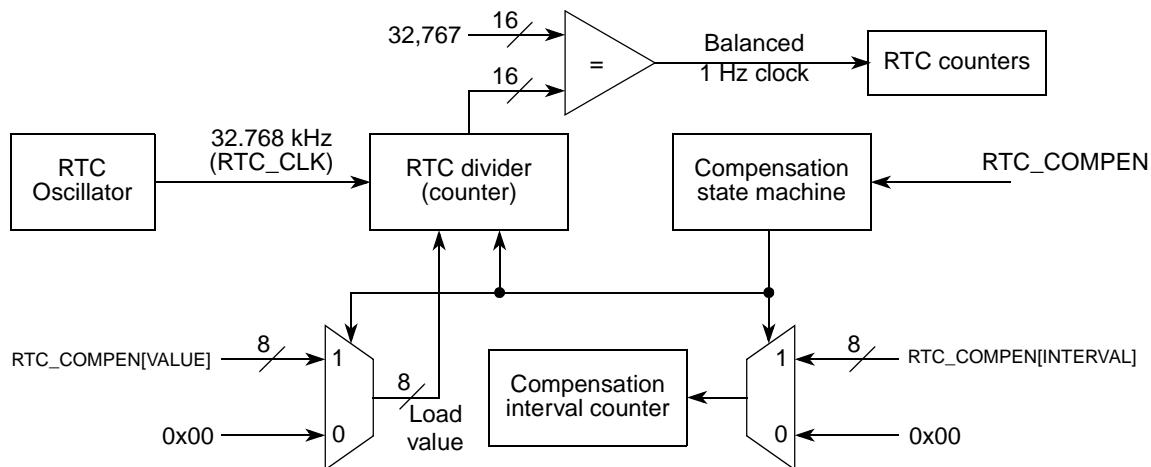


Figure 37-24. Compensation Logic Block Diagram

### 37.6.1.3 Recommendation for Optimal Compensation

Since the addition and removal of pulses is done during the first second of the compensation interval, you have the option of finding the compensation factor over a period of time. Then calculate the correction

factor and enable the compensation hardware every second for better accuracy. As a result, the 1 Hz clock generated has a uniform period.

## 37.6.2 Write Protection

This logic protects the RTC registers and standby RAM from any spurious updates that can occur with run-away code. The logic is based on a state machine that monitors the values written to RTC\_CR[WE]. By default, unconditional write access is allowed to these bits only.

### 37.6.2.1 Write Protection Flow

To enable write protection, write 10 to these bits. To disable write protection, write the sequence 00, 01, 11, 10 to these bits.

After a power-on reset, the write-protect mechanism is disabled, allowing user code to calibrate the RTC clock, set the time in the clock registers, and set the date in the calendar registers. When that is complete, enable write protection mode. If not, the registers are placed into write protect mode 15 seconds after power on. After disabling write protect mode to update registers, write protect mode is automatically enabled after two seconds.

The internal bus clock is essentially asynchronous to the 1 Hz clock that generates the timeouts. Hence after unlocking the registers, the actual duration of unlock is less than two seconds. To have a complete two second duration, unlock the registers at the seconds boundary indicated by the 1 Hz interrupt (see RTC\_ISR[1HZ] in [Section 37.4.11, “RTC Interrupt Status Register \(RTC\\_ISR\)”](#)).

A write access made to the register space when write protection is enabled generates a transfer error.

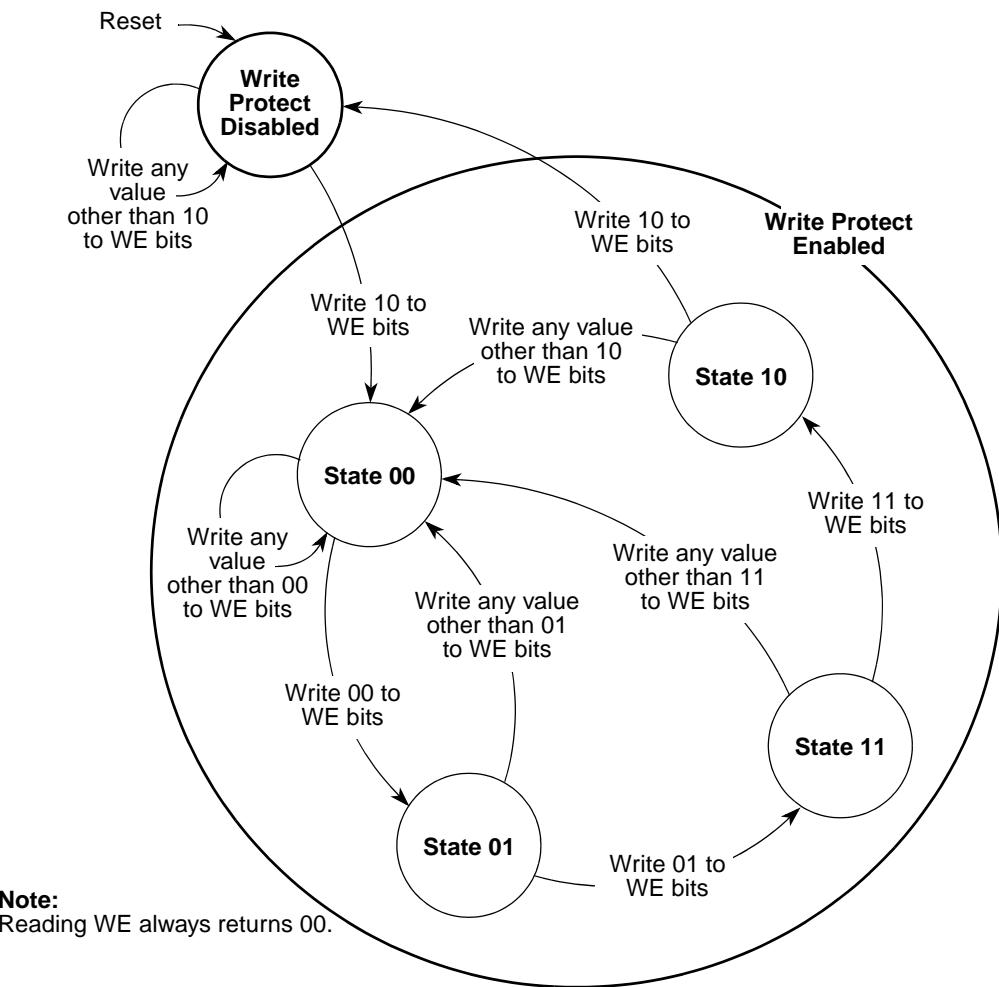


Figure 37-25. Write Protect State Machine

# Chapter 38

## Programmable Interrupt Timers (PIT0–PIT3)

### 38.1 Introduction

This chapter describes the operation of the four programmable interrupt timer modules: PIT0–PIT3.

#### 38.1.1 Overview

Each PIT is a 16-bit timer that provides precise interrupts at regular intervals with minimal processor intervention. The timer can count down from the value written in the modulus register or it can be a free-running down-counter.

#### 38.1.2 Block Diagram

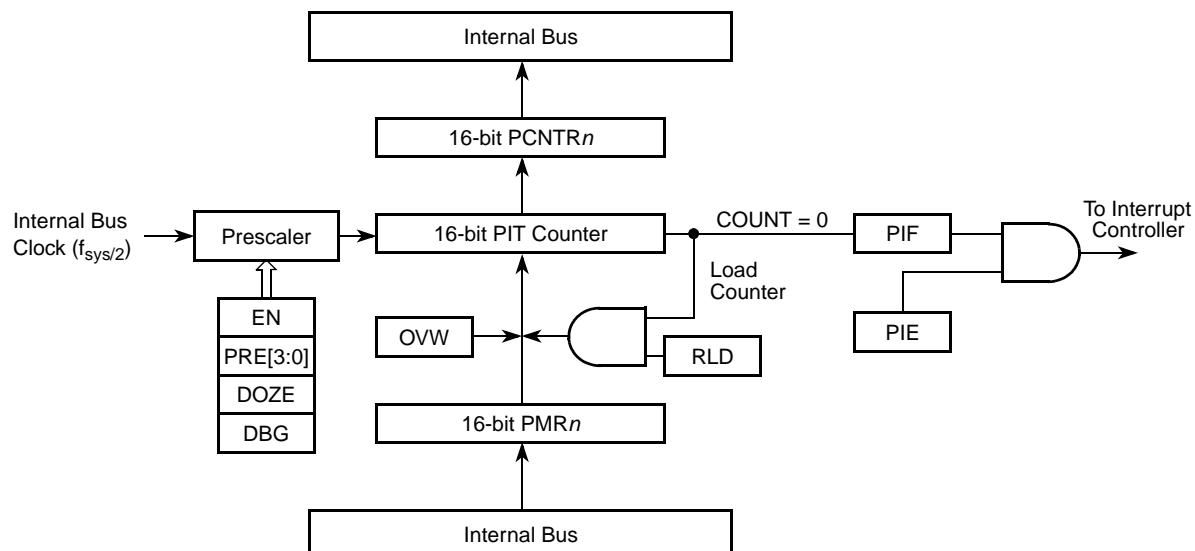


Figure 38-1. PIT Block Diagram

#### 38.1.3 Low-Power Mode Operation

This subsection describes the operation of the PIT modules in low-power modes and debug mode of operation. Low-power modes are described in the power management module, [Chapter 9, “Power Management.”](#) Table 38-1 shows the PIT module operation in low-power modes and how it can exit from each mode.

**NOTE**

The low-power interrupt control register (LPICR) in the system control module specifies the interrupt level at or above which the device can be brought out of a low-power mode.

**Table 38-1. PIT Module Operation in Low-power Modes**

Low-power Mode	PIT Operation	Mode Exit
Wait	Normal	N/A
Doze	Normal if PCSR $n$ [DOZE] cleared, stopped otherwise	Any interrupt at or above level in LPICR, exit doze mode if PCSR $n$ [DOZE] is set. Otherwise interrupt assertion has no effect.
Stop	Stopped	No
Debug	Normal if PCSR $n$ [DBG] cleared, stopped otherwise	No. Any interrupt is serviced upon normal exit from debug mode

In wait mode, the PIT module continues to operate as in run mode and can be configured to exit the low-power mode by generating an interrupt request. In doze mode with the PCSR $n$ [DOZE] bit set, PIT module operation stops. In doze mode with the PCSR $n$ [DOZE] bit cleared, doze mode does not affect PIT operation. When doze mode is exited, PIT continues operating in the state it was in prior to doze mode. In stop mode, the internal bus clock is absent and PIT module operation stops.

In debug mode with the PCSR $n$ [DBG] bit set, PIT module operation stops. In debug mode with the PCSR $n$ [DBG] bit cleared, debug mode does not affect PIT operation. When debug mode is exited, the PIT continues to operate in its pre-debug mode state, but any updates made in debug mode remain.

## 38.2 Memory Map/Register Definition

This section contains a memory map (see [Table 38-2](#)) and describes the register structure for PIT0–PIT3.

**NOTE**

Longword accesses to any of the programmable interrupt timer registers results in a bus error. Only byte and word accesses are allowed.

**Table 38-2. Programmable Interrupt Timer Modules Memory Map**

Address	Register	Width (bits)	Access <sup>1</sup>	Reset Value	Section/Page
PIT 0 PIT 1 PIT 2 PIT 3					
<b>Supervisor Access Only Registers<sup>2</sup></b>					
0xFC08_0000 0xFC08_4000 0xFC08_8000 0xFC08_C000	PIT Control and Status Register (PCSR $n$ )	16	R/W	0x0000	<a href="#">38.2.1/38-3</a>

**Table 38-2. Programmable Interrupt Timer Modules Memory Map (continued)**

Address	Register	Width (bits)	Access <sup>1</sup>	Reset Value	Section/Page
PIT 0 PIT 1 PIT 2 PIT 3					
0xFC08_0002 0xFC08_4002 0xFC08_8002 0xFC08_C002	PIT Modulus Register (PMR $n$ )	16	R/W	0xFFFF	<a href="#">38.2.2/38-5</a>
<b>User/Supervisor Access Registers</b>					
0xFC08_0004 0xFC08_4004 0xFC08_8004 0xFC08_C004	PIT Count Register (PCNTR $n$ )	16	R	0xFFFF	<a href="#">38.2.3/38-5</a>

<sup>1</sup> Accesses to reserved address locations have no effect and result in a cycle termination transfer error.

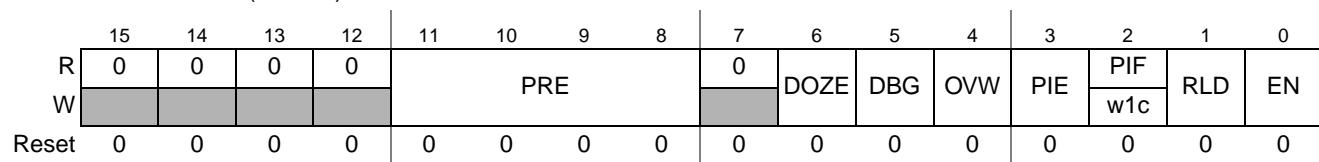
<sup>2</sup> User mode accesses to supervisor only addresses have no effect and result in a cycle termination transfer error.

### 38.2.1 PIT Control and Status Register (PCSR $n$ )

The PCSR $n$  registers configure the corresponding timer's operation.

Address: 0xFC08\_0000 (PCSR0)  
0xFC08\_4000 (PCSR1)  
0xFC08\_8000 (PCSR2)  
0xFC08\_C000 (PCSR3)

Access: Supervisor  
read/write

**Figure 38-2. PCSR $n$  Register**

**Table 38-3. PCSR $n$  Field Descriptions**

Field	Description		
15–12	Reserved, must be cleared.		
11–8 PRE	Prescaler. The read/write prescaler bits select the internal bus clock divisor to generate the PIT clock. To accurately predict the timing of the next count, change the PRE[3:0] bits only when the enable bit (EN) is clear. Changing PRE[3:0] resets the prescaler counter. System reset and the loading of a new value into the counter also reset the prescaler counter. Setting the EN bit and writing to PRE[3:0] can be done in this same write cycle. Clearing the EN bit stops the prescaler counter.		
PRE	Internal Bus Clock Divisor	Decimal Equivalent	
0000	$2^0$	1	
0001	$2^1$	2	
0010	$2^2$	4	
...	...	...	
1101	$2^{13}$	8192	
1110	$2^{14}$	16384	
1111	$2^{15}$	32768	
7	Reserved, must be cleared.		
6 DOZE	Doze Mode Bit. The read/write DOZE bit controls the function of the PIT in doze mode. Reset clears DOZE. 0 PIT function not affected in doze mode 1 PIT function stopped in doze mode. When doze mode is exited, timer operation continues from the state it was in before entering doze mode.		
5 DBG	Debug mode bit. Controls the function of PIT in halted/debug mode. Reset clears DBG. During debug mode, register read and write accesses function normally. When debug mode is exited, timer operation continues from the state it was in before entering debug mode, but any updates made in debug mode remain. 0 PIT function not affected in debug mode 1 PIT function stopped in debug mode <b>Note:</b> Changing the DBG bit from 1 to 0 during debug mode starts the PIT timer. Likewise, changing the DBG bit from 0 to 1 during debug mode stops the PIT timer.		
4 OVW	Overwrite. Enables writing to PMR $n$ to immediately overwrite the value in the PIT counter. 0 Value in PMR $n$ replaces value in PIT counter when count reaches 0x0000. 1 Writing PMR $n$ immediately replaces value in PIT counter.		
3 PIE	PIT interrupt enable. This read/write bit enables PIF flag to generate interrupt requests. 0 PIF interrupt requests disabled 1 PIF interrupt requests enabled		
2 PIF	PIT interrupt flag. This read/write bit is set when PIT counter reaches 0x0000. Clear PIF by writing a 1 to it or by writing to PMR. Writing 0 has no effect. Reset clears PIF. 0 PIT count has not reached 0x0000. 1 PIT count has reached 0x0000.		

**Table 38-3. PCSR $n$  Field Descriptions (continued)**

Field	Description
1 RLD	Reload bit. The read/write reload bit enables loading the value of PMR $n$ into PIT counter when the count reaches 0x0000. 0 Counter rolls over to 0xFFFF on count of 0x0000 1 Counter reloaded from PMR $n$ on count of 0x0000
0 EN	PIT enable bit. Enables PIT operation. When PIT is disabled, counter and prescaler are held in a stopped state. This bit is read anytime, write anytime. 0 PIT disabled 1 PIT enabled

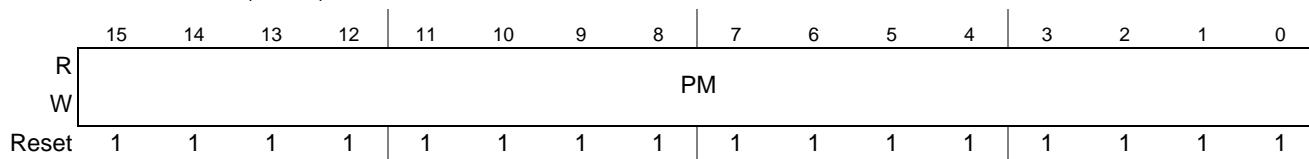
### 38.2.2 PIT Modulus Register (PMR $n$ )

The 16-bit read/write PMR $n$  contains the timer modulus value loaded into the PIT counter when the count reaches 0x0000 and the PCSR $n$ [RLD] bit is set.

When the PCSR $n$ [OVW] bit is set, PMR $n$  is transparent, and the value written to PMR $n$  is immediately loaded into the PIT counter. The prescaler counter is reset (0xFFFF) anytime a new value is loaded into the PIT counter and also during reset. Reading the PMR $n$  returns the value written in the modulus latch. Reset initializes PMR $n$  to 0xFFFF.

Address: 0xFC08\_0002 (PMR0)  
0xFC08\_4002 (PMR1)  
0xFC08\_8002 (PMR2)  
0xFC08\_C002 (PMR3)

Access: Supervisor  
read/write

**Figure 38-3. PIT Modulus Register (PMR $n$ )****Table 38-4. PMR $n$  Field Descriptions**

Field	Description
15–0 PM	Timer modulus. The value of this register is loaded into the PIT counter when the count reaches zero and the PCSR $n$ [RLD] bit is set. However, if PCSR $n$ [OVW] is set, the value written to this field is immediately loaded into the counter. Reading this field returns the value written.

### 38.2.3 PIT Count Register (PCNTR $n$ )

The 16-bit, read-only PCNTR $n$  contains the counter value. Reading the 16-bit counter with two 8-bit reads is not guaranteed coherent. Writing to PCNTR $n$  has no effect, and write cycles are terminated normally.

## Programmable Interrupt Timers (PIT0–PIT3)

Address: 0xFC08\_0004 (PCNTR0)  
 0xFC08\_4004 (PCNTR1)  
 0xFC08\_8004 (PCNTR2)  
 0xFC08\_C004 (PCNTR3)

Access: User read only

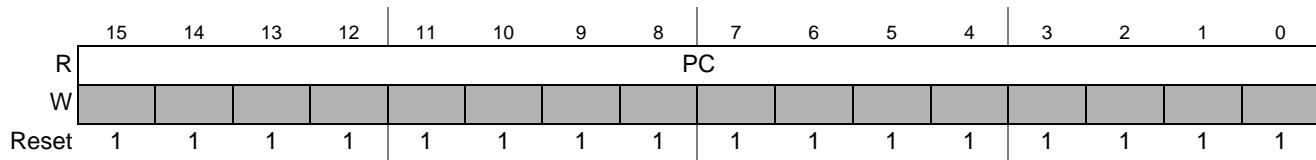


Figure 38-4. PIT Count Register (PCNTRn)

Table 38-5. PCNTRn Field Descriptions

Field	Description
15–0 PC	Counter value. Reading this field with two 8-bit reads is not guaranteed coherent. Writing to PCNTRn has no effect, and write cycles are terminated normally.

## 38.3 Functional Description

This section describes the PIT functional operation.

### 38.3.1 Set-and-Forget Timer Operation

This mode of operation is selected when the RLD bit in the PCSR register is set.

When PIT counter reaches a count of 0x0000, PIF flag is set in PCSRn. The value in the modulus register loads into the counter, and the counter begins decrementing toward 0x0000. If the PCSRn[PIE] bit is set, the PIF flag issues an interrupt request to the CPU.

When the PCSRn[OVW] bit is set, the counter can be directly initialized by writing to PMRn without having to wait for the count to reach 0x0000.

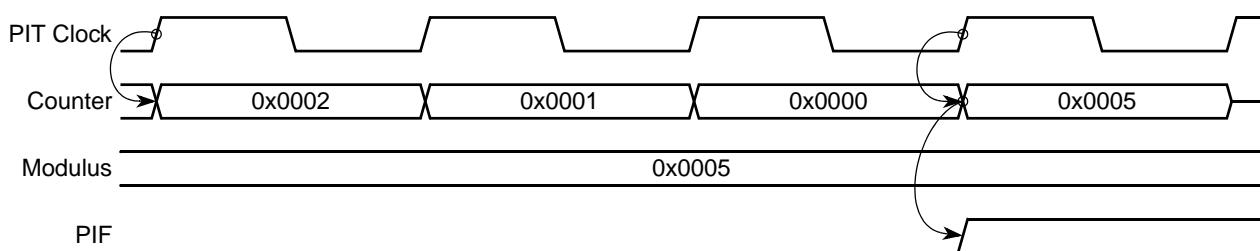


Figure 38-5. Counter Reloading from the Modulus Latch

### 38.3.2 Free-Running Timer Operation

This mode of operation is selected when the PCSRn[RLD] bit is clear. In this mode, the counter rolls over from 0x0000 to 0xFFFF without reloading from the modulus latch and continues to decrement.

When the counter reaches a count of 0x0000, PCSRn[PIF] flag is set. If the PCSRn[PIE] bit is set, PIF flag issues an interrupt request to the CPU.

When the PCSR $n$ [OVW] bit is set, counter can be directly initialized by writing to PMR $n$  without having to wait for the count to reach 0x0000.

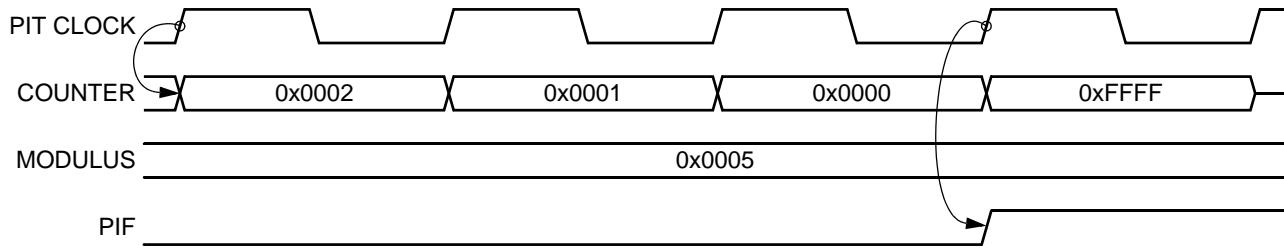


Figure 38-6. Counter in Free-Running Mode

### 38.3.3 Timeout Specifications

The 16-bit PIT counter and prescaler supports different timeout periods. The prescaler divides the internal bus clock period as selected by the PCSR $n$ [PRE] bits. The PMR $n$ [PM] bits select the timeout period.

$$\text{Timeout period} = \frac{2^{\text{PCSR}_n[\text{PRE}]} \times (\text{PMR}_n[\text{PM}] + 1)}{f_{\text{sys}/2}} \quad \text{Eqn. 38-1}$$

### 38.3.4 Interrupt Operation

[Table 38-6](#) shows the interrupt request generated by the PIT.

Table 38-6. PIT Interrupt Requests

Interrupt Request	Flag	Enable Bit
Timeout	PIF	PIE

The PIF flag is set when the PIT counter reaches 0x0000. The PIE bit enables the PIF flag to generate interrupt requests. Clear PIF by writing a 1 to it or by writing to the PMR.



# Chapter 39

## DMA Timers (DTIM0–DTIM3)

### 39.1 Introduction

This chapter describes the configuration and operation of the four direct memory access (DMA) timer modules (DTIM0, DTIM1, DTIM2, and DTIM3). These 32-bit timers provide input capture and reference compare capabilities with optional signaling of events using interrupts or DMA triggers. Additionally, programming examples are included.

#### NOTE

The designation *n* appears throughout this section to refer to registers or signals associated with one of the four identical timer modules: DTIM0, DTIM1, DTIM2, or DTIM3.

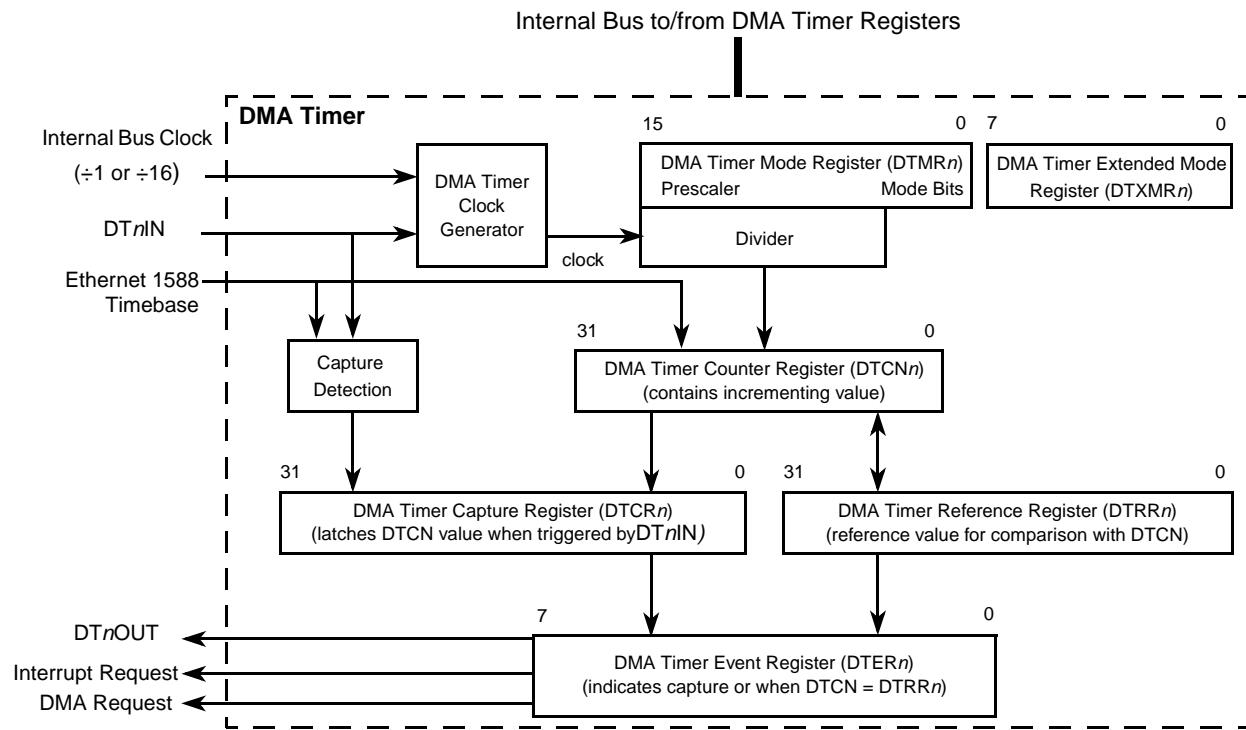
#### 39.1.1 Overview

Each DMA timer module has a separate register set for configuration and control. The timers can be configured to operate from the internal bus clock (*f<sub>sys</sub>*) or from an external clocking source using the DT*n*IN signal. If the internal bus clock is selected, it can be divided by 16 or 1. The selected clock source is routed to an 8-bit programmable prescaler that clocks the actual DMA timer counter register (DTCN*n*). The Ethernet assembly's IEEE 1588 timebase (which is asynchronous to the internal bus clock) can optionally drive the timers. Using the DTMR*n*, DTXMR*n*, DTCR*n*, and DTRR*n* registers, the DMA timer may be configured to assert an output signal, generate an interrupt, or request a DMA transfer on a particular event.

#### NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to [Chapter 15, “Pin-Multiplexing and Control”](#)) prior to configuring the DMA Timers.

Figure 39-1 is a block diagram of one of the four identical timer modules.



**Figure 39-1. DMA Timer Block Diagram**

### 39.1.2 Features

Each DMA timer module has:

- Maximum timeout period of 140,737 seconds at 125 MHz (~39 hours)
- 8-ns resolution at 125 MHz
- Programmable sources for the clock input, including external clock
- Programmable prescaler
- Input-capture capability with programmable trigger edge on input pin
- Programmable mode for the output pin on reference compare
- Free run and restart modes
- Programmable interrupt or DMA request on input capture or reference-compare
- Ability to stop the timer from counting when the ColdFire core is halted
- Configuration bit to enable use of the 1588 timebase and count value

## 39.2 Memory Map/Register Definition

The timer module registers, shown in [Table 39-1](#), can be modified at any time.

### NOTE

Due to the additional IEEE 1588 timebase logic, if DTXMR $n$ [EN1588] is set, reads and writes to the timer registers may take multiple bus cycles.

**Table 39-1. DMA Timer Module Memory Map**

Address	Register	Width (bits)	Access	Reset Value	Section/Page
DMA Timer 0 DMA Timer 1 DMA Timer 2 DMA Timer 3					
0xFC07_0000 0xFC07_4000 0xFC07_8000 0xFC07_C000	DMA Timer $n$ Mode Register (DTMR $n$ )	16	R/W	0x0000	<a href="#">39.2.1/39-4</a>
0xFC07_0002 0xFC07_4002 0xFC07_8002 0xFC07_C002	DMA Timer $n$ Extended Mode Register (DTXMR $n$ )	8	R/W	0x00	<a href="#">39.2.2/39-5</a>
0xFC07_0003 0xFC07_4003 0xFC07_8003 0xFC07_C003	DMA Timer $n$ Event Register (DTER $n$ )	8	R/W	0x00	<a href="#">39.2.3/39-6</a>
0xFC07_0004 0xFC07_4004 0xFC07_8004 0xFC07_C004	DMA Timer $n$ Reference Register (DTRR $n$ )	32	R/W	0xFFFF_FFFF	<a href="#">39.2.4/39-7</a>
0xFC07_0008 0xFC07_4008 0xFC07_8008 0xFC07_C008	DMA Timer $n$ Capture Register (DTCR $n$ )	32	R/W	0x0000_0000	<a href="#">39.2.5/39-8</a>
0xFC07_000C 0xFC07_400C 0xFC07_800C 0xFC07_C00C	DMA Timer $n$ Counter Register (DTCN $n$ )	32	R	0x0000_0000	<a href="#">39.2.6/39-8</a>

### 39.2.1 DMA Timer Mode Registers (DTMR $n$ )

The DTMR $n$  registers program the prescaler and various timer modes.

Address:	0xFC07_0000 (DTMR0)	Access:	User read/write
	0xFC07_4000 (DTMR1)		
	0xFC07_8000 (DTMR2)		
	0xFC07_C000 (DTMR3)		
R	15 14 13 12   11 10 9 8	7 6 5 4	3 2 1 0
W	PS	CE OM ORRI	FRR CLK RST
Reset	0 0 0 0   0 0 0 0	0 0 0 0	0 0 0 0

Figure 39-2. DTMR $n$  Registers

Table 39-2. DTMR $n$  Field Descriptions

Field	Description
15–8 PS	Prescaler value. Divides the clock input (internal bus clock/(16 or 1) or clock on DT $n$ IN). Unused when DTXMR $n$ [EN1588] is set. 0x00 1 ... 0xFF 256
7–6 CE	Capture edge. 00 Disable capture event output. Timer in reference mode. 01 Capture on rising edge only 10 Capture on falling edge only 11 Capture on any edge
5 OM	Output mode. 0 Active-low pulse for one internal bus clock cycle (8-ns resolution at 125 MHz) if DTXMR $n$ [EN1588] is cleared or one 1588 timebase clock cycle if DTXMR $n$ [EN1588] is set 1 Toggle output.
4 ORRI	Output reference request, interrupt enable. If ORRI is set when DTER $n$ [REF] is set, a DMA request or an interrupt occurs, depending on the value of DTXMR $n$ [DMAEN] (DMA request if set, interrupt if cleared). 0 Disable DMA request or interrupt for reference reached (does not affect DMA request or interrupt on capture function). 1 Enable DMA request or interrupt upon reaching the reference value.
3 FRR	Free run/restart. Unused when DTXMR $n$ [EN1588] is set. 0 Free run. Timer count continues incrementing after reaching the reference value. 1 Restart. Timer count is reset immediately after reaching the reference value.

**Table 39-2. DTMR $n$  Field Descriptions (continued)**

Field	Description
2–1 CLK	Input clock source for the timer. Avoid setting CLK when RST is already set. Doing so causes CLK to zero (stop counting). Unused when DTXMR $n$ [EN1588] is set. 00 Stop count 01 Internal bus clock divided by 1 10 Internal bus clock divided by 16. This clock source is not synchronized with the timer; therefore, successive time-outs may vary slightly. 11 DT $n$ IN pin (falling edge)
0 RST	Reset timer. Performs a software timer reset similar to an external reset, although other register values can be written while RST is cleared. A transition of RST from 1 to 0 resets register values. The timer counter or 1588 timer logic is not clocked unless the timer is enabled. 0 Reset timer (software reset) 1 Enable timer

### 39.2.2 DMA Timer Extended Mode Registers (DTXMR $n$ )

The DTXMR $n$  registers program DMA request and increment modes for the timers.

Address:	0xFC07_0002 (DTXMR0) 0xFC07_4002 (DTXMR1) 0xFC07_8002 (DTXMR2) 0xFC07_C002 (DTXMR3)	Access:	User read/write
R W	7 DMAEN      6 HALTED      5 EN1588      4 EPD	3 0      2 0      1 0      0	0 MODE16

Reset: 0 0 0 0 | 0 0 0 0

**Figure 39-3. DTXMR $n$  Registers****Table 39-3. DTXMR $n$  Field Descriptions**

Field	Description
7 DMAEN	DMA request. Enables DMA request output on counter reference match or capture edge event. 0 DMA request disabled 1 DMA request enabled
6 HALTED	Controls the counter when the core is halted. This allows debug mode to be entered without timer interrupts affecting the debug flow. Unused when DTXMR $n$ [EN1588] is set. 0 Timer function is not affected by core halt. 1 Timer stops counting while the core is halted. <b>Note:</b> This bit is only applicable in reference compare mode, see <a href="#">Section 39.3.3, “Reference Compare.”</a>
5 EN1588	Enable IEEE 1588 timebase support. 0 Disable 1 Enable. Several bits in the DTXMR $n$ and DTMR $n$ registers are ignored. See the bit descriptions for which are not used.
4 EPD	Enable programmable delay. Unused when DTXMR $n$ [EN1588] is set. 0 Disable 1 Enable

**Table 39-3. DTXMR $n$  Field Descriptions (continued)**

Field	Description
3–1	Reserved, must be cleared.
0 MODE16	Selects the increment mode for the timer. Setting MODE16 is intended to exercise the upper bits of the 32-bit timer in diagnostic software without requiring the timer to count through its entire dynamic range. When set, the counter's upper 16 bits mirror its lower 16 bits. All 32 bits of the counter remain compared to the reference value. Unused when DTXMR $n$ [EN1588] is set. 0 Increment timer by 1 1 Increment timer by 65,537

### 39.2.3 DMA Timer Event Registers (DTER $n$ )

DTER $n$ , shown in [Figure 39-4](#), reports capture or reference events by setting DTER $n$ [CAP] or DTER $n$ [REF]. This reporting happens regardless of the corresponding DMA request or interrupt enable values, DTXMR $n$ [DMAEN] and DTMR $n$ [ORRI,CE].

Writing a 1 to DTER $n$ [REF] or DTER $n$ [CAP] clears it (writing a 0 does not affect bit value); both bits can be cleared at the same time. If configured to generate an interrupt request, clear REF and CAP early in the interrupt service routine so the timer module can negate the interrupt request signal to the interrupt controller. If configured to generate a DMA request, processing of the DMA data transfer automatically clears the REF and CAP flags via the internal DMA ACK signal.

Address:	0xFC07_0003 (DTER0) 0xFC07_4003 (DTER1) 0xFC07_8003 (DTER2) 0xFC07_C003 (DTER3)	Access:	User read/write
R	7            6            5            4                         3            2            1            0		
W	0            0            0            0                         0            0            REF            CAP		
Reset:	0            0            0            0                         0            0            0            0		

**Figure 39-4. DTER $n$  Registers**

**Table 39-4. DTER $n$  Field Descriptions**

Field	Description			
7–2	Reserved, must be cleared.			
1 REF	Output reference event. The counter value (DTCN $n$ ) equals DTRR $n$ , or if DTXMR $n$ [EN1588] is set the 1588 timebase counter is greater than the reference value. Writing a 1 to REF clears the event condition. Writing a 0 has no effect. To prevent spurious assertions of the reference compare, once set, the logic associated with the setting of this bit is not re-enabled until the DTRR $n$ register is updated, the 1588 timebase counter rolls over, or a timer reset (DTMR $n$ [RST] = 0) occurs.			
		<b>REF</b>	<b>DTMR<math>n</math>[ORRI]</b>	<b>DTXMR<math>n</math>[DMAEN]</b>
		0	X	X
		1	0	0
		1	0	1
		1	1	0
		1	1	1
0 CAP	Capture event. The counter value has been latched into DTCR $n$ . Writing a 1 to CAP clears the event condition. Writing a 0 has no effect.			
		<b>CAP</b>	<b>DTMR<math>n</math>[CE]</b>	<b>DTXMR<math>n</math> [DMAEN]</b>
		0	XX	X
		1	00	0
		1	00	1
		1	01	0
		1	01	1
		1	10	0
		1	10	1
		1	11	0
		1	11	1

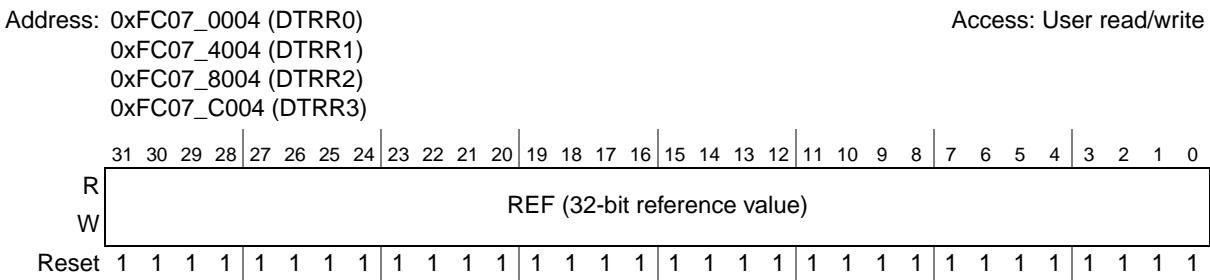
### 39.2.4 DMA Timer Reference Registers (DTRR $n$ )

As part of the output-compare function, each DTRR $n$  contains the reference value compared with the respective free-running timer counter (DTCN $n$ ) if DTXMR $n$ [EN1588] is cleared or the 1588 timebase counter if DTXMR $n$ [EN1588] is set.

If DTXMR $n$ [EN1588] is cleared, the reference value is matched when DTCN $n$  equals DTRR $n$ . The prescaler indicates that DTCN $n$  should be incremented again. Therefore, the reference register is matched after DTRR $n$  + 1 time intervals.

## DMA Timers (DTIM0–DTIM3)

If  $DTXMRn[EN1588]$  is set, the reference value is matched when the 1588 timebase counter is greater than  $DTRRn$ . When the reference compare occurs and  $DTERn[REF]$  is set, the logic associated with the setting of this bit is not re-enabled until  $DTRRn$  is updated, the 1588 timebase counter rolls over, or a timer reset ( $DTMRn[RST] = 0$ ) occurs. This prevents spurious assertions of  $DTERn[REF]$ .



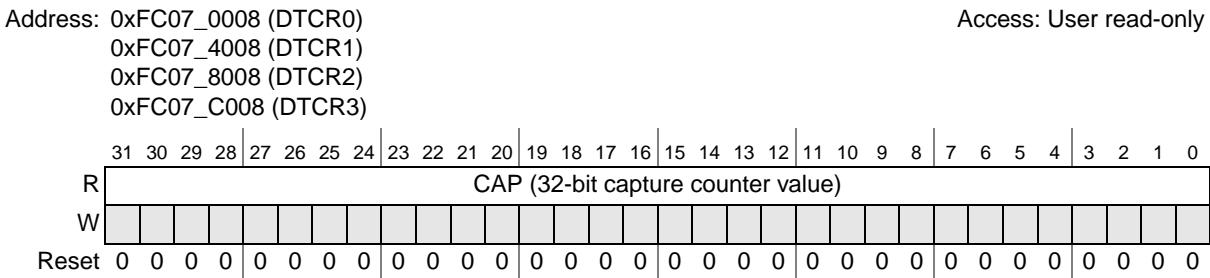
**Figure 39-5. DTRR $n$  Registers**

**Table 39-5. DTRR $n$  Field Descriptions**

Field	Description
31–0 REF	Reference value compared with the respective free-running timer counter (DTCN $n$ ) or 1588 timebase count value as part of the output-compare function.

## 39.2.5 DMA Timer Capture Registers (DTCR $n$ )

Each DTCR $n$  latches the corresponding DTCN $n$  or 1588 timebase count value during a capture operation when an edge occurs on DT $n$ IN, as programmed in DTMR $n$ . The internal bus clock is assumed to be the clock source. DT $n$ IN cannot simultaneously function as a clocking source and as an input capture pin. Indeterminate operation results if DT $n$ IN is set as the clock source when the input capture mode is used.



**Figure 39-6. DTCR $n$  Registers**

**Table 39-6. DTCR $n$  Field Descriptions**

Field	Description
31–0 CAP	Captures the corresponding DTCN $n$ value during a capture operation when an edge occurs on DT $n$ IN, as programmed in DTMR $n$ .

## 39.2.6 DMA Timer Counters (DTCN $n$ )

The current value of the 32-bit timer counter or IEEE 1588 timebase counter can be read at anytime without affecting counting. Writes to DTCN $n$  clear the timer counter if  $DTXMRn[EN1588]$  is cleared. If

EN1588 is set, writes are not allowed. The timer counter increments on the clock source rising edge (internal bus clock divided by 1, internal bus clock divided by 16, or DTnIN) if EN1588 is cleared.

**Figure 39-7. DMA Timer Counters (DTCN $n$ )**

**Table 39-7. DTCN<sub>n</sub> Field Descriptions**

Field	Description
31-0 CNT	Timer counter. Can be read at anytime without affecting counting. Writes to this register are allowed only if DTXMR $n$ [EN1588] is cleared and any write to this field clears it.

### 39.3 Functional Description

### 39.3.1 Prescaler

The prescaler clock input is selected from the internal bus clock ( $f_{sys/2}$  divided by 1 or 16) or from the corresponding timer input, DTnIN. DTnIN is synchronized to the internal bus clock, and the synchronization delay is between two and three internal bus clocks. The corresponding DTMRn[CLK] selects the clock input source. A programmable prescaler divides the clock input by values from 1 to 256. The prescaler output is an input to the 32-bit counter, DTCNn.

If DTXMR $n$ [EN1588] is set, the prescaler is not used.

### 39.3.2 Capture Mode

Each DMA timer has a 32-bit timer capture register (DTCR $n$ ) that latches the counter value when the corresponding input capture edge detector senses a defined DTnIN transition. The capture edge bits (DTMR $n$ [CE]) select the type of transition that triggers the capture and sets the timer event register capture event bit, DTER $n$ [CAP]. If DTER $n$ [CAP] and DTXMR $n$ [DMAEN] are set, a DMA request is asserted. If DTER $n$ [CAP] is set and DTXMR $n$ [DMAEN] is cleared, an interrupt is asserted.

### 39.3.3 Reference Compare

Each DMA timer can be configured to count up to a reference value. If  $\text{DTXMR}_n[\text{EN1588}]$  is cleared, when the reference value is met,  $\text{DTER}_n[\text{REF}]$  is set. If  $\text{DTXMR}_n[\text{EN1588}]$  is set, the reference value is matched when the 1588 timebase counter is greater than  $\text{DTRR}_n$ .

- If DTMR $n$ [ORRI] is set and DTXMR $n$ [DMAEN] is cleared, an interrupt is asserted.
  - If DTMR $n$ [ORRI] and DTXMR $n$ [DMAEN] are set, a DMA request is asserted.

If the free run/restart bit (DTMR $n$ [FRR]) is set, a new count starts. If it is clear, the timer keeps running. This bit is not used in 1588 mode.

### 39.3.4 Output Mode

When a timer reaches the reference value selected by DTRR, it can send an output signal on DT $n$ OUT. DT $n$ OUT can be an active-low pulse or a toggle of the current output, as selected by the DTMR $n$ [OM] bit.

### 39.3.5 Programmable Delay Mode

To properly enable programmable delay mode, the DTMR must be set appropriately as shown in [Table 39-8](#).

**Table 39-8. Valid DTMR $n$  Values in Programmable Delay Mode**

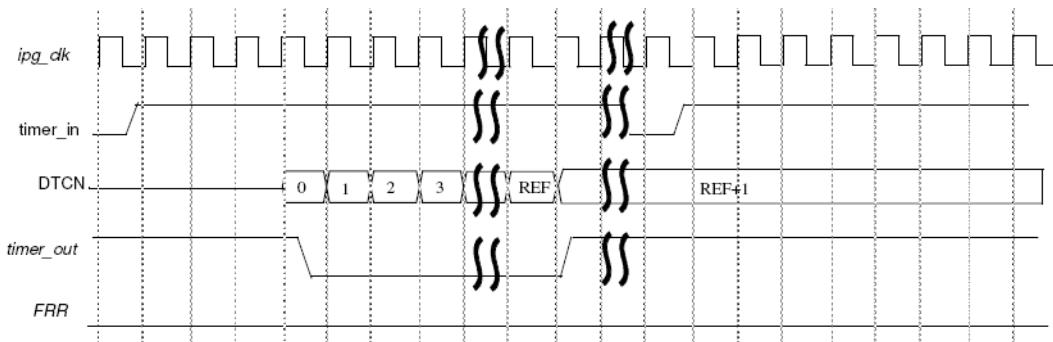
DTMR Field		Valid Values
15–8	PS	Don't care
7–6	CE	≠ 00
5	OM	0
4	ORRI	Don't care
3	FRR	0 or 1
2–1	CLK	01
0	RST	1

Depending on the configured timer input edge capture (DTMR[CE]), when the trigger occurs, the timer counter (DTCN), operating at the platform frequency, starts incrementing and the DT $n$ OUT signal is negated. When the timer counter reaches the reference value (DTRR), DT $n$ OUT is asserted. The low-to-high edge of DT $n$ OUT provides the programmable delay trigger.

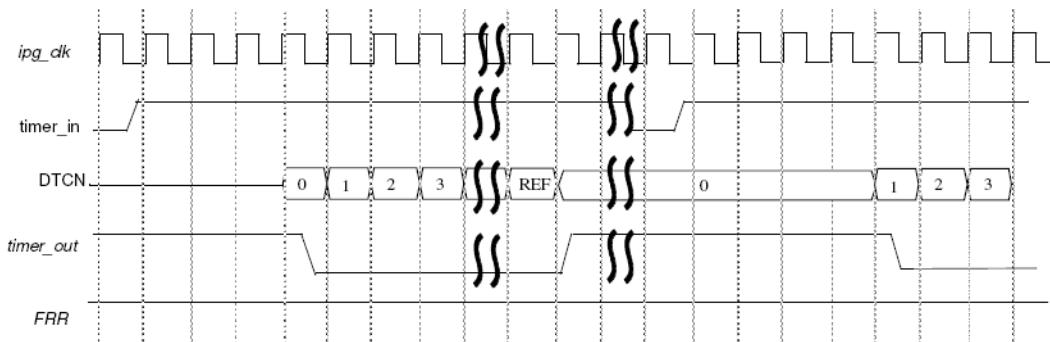
The values in DTMR[ORRI] and DTXMR[DMAEN] control the timer interrupt and DMA requests in this mode. The only configurable feature in this mode is based on the value in DTMR[FRR]:

- If DTMR[FRR] = 0, the programmable delay function only occurs once. When the reference value is reached, the timer counter stops
- If DTMR[FRR] = 1, the programmable delay function restarts waiting for another edge to capture. For this case, the DTCN value resets immediately after reaching the reference value defined by DTRR.

The following figures depict the programmable delay timing when FRR = 0 and FRR = 1, respectively.



**Figure 39-8. Programmable Delay Timing (FRR = 0)**



**Figure 39-9. Programmable Delay Timing (FRR = 1)**

#### NOTE

The time between the transition of the *timer\_in* signal and the incrementing of the *DTCN* is fixed (4 IPS cycles) and is due to the cascaded registers performing the clock domain synchronization and input edge capture logic.

If another capture edge occurs while the timer counter is incrementing, it is ignored.

### 39.3.6 IEEE 1588 Support

The DMA timers on this device can use the Ethernet assembly's IEEE-1588 timebase count value as its clock source. This feature supports triggering events via processor interrupts or DMA requests based on network time values.

To use the 1588 timebase as the clock source follow the below sequence:

1. Clear/disable *DTMRn* (*CE* = 00, *ORRI* = 0, *RST* = 0)
2. Enable 1588 mode in *DTXMRn* (*DTXMRn[EN1588]* = 1)
3. Program the reference value in *DTRRn*
4. Program *DTMRn* with *RST* set

**NOTE**

If the value programmed in DTRR $n$  is less than the value of the 1588 timebase count, then DTER $n$ [REF] sets immediately after RST is set.

## 39.4 Initialization/Application Information

The general-purpose timer modules typically, but not necessarily, follow this program order:

- The DTMR $n$  and DTXMR $n$  registers are configured for the desired function and behavior.
  - Count and compare to a reference value stored in the DTRR $n$  register
  - Capture the timer value on an edge detected on DT $n$ IN
  - Configure DT $n$ OUT output mode
  - Increment counter by 1 or by 65,537 (16-bit mode)
  - Enable/disable interrupt or DMA request on counter reference match or capture edge
- The DTMR $n$ [CLK] register is configured to select the clock source to be routed to the prescaler.
  - Internal bus clock (can be divided by 1 or 16)
  - DT $n$ IN, the maximum value of DT $n$ IN is 1/5 of the internal bus clock, as described in the device's electrical characteristics

**NOTE**

DT $n$ IN may not be configured as a clock source when the timer capture mode is selected or indeterminate operation results.

- The 8-bit DTMR $n$ [PS] prescaler value is set.
- Using DTMR $n$ [RST], counter is cleared and started.
- Timer events are managed with an interrupt service routine, a DMA request, or by a software polling mechanism.

### 39.4.1 Code Example

The following code provides an example of how to initialize and use DMA Timer0 for counting time-out periods.

```

DTMR0 EQU 0xFC07_0000 ;Timer0 mode register
DTMR1 EQU 0xFC07_4000 ;Timer1 mode register
DTRR0 EQU 0xFC07_0004 ;Timer0 reference register
DTRR1 EQU 0xFC07_4004 ;Timer1 reference register
DTCR0 EQU 0xFC07_0008 ;Timer0 capture register
DTCR1 EQU 0xFC07_4008 ;Timer1 capture register
DTCN0 EQU 0xFC07_000C ;Timer0 counter register
DTCN1 EQU 0xFC07_400C ;Timer1 counter register
DTER0 EQU 0xFC07_0003 ;Timer0 event register
DTER1 EQU 0xFC07_4003 ;Timer1 event register
* TMRO is defined as: *
*[PS] = 0xFF,      divide clock by 256
*[CE] = 00         disable capture event output
*[OM] = 0          output=active-low pulse
*[ORRI] = 0,       disable ref. match output

```

```
*[FRR] = 1,      restart mode enabled
*[CLK] = 10,     internal bus clock/16
*[RST] = 0,      timer0 disabled

move.w #0xFF0C,D0
move.w D0,TMR0
move.l #0x0000,D0;writing to the timer counter with any
move.l D0,TCN0 ;value resets it to zero
move.l #0xAFAF,DO ;set the timer0 reference to be
move.l #D0,TRR0 ;defined as 0xAFAF
```

The simple example below uses Timer0 to count time-out loops. A time-out occurs when the reference value, 0xAFAF, is reached.

```
timer0_ex
    clr.l D0
    clr.l D1
    clr.l D2
    move.l #0x0000,D0
    move.l D0,TCN0          ;reset the counter to 0x0000
    move.b #0x03,D0          ;writing ones to TER0[REF,CAP]
    move.b D0,TER0          ;clears the event flags
    move.w TMR0,D0          ;save the contents of TMR0 while setting
    bset #0,D0              ;the 0 bit. This enables timer 0 and starts counting
    move.w D0,TMR0          ;load the value back into the register, setting TMR0[RST]

T0_LOOP
    move.b TER0,D1          ;load TER0 and see if
    btst #1,D1              ;TER0[REF] has been set
    beq T0_LOOP
    addi.l #1,D2              ;Increment D2
    cmp.l #5,D2              ;Did D2 reach 5? (i.e. timer ref has timed)
    beq T0_FINISH            ;If so, end timer0 example. Otherwise jump back.
    move.b #0x02,D0          ;writing one to TER0[REF] clears the event flag
    move.b D0,TER0
    jmp T0_LOOP

T0_FINISH
    HALT                    ;End processing. Example is finished
```

### 39.4.2 Calculating Time-Out Values

Equation 39-1 determines time-out periods for various reference values:

$$\text{Timeout period} = (1/\text{clock frequency}) \times (1 \text{ or } 16) \times (\text{DTMR}_n[\text{PS}] + 1) \times (\text{DTRR}_n[\text{REF}] + 1) \quad \text{Eqn. 39-1}$$

When calculating time-out periods, add one to the prescaler to simplify calculating, because DTMR<sub>n</sub>[PS] equal to 0x00 yields a prescaler of one, and DTMR<sub>n</sub>[PS] equal to 0xFF yields a prescaler of 256.

For example, if a 125-MHz timer clock is divided by 16, DTMR<sub>n</sub>[PS] equals 0x7F, and the timer is referenced at 0x1DCD6 (122,070 decimal), the time-out period is:

$$\text{Timeout period} = \frac{1}{125 \times 10^6} \times 16 \times (127 + 1) \times (122070 + 1) = 2.00 \text{ seconds} \quad \text{Eqn. 39-2}$$



# Chapter 40

## DMA Serial Peripheral Interface (DSPI)

### 40.1 Introduction

This chapter describes the DMA serial peripheral interface (DSPI), which provides a synchronous serial bus for communication between the MCU and an external peripheral device. This device contains four identical DSPI modules. However, the number of chip selects they contain may vary.

#### 40.1.1 Block Diagram

Figure 40-1 shows a block diagram of the DSPI.

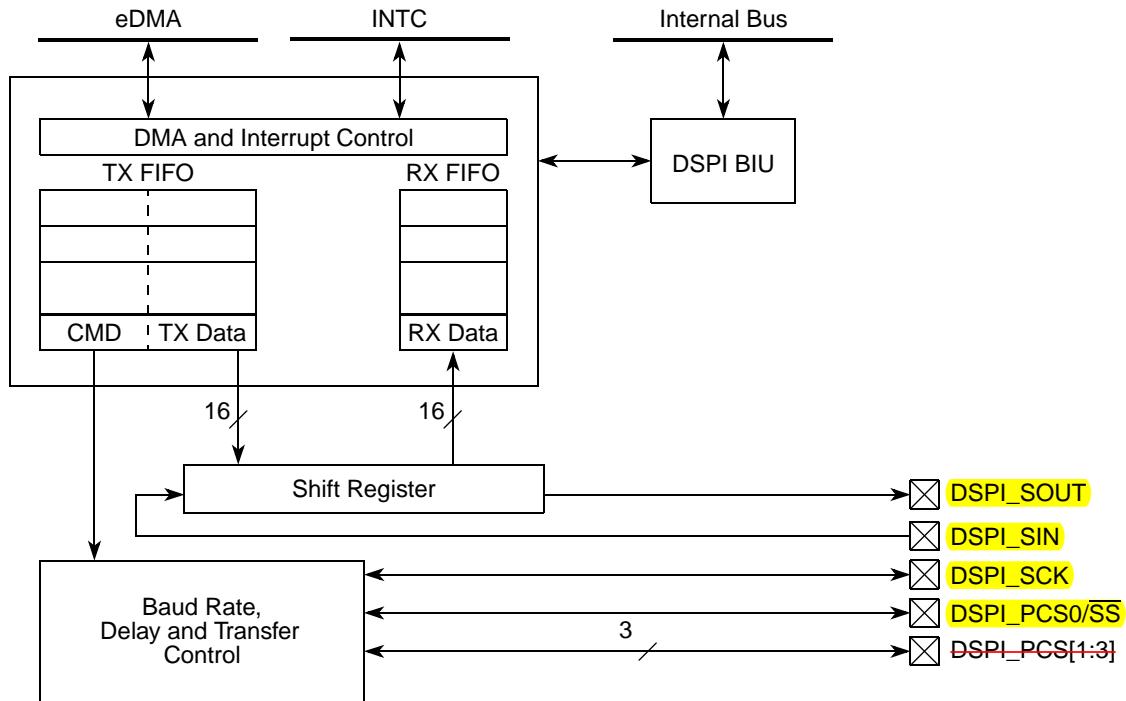


Figure 40-1. DSPI Block Diagram

#### 40.1.2 Overview

The DMA serial peripheral interface (DSPI) block provides a synchronous serial bus for communication between an MCU and an external peripheral device. ~~The DSPI supports up to 32 queued SPI transfers (16 receive and 16 transmit) in the DSPI resident FIFOs eliminating CPU intervention between transfers.~~

~~For queued operations, the SPI queues reside in system RAM external to the DSPI. Data transfers between the queues and the DSPI FIFOs are accomplished through the use of a DMA controller or through host software.~~

### NOTE

The pin multiplexing and control module must be configured to enable the peripheral function of the appropriate pins (refer to [Chapter 15, “Pin-Multiplexing and Control”](#)) prior to configuring the DSPI.

#### 40.1.3 Features

The DSPI module supports these SPI features:

- Full-duplex, three-wire synchronous transfers
- Master and slave mode
- Buffered transmit and receive operation using the TX and RX FIFOs, with depths of 16 entries
- Visibility into 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
  - Eight clock and transfer attribute registers
  - Serial clock with programmable polarity and phase
  - Programmable delays
    - PCS to SCK delay
    - SCK to PCS delay
    - Delay between frames
  - Programmable serial frame size of 4 to 16 bits, expandable with software control
  - Continuously held chip select capability
- Up to four peripheral chip selects, expandable to 16 with external demultiplexer
- Two DMA conditions for SPI queues residing in RAM or Flash
  - TX FIFO is not full (TFFF)
  - RX FIFO is not empty (RFDF)
- Eight interrupt conditions
  - End of queue reached (EOQF)
  - TX FIFO is not full (TFFF)
  - Transfer of current frame complete (TCF)
  - FIFO underflow (slave only, the slave is asked to transfer data when the TX FIFO is empty) (TFUF)
  - RX FIFO is not empty (RFDF)
  - FIFO overflow (attempt to transmit with an empty TX FIFO or serial frame received while RX FIFO is full) (RFOF)
  - FIFO overrun (logical OR of RX overflow and TX underflow interrupts)

- General DSPI interrupt (logical OR of the seven above conditions)
- Modified SPI transfer formats for communication with slower peripheral devices
- Continuous serial communications clock (DSPI\_SCK)

#### 40.1.4 Modes of Operation

The DSPI module has four available distinct modes:

- Master mode
- Slave mode
- Module disable mode
- Debug mode

Master, slave, and module disable modes are module-specific modes while debug mode is a device-specific mode.

Bits in the DSPI\_MCR register determine the module-specific modes. Debug mode is a mode that the entire device can enter in parallel with the DSPI being configured in one of its module-specific modes.

##### 40.1.4.1 Master Mode

In master mode, the DSPI can initiate communications with peripheral devices. The DSPI operates as bus master when the DSPI\_MCR[MSTR] bit is set. The serial communications clock (DSPI\_SCK) is controlled by the master DSPI.

Master mode transfer attributes are controlled by the SPI command in the current TX FIFO entry. The CTAS field in the SPI command selects which of the eight DSPI\_CTARs sets the transfer attributes. Transfer attribute control is on a frame by frame basis. See [Section 40.4.2, “Serial Peripheral Interface \(SPI\) Configuration”](#) for more details.

##### 40.1.4.2 Slave Mode

In slave mode, the DSPI responds to transfers initiated by an SPI master. The DSPI operates as bus slave when the DSPI\_MCR[MSTR] bit is cleared. A bus master selects the DSPI slave by having the slave's DSPI\_SS signal asserted. In slave mode, the bus master provides DSPI\_SCK. The bus master controls all transfer attributes, but clock polarity, clock phase, and numbers of bits to transfer must be configured in the DSPI slave for proper communications.

In slave mode, data transfers MSB first. The LSBFE field of the associated CTAR register is ignored.

##### 40.1.4.3 Module Disable Mode

The module disable mode is used for MCU power management. The clock to the non-memory mapped logic in the DSPI stops while in module disable mode. The DSPI enters the module disable mode when the DSPI\_MCR[MDIS] bit is set. See [Section 40.4.7, “Power Saving Features,”](#) for more details on the module disable mode.

#### 40.1.4.4 Debug Mode

Debug mode is used for system development and debugging. If the device enters debug mode while the DSPI\_MCR[FRZ] bit is set, the DSPI halts operation on the next frame boundary. If the device enters debug mode while the FRZ bit is cleared, the DSPI behavior is unaffected and remains dictated by the module-specific mode and configuration of the DSPI. See [Figure 40-12](#) for a state diagram.

## 40.2 External Signal Description

### 40.2.1 Signal Overview

[Table 40-1](#) lists the DSPI signals.

**Table 40-1. DSPI Signal Properties**

Name	Function			
	Master Mode	I/O	Slave Mode	I/O
DSPIx_PCS0/SS	Peripheral chip select 0	Output	Slave select	Input
DSPIx_PCS[1:3]	Peripheral chip select 1–3	Output	Unused	—
DSPIx_SIN	Serial data in	Input	Serial data in	Input
DSPIx_SOUT	Serial data out	Output	Serial data out	Output
DSPIx_SCK	Serial clock	Output	Serial clock	Input

### 40.2.2 Peripheral Chip Select/Slave Select (DSPIx\_PCS0/SS)

In master mode, the DSPIx\_PCS0 signal is a peripheral chip select output that selects the slave device to which the current transmission is intended. In slave mode, the DSPIx\_SS signal is a slave select input signal allowing an SPI master to select the DSPI as the target for transmission.

### 40.2.3 Peripheral Chip Selects 1–3 (DSPIx\_PCS[1:3])

The DSPIx\_PCS[1:3] signals are peripheral chip select output signals in master mode. In slave mode, these signals are not used.

### 40.2.4 Serial Input (DSPIx\_SIN)

DSPIx\_SIN is a serial data input signal.

### 40.2.5 Serial Output (DSPIx\_SOUT)

DSPIx\_SOUT is a serial data output signal.

## 40.2.6 Serial Clock (DSPIx\_SCK)

DSPIx\_SCK is a serial communication clock signal. In master mode, DSPI generates DSPIx\_SCK. In slave mode, DSPIx\_SCK is an input from an external bus master.

## 40.3 Memory Map/Register Definition

This device contains four DSPI modules. Their base addresses are listed below:

**Table 40-2. DSPI Base Addresses**

Base Address	Module
0xFC05_C000	DSPI 0
0xFC03_C000	DSPI 1
0xEC03_8000	DSPI 2
0xEC03_C000	DSPI 3

Table 40-3 shows the DSPI memory map.

**Table 40-3. DSPI Module Memory Map**

Address	Register	Width	Access	Reset Value	Section/Page
DSPI 0 ... DSPI 3					
0xFC05_C000 0xFC03_C000 0xEC03_8000 0xEC03_C000	DSPI module configuration register (DSPIx_MCR)	32	R/W	0x0000_4001	<a href="#">40.3.1/40-6</a>
0xFC05_C008 0xFC03_C008 0xEC03_8008 0xEC03_C008	DSPI transfer count register (DSPIx_TCR)	32	R/W	0x0000_0000	<a href="#">40.3.2/40-9</a>
0xFC05_C00C + (n × 0x04) 0xFC03_C00C + (n × 0x04) 0xEC03_800C + (n × 0x04) 0xEC03_C00C + (n × 0x04)	DSPI clock and transfer attributes registers (DSPIx_CTARn), n=0:7	32	R/W	0x7800_0000	<a href="#">40.3.3/40-9</a>
0xFC05_C02C 0xFC03_C02C 0xEC03_802C 0xEC03_C02C	DSPI status register (DSPIx_SR)	32	R/W	0x0000_0000	<a href="#">40.3.4/40-14</a>
0xFC05_C030 0xFC03_C030 0xEC03_8030 0xEC03_C030	DSPI DMA/interrupt request select and enable register (DSPIx_RSER)	32	R/W	0x0000_0000	<a href="#">40.3.5/40-16</a>

**Table 40-3. DSPI Module Memory Map (continued)**

Address	Register	Width	Access	Reset Value	Section/Page
<b>DSPI 0</b> ... <b>DSPI 3</b>					
0xFC05_C034 0xFC03_C034 0xEC03_8034 0xEC03_C034	DSPI push TX FIFO register (DSPIx_PUSHR)	32	R/W	0x0000_0000	<a href="#">40.3.6/40-17</a>
0xFC05_C038 0xFC03_C038 0xEC03_8038 0xEC03_C038	DSPI pop RX FIFO register (DSPIx_POPR)	32	R	0x0000_0000	<a href="#">40.3.7/40-19</a>
0xFC05_C03C + (n × 0x04) 0xFC03_C03C + (n × 0x04) 0xEC03_803C + (n × 0x04) 0xEC03_C03C + (n × 0x04)	DSPI transmit FIFO registers (DSPIx_TXFR $n$ ), $n=0:15$	32	R	0x0000_0000	<a href="#">40.3.8/40-19</a>
0xFC05_C07C + (n × 0x04) 0xFC03_C07C + (n × 0x04) 0xEC03_807C + (n × 0x04) 0xEC03_C07C + (n × 0x04)	DSPI receive FIFO registers (DSPIx_RXFR $n$ ), $n=0:15$	32	R	0x0000_0000	<a href="#">40.3.9/40-20</a>

### 40.3.1 DSPI Module Configuration Register (DSPI\_MCR)

The DSPI\_MCR contains bits that 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 may be changed while the DSPI is running.

#### NOTE

The DSPI\_MCR[MDIS] bit is set at reset.

Address: 0xFC05\_C000 (DSPI0\_MCR)  
 0xFC03\_C000 (DSPI1\_MCR)  
 0xEC03\_8000 (DSPI2\_MCR)  
 0xEC03\_C000 (DSPI3\_MCR)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MSTR	CONT_SCKE		DCONF	FRZ	MTFE	0	RO OE	PCS_IS7	PCS_IS6	PCS_IS5	PCS_IS4	PCS_IS3	PCS_IS2	PCS_IS1	PCS_IS0
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		MDIS	DIS_TXF	DIS_RXF	0	0	SMPL_PT	0	0	0	0	0	0	0	HALT
W						CLR_TXF	CLR_RXF									
Reset	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 40-2. DSPI Module Configuration Register (DSPIx\_MCR)

Table 40-4. DSPIx\_MCR Field Descriptions

Field	Description
31 <b>MSTR</b>	Master/slave mode select. Configures the DSPI for master mode or slave mode. <b>Note:</b> This bit's value must only be changed when the DSPI_MCR[HALT] bit is set. Otherwise, improper operation may occur. 0 Slave mode 1 Master mode
30 CONT_SCKE	Continuous SCK enable. Enables the serial communication clock (DSPI_SCK) to run continuously. See <a href="#">Section 40.4.5, “Continuous Serial Communications Clock,”</a> for details. 0 Continuous SCK disabled 1 Continuous SCK enabled
29–28 <b>DCONF</b>	DSPI configuration. Selects between the different configurations of the DSPI. 00 SPI 01 Reserved 10 Reserved 11 Reserved <b>Note:</b> All values except 00 are reserved. This field must be configured for SPI mode for the DSPI module to operate correctly.
27 FRZ	Freeze. Enables the DSPI transfers to be stopped on the next frame boundary when the device enters debug mode. 0 Do not halt serial transfers 1 Halt serial transfers
26 MTFE	Modified timing format enable. Enables a modified transfer format to be used. See <a href="#">Section 40.4.4.4, “Modified SPI Transfer Format (MTFE = 1, CPHA = 1),”</a> for more information. 0 Modified SPI transfer format disabled 1 Modified SPI transfer format enabled
25	Reserved, must be cleared.

**Table 40-4. DSPIx\_MCR Field Descriptions (continued)**

Field	Description
24 ROOE	Receive FIFO overflow overwrite enable. Enables an 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, data from the transfer that generated the overflow is ignored or shifted in to the shift register. If the ROOE bit is set, incoming data is shifted into the shift register. If the ROOE bit is cleared, incoming data is ignored. See <a href="#">Section 40.4.6.6, "Receive FIFO Overflow Interrupt Request (RFOF)"</a> , for more information. 0 Incoming data is ignored 1 Incoming data is shifted in to the shift register
23–16 <b>PCSI<sub>n</sub></b>	Peripheral chip select inactive state. Determines the inactive state of the DSPI_PCS <sub>n</sub> signal. 0 The inactive state of DSPI_PCS <sub>n</sub> is low 1 The inactive state of DSPI_PCS <sub>n</sub> is high <b>Note:</b> The availability of the DSPI_PCS <sub>n</sub> signals varies depending upon which of the four DSPI modules are used. Specifically, DSPI0 contains DSPI0_PCS[3:0], DSPI1—DSPI1_PCS[3,2,0], DSPI2—DSPI2_PCS0, and DSPI3—DSPI_PCS0. The corresponding unused bits are reserved. <b>Note:</b> DSPI_PCS0/SS must be configured as inactive high for slave mode operation.
15	Reserved, must be cleared.
14 MDIS	Module disable. Allows the clock to be stopped to non-memory mapped logic in DSPI effectively putting DSPI in a software controlled power-saving state. See <a href="#">Section 40.4.7, "Power Saving Features"</a> , for more information. This bit is set at reset. 0 Enable DSPI clocks 1 Allow external logic to disable DSPI clocks
13 DIS_TXF	Disable transmit FIFO. When the TX FIFO is disabled, transmit part of the DSPI operates as a simplified double-buffered SPI. See <a href="#">Section 40.4.2.3, "FIFO Disable Operation"</a> , for details. 0 TX FIFO is enabled 1 TX FIFO is disabled
12 DIS_RXF	Disable receive FIFO. When the RX FIFO is disabled, receive part of the DSPI operates as a simplified double-buffered SPI. See <a href="#">Section 40.4.2.3, "FIFO Disable Operation"</a> for details. 0 RX FIFO is enabled 1 RX FIFO is disabled
11 CLR_TXF	Clear TX FIFO. Flushes the TX FIFO. The CLR_TXF bit is always read as zero. 0 Do not clear the TX FIFO counter 1 Clear the TX FIFO counter <b>Note:</b> When the respective FIFO is disabled, this bit does has no effect.
10 CLR_RXF	Clear RX FIFO. Flushes the RX FIFO. The CLR_RXF bit is always read as zero. 0 Do not clear the RX FIFO counter 1 Clear the RX FIFO counter <b>Note:</b> When the respective FIFO is disabled, this bit does has no effect.
9–8 SMPL_PT	Sample point. Allows host software to select when the DSPI master samples SIN in modified transfer format. <a href="#">Figure 40-16</a> shows where the master can sample the SIN pin. 00 0 system clocks between DSPI_SCK edge and DSPI_SIN sample 01 1 system clock between DSPI_SCK edge and DSPI_SIN sample 10 2 system clocks between DSPI_SCK edge and DSPI_SIN sample 11 Reserved
7–1	Reserved, must be cleared.
0 <b>HALT</b>	Halt. Starts and stops DSPI transfers. See <a href="#">Section 40.4.1, "Start and Stop of DSPI Transfers"</a> , for details on the operation of this bit. 0 Start transfers 1 Stop transfers

### 40.3.2 DSPI Transfer Count Register (DSPIx\_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 to the DSPI TCR while the DSPI is running.

Address: 0xFC05\_C008 (DSPI0\_TCR)  
 0xFC03\_C008 (DSPI1\_TCR)  
 0xEC03\_8008 (DSPI2\_TCR)  
 0xEC03\_C008 (DSPI3\_TCR)

Access: User read/write

R	SPI_TCNT																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 40-3. DSPI Transfer Count Register (DSPIx\_TCR)

Table 40-5. DSPIx\_TCR Field Descriptions

Field	Description
31–16 SPI_TCNT	SPI transfer counter. Counts the number of SPI transfers the DSPI makes. The SPI_TCNT field increments every time the last bit of an SPI frame transmits. A value written to SPI_TCNT presets the counter to that value. SPI_TCNT is reset to 0 at the beginning of the frame when the CTCNT field is set in the executing SPI command. The transfer counter wraps around. Incrementing the counter past 65535 resets the counter to 0.
15–0	Reserved, must be cleared

### 40.3.3 DSPI Clock and Transfer Attributes Registers 0–7 (DSPIx\_CTARn)

DSPI modules each contain eight clock and transfer attribute registers (DSPI\_CTARn) used to define different transfer attribute configurations. Each DSPI\_CTAR controls:

- Frame size
- Baud rate and transfer delay values
- Clock phase
- Clock polarity
- MSB/LSB first

DSPI\_CTARs support compatibility with the QSPI module in the ColdFire family of MCUs. At the initiation of an SPI transfer, control logic selects the DSPI\_CTAR that contains the transfer's attributes. Do not write to the DSPI CTARs while the DSPI is running.

In master mode, the DSPI\_CTARn registers define combinations of transfer attributes such as frame size, clock phase and polarity, data bit ordering, baud rate, and various delays. When DSPI is configured as an SPI master, the DSPI\_PUSH[CTAS] field in the command portion of the TX FIFO entry selects which of the DSPI\_CTAR registers is used on a per-frame basis.

In slave mode, a subset of the bit fields in the DSPI\_CTAR0 registers sets the slave transfer attributes. See the individual bit descriptions for details on which bits are used in slave modes.

## DMA Serial Peripheral Interface (DSPI)

Address: 0xFC05_C00C (DSPI0_CTAR0)	0xFC03_C00C + (n × 0x04) (DSPI1_CTAR0–7)	Access: User read/write
0xFC05_C010 (DSPI0_CTAR1)	0xEC03_800C + (n × 0x04) (DSPI2_CTAR0–7)	
0xFC05_C014 (DSPI0_CTAR2)	0xEC03_C00C + (n × 0x04) (DSPI3_CTAR0–7)	
0xFC05_C018 (DSPI0_CTAR3)		
0xFC05_C01C (DSPI0_CTAR4)		
0xFC05_C020 (DSPI0_CTAR5)		
0xFC05_C024 (DSPI0_CTAR6)		
0xFC05_C028 (DSPI0_CTAR7)		

**Figure 40-4. DSPI Clock and Transfer Attributes Registers 0–7 (DSPIx\_CTARn)**

**Table 40-6. DSPIx\_CTARn Field Description**

Field	Description																																								
31 DBR	<p>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 below. See the BR field below and <a href="#">Section 40.4.3.1, “Baud Rate Generator”</a> for details on how to compute the baud rate. If the overall baud rate is divided by two or three of the system clock, the continuous SCK enable or the modified timing format enable bits must not be set.</p> <ul style="list-style-type: none"> <li>0 The baud rate is computed normally with a 50/50 duty cycle</li> <li>1 Baud rate is doubled with the duty cycle depending on the baud rate prescaler</li> </ul> <table border="1" data-bbox="473 1315 1281 1714"> <thead> <tr> <th data-bbox="479 1320 662 1336">DBR</th><th data-bbox="479 1320 850 1336">CPHA</th><th data-bbox="479 1320 1080 1336">PBR</th><th data-bbox="479 1320 1276 1336">SCK Duty Cycle</th></tr> </thead> <tbody> <tr> <td data-bbox="479 1336 662 1351">0</td><td data-bbox="479 1336 850 1351">any</td><td data-bbox="479 1336 1080 1351">any</td><td data-bbox="479 1336 1276 1351">50/50</td></tr> <tr> <td data-bbox="479 1351 662 1377">1</td><td data-bbox="479 1351 850 1377">0</td><td data-bbox="479 1351 1080 1377">00</td><td data-bbox="479 1351 1276 1377">50/50</td></tr> <tr> <td data-bbox="479 1377 662 1402">1</td><td data-bbox="479 1377 850 1402">0</td><td data-bbox="479 1377 1080 1402">01</td><td data-bbox="479 1377 1276 1402">33/66</td></tr> <tr> <td data-bbox="479 1402 662 1427">1</td><td data-bbox="479 1402 850 1427">0</td><td data-bbox="479 1402 1080 1427">10</td><td data-bbox="479 1402 1276 1427">40/60</td></tr> <tr> <td data-bbox="479 1427 662 1453">1</td><td data-bbox="479 1427 850 1453">0</td><td data-bbox="479 1427 1080 1453">11</td><td data-bbox="479 1427 1276 1453">43/57</td></tr> <tr> <td data-bbox="479 1453 662 1478">1</td><td data-bbox="479 1453 850 1478">1</td><td data-bbox="479 1453 1080 1478">00</td><td data-bbox="479 1453 1276 1478">50/50</td></tr> <tr> <td data-bbox="479 1478 662 1501">1</td><td data-bbox="479 1478 850 1501">1</td><td data-bbox="479 1478 1080 1501">01</td><td data-bbox="479 1478 1276 1501">66/33</td></tr> <tr> <td data-bbox="479 1501 662 1526">1</td><td data-bbox="479 1501 850 1526">1</td><td data-bbox="479 1501 1080 1526">10</td><td data-bbox="479 1501 1276 1526">60/40</td></tr> <tr> <td data-bbox="479 1526 662 1552">1</td><td data-bbox="479 1526 850 1552">1</td><td data-bbox="479 1526 1080 1552">11</td><td data-bbox="479 1526 1276 1552">57/43</td></tr> </tbody> </table>	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
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 40-6. DSPIx\_CTARn Field Description (continued)**

Field	Description																																				
30–27 FMSZ	<p>Frame size. Selects the number of bits transferred per frame. The FMSZ field is used in master mode and slave mode. The table below lists the frame sizes.</p> <table border="1" data-bbox="496 369 845 813"> <thead> <tr> <th data-bbox="496 369 621 411">FMSZ</th><th data-bbox="621 369 845 411">Framesize</th></tr> </thead> <tbody> <tr><td data-bbox="496 411 621 454">0000</td><td data-bbox="621 411 845 454">Reserved</td></tr> <tr><td data-bbox="496 454 621 496">0001</td><td data-bbox="621 454 845 496">Reserved</td></tr> <tr><td data-bbox="496 496 621 538">0010</td><td data-bbox="621 496 845 538">Reserved</td></tr> <tr><td data-bbox="496 538 621 580">0011</td><td data-bbox="621 538 845 580">4</td></tr> <tr><td data-bbox="496 580 621 623">0100</td><td data-bbox="621 580 845 623">5</td></tr> <tr><td data-bbox="496 623 621 665">0101</td><td data-bbox="621 623 845 665">6</td></tr> <tr><td data-bbox="496 665 621 707">0110</td><td data-bbox="621 665 845 707">7</td></tr> <tr><td data-bbox="496 707 621 749">0111</td><td data-bbox="621 707 845 749">8</td></tr> </tbody> </table> <table border="1" data-bbox="878 369 1228 813"> <thead> <tr> <th data-bbox="878 369 1003 411">FMSZ</th><th data-bbox="1003 369 1228 411">Framesize</th></tr> </thead> <tbody> <tr><td data-bbox="878 411 1003 454">1000</td><td data-bbox="1003 411 1228 454">9</td></tr> <tr><td data-bbox="878 454 1003 496">1001</td><td data-bbox="1003 454 1228 496">10</td></tr> <tr><td data-bbox="878 496 1003 538">1010</td><td data-bbox="1003 496 1228 538">11</td></tr> <tr><td data-bbox="878 538 1003 580">1011</td><td data-bbox="1003 538 1228 580">12</td></tr> <tr><td data-bbox="878 580 1003 623">1100</td><td data-bbox="1003 580 1228 623">13</td></tr> <tr><td data-bbox="878 623 1003 665">1101</td><td data-bbox="1003 623 1228 665">14</td></tr> <tr><td data-bbox="878 665 1003 707">1110</td><td data-bbox="1003 665 1228 707">15</td></tr> <tr><td data-bbox="878 707 1003 749">1111</td><td data-bbox="1003 707 1228 749">16</td></tr> </tbody> </table>	FMSZ	Framesize	0000	Reserved	0001	Reserved	0010	Reserved	0011	4	0100	5	0101	6	0110	7	0111	8	FMSZ	Framesize	1000	9	1001	10	1010	11	1011	12	1100	13	1101	14	1110	15	1111	16
FMSZ	Framesize																																				
0000	Reserved																																				
0001	Reserved																																				
0010	Reserved																																				
0011	4																																				
0100	5																																				
0101	6																																				
0110	7																																				
0111	8																																				
FMSZ	Framesize																																				
1000	9																																				
1001	10																																				
1010	11																																				
1011	12																																				
1100	13																																				
1101	14																																				
1110	15																																				
1111	16																																				
26 CPOL	<p>Clock polarity. Selects the inactive state of the serial communications clock (DSPI_SCK). This bit is used in master and slave mode. For successful communication between serial devices, the devices must have identical clock polarities. When the continuous selection format is selected (CONT or DCONT is set), 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. For more information on continuous selection format, refer to <a href="#">Section 40.4.4.5, “Continuous Selection Format.”</a></p> <p>0 The inactive state value of DSPI_SCK is low 1 The inactive state value of DSPI_SCK is high</p>																																				
25 CPHA	<p>Clock phase. Selects which edge of DSPI_SCK causes data to change and which edge causes data to be captured. This bit is used in master and slave mode. For successful communication between serial devices, the devices must have identical clock phase settings.</p> <p><b>Note:</b> When the continuous selection format is selected (CONT or DCONT is set), switching between clock phases without stopping the DSPI can cause errors in the transfer.</p> <p>0 Data is captured on the leading edge of DSPI_SCK and changed on the following edge 1 Data is changed on the leading edge of DSPI_SCK and captured on the following edge</p>																																				
24 LSBFE	<p>LSB first enable. Selects if the LSB or MSB of the frame is transferred first. This bit is only used in master mode.</p> <p>0 Data is transferred MSB first 1 Data is transferred LSB first</p>																																				
23–22 PCSSCK	<p>PCS to SCK delay prescaler. Selects the prescaler value for the delay between assertion of DSPI_PCS and the first edge of the DSPI_SCK. This field is only used in master mode.</p> <p><b>Note:</b> When the continuous selection format is selected (CONT or DCONT is set), switching the PCS to SCK delay prescaler without stopping the DSPI can cause errors in the transfer.</p> <p><b>Note:</b> See <a href="#">Section 40.4.3.2, “PCS to SCK Delay (tCSC),”</a> for details on calculating the PCS to SCK delay.</p> <p>00 1 clock DSPI_PCS to DSPI_SCK delay prescaler 01 3 clock DSPI_PCS to DSPI_SCK delay prescaler 10 5 clock DSPI_PCS to DSPI_SCK delay prescaler 11 7 clock DSPI_PCS to DSPI_SCK delay prescaler</p>																																				

**Table 40-6. DSPI<sub>x</sub>\_CTAR<sub>n</sub> Field Description (continued)**

Field	Description																																				
21–20 PASC	<p>After SCK delay prescaler. Selects the prescaler value for the delay between the last edge of DSPI_SCK and the negation of DSPI_PCS. This field is only used in master mode. The ASC field description in <a href="#">Table 40-6</a> explains how to compute the after SCK delay.</p> <p>00 1 clock delay between last edge of DSPI_SCK and DSPI_PCS negation prescaler            01 3 clock delay between last edge of DSPI_SCK and DSPI_PCS negation prescaler            10 5 clock delay between last edge of DSPI_SCK and DSPI_PCS negation prescaler            11 7 clock delay between last edge of DSPI_SCK and DSPI_PCS negation prescaler</p>																																				
19–18 PDT	<p>Delay after transfer prescaler. The PDT field selects the prescaler value for the delay between the negation of the DSPI_PCS signal at the end of a frame and the assertion of DSPI_PCS at the beginning of the next frame. The PDT field is only used in master mode. The DT field description in <a href="#">Table 40-6</a> explains how to compute the delay after transfer.</p> <p>00 1 clock delay between negation of DSPI_PCS to assertion of next DSPI_PCS prescaler            01 3 clock delay between negation of DSPI_PCS to assertion of next DSPI_PCS prescaler            10 5 clock delay between negation of DSPI_PCS to assertion of next DSPI_PCS prescaler            11 7 clock delay between negation of DSPI_PCS to assertion of next DSPI_PCS prescaler</p>																																				
17–16 PBR	<p>Baud rate prescaler. 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 (DSPI_SCK). The system clock is divided by the prescaler value before the baud rate selection takes place. The description for <a href="#">Section 40.4.3.1, “Baud Rate Generator”</a> details how to compute the baud rate.</p> <p>00 2 clock prescaler to divide system clock            01 3 clock prescaler to divide system clock            10 5 clock prescaler to divide system clock            11 7 clock prescaler to divide system clock</p>																																				
15–12 CSSCK	<p>PCS to SCK delay scaler. 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 DSPI_PCS and the first edge of the DSPI_SCK. The table below lists the scaler values.</p> <p><b>Note:</b> When the continuous selection format is selected (CONT or DCONT is set), switching the PCS to SCK delay prescaler without stopping the DSPI can cause errors in the transfer.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;">CSSCK</th> <th style="text-align: center;">PCS to SCK Delay Scaler Value</th> </tr> </thead> <tbody> <tr><td style="text-align: center;">0000</td><td style="text-align: center;">2</td></tr> <tr><td style="text-align: center;">0001</td><td style="text-align: center;">4</td></tr> <tr><td style="text-align: center;">0010</td><td style="text-align: center;">8</td></tr> <tr><td style="text-align: center;">0011</td><td style="text-align: center;">16</td></tr> <tr><td style="text-align: center;">0100</td><td style="text-align: center;">32</td></tr> <tr><td style="text-align: center;">0101</td><td style="text-align: center;">64</td></tr> <tr><td style="text-align: center;">0110</td><td style="text-align: center;">128</td></tr> <tr><td style="text-align: center;">0111</td><td style="text-align: center;">256</td></tr> </tbody> </table> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;">CSSCK</th> <th style="text-align: center;">PCS to SCK Delay Scaler Value</th> </tr> </thead> <tbody> <tr><td style="text-align: center;">1000</td><td style="text-align: center;">512</td></tr> <tr><td style="text-align: center;">1001</td><td style="text-align: center;">1024</td></tr> <tr><td style="text-align: center;">1010</td><td style="text-align: center;">2048</td></tr> <tr><td style="text-align: center;">1011</td><td style="text-align: center;">4096</td></tr> <tr><td style="text-align: center;">1100</td><td style="text-align: center;">8192</td></tr> <tr><td style="text-align: center;">1101</td><td style="text-align: center;">16384</td></tr> <tr><td style="text-align: center;">1110</td><td style="text-align: center;">32768</td></tr> <tr><td style="text-align: center;">1111</td><td style="text-align: center;">65536</td></tr> </tbody> </table> <p><b>Note:</b> See <a href="#">Section 40.4.3.2, “PCS to SCK Delay (tCSC),”</a> for details on calculating the PCS to SCK delay.</p>	CSSCK	PCS to SCK Delay Scaler Value	0000	2	0001	4	0010	8	0011	16	0100	32	0101	64	0110	128	0111	256	CSSCK	PCS to SCK Delay Scaler Value	1000	512	1001	1024	1010	2048	1011	4096	1100	8192	1101	16384	1110	32768	1111	65536
CSSCK	PCS to SCK Delay Scaler Value																																				
0000	2																																				
0001	4																																				
0010	8																																				
0011	16																																				
0100	32																																				
0101	64																																				
0110	128																																				
0111	256																																				
CSSCK	PCS to SCK Delay Scaler Value																																				
1000	512																																				
1001	1024																																				
1010	2048																																				
1011	4096																																				
1100	8192																																				
1101	16384																																				
1110	32768																																				
1111	65536																																				

**Table 40-6. DSPIx\_CTARn Field Description (continued)**

Field	Description																																					
11–8 ASC	<p>After SCK delay scaler. 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 DSPI_SCK and the negation of DSPI_PCS. The table below lists the scaler values.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;">ASC</th> <th style="text-align: center;">After SCK Delay Scaler Value</th> </tr> </thead> <tbody> <tr><td style="text-align: center;">0000</td><td style="text-align: center;">2</td></tr> <tr><td style="text-align: center;">0001</td><td style="text-align: center;">4</td></tr> <tr><td style="text-align: center;">0010</td><td style="text-align: center;">8</td></tr> <tr><td style="text-align: center;">0011</td><td style="text-align: center;">16</td></tr> <tr><td style="text-align: center;">0100</td><td style="text-align: center;">32</td></tr> <tr><td style="text-align: center;">0101</td><td style="text-align: center;">64</td></tr> <tr><td style="text-align: center;">0110</td><td style="text-align: center;">128</td></tr> <tr><td style="text-align: center;">0111</td><td style="text-align: center;">256</td></tr> </tbody> </table> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;">ASC</th> <th style="text-align: center;">After SCK Delay Scaler Value</th> </tr> </thead> <tbody> <tr><td style="text-align: center;">1000</td><td style="text-align: center;">512</td></tr> <tr><td style="text-align: center;">1001</td><td style="text-align: center;">1024</td></tr> <tr><td style="text-align: center;">1010</td><td style="text-align: center;">2048</td></tr> <tr><td style="text-align: center;">1011</td><td style="text-align: center;">4096</td></tr> <tr><td style="text-align: center;">1100</td><td style="text-align: center;">8192</td></tr> <tr><td style="text-align: center;">1101</td><td style="text-align: center;">16384</td></tr> <tr><td style="text-align: center;">1110</td><td style="text-align: center;">32768</td></tr> <tr><td style="text-align: center;">1111</td><td style="text-align: center;">65536</td></tr> </tbody> </table>		ASC	After SCK Delay Scaler Value	0000	2	0001	4	0010	8	0011	16	0100	32	0101	64	0110	128	0111	256	ASC	After SCK Delay Scaler Value	1000	512	1001	1024	1010	2048	1011	4096	1100	8192	1101	16384	1110	32768	1111	65536
ASC	After SCK Delay Scaler Value																																					
0000	2																																					
0001	4																																					
0010	8																																					
0011	16																																					
0100	32																																					
0101	64																																					
0110	128																																					
0111	256																																					
ASC	After SCK Delay Scaler Value																																					
1000	512																																					
1001	1024																																					
1010	2048																																					
1011	4096																																					
1100	8192																																					
1101	16384																																					
1110	32768																																					
1111	65536																																					
7–4 DT	<p>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 DSPI_PCS signal at the end of a frame and the assertion of DSPI_PCS at the beginning of the next frame. The table below lists the scaler values.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;">DT</th> <th style="text-align: center;">Delay after Transfer Scaler Value</th> </tr> </thead> <tbody> <tr><td style="text-align: center;">0000</td><td style="text-align: center;">2</td></tr> <tr><td style="text-align: center;">0001</td><td style="text-align: center;">4</td></tr> <tr><td style="text-align: center;">0010</td><td style="text-align: center;">8</td></tr> <tr><td style="text-align: center;">0011</td><td style="text-align: center;">16</td></tr> <tr><td style="text-align: center;">0100</td><td style="text-align: center;">32</td></tr> <tr><td style="text-align: center;">0101</td><td style="text-align: center;">64</td></tr> <tr><td style="text-align: center;">0110</td><td style="text-align: center;">128</td></tr> <tr><td style="text-align: center;">0111</td><td style="text-align: center;">256</td></tr> </tbody> </table> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;">DT</th> <th style="text-align: center;">Delay after Transfer Scaler Value</th> </tr> </thead> <tbody> <tr><td style="text-align: center;">1000</td><td style="text-align: center;">512</td></tr> <tr><td style="text-align: center;">1001</td><td style="text-align: center;">1024</td></tr> <tr><td style="text-align: center;">1010</td><td style="text-align: center;">2048</td></tr> <tr><td style="text-align: center;">1011</td><td style="text-align: center;">4096</td></tr> <tr><td style="text-align: center;">1100</td><td style="text-align: center;">8192</td></tr> <tr><td style="text-align: center;">1101</td><td style="text-align: center;">16384</td></tr> <tr><td style="text-align: center;">1110</td><td style="text-align: center;">32768</td></tr> <tr><td style="text-align: center;">1111</td><td style="text-align: center;">65536</td></tr> </tbody> </table>		DT	Delay after Transfer Scaler Value	0000	2	0001	4	0010	8	0011	16	0100	32	0101	64	0110	128	0111	256	DT	Delay after Transfer Scaler Value	1000	512	1001	1024	1010	2048	1011	4096	1100	8192	1101	16384	1110	32768	1111	65536
DT	Delay after Transfer Scaler Value																																					
0000	2																																					
0001	4																																					
0010	8																																					
0011	16																																					
0100	32																																					
0101	64																																					
0110	128																																					
0111	256																																					
DT	Delay after Transfer Scaler Value																																					
1000	512																																					
1001	1024																																					
1010	2048																																					
1011	4096																																					
1100	8192																																					
1101	16384																																					
1110	32768																																					
1111	65536																																					

**Note:** See [Section 40.4.3.3, “After SCK Delay \(tASC\),”](#) for more details on calculating the after SCK delay.

**Note:** See [Section 40.4.3.4, “Delay after Transfer \(tDT\),”](#) for more details on calculating the delay after transfer.

**Table 40-6. DSPIx\_CTARn Field Description (continued)**

Field	Description																																					
3–0 BR	<p>Baud rate scaler. Selects the scaler value for the baud rate. This field is only used in master mode. The pre-scaled system clock is divided by the baud rate scaler to generate the frequency of the DSPI_SCK. The table below lists the baud rate scaler values.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;">BR</th> <th style="text-align: center;">Baud Rate Scaler Value</th> </tr> </thead> <tbody> <tr><td style="text-align: center;">0000</td><td style="text-align: center;">2</td></tr> <tr><td style="text-align: center;">0001</td><td style="text-align: center;">4</td></tr> <tr><td style="text-align: center;">0010</td><td style="text-align: center;">6</td></tr> <tr><td style="text-align: center;">0011</td><td style="text-align: center;">8</td></tr> <tr><td style="text-align: center;">0100</td><td style="text-align: center;">16</td></tr> <tr><td style="text-align: center;">0101</td><td style="text-align: center;">32</td></tr> <tr><td style="text-align: center;">0110</td><td style="text-align: center;">64</td></tr> <tr><td style="text-align: center;">0111</td><td style="text-align: center;">128</td></tr> </tbody> </table> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;">BR</th> <th style="text-align: center;">Baud Rate Scaler Value</th> </tr> </thead> <tbody> <tr><td style="text-align: center;">1000</td><td style="text-align: center;">256</td></tr> <tr><td style="text-align: center;">1001</td><td style="text-align: center;">512</td></tr> <tr><td style="text-align: center;">1010</td><td style="text-align: center;">1024</td></tr> <tr><td style="text-align: center;">1011</td><td style="text-align: center;">2048</td></tr> <tr><td style="text-align: center;">1100</td><td style="text-align: center;">4096</td></tr> <tr><td style="text-align: center;">1101</td><td style="text-align: center;">8192</td></tr> <tr><td style="text-align: center;">1110</td><td style="text-align: center;">16384</td></tr> <tr><td style="text-align: center;">1111</td><td style="text-align: center;">32768</td></tr> </tbody> </table> <p><b>Note:</b> See <a href="#">Section 40.4.3.1, “Baud Rate Generator,”</a> for more details on calculating the baud rate.</p>	BR	Baud Rate Scaler Value	0000	2	0001	4	0010	6	0011	8	0100	16	0101	32	0110	64	0111	128	BR	Baud Rate Scaler Value	1000	256	1001	512	1010	1024	1011	2048	1100	4096	1101	8192	1110	16384	1111	32768	
BR	Baud Rate Scaler Value																																					
0000	2																																					
0001	4																																					
0010	6																																					
0011	8																																					
0100	16																																					
0101	32																																					
0110	64																																					
0111	128																																					
BR	Baud Rate Scaler Value																																					
1000	256																																					
1001	512																																					
1010	1024																																					
1011	2048																																					
1100	4096																																					
1101	8192																																					
1110	16384																																					
1111	32768																																					

#### 40.3.4 DSPI Status Register (DSPIx\_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.

Address 0xFC05_C02C (DSPI0_SR) : 0xFC03_C02C (DSPI1_SR) 0xEC03_802C (DSPI2_SR) 0xEC03_C02C (DSPI3_SR)																Access: User Read/Write					
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16					
W	TCF	TXRXS	0	EOQF	TFUF	0	FFFF	0	0	0	0	0	RFOF	0	RFDF	0					
Reset	w1c				w1c	w1c							w1c		w1c						
R										0	0	0	0	0	0	0	0	0	0	0	
W										0	0	0	0	0	0	0	0	0	0	0	
Reset										0	0	0	0	0	0	0	0	0	0	0	
R										15	14	13	12	11	10	9	8	7	6	5	4
W										TXCTR				TXNXTPTR				RXCTR		POPNXTPTR	
Reset										0	0	0	0	0	0	0	0	0	0	0	0

**Figure 40-5. DSPI Status Register (DSPIx\_SR)**

**Table 40-7. DSPIx\_SR Field Descriptions**

Field	Description
31 <b>TCF</b>	Transfer complete flag. Indicates all bits in a frame have been shifted out. The TCF bit is set after the last incoming databit is sampled, but before the $t_{ASC}$ delay starts. Refer to <a href="#">Section 40.4.4.1, "Classic SPI Transfer Format (CPHA = 0)"</a> for details. <u>The TCF bit is cleared by writing 1 to it.</u> 0 Transfer not complete 1 Transfer complete
30 <b>TXRXS</b>	TX and RX status. Reflects the status of the DSPI. See <a href="#">Section 40.4.1, "Start and Stop of DSPI Transfers"</a> for information on what causes this bit to be cleared or set. 0 TX and RX operations are disabled (DSPI is in stopped state) 1 TX and RX operations are enabled (DSPI is in running state)
29	Reserved, must be cleared.
28 <b>EOQF</b>	End of queue flag. Indicates transmission in progress is the last entry in a queue. The EOQF bit is set when the TX FIFO entry has the EOQ bit set in the command halfword and after the last incoming databit is sampled, but before the $t_{ASC}$ delay starts. Refer to <a href="#">Section 40.4.4.1, "Classic SPI Transfer Format (CPHA = 0)"</a> for details.  <u>The EOQF bit is cleared by writing 1 to it. When the EOQF bit is set, the TXRXS bit is automatically cleared.</u> 0 EOQ is not set in the executing SPI command 1 EOQ bit is set in the executing SPI command <b>Note:</b> EOQF does not function in slave mode.
27 <b>TFUF</b>	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. The TFUF bit is set when the TX FIFO of a DSPI operating in slave mode is empty, and a transfer is initiated by an external SPI master. The TFUF bit is cleared by writing 1 to it. 0 TX FIFO underflow has not occurred 1 TX FIFO underflow has occurred
26	Reserved, must be cleared.
25 <b>TFFF</b>	Transmit FIFO fill flag. Indicates that the TX FIFO can be filled. 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. Therefore, this bit is set after DSPI_MCR[MDIS] is cleared after a reset. The TFFF bit can be cleared by writing 1 to it or by an acknowledgement from the eDMA controller when the TX FIFO is full. 0 TX FIFO is full 1 TX FIFO is not full
24–20	Reserved, must be cleared.
19 <b>RFOF</b>	Receive FIFO overflow flag. 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 is cleared by writing 1 to it. 0 RX FIFO overflow has not occurred 1 RX FIFO overflow has occurred
18	Reserved, must be cleared.
17 <b>RFDF</b>	Receive FIFO drain flag. Indicates that the RX FIFO can be drained. 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 an acknowledgement from the eDMA controller when the RX FIFO is empty. 0 RX FIFO is empty 1 RX FIFO is not empty <b>Note:</b> In the interrupt service routine, RFDF must be cleared only after the DSPI_POPR register is read.
16	Reserved, must be cleared.

**Table 40-7. DSPIx\_SR Field Descriptions (continued)**

Field	Description
15–12 TXCTR	TX FIFO counter. 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 an SPI command is executed and the SPI data is transferred to the shift register.
11–8 TXNXTPTR	Transmit next pointer. 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 <a href="#">Section 40.4.2.4, “TX FIFO Buffering Mechanism,”</a> for more details.
7–4 RXCTR	RX FIFO counter. Indicates the number of entries in the RX FIFO. The RXCTR is decremented every time the DSPI_POPR is read. The RXCTR is incremented after the last incoming databit is sampled, but before the $t_{ASC}$ delay starts. Refer to <a href="#">Section 40.4.4.1, “Classic SPI Transfer Format (CPHA = 0)”</a> for details.
3–0 POPNXTPTR	Pop next pointer. Contains a pointer to the RX FIFO entry that is returned when the DSPI_POPR is read. The POPNXTPTR is updated when the DSPI_POPR is read. See <a href="#">Section 40.4.2.5, “RX FIFO Buffering Mechanism”</a> for more details.

### 40.3.5 DSPI DMA/Interrupt Request Select and Enable Register (DSPIx\_RSER)

The DSPI\_RSER serves two purposes. It enables flag bits in the DSPI\_SR to generate DMA requests or interrupt requests. The DSPI\_RSER also selects the type of request to be generated. See the individual bit descriptions for information on the types of requests the bits support. Do not write to the DSPI\_RSER while the DSPI is running.

Address 0xFC05\_C030 (DSPI0\_RSER) Access: User Read/Write  
 : 0xFC03\_C030 (DSPI1\_RSER)  
 0xEC03\_8030 (DSPI2\_RSER)  
 0xEC03\_C030 (DSPI3\_RSER)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	TCF _RE	0	0	EOQF _RE	TFUF _RE	0	TFFF _RE	TFFF _DIRS	0	0	0	0	RFOF _RE	0	RFDF _RE	RFDF _DIRS
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	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 40-6. DSPI DMA/Interrupt Request Select and Enable Register (DSPIx\_RSER)**

**Table 40-8. DSPIx\_RSER Field Descriptions**

<b>Field</b>	<b>Description</b>
31 TCF_RE	Transmission complete request enable. Enables DSPI_SR[TCF] flag to generate an interrupt request. 0 TCF interrupt requests are disabled 1 TCF interrupt requests are enabled
30–29	Reserved, must be cleared.
28 EOQF_RE	DSPI finished request enable. Enables the DSPI_SR[EOQF] flag to generate an interrupt request. 0 EOQF interrupt requests are disabled 1 EOQF interrupt requests are enabled
27 TFUF_RE	Transmit FIFO underflow request enable. Enables the DSPI_SR[TFUF] flag to generate an interrupt request. 0 TFUF interrupt requests are disabled 1 TFUF interrupt requests are enabled
26	Reserved, must be cleared.
25 TFFF_RE	Transmit FIFO fill request enable. Enables the DSPI_SR[TFFF] flag to generate a request. The TFFF_DIRS bit selects between generating an interrupt request or a DMA requests. 0 TFFF interrupt or DMA requests are disabled 1 TFFF interrupt or DMA requests are enabled
24 TFFF_DIRS	Transmit FIFO fill DMA or interrupt request select. Selects between generating a DMA request or an interrupt request. When the DSPI_SR[TFFF] flag bit and the DSPI_RSER[TFFF_RE] bit are set, this bit selects between generating an interrupt request or a DMA request. 0 TFFF flag generates interrupt requests 1 TFFF flag generates DMA requests
23–20	Reserved, must be cleared.
19 RFOF_RE	Receive FIFO overflow request enable. Enables the DSPI_SR[RFOF] flag to generate an interrupt request. 0 RFOF interrupt requests are disabled 1 RFOF interrupt requests are enabled
18	Reserved, must be cleared.
17 RFDF_RE	Receive FIFO drain request enable. Enables the DSPI_SR[RFDF] flag to generate a request. The RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 RFDF interrupt or DMA requests are disabled 1 RFDF interrupt or DMA requests are enabled
16 RFDF_DIRS	Receive FIFO drain DMA or interrupt request select. Selects between generating a DMA request or an interrupt request. When the DSPI_SR[RFDF] flag bit and the DSPI_RSER[RFDF_RE] bit are set, the RFDF_DIRS bit selects between generating an interrupt request or a DMA request. 0 RFDF flag generates interrupt requests 1 RFDF flag generates DMA requests
15–0	Reserved, must be cleared.

### 40.3.6 DSPI Push Transmit FIFO Register (DSPIx\_PUSHR)

The DSPI\_PUSHR provides a means to write to the TX FIFO. SPI commands and data written to this register is transferred to the TX FIFO. See [Section 40.4.2.4, “TX FIFO Buffering Mechanism,”](#) for more information. Write accesses of 8- or 16-bits to the DSPI\_PUSHR transfer 32 bits to the TX FIFO.

#### NOTE

Only the TXDATA field is used for DSPI slaves.

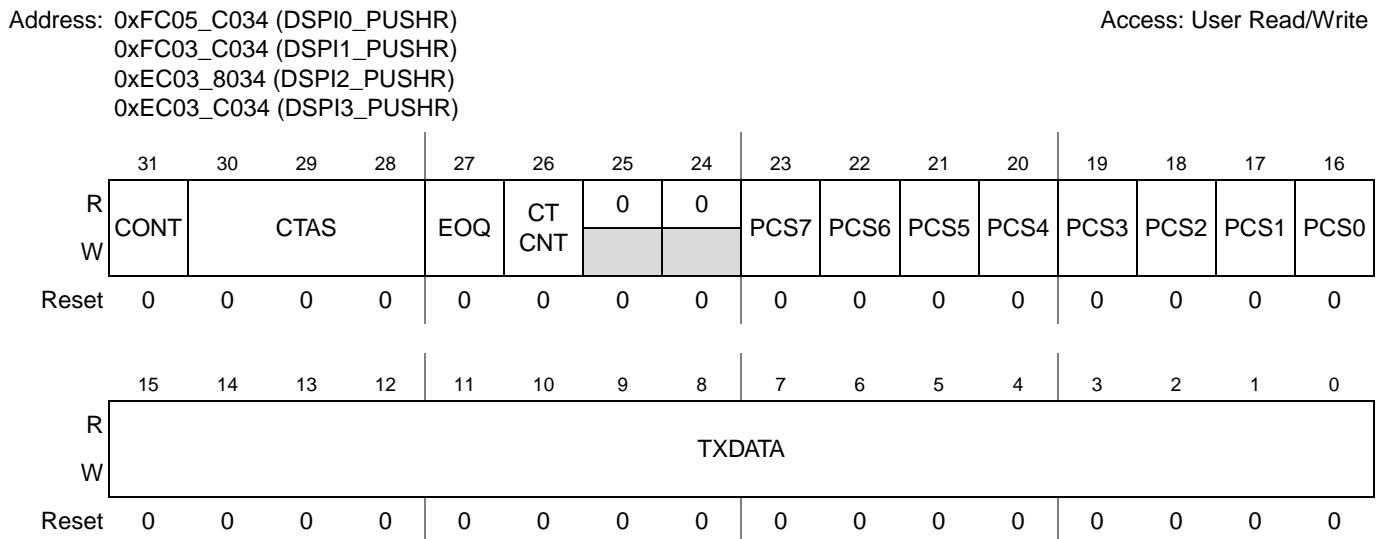


Figure 40-7. DSPI Push Transmit FIFO Register (DSPIx\_PUSHHR)

Table 40-9. DSPIx\_PUSHHR Field Descriptions

Field	Description
31 CONT	Continuous peripheral chip select enable. 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 <a href="#">Section 40.4.4.5, "Continuous Selection Format,"</a> for more information. 0 Return DSPI_PCSn signals to their inactive state between transfers 1 Keep DSPI_PCSn signals asserted between transfers
30–28 CTAS	Clock and transfer attributes select. Selects which of the DSPI_CTARn registers is used to set the transfer attributes for the associated SPI frame. This field is used only in SPI master mode. In SPI slave mode, DSPI_CTAR0 is used instead. 000 DSPI_CTAR0 001 DSPI_CTAR1 010 DSPI_CTAR2 011 DSPI_CTAR3 100 DSPI_CTAR4 101 DSPI_CTAR5 110 DSPI_CTAR6 111 DSPI_CTAR7
27 EOQ	End of queue. 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 DSPI_SR[EOQF] bit is set. This bit is used only in SPI master mode. 0 The SPI data is not the last data to transfer 1 The SPI data is the last data to transfer
26 CTCNT	Clear SPI_TCNT. Provides a means for host software to clear the SPI transfer counter. The CTCNT bit clears the DSPI_TCR[SPI_TCNT] field. The SPI_TCNT field is cleared before transmission of the current SPI frame begins. This bit is used only in SPI master mode. 0 Do not clear DSPI_TCR[SPI_TCNT] field 1 Clear DSPI_TCR[SPI_TCNT] field
25–24	Reserved, must be cleared.

**Table 40-9. DSPIx\_PUSHR Field Descriptions (continued)**

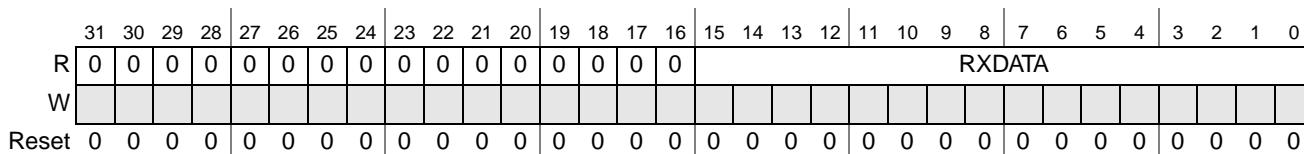
Field	Description
23–16 PCSn	<p>Peripheral chip select <math>n</math>. Selects which DSPI_PCS<math>n</math> signals are asserted for the transfer. This bit is used only in SPI master mode.</p> <ul style="list-style-type: none"> <li>0 Negate the DSPI_PCS<math>n</math> signal</li> <li>1 Assert the DSPI_PCS<math>n</math> signal</li> </ul> <p><b>Note:</b> The availability of the DSPI_PCS<math>n</math> signals varies depending upon which of the four DSPI modules are used. Specifically, DSPI0 contains DSPI_PCS[3:0], DSPI1—DSPI_PCS[3,2,0], DSPI2 and DSPI3—DSPI_PCS0. The corresponding unused bits are reserved.</p>
15–0 TXDATA	<p>Transmit data. Holds SPI data to be transferred according to the associated SPI command.</p> <p><b>Note:</b> TXDATA is used in slave mode.</p>

#### 40.3.7 DSPI Pop Receive FIFO Register (DSPIx\_POPR)

The DSPI\_POPR provides a means to read the RX FIFO. See [Section 40.4.2.5, “RX FIFO Buffering Mechanism”](#) for a description of the RX FIFO operations. Eight or 16-bit read accesses to the DSPI\_POPR read from the RX FIFO and update the counter and pointer.

Address: 0xFC05\_C038 (DSPI\_POPR)  
0xFC03\_C038 (DSPI1\_POPR)  
0xEC03\_8038 (DSPI2\_POPR)  
0xEC03\_C038 (DSPI3\_POPR)

Access: User read-only



**Figure 40-8. DSPI Pop Receive FIFO Register (DSPIx\_POPR)**

**Table 40-10. DSPIx\_POPR Field Descriptions**

Field	Description
31–16	Reserved, must be cleared.
15–0 RXDATA	Received data. Contains the SPI data from the RX FIFO entry pointed to by the pop next data pointer (DSPI_SR[POPNXTPTR]).

#### 40.3.8 DSPI Transmit FIFO Registers 0–15 (DSPIx\_TXFR $n$ )

The DSPI\_TXFR $n$  registers provide visibility into TX FIFO for debugging purposes. Each register is an entry in TX FIFO. The registers are read-only and cannot be modified. Reading the DSPI\_TXFR $n$  registers does not alter the state of TX FIFO. The 16-entry deep FIFO is implemented with 16 registers, DSPI\_TXFR0–15.

Address: 0xFC05_C03C (DSPI_TXFR0)	0xFC05_C05C (DSPI_TXFR8)	0xFC03_C03C + (n × 0x04) (DSPI1_RXFR0–15)	Access: User read-only
0xFC05_C040 (DSPI_TXFR1)	0xFC05_C060 (DSPI_TXFR9)	0xEC03_803C + (n × 0x04) (DSPI2_RXFR0–15)	
0xFC05_C044 (DSPI_TXFR2)	0xFC05_C064 (DSPI_TXFR10)	0xEC03_C03C + (n × 0x04) (DSPI3_RXFR0–15)	
0xFC05_C048 (DSPI_TXFR3)	0xFC05_C068 (DSPI_TXFR11)		
0xFC05_C04C (DSPI_TXFR4)	0xFC05_C06C (DSPI_TXFR12)		
0xFC05_C050 (DSPI_TXFR5)	0xFC05_C070 (DSPI_TXFR13)		
0xFC05_C054 (DSPI_TXFR6)	0xFC05_C074 (DSPI_TXFR14)		
0xFC05_C058 (DSPI_TXFR7)	0xFC05_C078 (DSPI_TXFR15)		

	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	TXCMD
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 40-9. DSPI Transmit FIFO Registers 0–15 (DSPIx\_RXFRn)

Table 40-11. DSPIx\_RXFRn Field Descriptions

Field	Description
31–16 TXCMD	Transmit command. Contains the command that sets the transfer attributes for the SPI data. See <a href="#">Section 40.3.6, “DSPI Push Transmit FIFO Register (DSPIx_PUSHR),”</a> for details on the command field.
15–0 TXDATA	Transmit data. Contains the SPI data to be shifted out.

### 40.3.9 DSPI Receive FIFO Registers 0–15 (DSPIx\_RXFRn)

The DSPI\_RXFRn 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\_RXFRn registers does not alter the state of the RX FIFO. The device uses 16 registers to implement the RX FIFO; DSPI\_RXFR0–15 are used.

Address: 0xFC05_C07C (DSPI_RXFR0)	0xFC05_C09C (DSPI_RXFR8)	0xFC03_C07C + (n × 0x04) (DSPI1_RXFR0–15)	Access: User read-only
0xFC05_C080 (DSPI_RXFR1)	0xFC05_C0A0 (DSPI_RXFR9)	0xEC03_807C + (n × 0x04) (DSPI2_RXFR0–15)	
0xFC05_C084 (DSPI_RXFR2)	0xFC05_C0A4 (DSPI_RXFR10)	0xEC03_C07C + (n × 0x04) (DSPI3_RXFR0–15)	
0xFC05_C088 (DSPI_RXFR3)	0xFC05_C0A8 (DSPI_RXFR11)		
0xFC05_C08C (DSPI_RXFR4)	0xFC05_C0AC (DSPI_RXFR12)		
0xFC05_C090 (DSPI_RXFR5)	0xFC05_C0B0 (DSPI_RXFR13)		
0xFC05_C094 (DSPI_RXFR6)	0xFC05_C0B4 (DSPI_RXFR14)		
0xFC05_C098 (DSPI_RXFR7)	0xFC05_C0B8 (DSPI_RXFR15)		

	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	RXDATA
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 40-10. DSPI Receive FIFO Registers (DSPIx\_RXFRn)

Table 40-12. DSPI\_RXFRn Field Description

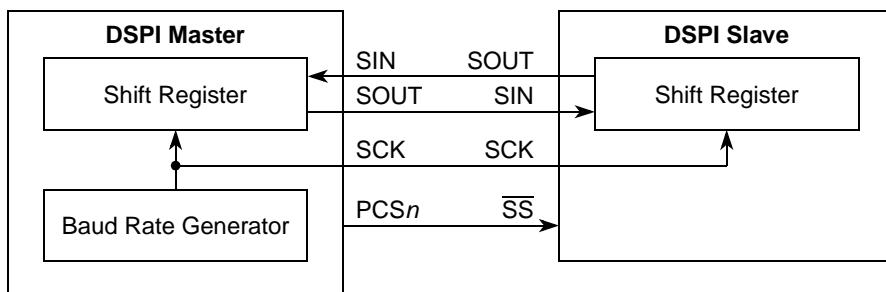
Field	Description
31–16	Reserved, must be cleared.
15–0 RXDATA	Receive data. Contains the received SPI data.

## 40.4 Functional Description

The DSPI supports full-duplex, synchronous serial communications between the MCU and external peripheral devices. The DSPI supports up to 32 queued SPI transfers at once (16 transmit and 16 receive) in the DSPI resident FIFOs, thereby eliminating CPU intervention between transfers.

The DSPI\_CTAR $n$  registers hold clock and transfer attributes. The SPI configuration can select which CTAR to use on a frame by frame basis by setting the DSPI\_PUSHR[CTAS] field. See [Section 40.3.3, “DSPI Clock and Transfer Attributes Registers 0–7 \(DSPIx\\_CTAR \$n\$ \)](#),” for information on DSPI\_CTAR $n$  fields.

The 16-bit shift register in the master and the 16-bit shift register in the slave are linked by the SOUT and SIN signals to form a distributed 32-bit register. When a data transfer operation is performed, data is serially shifted a pre-determined number of bit positions. Because the registers are linked, data exchanged between the master and the slave; the data that was in the master’s shift register is now in the shift register of the slave and vice versa. At the end of a transfer, the DSPI\_SR[TCF] bit is set to indicate a completed transfer. [Figure 40-11](#) illustrates how master and slave data is exchanged.



**Figure 40-11. SPI Serial Protocol Overview**

The DSPI has peripheral chip select (DSPI\_PCS $n$ ) signals that select which of the slaves to communicate with.

Transfer protocols and timing properties are shared by the three DSPI configurations; these properties are described independently of the configuration in [Section 40.4.4, “Transfer Formats.”](#) The transfer rate and delay settings are described in section [Section 40.4.3, “DSPI Baud Rate and Clock Delay Generation.”](#)

See [Section 40.4.7, “Power Saving Features”](#) for information on the power-saving features of the DSPI.

### 40.4.1 Start and Stop of DSPI Transfers

The DSPI has two operating states; stopped and running. 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. Master/slave mode must only be changed when the DSPI is halted (DSPI\_MCR[HALT] is set). The DSPI\_SR[TXRXS] bit is cleared in this state. In the running state, serial transfers take place. The DSPI\_SR[TXRXS] bit is set in the running state. [Figure 40-12](#) shows a state diagram of the start and stop mechanism. The transitions are described in [Table 40-13](#).

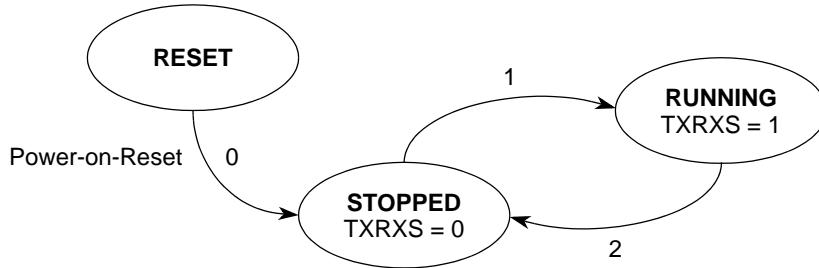


Figure 40-12. DSPI Start and Stop State Diagram

Table 40-13. State Transitions for Start and Stop of DSPI Transfers

Transition #	Current State	Next State	Description
0	RESET	STOPPED	Generic power-on-reset transition
1	STOPPED	RUNNING	The DSPI is started (DSPI transitions to running) when all of the following conditions are true: <ul style="list-style-type: none"><li>• EOQF bit is clear</li><li>• Debug mode is unselected or the FRZ bit is clear</li><li>• HALT bit is clear</li></ul>
2	RUNNING	STOPPED	The DSPI stops (transitions from running to stopped) after the current frame for any one of the following conditions: <ul style="list-style-type: none"><li>• EOQF bit is set</li><li>• Debug mode is selected and the FRZ bit is set</li><li>• HALT bit is set</li></ul>

State transitions from running to stopped occur on the next frame boundary if a transfer is in progress or on the next system clock cycle if no transfers are in progress.

#### 40.4.2 Serial Peripheral Interface (SPI) Configuration

The SPI configuration transfers data serially using a shift register and a selection of programmable transfer attributes. The SPI frames can be from 4–16 bits long. The data transmitted can come from queues stored in RAM external to the DSPI. Host software or the eDMA controller can transfer the SPI data from the queues to a first-in first-out (FIFO) buffer. The received data is stored in entries in the receive FIFO (RX FIFO) buffer. Host software or the eDMA controller transfers the received data from the RX FIFO to memory external to the DSPI. The FIFO buffer operations are described in [Section 40.4.2.4, “TX FIFO Buffering Mechanism,”](#) and [Section 40.4.2.5, “RX FIFO Buffering Mechanism.”](#) The interrupt and DMA request conditions are described in [Section 40.4.6, “Interrupts/DMA Requests.”](#)

The SPI configuration supports two module-specific modes; master mode and slave mode. The FIFO operations are similar for both modes. In master mode, the DSPI initiates and controls the transfer according to 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 of the TX FIFO entry is ignored. For information on switching between master and slave modes see [Section 40.5.2, “Switching Master and Slave Mode.”](#)

#### 40.4.2.1 Master Mode

In master mode, the DSPI initiates the serial transfers by controlling the serial communications clock (DSPI\_SCK) and the peripheral chip select (DSPI\_PCS $n$ ) signals. The SPI command field in the executing TX FIFO entry determines which DSPI\_CTAR $n$  register sets the transfer attributes and which DSPI\_PCS $n$  signal to assert. The command field also contains various bits that help with queue management and transfer protocol. See [Section 40.3.6, “DSPI Push Transmit FIFO Register \(DSPIx\\_PUSHR\),”](#) 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 (DSPI\_SOUT) pin. In 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.

#### 40.4.2.2 Slave Mode

In slave mode, the DSPI responds to transfers initiated by an 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 an SPI master. The slave mode transfer attributes are set in the DSPI\_CTAR0 register.

#### 40.4.2.3 FIFO Disable Operation

The FIFO disable mechanisms allow SPI transfers without using the TX or RX FIFOs. 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, DSPI\_SR[TFFF, TFUF, and TXCTR] fields behave as if there is a one-entry FIFO, but the contents of DSPI\_TXFRs and TXNXTPTR are undefined. Likewise, when RX FIFO is disabled, DSPI\_SR[RFDF, RFOF, and RXCTR] fields behave as if there is a one-entry FIFO, but the contents of DSPI\_RXFRs and POPNXTPTR are undefined.

The TX and RX FIFOs should be disabled only if the application's operating mode requires FIFO to be disabled. A FIFO must be disabled before it is accessed. Failure to disable a FIFO prior to a first FIFO access is not supported and may result in incorrect results.

#### NOTE

When the FIFOs are disabled, the respective DSPI\_MCR[CLR\_TXF, CLR\_RXF] bits have no effect.

#### 40.4.2.4 TX FIFO Buffering Mechanism

The TX FIFO functions as a buffer of SPI data and SPI commands for transmission. The TX FIFO holds 16 entries, each consisting of a command field and a data field. SPI commands and data are added to the TX FIFO by writing to the DSPI push TX FIFO register (DSPI\_PUSHR). For more information on DSPI\_PUSHR, refer to [Section 40.3.6, “DSPI Push Transmit FIFO Register \(DSPIx\\_PUSHR\),”](#) 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 (TXCTR) in the DSPI status register (DSPI\_SR) indicates the number of valid entries in the TX FIFO. The TXCTR is updated every time the DSPI\_PUSHR is written or SPI data transfers into the shift register from the TX FIFO. For more information on DSPI\_SR, refer to [Section 40.3.4, “DSPI Status Register \(DSPIx\\_SR\).”](#)

The DSPI\_SR[TXNXTPTR] field indicates which TX FIFO entry is transmitted during the next transfer. The TXNXTPTR contains the positive offset from DSPI\_TXFR0 in number of 32-bit registers. For example, TXNXTPTR equal to two means DSPI\_TXFR2 contains the SPI data and command for the next transfer. The TXNXTPTR field increments every time SPI data transfers from TX FIFO to shift register.

#### 40.4.2.4.1 Filling the TX FIFO

Host software or the eDMA controller can add (push) entries to the TX FIFO by writing to the DSPI\_PUSHR register. When the TX FIFO is not full, the TX FIFO fill flag, DSPI\_SR[TFFF], is set. The TFFF bit is cleared when the TX FIFO is full and the eDMA controller indicates that a write to DSPI\_PUSHR is complete. Host software writing a 1 to the DSPI\_SR[TFFF] bit can also clear the TFFF bit. The TFFF can generate a DMA request or an interrupt request. See [Section 40.4.6.2, “Transmit FIFO Fill Interrupt or DMA Request \(TFFF\),”](#) for details.

The DSPI ignores attempts to push data to a full TX FIFO; in other words, the state of the TX FIFO is unchanged and no error condition is indicated.

#### 40.4.2.4.2 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, the DSPI\_SR[TCF] bit is set to indicate completion of a transfer. The TX FIFO is flushed by writing a 1 to the DSPI\_MCR[CLR\_TXF] bit.

If an external SPI bus master initiates a transfer with a DSPI slave while the slave’s DSPI TX FIFO is empty, the slave’s transmit FIFO underflow flag, DSPI\_SR[TFUF], is set. See [Section 40.4.6.4, “Transmit FIFO Underflow Interrupt Request \(TFUF\),”](#) for details.

#### 40.4.2.5 RX FIFO Buffering Mechanism

The RX FIFO functions as a buffer for data received on the DSPI\_SIN pin. The RX FIFO holds 16 received SPI data frames. 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 is removed (popped) from the RX FIFO by reading the DSPI\_POPR register. RX FIFO entries can only be removed from the RX FIFO by reading the DSPI\_POPR or by flushing the RX FIFO. For more information on the DSPI\_POPR, refer to [Section 40.3.7, “DSPI Pop Receive FIFO Register \(DSPIx\\_POPR\).”](#)

The RX FIFO counter field, DSPI\_SR[RXCTR], indicates the number of valid entries in the RX FIFO. The RXCTR is updated every time the DSPI\_POPR is read or SPI data is copied from the shift register to the RX FIFO.

The DSPI\_SR[POPNXTPTR] field points to the RX FIFO entry returned when the DSPI\_POPR is read. The POPNXTPTR contains the positive, 32-bit word offset from DSPI\_RXFR0. For example,

POPNXTPTR equal to two means that the DSPI\_RXFR2 contains the received SPI data that is returned when DSPI\_POPR is read. The POPNXTPTR field increments every time the DSPI\_POPR is read. POPNXTPTR rolls over every four frames on the MCU.

#### 40.4.2.5.1 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 an SPI frame is transferred to the RX FIFO, the RX FIFO counter increments by one.

If the RX FIFO and shift register are full and a transfer is initiated, the DSPI\_SR[RFOF] bit is asserted indicating an overflow condition. Depending on the state of the DSPI\_MCR[ROOE] bit, data from the transfer that generated the overflow is ignored or shifted in to the shift register. If the ROOE bit is set, incoming data is shifted in to the shift register. If the ROOE bit is cleared, the incoming data is ignored.

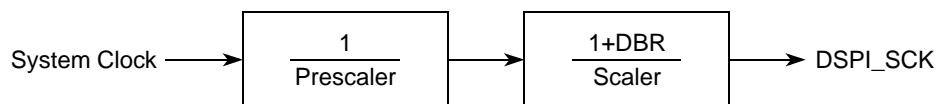
#### 40.4.2.5.2 Draining the RX FIFO

Host software or the eDMA can remove (pop) entries from the RX FIFO by reading the DSPI\_POPR. For more information on DSPI\_POPR, refer to [Section 40.3.7, “DSPI Pop Receive FIFO Register \(DSPIx\\_POPR\).”](#) 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 returned from reading an empty RX FIFO is undetermined.

When the RX FIFO is not empty, the RX FIFO drain flag, DSPI\_SR[RFDF], is set. The RFDF bit is cleared when the RX\_FIFO is empty and the eDMA controller indicates that a read from DSPI\_POPR is complete. Alternatively, the RFDF bit can be cleared by software writing a 1 to it.

### 40.4.3 DSPI Baud Rate and Clock Delay Generation

The DSPI\_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 of doubling the baud rate. [Figure 40-13](#) shows conceptually how the DSPI\_SCK signal is generated.



**Figure 40-13. Communications Clock Prescalers and Scalers**

#### 40.4.3.1 Baud Rate Generator

The baud rate is the frequency of the serial communication clock (DSPI\_SCK). The system clock is divided by a baud rate prescaler (defined by DSPI\_CTARn[PBR]) and baud rate scaler (defined by DSPI\_CTARn[BR]) to produce DSPI\_SCK with the possibility of doubling the baud rate. The DBR, PBR, and BR fields in the DSPI\_CTARn select the frequency of DSPI\_SCK using the following formula:

$$\text{SCK baud rate} = \frac{f_{\text{SYS}/2}}{\text{PBR Prescaler Value}} \times \frac{1 + \text{DBR}}{\text{BR Scaler Value}}$$
**Eqn. 40-1**

Table 40-14 shows an example of a computed baud rate.

**Table 40-14. Baud Rate Computation Example**

$f_{SYS/2}$	PBR	Prescaler Value	BR	Scaler Value	DBR Value	Baud Rate
100 MHz	00	2	0000	2	0	25 Mb/s
20 MHz	00	2	0000	2	1	10 Mb/s

#### 40.4.3.2 PCS to SCK Delay ( $t_{CSC}$ )

The PCS to SCK delay is the length of time from assertion of DSPI\_PCS signal to the first DSPI\_SCK edge. See Figure 40-14 for an illustration of the PCS to SCK delay. The DSPI\_CTARn[PCSSCK, CSSCK] fields select the PCS to SCK delay, and the relationship is expressed by the following:

$$t_{CSC} = \frac{1}{f_{SYS/2}} \times PCSSCK \times CSSCK \quad \text{Eqn. 40-2}$$

Table 40-15 shows an example of the computed PCS to SCK delay.

**Table 40-15. PCS to SCK Delay Computation Example**

PCSSCK	Prescaler Value	CSSCK	Scaler Value	$f_{SYS/2}$	PCS to SCK Delay
01	3	0100	32	100 MHz	0.96 $\mu$ s

#### 40.4.3.3 After SCK Delay ( $t_{ASC}$ )

The after SCK delay is the length of time between the last edge of DSPI\_SCK and negation of DSPI\_PCS. See Figure 40-14 and Figure 40-15 for illustrations of the after SCK delay. The DSPI\_CTARn[PASC, ASC] fields select the after SCK delay. The relationship between these variables is given in the following:

$$t_{ASC} = \frac{1}{f_{SYS/2}} \times PASC \times ASC \quad \text{Eqn. 40-3}$$

Table 40-16 shows an example of the computed after SCK delay.

**Table 40-16. After SCK Delay Computation Example**

PASC	Prescaler Value	ASC	Scaler Value	$f_{SYS/2}$	After SCK Delay
01	3	0100	32	100 MHz	0.96 us

#### 40.4.3.4 Delay after Transfer ( $t_{DT}$ )

The delay after transfer is the length of time between negation of DSPI\_PCS signal for a frame and the assertion of DSPI\_PCS signal for the next frame. See Figure 40-14 for an illustration of the delay after transfer. DSPI\_CTARn[PDT, DT] fields select the delay after transfer by the formula:

$$t_{DT} = \frac{1}{f_{SYS/2}} \times PDT \times DT \quad \text{Eqn. 40-4}$$

[Table 40-17](#) shows an example of the computed delay after transfer.

**Table 40-17. Delay after Transfer Computation Example**

PDT	Prescaler Value	DT	Scaler Value	f <sub>SYS/2</sub>	Delay after Transfer
01	3	1110	32768	100 MHz	0.98 ms

#### 40.4.4 Transfer Formats

The serial communications clock (DSPI\_SCK) signal and the DSPI\_PCS $n$  signals control the SPI serial communication. The DSPI\_SCK signal provided by the master device synchronizes shifting and sampling of the data by the DSPI\_SIN and DSPI\_SOUT pins. The DSPI\_PCS $n$  signals serve as enable signals for the slave devices.

When the DSPI is the bus master, the DSPI\_CTAR $n$ [CPOL, CPHA] bits select the polarity and phase of the DSPI\_SCK signal. The polarity bit selects the idle state of the DSPI\_SCK. The clock phase bit selects if the data on DSPI\_SOUT is valid before or on the first DSPI\_SCK edge.

When the DSPI is the bus slave, the DSPI\_CTAR0[CPOL, CPHA] bits select the polarity and phase of the serial clock. Even though the bus slave does not control the DSPI\_SCK signal, clock polarity, clock phase, and number of bits to transfer must be identical for the master device and the slave device 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 DSPI\_MCR[MTFE] bit selects between classic SPI format and modified transfer format. The classic SPI formats are described in [Section 40.4.4.1, “Classic SPI Transfer Format \(CPHA = 0\)”](#) and [Section 40.4.4.2, “Classic SPI Transfer Format \(CPHA = 1\).”](#) The modified transfer formats are described in [Section 40.4.4.3, “Modified SPI Transfer Format \(MTFE = 1, CPHA = 0\)”](#) and [Section 40.4.4.4, “Modified SPI Transfer Format \(MTFE = 1, CPHA = 1\).”](#)

##### 40.4.4.1 Classic SPI Transfer Format (CPHA = 0)

The transfer format shown in [Figure 40-14](#) communicates 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 DSPI\_SIN pins on the odd-numbered DSPI\_SCK edges and change the data on their DSPI\_SOUT pins on the even-numbered DSPI\_SCK edges.

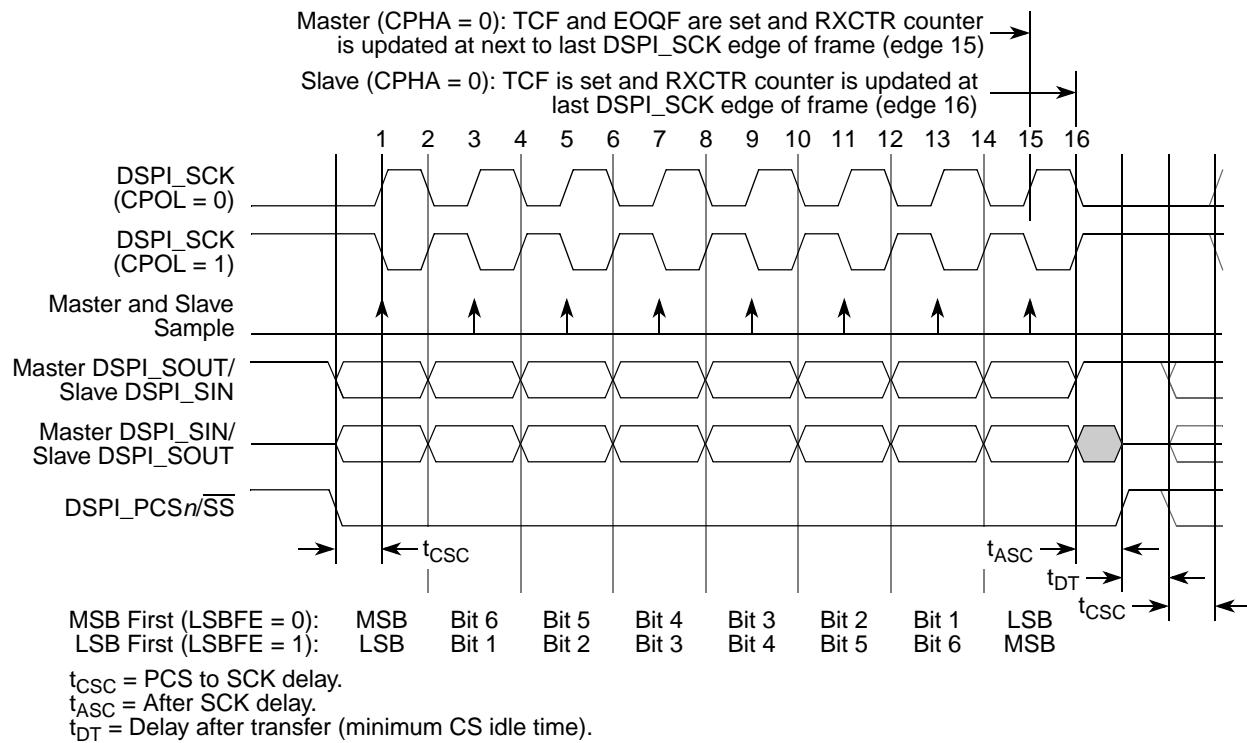


Figure 40-14. DSPI Transfer Timing Diagram (MTFE = 0, CPHA = 0, FMSZ = 8)

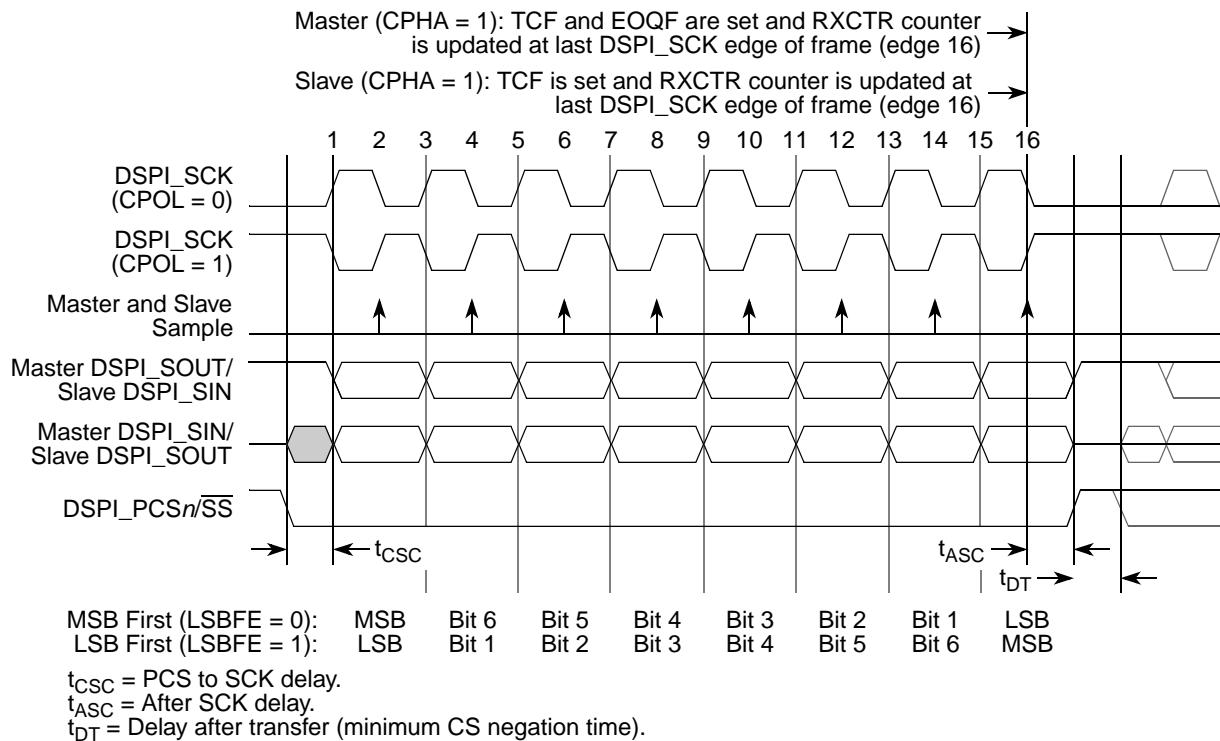
The master initiates the transfer by placing its first data bit on the DSPI\_SOUT pin and asserting the appropriate peripheral chip select signals to the slave device. The slave responds by placing its first data bit on its DSPI\_SOUT pin. After the  $t_{CSC}$  delay elapses, the master outputs the first edge of DSPI\_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 DSPI\_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 DSPI\_SIN pins on the odd-numbered clock edges and change the data on their DSPI\_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 DSPI\_PCSn signals. A delay of  $t_{DT}$  is inserted before a new frame transfer can be initiated by the master.

If DSPI\_CTARn[CPHA] is cleared:

- At the next to last serial clock edge of the frame (edge 15 of Figure 40-14)
  - Master's TCF and EOQF are set and RXCTR counter is updated
- At the last serial clock edge of the frame (edge 16 of Figure 40-14)
  - Slave's TCF is set and RXCTR counter is updated

#### 40.4.4.2 Classic SPI Transfer Format (CPHA = 1)

The transfer format shown in Figure 40-15 communicates with peripheral SPI slave devices that require the first DSPI\_SCK edge before the first data bit becomes available on the slave DSPI\_SOUT pin. In this format, the master and slave devices change the data on their DSPI\_SOUT pins on the odd-numbered DSPI\_SCK edges and sample the data on their DSPI\_SIN pins on the even-numbered DSPI\_SCK edges.



**Figure 40-15. DSPI Transfer Timing Diagram (MTFE = 0, CPHA = 1, FMSZ = 8)**

The master initiates the transfer by asserting the DSPI\_PCS $n$  signal to the slave. After the  $t_{CSC}$  delay has elapsed, the master generates the first DSPI\_SCK edge and places valid data on the master DSPI\_SOUT pin. The slave responds to the first DSPI\_SCK edge by placing its first data bit on its slave DSPI\_SOUT pin.

At the second edge of the DSPI\_SCK, the master and slave sample their DSPI\_SIN pins. For the rest of the frame, the master and the slave change the data on their DSPI\_SOUT pins on the odd-numbered clock edges and sample their DSPI\_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 DSPI\_PCS $n$  signal. A delay of  $t_{DT}$  is inserted before a new frame transfer can be initiated by the master.

If DSPI\_CTAR $n$ [CPHA] is set:

- At the last serial clock edge (edge 16 of Figure 40-15)
  - Master's EOQF and TCF are set
  - Slave's TCF is set
  - Master's and slave's RXCTR counters are updated

#### 40.4.4.3 Modified SPI Transfer Format (MTFE = 1, CPHA = 0)

In this modified transfer format, the master and the slave sample later in the DSPI\_SCK period than in classic SPI mode to allow for delays in device pads and board traces. These delays become a more significant fraction of the DSPI\_SCK period as the DSPI\_SCK period decreases with increasing baud rates.

**NOTE**

For correct operation of the modified transfer format, thoroughly analyze the SPI link timing budget.

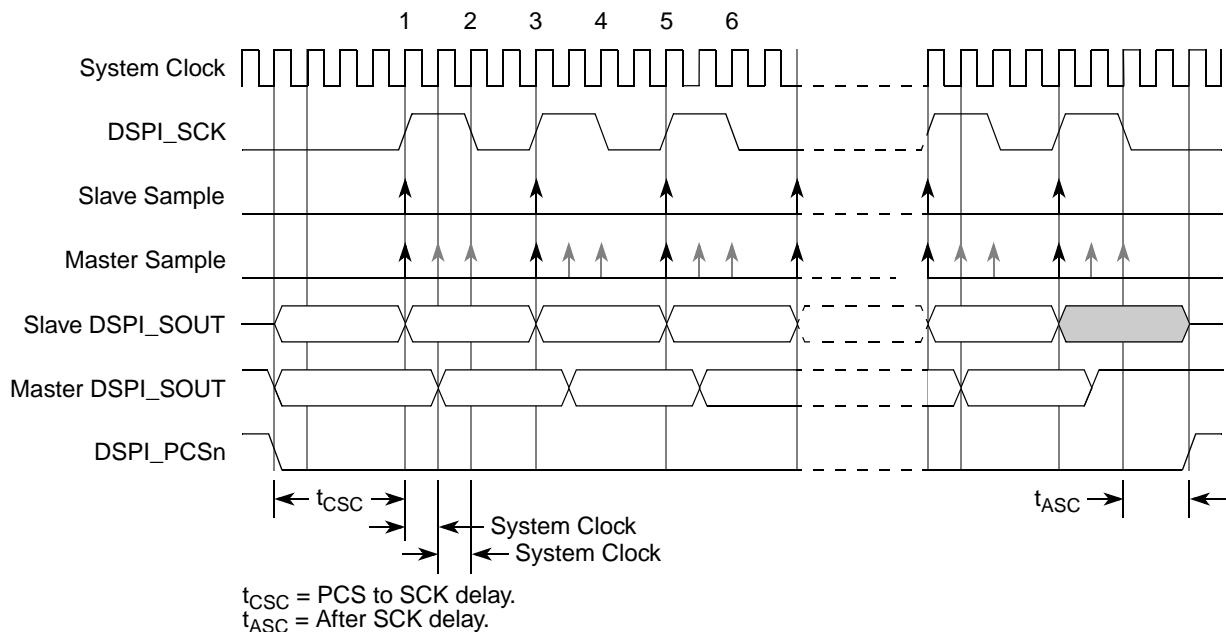
The master and the slave place data on the DSPI\_SOUT pins at the assertion of the DSPI\_PCSn signal. After the PCS to SCK delay has elapsed, the first DSPI\_SCK edge is generated. The slave samples the master DSPI\_SOUT signal on every odd numbered DSPI\_SCK edge. The slave also places new data on the slave DSPI\_SOUT on every odd numbered clock edge.

The master places its second data bit on the DSPI\_SOUT line one system clock after odd numbered SDSPI\_CK edge. Writing to the DSPI\_MCR[SMPL\_PT] field selects the point where the master samples the slave DSPI\_SOUT. [Table 40-18](#) lists the number of system clock cycles between the active edge of DSPI\_SCK and the master sample point for different values of the SMPL\_PT bit field. The master sample point can be delayed by one or two system clock cycles.

**Table 40-18. Delayed Master Sample Point**

SMPL_PT	Number of System Clock Cycles between Odd-numbered Edge of SCK and Sampling of SIN
00	0
01	1
10	2
11	Reserved

[Figure 40-16](#) shows the modified transfer format for CPHA is cleared. Only the condition where CPOL is cleared is illustrated. The delayed master sample points are indicated with a lighter shaded arrow.



**Figure 40-16. DSPI Modified Transfer Format (MTFE = 1, CPHA = 0, Fsck = Fsys/4)**

#### 40.4.4.4 Modified SPI Transfer Format (MTFE = 1, CPHA = 1)

Figure 40-17 shows the modified transfer format for CPHA is set. Only the condition where CPOL is cleared is described. At the start of a transfer, the DSPI asserts the DSPI\_PCS $n$  signal to the slave device. After the PCS to SCK delay has elapsed the master and the slave put data on their DSPI\_SOUT pins at the first edge of DSPI\_SCK. The slave samples the master DSPI\_SOUT signal on the even numbered edges of DSPI\_SCK. The master samples the slave DSPI\_SOUT signal on the odd numbered DSPI\_SCK edges starting with the third DSPI\_SCK edge. The slave samples the last bit on the last edge of the DSPI\_SCK. The master samples the last slave DSPI\_SOUT bit one half DSPI\_SCK cycle after the last edge of DSPI\_SCK. No clock edge is visible on the master DSPI\_SCK pin during the sampling of the last bit. The SCK to PCS delay must be greater or equal to half of the DSPI\_SCK period.

##### NOTE

For correct operation of the modified transfer format, the user must thoroughly analyze the SPI link timing budget.

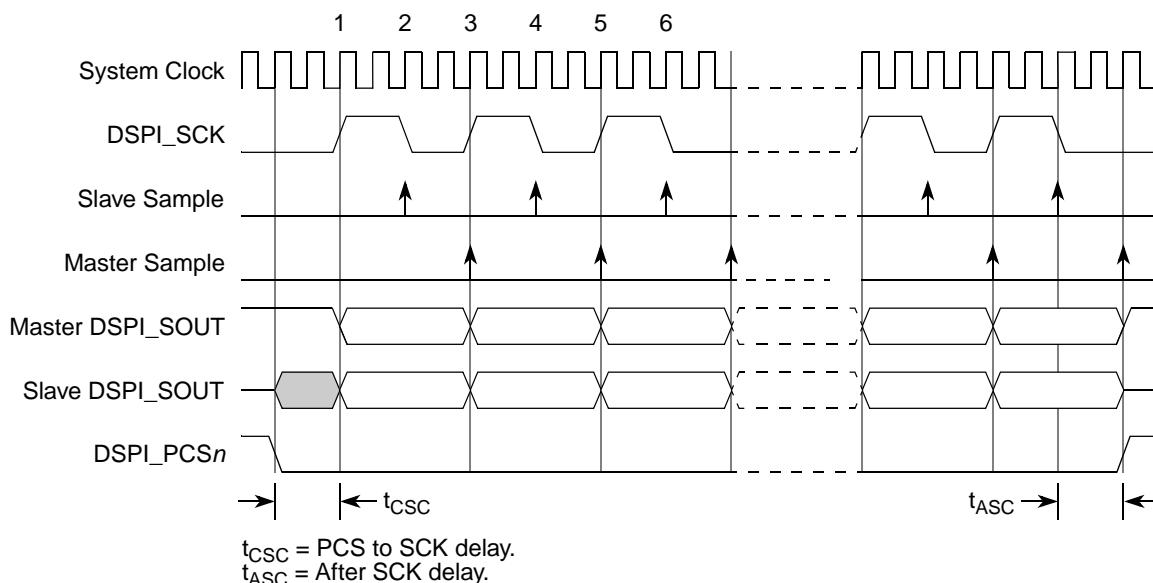
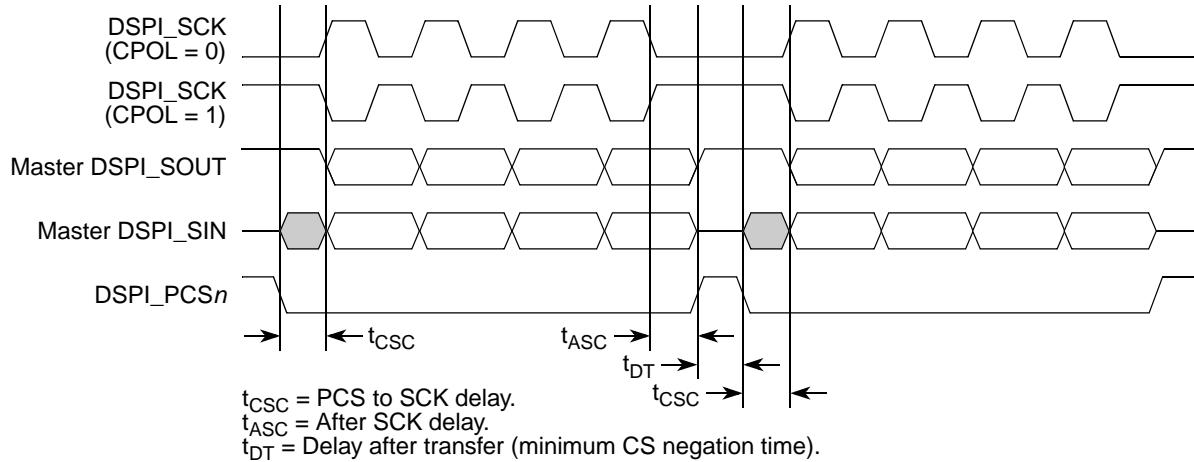


Figure 40-17. DSPI Modified Transfer Format (MTFE = 1, CPHA = 1,  $F_{SCK} = F_{sys}/4$ )

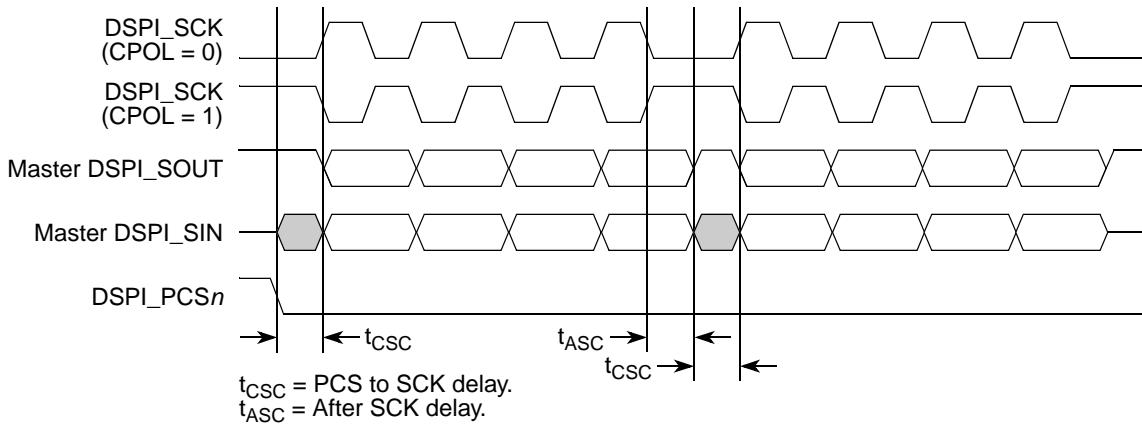
#### 40.4.4.5 Continuous Selection Format

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 DSPI\_PUSHR[CONT] bit.

When CONT is cleared, 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 DSPI\_MCR[PCSISS] field. Figure 40-18 shows the timing diagram for two four-bit transfers with CPHA set and CONT cleared.

**Figure 40-18. Example of Non-Continuous Format (CPHA=1, CONT=0)**

When **CONT** is set and the **DSPI\_PCSn** signal for the next transfer the same as for the current transfer, **DSPI\_PCSn** signal remains asserted for the duration of the two transfers. The delay between transfers ( $t_{DT}$ ) is not inserted between the transfers. [Figure 40-19](#) shows the timing diagram for two four-bit transfers with CPHA and CONT set.

**Figure 40-19. Example of Continuous Transfer (CPHA = 1, CONT = 1)**

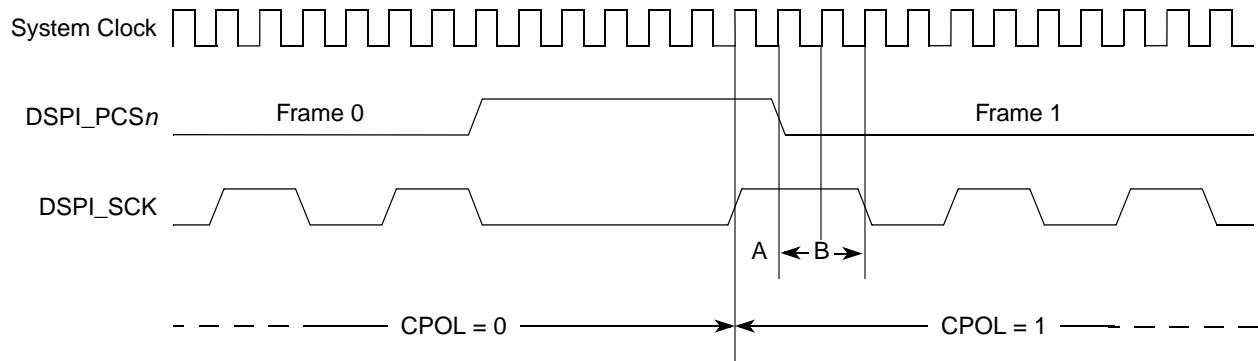
In [Figure 40-19](#), the period length at the start of the next transfer is the sum of  $t_{ASC}$  and  $t_{CSC}$ . It does not include a half-clock period. The default settings for these provide a total of four system clocks. In many situations,  $t_{ASC}$  and  $t_{CSC}$  must be increased if a full half-clock period is required.

Switching **DSPI\_CTARn** registers between frames while using continuous selection can cause errors in the transfer. The **DSPI\_PCSn** signal must be negated before **DSPI\_CTAR** is switched.

When **CONT** is set and the **DSPI\_PCSn** signals for the next transfer are different from the present transfer, the **DSPI\_PCSn** signals behave as if the **CONT** bit was cleared.

#### 40.4.4.6 Clock Polarity Switching between DSPI Transfers

If it is desired to switch polarity between non-continuous DSPI frames, the edge generated by the change in the idle state of the clock occurs one system clock before the assertion of the chip select for the next frame. In [Figure 40-20](#), time A shows the one clock interval. Time B is user programmable from a minimum of 2 system clocks. See [Section 40.3.3, “DSPI Clock and Transfer Attributes Registers 0–7 \(DSPIx\\_CTARn\).”](#)



**Figure 40-20. Polarity Switching between Frames**

#### 40.4.5 Continuous Serial Communications Clock

The DSPI provides the option of generating a continuous DSPI\_SCK signal for slave peripherals that require a continuous clock. Continuous SCK is enabled by setting the DSPI\_MCR[CONT\_SCKE] bit.

Continuous SCK is only supported if CPHA is set. Clearing CPHA is ignored if the CONT\_SCKE bit 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:

- DSPI\_CTAR0 is used initially. At the start of each SPI frame transfer, the DSPI\_CTAR $n$  specified by the CTAS field for the frame is used.
- The currently selected DSPI\_CTAR $n$  remains in use until the start of a frame with a different DSPI\_CTAR $n$  specified, or the continuous SCK mode is terminated.

It is recommended that the baud rate is the same for all transfers made 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 module disable mode.

Enabling continuous SCK disables the PCS to SCK delay and the after SCK delay. The delay after transfer is fixed at one DSPI\_SCK cycle. [Figure 40-21](#) shows timing diagram for continuous SCK format with continuous selection disabled.

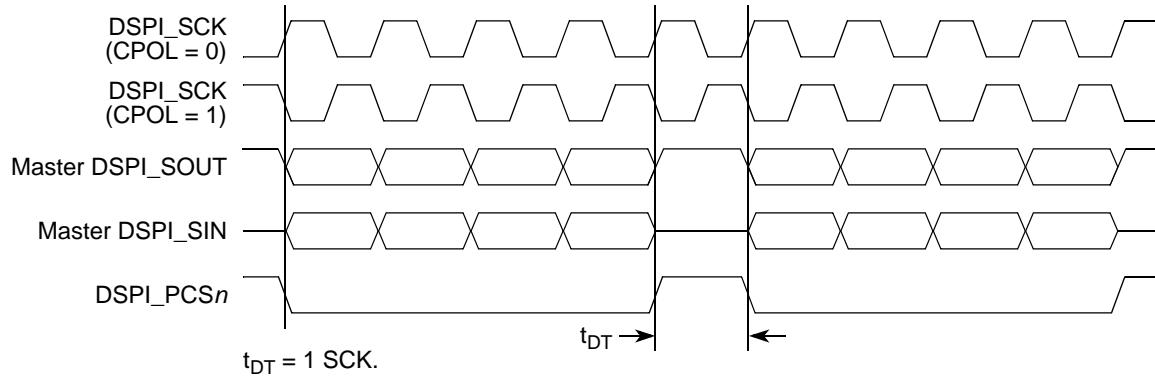


Figure 40-21. Continuous SCK Timing Diagram (CONT= 0)

If the CONT bit in the TX FIFO entry is set, DSPI\_PCSn remains asserted between the transfers when the DSPI\_PCSn signal for the next transfer is the same as for the current transfer. Figure 40-22 shows timing diagram for continuous SCK format with continuous selection enabled.

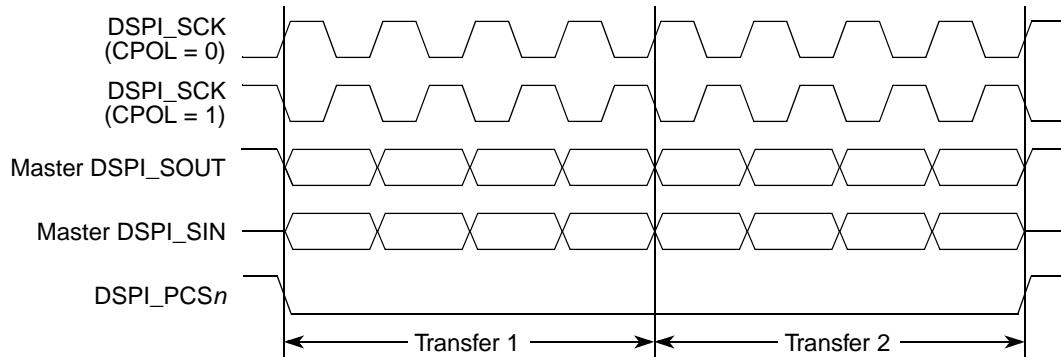


Figure 40-22. Continuous SCK Timing Diagram (CONT=1)

#### 40.4.6 Interrupts/DMA Requests

The DSPI has six conditions that can only generate interrupt requests and two conditions that can generate an interrupt or DMA request. Table 40-19 lists these conditions. On this device, the interrupt controller only supports the general DSPI condition for each DSPI module. The DSPIx\_SR register must be read to determine the cause of the interrupt.

Table 40-19. Interrupt and DMA Request Conditions

Condition	Flag	Interrupt	DMA
End of transfer queue has been reached (EOQ)	EOQF	X	—
TX FIFO is not full	TFFF	X	X
Current frame transfer is complete	TCF	X	—
TX FIFO underflow has occurred	TFUF	X	—
RX FIFO is not empty	RFDF	X	X
RX FIFO overflow has occurred	RFOF	X	—

**Table 40-19. Interrupt and DMA Request Conditions (continued)**

Condition	Flag	Interrupt	DMA
A FIFO overrun has occurred <sup>1</sup>	TFUF OR RFOF	X	—
General DSPI condition <sup>2</sup>	Any of the above	X	—

<sup>1</sup> The FIFO overrun condition is created by OR-ing the TFUF and RFOF flags together.

<sup>2</sup> OR'd condition of any of the six flags.

Each condition has a flag bit and a request enable bit. The flag bits are described in [Section 40.3.4, “DSPI Status Register \(DSPIx\\_SR\),”](#) and the request enable bits are described in [Section 40.3.5, “DSPI DMA/Interrupt Request Select and Enable Register \(DSPIx\\_RSER\).”](#) The TX FIFO fill flag (TFFF) and RX FIFO drain flag (RFDF) generate interrupt requests or DMA requests depending on the DSPI\_RSER[TFFF\_DIRS, RFDF\_DIRS] bits.

#### 40.4.6.1 End of Queue Interrupt Request (EOQF)

The end of queue request indicates end of a transmit queue is reached. The end of queue request is generated when the EOQ bit in the executing SPI command is asserted and the DSPI\_RSER[EOQF\_RE] bit is set. See the EOQ bit description in [Section 40.3.4, “DSPI Status Register \(DSPIx\\_SR\).”](#) Refer to [Figure 40-14](#) and [Figure 40-15](#) that illustrate when EOQF is set.

#### 40.4.6.2 Transmit FIFO Fill Interrupt or DMA Request (TFFF)

The transmit FIFO fill request indicates 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 DSPI\_RSER[TFFF\_RE] bit is set. The DSPI\_RSER[TFFF\_DIRS] bit selects whether a DMA request or an interrupt request is generated.

#### 40.4.6.3 Transfer Complete Interrupt Request (TCF)

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 DSPI\_RSER[TCF\_RE] bit is set. See the TCF bit description in [Section 40.3.4, “DSPI Status Register \(DSPIx\\_SR\).”](#) Refer to [Figure 40-14](#) and [Figure 40-15](#) that illustrate when TCF is set.

#### 40.4.6.4 Transmit FIFO Underflow Interrupt Request (TFUF)

The transmit FIFO underflow request 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. The TFUF bit is set when the TX FIFO of a DSPI operating in slave mode is empty, and a transfer is initiated from an external SPI master. If the TFUF bit is set while the DSPI\_RSER[TFUF\_RE] bit is set, an interrupt request is generated.

#### 40.4.6.5 Receive FIFO Drain Interrupt or DMA Request (RFDF)

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 DSPI\_RSER[RFDF\_RE] bit is set. The DSPI\_RSER[RFDF\_DIRS] bit selects whether a DMA request or an interrupt request is generated.

#### 40.4.6.6 Receive FIFO Overflow Interrupt Request (RFOF)

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 DSPI\_RSER[RFOF\_RE] bit must be set for the interrupt request to be generated.

Depending on the state of the DSPI\_MCR[ROOE] bit, data from the transfer that generated overflow is 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, incoming data is ignored.

#### 40.4.6.7 FIFO Overrun Request (TFUF) or (RFOF)

The FIFO overrun request indicates at least one of the FIFOs in the DSPI has exceeded its capacity. The FIFO overrun request is generated by logically OR'ing the RX FIFO overflow and TX FIFO underflow signals.

### 40.4.7 Power Saving Features

The DSPI supports two power-saving strategies:

- Module disable mode—clock gating of non-memory mapped logic
- Clock gating of slave interface signals and clock to memory-mapped logic

#### 40.4.7.1 Module Disable Mode

Module disable mode is a mode the DSPI can enter to save power. Host software can initiate the module disable mode by setting DSPI\_MCR[MDIS]. The MDIS bit is set at reset.

In module disable mode, the DSPI is in a dormant state, but the memory-mapped registers remain accessible. Certain read or write operations have a different affect 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 does not have any affect in module disable mode. Changes to the DSPI\_MCR[DIS\_TXF, DIS\_RXF] fields do not have any affect in module disable mode. In 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 does not have any affect. Interrupt and DMA request signals cannot be cleared while in module disable mode.

#### 40.4.7.2 Slave Interface Signal Gating

The DSPI's module enable signal gates slave interface signals such as address, byte enable, read/write and data. This prevents toggling slave interface signals from consuming power unless the DSPI is accessed.

## 40.5 Initialization/Application Information

### 40.5.1 How to Change Queues

DSPI queues are not part of the DSPI module, but the DSPI includes features in support of queue management. This section presents an example of how to change queues for the DSPI.

1. The last command word from a queue is executed. 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 EOQF flag disables serial transmission and serial reception of data, putting the DSPI in the stopped state. The TXRXS bit is cleared to indicate the stopped state.
4. The eDMA continues 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 eDMA controller.
6. Ensure all received data in RX FIFO has been transferred to memory receive queue by reading the DSPI\_SR[RXCNT] bit or by checking the DSPI\_SR[RFDF] bit after each read operation of the DSPI\_POPR register.
7. Modify DMA descriptor of TX and RX channels for new queues.
8. Flush TX FIFO by writing a 1 to the DSPI\_MCR[CLR\_TXF] bit; Flush RX FIFO by writing a 1 to the DSPI\_MCR[CLR\_RXF] bit.
9. Clear transfer count by setting the CTCNT bit in the command word of the first entry in the new queue or via CPU writing directly to the DSPI\_TCR[SPI\_TCNT] field.
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.

### 40.5.2 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].

### 40.5.3 Baud Rate Settings

**Table 40-20** shows the baud rate generated based on the combination of the baud rate prescaler PBR and the baud rate scaler BR in the DSPI\_CTARn registers. The values calculated assume a 100 MHz system frequency.

**Table 40-20. Baud Rate Values**

	Baud Rate Divider Prescaler Values (DSPI_CTARn[PBR])				
	2	3	5	7	
Baud Rate Scaler Values (DSPI_CTARn[BRI])	<b>2</b>	25.0MHz	16.7MHz	10.0MHz	7.14MHz
	<b>4</b>	12.5MHz	8.33MHz	5.00MHz	3.57MHz
	<b>6</b>	8.33MHz	5.56MHz	3.33MHz	2.38MHz
	<b>8</b>	6.25MHz	4.17MHz	2.50MHz	1.79MHz
	<b>16</b>	3.12MHz	2.08MHz	1.25MHz	893kHz
	<b>32</b>	1.56MHz	1.04MHz	625kHz	446kHz
	<b>64</b>	781kHz	521kHz	312kHz	223kHz
	<b>128</b>	391kHz	260kHz	156kHz	112kHz
	<b>256</b>	195kHz	130kHz	78.1kHz	55.8kHz
	<b>512</b>	97.7kHz	65.1kHz	39.1kHz	27.9kHz
	<b>1024</b>	48.8kHz	32.6kHz	19.5kHz	14.0kHz
	<b>2048</b>	24.4kHz	16.3kHz	9.77kHz	6.98kHz
	<b>4096</b>	12.2kHz	8.14kHz	4.88kHz	3.49kHz
	<b>8192</b>	6.10kHz	4.07kHz	2.44kHz	1.74kHz
	<b>16384</b>	3.05kHz	2.04kHz	1.22kHz	872Hz
	<b>32768</b>	1.53kHz	1.02kHz	610Hz	436Hz

#### 40.5.4 Delay Settings

Table 40-21 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\_CTARn registers. The values calculated assume a 100 MHz system frequency.

**Table 40-21. Delay Values**

	Delay Prescaler Values (DSPI_CTARn[PBR])				
	1	3	5	7	
Delay Scaler Values (DSPI_CTARn[DT])	<b>2</b>	20.0 ns	60.0 ns	100.0 ns	140.0 ns
	<b>4</b>	40.0 ns	120.0 ns	200.0 ns	280.0 ns
	<b>8</b>	80.0 ns	240.0 ns	400.0 ns	560.0 ns
	<b>16</b>	160.0 ns	480.0 ns	800.0 ns	1.1 µs
	<b>32</b>	320.0 ns	960.0 ns	1.6 µs	2.2 µs
	<b>64</b>	640.0 ns	1.9 µs	3.2 µs	4.5 µs
	<b>128</b>	1.3 µs	3.8 µs	6.4 µs	9.0 µs
	<b>256</b>	2.6 µs	7.7 µs	12.8 µs	17.9 µs
	<b>512</b>	5.1 µs	15.4 µs	25.6 µs	35.8 µs
	<b>1024</b>	10.2 µs	30.7 µs	51.2 µs	71.7 µs
	<b>2048</b>	20.5 µs	61.4 µs	102.4 µs	143.4 µs
	<b>4096</b>	41.0 µs	122.9 µs	204.8 µs	286.7 µs
	<b>8192</b>	81.9 µs	245.8 µs	409.6 µs	573.4 µs
	<b>16384</b>	163.8 µs	491.5 µs	819.2 µs	1.1 ms
	<b>32768</b>	327.7 µs	983.0 µs	1.6 ms	2.3 ms
	<b>65536</b>	655.4 µs	2.0 ms	3.3 ms	4.6 ms

## 40.5.5 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 40-23 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 40.4.2.4, “TX FIFO Buffering Mechanism,”](#) and [Section 40.4.2.5, “RX FIFO Buffering Mechanism,”](#) for details on the FIFO operation.

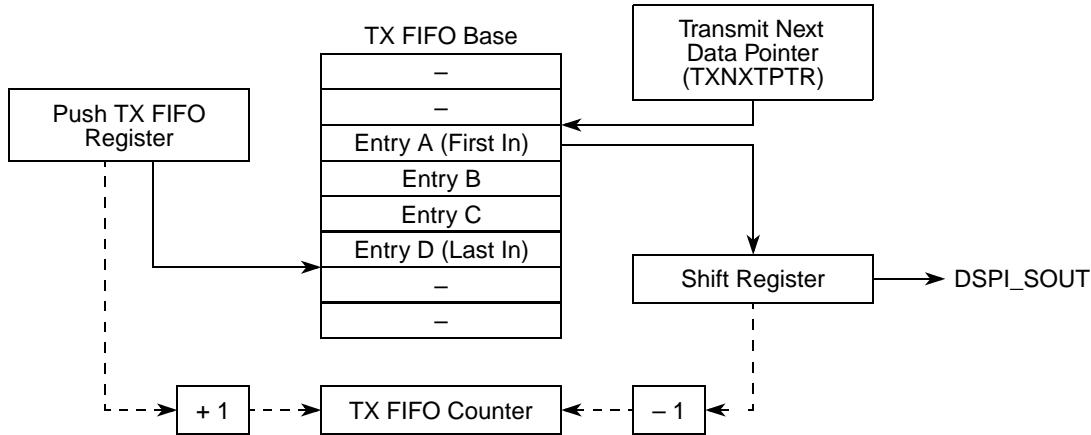


Figure 40-23. TX FIFO Pointers and Counter

#### 40.5.5.1 Address Calculation for the First-in and Last-in Entries in the TX FIFO

The memory address of the first-in entry in the TX FIFO is computed by the following equation:

$$\text{First-in entry address} = \text{TXFIFO base} + 4 \times (\text{TXNXTPTR})$$

The memory address of the last-in entry in the TX FIFO is computed by the following equation:

$$\text{Last-in entry address} = \text{TX FIFO base} + 4 \times [(\text{TXCTR} + \text{TXNXTPTR} - 1) \bmod \text{TX FIFO depth}]$$

where:

TX FIFO base: base address of TX FIFO

TXCTR: TX FIFO counter

TXNXTPTR: transmit next pointer

TX FIFO depth: 16

#### 40.5.5.2 Address Calculation for the First-in and Last-in Entries in the RX FIFO

The memory address of the first-in entry in the RX FIFO is computed by the following equation:

$$\text{First-in entry address} = \text{RX FIFO base} + 4 \times (\text{POPNXTPTR})$$

The memory address of the last-in entry in the RX FIFO is computed by the following equation:

$$\text{Last-in entry address} = \text{RX FIFO base} + 4 \times [(\text{RXCTR} + \text{POPNXTPTR} - 1) \bmod \text{RX FIFO depth}]$$

RX FIFO base: base address of RX FIFO

RXCTR: RX FIFO counter

POPNXTPTR: pop next pointer

RX FIFO depth: 16

# Chapter 41

## UART Modules

### 41.1 Introduction

This chapter describes the use of the ten universal asynchronous receiver/transmitters (UARTs) and includes programming examples.

#### NOTE

The designation  $n$  appears throughout this section to refer to registers or signals associated with one of the ten identical UART modules: UART0, UART1,... UART9.

#### 41.1.1 Overview

The internal bus clock can clock each of the ten independent UARTs, eliminating the need for an external UART clock. As [Figure 41-1](#) shows, each UART module interfaces directly to the CPU and consists of:

- Serial communication channel
- Programmable clock generation
- Interrupt control logic and DMA request logic
- Internal channel control logic

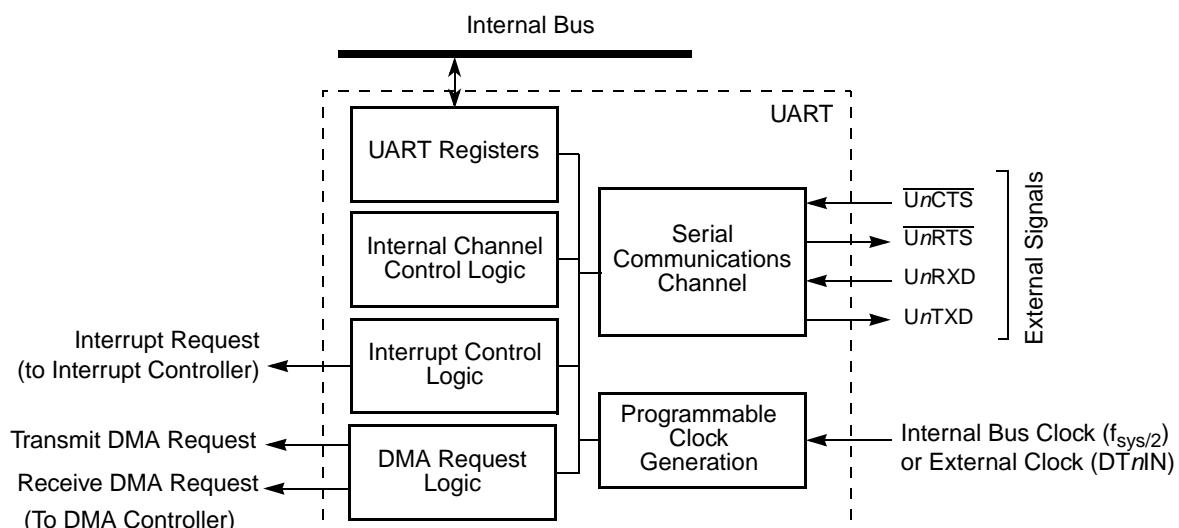


Figure 41-1. UART Block Diagram

**NOTE**

The DT<sub>n</sub>IN pin can clock UART<sub>n</sub>. However, if the timers are operating and the UART uses DT<sub>n</sub>IN as a clock source, input capture mode is not available for that timer.

Here are the available connections between the ten UARTs and four DMA timers:

- DT0IN → UART0, 4, and 8
- DT1IN → UART1, 5, and 9
- DT2IN → UART2, 6
- DT3IN → UART3, 7

The serial communication channel provides a full-duplex asynchronous/synchronous receiver and transmitter deriving an operating frequency from the internal bus clock or an external clock using the timer pin. The transmitter converts parallel data from the CPU to a serial bit stream, inserting appropriate start, stop, and parity bits. It outputs the resulting stream on the transmitter serial data output (UnTXD). See [Section 41.4.2.1, “Transmitter.”](#)

The receiver converts serial data from the receiver serial data input (UnRXD) to parallel format, checks for a start, stop, and parity bits, or break conditions, and transfers the assembled character onto the bus during read operations. The receiver may be polled, interrupt driven, or use DMA requests for servicing. See [Section 41.4.2.2, “Receiver.”](#)

**NOTE**

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to [Chapter 15, “Pin-Multiplexing and Control”](#)) prior to configuring the UART module.

### 41.1.2 Features

The device contains ten independent UART modules with:

- Each clocked by external clock or internal bus clock (eliminates need for an external UART clock)
- Full-duplex asynchronous/synchronous receiver/transmitter
- Quadruple-buffered receiver
- Double-buffered transmitter
- Independently programmable receiver and transmitter clock sources
- Programmable data format:
  - 5–8 data bits plus parity
  - Odd, even, no parity, or force parity
  - One, one-and-a-half, or two stop bits
- Each serial channel programmable to normal (full-duplex), automatic echo, local loopback, or remote loopback mode
- Single-wire mode with polarity control

- Automatic wake-up mode for multidrop applications
- Four maskable interrupt conditions
- All ten UARTs have DMA request capability
- Parity, framing, and overrun error detection
- False-start bit detection
- Line-break detection and generation
- Detection of breaks originating in the middle of a character
- Start/end break interrupt/status

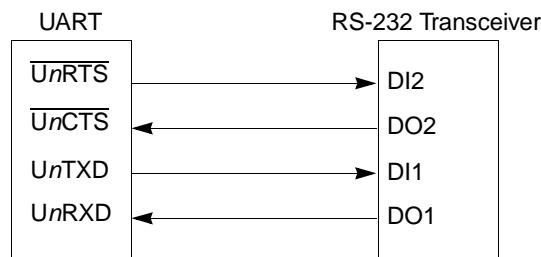
## 41.2 External Signal Description

[Table 41-1](#) briefly describes the UART module signals.

**Table 41-1. UART Module External Signals**

Signal	Description
UnTXD	Transmitter Serial Data Output. UnTXD is held high (mark condition) when the transmitter is disabled, idle, or operating in the local loopback mode. Data is shifted out on UnTXD on the falling edge of the clock source, with the least significant bit (lsb) sent first.
UnRXD	Receiver Serial Data Input. Data received on UnRXD is sampled on the rising edge of the clock source, with the lsb received first.
UnCTS	Clear-to- Send. This input can generate an interrupt on a change of state.
UnRTS	Request-to-Send. This output can be programmed to be negated or asserted automatically by the receiver or the transmitter. When connected to a transmitter's UnCTS, UnRTS can control serial data flow.

[Figure 41-2](#) shows a signal configuration for a UART/RS-232 interface.



**Figure 41-2. UART/RS-232 Interface**

## 41.3 Memory Map/Register Definition

This section contains a detailed description of each register and its specific function. Flowcharts in [Section 41.5, “Initialization/Application Information,”](#) describe basic UART module programming. Writing control bytes into the appropriate registers controls the operation of the UART module.

### NOTE

UART registers are accessible only as bytes.

**NOTE**

Interrupt can mean an interrupt request asserted to the CPU or a DMA request.

This device contains ten UART modules. Their base addresses are listed below:

**Table 41-2. UART Module Base Addresses**

Base Address	Module	Base Address	Module
0xFC06_0000	UART 0	0xEC06_4000	UART 5
0xFC06_4000	UART 1	0xEC06_8000	UART 6
0xFC06_8000	UART 2	0xEC06_C000	UART 7
0xFC06_C000	UART 3	0xEC07_0000	UART 8
0xEC06_0000	UART 4	0xEC07_4000	UART 9

**Table 41-3. UART Module Memory Map**

UART Base Offset	Register	Width (bit)	Access	Reset Value	Section/Page
0x00	UART Mode Registers <sup>1</sup> (UMR1 $n$ ), (UMR2 $n$ )	8	R/W	0x00	<a href="#">41.3.1/41-5</a> <a href="#">41.3.2/41-6</a>
0x04	UART Status Register (USR $n$ )	8	R	0x00	<a href="#">41.3.3/41-8</a>
	UART Clock Select Register <sup>1</sup> (UCSR $n$ )	8	W	See Section	<a href="#">41.3.4/41-9</a>
0x08	UART Command Registers (UCR $n$ )	8	W	0x00	<a href="#">41.3.5/41-10</a>
0x0C	UART Receive Buffers (URB $n$ )	8	R	0xFF	<a href="#">41.3.6/41-12</a>
	UART Transmit Buffers (UTB $n$ )	8	W	0x00	<a href="#">41.3.7/41-13</a>
0x10	UART Input Port Change Register (UIPCR $n$ )	8	R	See Section	<a href="#">41.3.8/41-14</a>
	UART Auxiliary Control Register (UACR $n$ )	8	W	0x00	<a href="#">41.3.9/41-14</a>
0x14	UART Interrupt Status Register (UISR $n$ )	8	R	0x00	<a href="#">41.3.10/41-15</a>
	UART Interrupt Mask Register (UIMR $n$ )	8	W	0x00	
0x18	UART Baud Rate Generator Register (UBG1 $n$ )	8	W <sup>2</sup>	0x00	<a href="#">41.3.11/41-16</a>
0x1C	UART Baud Rate Generator Register (UBG2 $n$ )	8	W <sup>2</sup>	0x00	<a href="#">41.3.11/41-16</a>
0x34	UART Input Port Register (UIP $n$ )	8	R	0xFF	<a href="#">41.3.12/41-17</a>
0x38	UART Output Port Bit Set Command Register (UOP1 $n$ )	8	W <sup>2</sup>	0x00	<a href="#">41.3.13/41-18</a>
0x3C	UART Output Port Bit Reset Command Register (UOP0 $n$ )	8	W <sup>2</sup>	0x00	<a href="#">41.3.13/41-18</a>

<sup>1</sup> UMR1 $n$ , UMR2 $n$ , and UCSR $n$  must be changed only after the receiver/transmitter is issued a software reset command. If operation is not disabled, undesirable results may occur.

<sup>2</sup> Reading this register results in undesired effects and possible incorrect transmission or reception of characters. Register contents may also be changed.

### 41.3.1 UART Mode Registers 1 (UMR1 $n$ )

The UMR1 $n$  registers control UART module configuration. UMR1 $n$  can be read or written when the mode register pointer points to it, at RESET or after a RESET MODE REGISTER POINTER command using UCRn[MISC]. After UMR1 $n$  is read or written, the pointer points to UMR2 $n$ .

Address:	0xFC06_0000 (UMR10)	0xEC06_4000 (UMR15)	Access: User read/write <sup>1</sup>					
	0xFC06_4000 (UMR11)	0xEC06_8000 (UMR16)						
	0xFC06_8000 (UMR12)	0xEC06_C000 (UMR17)						
	0xFC06_C000 (UMR13)	0xEC07_0000 (UMR18)						
	0xEC06_0000 (UMR14)	0xEC07_4000 (UMR19)						
	7	6	5	4	3	2	1	0
R	RXRTS	RXIRQ/ FFULL	ERR	PM	PT	B/C		
W								
Reset:	0	0	0	0	0	0	0	0

<sup>1</sup> After UMR1 $n$  is read or written, the pointer points to UMR2 $n$

Figure 41-3. UART Mode Registers 1 (UMR1 $n$ )

Table 41-4. UMR1 $n$  Field Descriptions

Field	Description
7 RXRTS	Receiver request-to-send. Allows the $\overline{UnRTS}$ output to control the $\overline{UnCTS}$ input of the transmitting device to prevent receiver overrun. If the receiver and transmitter are incorrectly programmed for $\overline{UnRTS}$ control, $\overline{UnRTS}$ control is disabled for both. Transmitter RTS control is configured in UMR2 $n$ [TXRTS]. 0 The receiver has no effect on $\overline{UnRTS}$ . 1 When a valid start bit is received, $\overline{UnRTS}$ is negated if the UART's FIFO is full. $\overline{UnRTS}$ is reasserted when the FIFO has an empty position available.
6 RXIRQ/ FFULL	Receiver interrupt select. 0 RXRDY is the source generating interrupt or DMA requests. 1 FFULL is the source generating interrupt or DMA requests.
5 ERR	Error mode. Configures the FIFO status bits, USR $n$ [RB,FE,PE]. 0 Character mode. The USR $n$ values reflect the status of the character at the top of the FIFO. ERR must be 0 for correct A/D flag information when in multidrop mode. 1 Block mode. The USR $n$ values are the logical OR of the status for all characters reaching the top of the FIFO since the last RESET ERROR STATUS command for the UART was issued. See <a href="#">Section 41.3.5, “UART Command Registers (UCRn).”</a>
4–3 PM	Parity mode. Selects the parity or multidrop mode for the UART. The parity bit is added to the transmitted character, and the receiver performs a parity check on incoming data. The value of PM affects PT, as shown below.

**Table 41-4. UMR1n Field Descriptions (continued)**

Field	Description																				
2 PT	Parity type. PM and PT together select parity type (PM = 0x) or determine whether a data or address character is transmitted (PM = 11).  <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;">PM</th> <th style="text-align: center;">Parity Mode</th> <th style="text-align: center;">Parity Type (PT= 0)</th> <th style="text-align: center;">Parity Type (PT= 1)</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">00</td><td style="text-align: center;">With parity</td><td style="text-align: center;">Even parity</td><td style="text-align: center;">Odd parity</td></tr> <tr> <td style="text-align: center;">01</td><td style="text-align: center;">Force parity</td><td style="text-align: center;">Low parity</td><td style="text-align: center;">High parity</td></tr> <tr> <td style="text-align: center;">10</td><td style="text-align: center;">No parity</td><td colspan="2" style="text-align: center;">N/A</td></tr> <tr> <td style="text-align: center;">11</td><td style="text-align: center;">Multidrop mode</td><td style="text-align: center;">Data character</td><td style="text-align: center;">Address character</td></tr> </tbody> </table>	PM	Parity Mode	Parity Type (PT= 0)	Parity Type (PT= 1)	00	With parity	Even parity	Odd parity	01	Force parity	Low parity	High parity	10	No parity	N/A		11	Multidrop mode	Data character	Address character
PM	Parity Mode	Parity Type (PT= 0)	Parity Type (PT= 1)																		
00	With parity	Even parity	Odd parity																		
01	Force parity	Low parity	High parity																		
10	No parity	N/A																			
11	Multidrop mode	Data character	Address character																		
1–0 B/C	Bits per character. Selects the number of data bits per character to be sent. The values shown do not include start, parity, or stop bits. 00 5 bits 01 6 bits 10 7 bits 11 8 bits																				

### 41.3.2 UART Mode Register 2 (UMR2n)

The UMR2n registers control UART module configuration. UMR2n can be read or written when the mode register pointer points to it, which occurs after any access to UMR1n. UMR2n accesses do not update the pointer.

Address:	0xFC06_0000 (UMR20)	0xEC06_4000 (UMR25)	Access: User read/write <sup>1</sup>
	0xFC06_4000 (UMR21)	0xEC06_8000 (UMR26)	
	0xFC06_8000 (UMR22)	0xEC06_C000 (UMR27)	
	0xFC06_C000 (UMR23)	0xEC07_0000 (UMR28)	
	0xEC06_0000 (UMR24)	0xEC07_4000 (UMR29)	
Reset:	0	0	0
	0	0	0
	0	0	0
	0	0	0

<sup>1</sup> After UMR1n is read or written, the pointer points to UMR2n

**Figure 41-4. UART Mode Registers 2 (UMR2n)**

**Table 41-5. UMR2n Field Descriptions**

Field	Description
7–6 CM	Channel mode. Selects a channel mode. <a href="#">Section 41.4.3, “Looping Modes,”</a> describes individual modes. 00 Normal 01 Automatic echo 10 Local loopback 11 Remote loopback
5 TXRTS	Transmitter ready-to-send. Controls negation of $\overline{\text{UnRTS}}$ to automatically terminate a message transmission. Attempting to program a receiver and transmitter in the same UART for $\overline{\text{UnRTS}}$ control is not permitted and disables $\overline{\text{UnRTS}}$ control for both. 0 The transmitter has no effect on $\overline{\text{UnRTS}}$ . 1 In applications where the transmitter is disabled after transmission completes, setting this bit automatically clears UOP[RTS] one bit time after any characters in the transmitter shift and holding registers are completely sent, including the programmed number of stop bits.
4 TXCTS	Transmitter clear-to-send. If TXCTS and TXRTS are set, TXCTS controls the operation of the transmitter. 0 $\overline{\text{UnCTS}}$ has no effect on the transmitter. 1 Enables clear-to-send operation. The transmitter checks the state of $\overline{\text{UnCTS}}$ each time it is ready to send a character. If $\overline{\text{UnCTS}}$ is asserted, the character is sent; if it is deasserted, the signal $\overline{\text{UnTXD}}$ remains in the high state and transmission is delayed until $\overline{\text{UnCTS}}$ is asserted. Changes in $\overline{\text{UnCTS}}$ as a character is being sent do not affect its transmission.
3–0 SB	Stop-bit length control. Selects length of stop bit appended to the transmitted character. Stop-bit lengths of 9/16 to 2 bits are programmable for 6–8 bit characters. Lengths of 1-1/16 to 2 bits are programmable for 5-bit characters. In all cases, the receiver checks only for a high condition at the center of the first stop-bit position, one bit time after the last data bit or after the parity bit, if parity is enabled. If an external 1x clock is used for the transmitter, clearing bit 3 selects one stop bit and setting bit 3 selects two stop bits for transmission.

SB	5 Bits	6–8 Bits
0000	1.063	0.563
0001	1.125	0.625
0010	1.188	0.688
0011	1.250	0.750
0100	1.313	0.813
0101	1.375	0.875
0110	1.438	0.938
0111	1.500	1.000

SB	5–8 Bits
1000	1.563
1001	1.625
1010	1.688
1011	1.750
1100	1.813
1101	1.875
1110	1.938
1111	2.000

### 41.3.3 UART Status Registers (USR $n$ )

The **USRn** registers show the status of the transmitter, the receiver, and the FIFO.

Address: 0xFC06_0004 (USR0)	0xEC06_4004 (USR5)	Access: User read-only
0xFC06_4004 (USR1)	0xEC06_8004 (USR6)	
0xFC06_8004 (USR2)	0xEC06_C004 (USR7)	
0xFC06_C004 (USR3)	0xEC07_0004 (USR8)	
0xEC06_0004 (USR4)	0xEC07_4004 (USR9)	

**Figure 41-5. UART Status Registers (USR $n$ )**

**Table 41-6. USR*n* Field Descriptions**

Field	Description
7 RB	<p>Received break. The received break circuit detects breaks originating in the middle of a received character. However, a break in the middle of a character must persist until the end of the next detected character time.</p> <ul style="list-style-type: none"> <li>0 No break was received.</li> <li>1 An all-zero character of the programmed length was received without a stop bit. Only a single FIFO position is occupied when a break is received. Further entries to the FIFO are inhibited until UnRXD returns to the high state for at least one-half bit time, which equals two successive edges of the UART clock. RB is valid only when RXRDY is set.</li> </ul>
6 FE	<p>Framing error.</p> <ul style="list-style-type: none"> <li>0 No framing error occurred.</li> <li>1 No stop bit was detected when the corresponding data character in the FIFO was received. The stop-bit check occurs in the middle of the first stop-bit position. FE is valid only when RXRDY is set.</li> </ul>
5 PE	<p>Parity error. Valid only if RXRDY is set.</p> <ul style="list-style-type: none"> <li>0 No parity error occurred.</li> <li>1 If UMR1n[PM] equals 0x (with parity or force parity), the corresponding character in the FIFO was received with incorrect parity. If UMR1n[PM] equals 11 (multidrop), PE stores the received address or data (A/D) bit. PE is valid only when RXRDY is set.</li> </ul>
4 OE	<p>Overrun error. Indicates whether an overrun occurs.</p> <ul style="list-style-type: none"> <li>0 No overrun occurred.</li> <li>1 One or more characters in the received data stream have been lost. OE is set upon receipt of a new character when the FIFO is full and a character is already in the shift register waiting for an empty FIFO position. When this occurs, the character in the receiver shift register and its break detect, framing error status, and parity error, if any, are lost. The RESET ERROR STATUS command in UCRn clears OE.</li> </ul>
3 TEMP	<p>Transmitter empty.</p> <ul style="list-style-type: none"> <li>0 The transmit buffer is not empty. A character is shifted out, or the transmitter is disabled. The transmitter is enabled/disabled by programming UCRn[TC].</li> <li>1 The transmitter has underrun (the transmitter holding register and transmitter shift registers are empty). This bit is set after transmission of the last stop bit of a character if there are no characters in the transmitter holding register awaiting transmission.</li> </ul>

**Table 41-6. USR $n$  Field Descriptions (continued)**

Field	Description
2 TXRDY	Transmitter ready. 0 The CPU loaded the transmitter holding register, or the transmitter is disabled. 1 The transmitter holding register is empty and ready for a character. TXRDY is set when a character is sent to the transmitter shift register or when the transmitter is first enabled. If the transmitter is disabled, characters loaded into the transmitter holding register are not sent.
1 FFULL	FIFO full. 0 The FIFO is not full but may hold up to two unread characters. 1 A character was received and the receiver FIFO is now full. Any characters received when the FIFO is full are lost.
0 RXRDY	Receiver ready. 0 The CPU has read the receive buffer and no characters remain in the FIFO after this read. 1 One or more characters were received and are waiting in the receive buffer FIFO.

### 41.3.4 UART Clock Select Registers (UCSR $n$ )

The UCSR $n$ s select an external clock on the DTIN input (divided by 1 or 16) or a prescaled internal bus clock as the clocking source for the transmitter and receiver. See [Section 41.4.1, “Transmitter/Receiver Clock Source.”](#) The transmitter and receiver can use different clock sources. To use the internal bus clock for both, set UCSR $n$  to 0xDD.

The DMA timers are assigned to the UARTs as follows:

- DT0IN → UART0, 4, and 8
- DT1IN → UART1, 5, and 9
- DT2IN → UART2, 6
- DT3IN → UART3, 7

Address: 0xFC06_0004 (UCSR0) 0xFC06_4004 (UCSR1) 0xFC06_8004 (UCSR2) 0xFC06_C004 (UCSR3) 0xEC06_0004 (UCSR4)	0xEC06_4004 (UCSR5) 0xEC06_8004 (UCSR6) 0xEC06_C004 (UCSR7) 0xEC07_0004 (UCSR8) 0xEC07_4004 (UCSR9)	Access: User write-only
R W	7        6        5        4                 3        2        1        0	
Reset: See Note		See Note

**Note:** The RCS and TCS reset values are set so the receiver and transmitter use the prescaled internal bus clock as their clock source.

**Figure 41-6. UART Clock Select Registers (UCSR $n$ )**

**Table 41-7. UCSR $n$  Field Descriptions**

Field	Description
7–4 RCS	Receiver clock select. Selects the clock source for the receiver. 1101 Prescaled internal bus clock ( $f_{sys}/2$ ) 1110 DTnIN divided by 16 1111 DTnIN
3–0 TCS	Transmitter clock select. Selects the clock source for the transmitter. 1101 Prescaled internal bus clock ( $f_{sys}/2$ ) 1110 DTnIN divided by 16 1111 DTnIN

### 41.3.5 UART Command Registers (UCR $n$ )

The UCRs supply commands to the UART. Only multiple commands that do not conflict can be specified in a single write to a UCR $n$ . For example, RESET TRANSMITTER and ENABLE TRANSMITTER cannot be specified in one command.

Address:	0xFC06_0008 (UCR0)	0xEC06_4008 (UCR5)	Access: User write-only
	0xFC06_4008 (UCR1)	0xEC06_8008 (UCR6)	
	0xFC06_8008 (UCR2)	0xEC06_C008 (UCR7)	
	0xFC06_C008 (UCR3)	0xEC07_0008 (UCR8)	
	0xEC06_0008 (UCR4)	0xEC07_4008 (UCR9)	
R	7	6	5
			4
W	0	MISC	TC
			RC
Reset:	0	0	0
		0	0
		0	0
		0	0

**Figure 41-7. UART Command Registers (UCR $n$ )**

Table 41-8 describes UCR $n$  fields and commands. Examples in [Section 41.4.2, “Transmitter and Receiver Operating Modes,”](#) show how these commands are used.

**Table 41-8. UCR $n$  Field Descriptions**

Field	Description	
7	Reserved, must be cleared.	
6–4 MISC	MISC Field (this field selects a single command)	
	Command	Description
000	NO COMMAND	—
001	RESET MODE REGISTER POINTER	Causes the mode register pointer to point to UMR1 $n$ .
010	RESET RECEIVER	Immediately disables the receiver, clears USR $n$ [FFULL,RXRDY], and reinitializes the receiver FIFO pointer. No other registers are altered. Because it places the receiver in a known state, use this command instead of RECEIVER DISABLE when reconfiguring the receiver.
011	RESET TRANSMITTER	Immediately disables the transmitter and clears USR $n$ [TXEMP,TXRDY]. No other registers are altered. Because it places the transmitter in a known state, use this command instead of TRANSMITTER DISABLE when reconfiguring the transmitter.
100	RESET ERROR STATUS	Clears USR $n$ [RB,FE,PE,OE]. Also used in block mode to clear all error bits after a data block is received.
101	RESET BREAK – CHANGE INTERRUPT	Clears the delta break bit, UISR $n$ [DB].
110	START BREAK	Forces UnTXD low. If the transmitter is empty, break may be delayed up to one bit time. If the transmitter is active, break starts when character transmission completes. Break is delayed until any character in the transmitter shift register is sent. Any character in the transmitter holding register is sent after the break. Transmitter must be enabled for the command to be accepted. This command ignores the state of $\overline{UnCTS}$ .
111	STOP BREAK	Causes UnTXD to go high (mark) within two bit times. Any characters in the transmit buffer are sent.

**Table 41-8. UCR $n$  Field Descriptions (continued)**

Field	Description		
3–2 TC	Transmit command field. Selects a single transmit command.		
	<b>Command</b>	<b>Description</b>	
	00	NO ACTION TAKEN	Causes the transmitter to stay in its current mode: if the transmitter is enabled, it remains enabled; if the transmitter is disabled, it remains disabled.
	01	TRANSMITTER ENABLE	Enables operation of the UART's transmitter. USR $n$ [TXEMP,TXRDY] are set. If the transmitter is already enabled, this command has no effect.
	10	TRANSMITTER DISABLE	Terminates transmitter operation and clears USR $n$ [TXEMP,TXRDY]. If a character is being sent when the transmitter is disabled, transmission completes before the transmitter becomes inactive. If the transmitter is already disabled, the command has no effect.
	11	—	Reserved, do not use.
1–0 RC	Receive command field. Selects a single receive command.		
	<b>Command</b>	<b>Description</b>	
	00	NO ACTION TAKEN	Causes the receiver to stay in its current mode. If the receiver is enabled, it remains enabled; if disabled, it remains disabled.
	01	RECEIVER ENABLE	If the UART module is not in multidrop mode (UMR1 $n$ [PM] ≠ 11), RECEIVER ENABLE enables the UART's receiver and forces it into search-for-start-bit state. If the receiver is already enabled, this command has no effect.
	10	RECEIVER DISABLE	Disables the receiver immediately. Any character being received is lost. The command does not affect receiver status bits or other control registers. If the UART module is programmed for local loopback or multidrop mode, the receiver operates even though this command is selected. If the receiver is already disabled, the command has no effect.
	11	—	Reserved, do not use.

### 41.3.6 UART Receive Buffers (URB $n$ )

The receive buffers contain one serial shift register and three receiver holding registers, which act as a FIFO. UnRXD is connected to the serial shift register. The CPU reads from the top of the FIFO while the receiver shifts and updates from the bottom when the shift register is full (see [Figure 41-18](#)). RB contains the character in the receiver.

**Figure 41-8. UART Receive Buffer (URBn)**

#### 41.3.7 UART Transmit Buffers (UTB $n$ )

The transmit buffers consist of the transmitter holding register and the transmitter shift register. The holding register accepts characters from the bus master if UART's  $\text{USR}_n[\text{TXRDY}]$  is set. A write to the transmit buffer clears  $\text{USR}_n[\text{TXRDY}]$ , inhibiting any more characters until the shift register can accept more data. When the shift register is empty, it checks if the holding register has a valid character to be sent ( $\text{TXRDY} = 0$ ). If there is a valid character, the shift register loads it and sets  $\text{USR}_n[\text{TXRDY}]$  again. Writes to the transmit buffer when the UART's TXRDY is cleared and the transmitter is disabled have no effect on the transmit buffer.

Figure 41-9 shows UTB $n$ . TB contains the character in the transmit buffer.

**Figure 41-9. UART Transmit Buffer (UTB $n$ )**

#### 41.3.8 UART Input Port Change Registers (UIPCR $n$ )

The UIPCRs hold the current state and the change-of-state for UnCTS.

**Figure 41-10. UART Input Port Changed Registers (UIPCR $n$ )**

**Table 41-9. UIPCRn Field Descriptions**

Field	Description
7–5	Reserved
4 COS	<p>Change of state (high-to-low or low-to-high transition).</p> <p>0 No change-of-state since the CPU last read UIPCRn. Reading UIPCRn clears UISRn[COS].</p> <p>1 A change-of-state longer than 25–50 µs occurred on the <math>\overline{\text{UnCTS}}</math> input. UACRn can be programmed to generate an interrupt to the CPU when a change of state is detected.</p>
3–1	Reserved
0 CTS	<p>Current state of clear-to-send. Starting two serial clock periods after reset, CTS reflects the state of <math>\overline{\text{UnCTS}}</math>. If <math>\overline{\text{UnCTS}}</math> is detected asserted at that time, COS is set, which initiates an interrupt if UACRn[IEC] is enabled.</p> <p>0 The current state of the <math>\overline{\text{UnCTS}}</math> input is asserted.</p> <p>1 The current state of the <math>\overline{\text{UnCTS}}</math> input is deasserted.</p>

#### 41.3.9 UART Auxiliary Control Register (UACR $n$ )

The UACRs control the input enable.

**Figure 41-11. UART Auxiliary Control Registers (UACR $n$ )**

**Table 41-10. UACR $n$  Field Descriptions**

Field	Description
7–1	Reserved, must be cleared.
0 IEC	Input enable control. 0 Setting the corresponding UIPCR $n$ bit has no effect on UISR $n$ [COS]. 1 UISR $n$ [COS] is set and an interrupt is generated when the UIPCR $n$ [COS] is set by an external transition on the $\overline{UnCTS}$ input (if UIMR $n$ [COS] = 1).

### 41.3.10 UART Interrupt Status/Mask Registers (UISR $n$ /UIMR $n$ )

The UISRs provide status for all potential interrupt sources. UISR $n$  contents are masked by UIMR $n$ . If corresponding UISR $n$  and UIMR $n$  bits are set, internal interrupt output is asserted. If a UIMR $n$  bit is cleared, state of the corresponding UISR $n$  bit has no effect on the output.

The UISR $n$  and UIMR $n$  registers share the same space in memory. Reading this register provides the user with interrupt status, while writing controls the mask bits.

#### NOTE

True status is provided in the UISR $n$  regardless of UIMR $n$  settings. UISR $n$  is cleared when the UART module is reset.

Address:	0xFC06_0014 (UISR0)	0xEC06_4014 (UISR5)	Access: User read/write																											
	0xFC06_4014 (UISR1)	0xEC06_8014 (UISR6)																												
	0xFC06_8014 (UISR2)	0xEC06_C014 (UISR7)																												
	0xFC06_C014 (UISR3)	0xEC07_0014 (UISR8)																												
	0xEC06_0014 (UISR4)	0xEC07_4014 (UISR9)																												
<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="width: 10px;"></td> <td style="width: 10px;">7</td> <td style="width: 10px;">6</td> <td style="width: 10px;">5</td> <td style="width: 10px;">4</td> <td style="width: 10px;">3</td> <td style="width: 10px;">2</td> <td style="width: 10px;">1</td> <td style="width: 10px;">0</td> </tr> <tr> <td style="text-align: right; vertical-align: middle;">R (UISR<math>n</math>)</td> <td>COS</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>DB</td> <td>FFULL/ RXRDY</td> <td>TXRDY</td> </tr> <tr> <td style="text-align: right; vertical-align: middle;">W (UIMR<math>n</math>)</td> <td>COS</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>DB</td> <td>FFULL/ RXRDY</td> <td>TXRDY</td> </tr> </table>					7	6	5	4	3	2	1	0	R (UISR $n$ )	COS	0	0	0	0	DB	FFULL/ RXRDY	TXRDY	W (UIMR $n$ )	COS	0	0	0	0	DB	FFULL/ RXRDY	TXRDY
	7	6	5	4	3	2	1	0																						
R (UISR $n$ )	COS	0	0	0	0	DB	FFULL/ RXRDY	TXRDY																						
W (UIMR $n$ )	COS	0	0	0	0	DB	FFULL/ RXRDY	TXRDY																						
Reset:	0	0	0	0	0	0	0	0																						

**Figure 41-12. UART Interrupt Status/Mask Registers (UISR $n$ /UIMR $n$ )****Table 41-11. UISR $n$ /UIMR $n$  Field Descriptions**

Field	Description
7 COS	Change-of-state. 0 UIPCR $n$ [COS] is not selected. 1 Change-of-state occurred on $\overline{UnCTS}$ and was programmed in UACR $n$ [IEC] to cause an interrupt.
6–3	Reserved, must be cleared.
2 DB	Delta break. 0 No new break-change condition to report. <a href="#">Section 41.3.5, “UART Command Registers (UCR<math>n</math>)</a> , describes the RESET BREAK-CHANGE INTERRUPT command. 1 The receiver detected the beginning or end of a received break.

Table 41-11. UISR $n$ /UIMR $n$  Field Descriptions

Field	Description			
1 FFULL/ RXRDY	Status of FIFO or receiver, depending on UMR1[FFULL/RXRDY] bit. Duplicate of USR $n$ [FIFO] and USR $n$ [RXRDY]			
	UIMR $n$ [FFULL/RXRDY]		UISR $n$ [FFULL/RXRDY]	
	0	0	0 (RXRDY)	1 (FIFO)
	1	0	Receiver not ready	FIFO not full
	0	1	Receiver not ready	FIFO is full, Do not interrupt
	1	1	Receiver is ready, interrupt	FIFO is full, interrupt
0 TXRDY	<p>Transmitter ready. This bit is the duplication of USR<math>n</math>[TXRDY].</p> <p>0 The transmitter holding register was loaded by the CPU or the transmitter is disabled. Characters loaded into the transmitter holding register when TXRDY is cleared are not sent.</p> <p>1 The transmitter holding register is empty and ready to be loaded with a character.</p>			

### 41.3.11 UART Baud Rate Generator Registers (UBG1 $n$ /UBG2 $n$ )

The UBG1 $n$  registers hold the MSB, and the UBG2 $n$  registers hold the LSB of the preload value. UBG1 $n$  and UBG2 $n$  concatenate to provide a divider to the internal bus clock for transmitter/receiver operation, as described in Section 41.4.1.2.1, “Internal Bus Clock Baud Rates.”

Address:	0xFC06_0018 (UBG10) 0xFC06_4018 (UBG11) 0xFC06_8018 (UBG12) 0xFC06_C018 (UBG13) 0xEC06_0018 (UBG14)	0xEC06_4018 (UBG15) 0xEC06_8018 (UBG16) 0xEC06_C018 (UBG17) 0xEC07_0018 (UBG18) 0xEC07_4018 (UBG19)	Access: User write-only
R	7 0	6 0	5 0
	4 0		3 0
			2 0
			1 0
			0 0
W	Divider MSB		
Reset:	0	0	0
	0	0	0
	0	0	0
	0	0	0

Figure 41-13. UART Baud Rate Generator Registers (UBG1 $n$ )

Address: 0xFC06_001C (UBG20)	0xEC06_401C (UBG25)	Access: User write-only
0xFC06_401C (UBG21)	0xEC06_801C (UBG26)	
0xFC06_801C (UBG22)	0xEC06_C01C (UBG27)	
0xFC06_C01C (UBG23)	0xEC07_001C (UBG28)	
0xEC06_001C (UBG24)	0xEC07_401C (UBG29)	

R								
W								
Divider LSB								
Reset:	0	0	0	0	0	0	0	0

Figure 41-14. UART Baud Rate Generator Registers (UBG2n)

**NOTE**

The minimum value loaded on the concatenation of UBG1 $n$  with UBG2 $n$  is 0x0002. The UBG2 $n$  reset value of 0x00 is invalid and must be written to before the UART transmitter or receiver are enabled. UBG1 $n$  and UBG2 $n$  are write-only and cannot be read by the CPU.

**41.3.12 UART Input Port Register (UIP $n$ )**

The UIP $n$  registers show the current state of the  $\overline{UnCTS}$  input.

Address: 0xFC06_0034 (UIP0)	0xEC06_4034 (UIP5)	Access: User read-only
0xFC06_4034 (UIP1)	0xEC06_8034 (UIP6)	
0xFC06_8034 (UIP2)	0xEC06_C034 (UIP7)	
0xFC06_C034 (UIP3)	0xEC07_0034 (UIP8)	
0xEC06_0034 (UIP4)	0xEC07_4034 (UIP9)	

R	1	1	1	1	1	1	1	CTS
W								
Reset:	1	1	1	1	1	1	1	1

Figure 41-15. UART Input Port Registers (UIP $n$ )Table 41-12. UIP $n$  Field Descriptions

Field	Description
7–1	Reserved
0 CTS	<p>Current state of clear-to-send. The <math>\overline{UnCTS}</math> value is latched and reflects the state of the input pin when UIP<math>n</math> is read.</p> <p><b>Note:</b> This bit has the same function and value as UIPCR<math>n</math>[CTS].</p> <p>0 The current state of the <math>\overline{UnCTS}</math> input is logic 0. 1 The current state of the <math>\overline{UnCTS}</math> input is logic 1.</p>

### 41.3.13 UART Output Port Command Registers (UOP1n/UOP0n)

The  $\overline{UnRTS}$  output can be asserted by writing a 1 to UOP1n[RTS] and negated by writing a 1 to UOP0n[RTS].

Address: 0xFC06_0038 (UOP10)	0xEC06_4038 (UOP15)	Access: User write-only
0xFC06_003C (UOP00)	0xEC06_403C (UOP05)	
0xFC06_4038 (UOP11)	0xEC06_8038 (UOP16)	
0xFC06_403C (UOP01)	0xEC06_803C (UOP06)	
0xFC06_8038 (UOP12)	0xEC06_C038 (UOP17)	
0xFC06_803C (UOP02)	0xEC06_C03C (UOP07)	
0xFC06_C038 (UOP13)	0xEC07_0038 (UOP18)	
0xFC06_C03C (UOP03)	0xEC07_003C (UOP08)	
0xEC06_0038 (UOP14)	0xEC07_4038 (UOP19)	
0xEC06_003C (UOP04)	0xEC07_403C (UOP09)	

	7	6	5	4		3	2	1	0
R									
W	0	0	0	0	0	0	0	0	RTS

Reset:	0	0	0	0	0	0	0	0
--------	---	---	---	---	---	---	---	---

Figure 41-16. UART Output Port Command Registers (UOP1n/UOP0n)

Table 41-13. UOP1n/UOP0n Field Descriptions

Field	Description
7–1	Reserved, must be cleared.
0 RTS	Output port output. Controls assertion (UOP1)/negation (UOP0) of $\overline{UnRTS}$ output. 0 Not affected. 1 Asserts $\overline{UnRTS}$ in UOP1. Negates $\overline{UnRTS}$ in UOP0.

## 41.4 Functional Description

This section describes operation of the clock source generator, transmitter, and receiver.

### 41.4.1 Transmitter/Receiver Clock Source

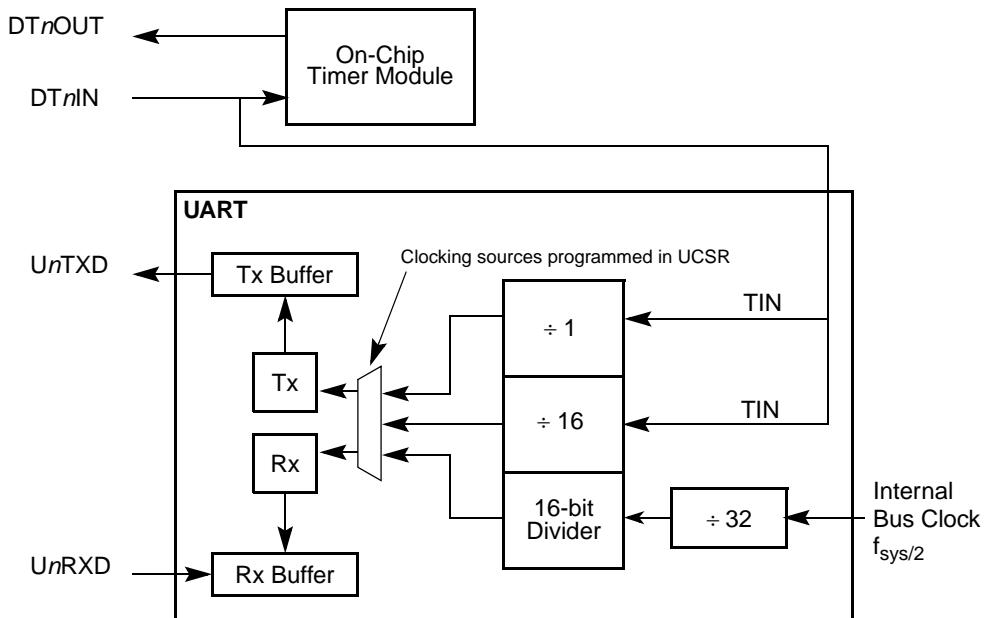
The internal bus clock serves as the basic timing reference for the clock source generator logic, which consists of a clock generator and a programmable 16-bit divider dedicated to each UART. The 16-bit divider is used to produce standard UART baud rates.

#### 41.4.1.1 Programmable Divider

As Figure 41-17 shows, the UARTn transmitter and receiver can use the following clock sources:

- An external clock signal on the DTnIN pin. When not divided, DTnIN provides a synchronous clock; when divided by 16, it is asynchronous.
- The internal bus clock supplies an asynchronous clock source divided by 32 and then divided by the 16-bit value programmed in UBG1n and UBG2n. See Section 41.3.11, “UART Baud Rate Generator Registers (UBG1n/UBG2n).”

The choice of DTIN or internal bus clock is programmed in the UCSR.



**Figure 41-17. Clocking Source Diagram**

#### NOTE

If DTnIN is a clocking source for the timer or UART, that timer module cannot use DTnIN for timer input capture.

### 41.4.1.2 Calculating Baud Rates

The following sections describe how to calculate baud rates.

#### 41.4.1.2.1 Internal Bus Clock Baud Rates

When the internal bus clock is the UART clocking source, it goes through a divide-by-32 prescaler and then passes through the 16-bit divider of the concatenated UBG1n and UBG2n registers. The baud-rate calculation is:

$$\text{Baudrate} = \frac{f_{\text{sys}/2}}{[32 \times \text{divider}]} \quad \text{Eqn. 41-1}$$

Using a 120-MHz internal bus clock and letting baud rate equal 9600, then

$$\text{Divider} = \frac{120\text{MHz}}{[32 \times 9600]} = 391(\text{decimal}) = 0x0187(\text{hexadecimal}) \quad \text{Eqn. 41-2}$$

Therefore, UBG1n equals 0x01 and UBG2n equals 0x87.

#### 41.4.1.2.2 External Clock

An external source clock (DTnIN) passes through a divide-by-1 or 16 prescaler. If  $f_{\text{extc}}$  is the external clock frequency, baud rate can be described with this equation:

$$\text{Baudrate} = \frac{f_{\text{extc}}}{(16 \text{ or } 1)} \quad \text{Eqn. 41-3}$$

## 41.4.2 Transmitter and Receiver Operating Modes

Figure 41-18 is a functional block diagram of the transmitter and receiver showing the command and operating registers, which are described generally in the following sections. For detailed descriptions, refer to [Section 41.3, “Memory Map/Register Definition.”](#)

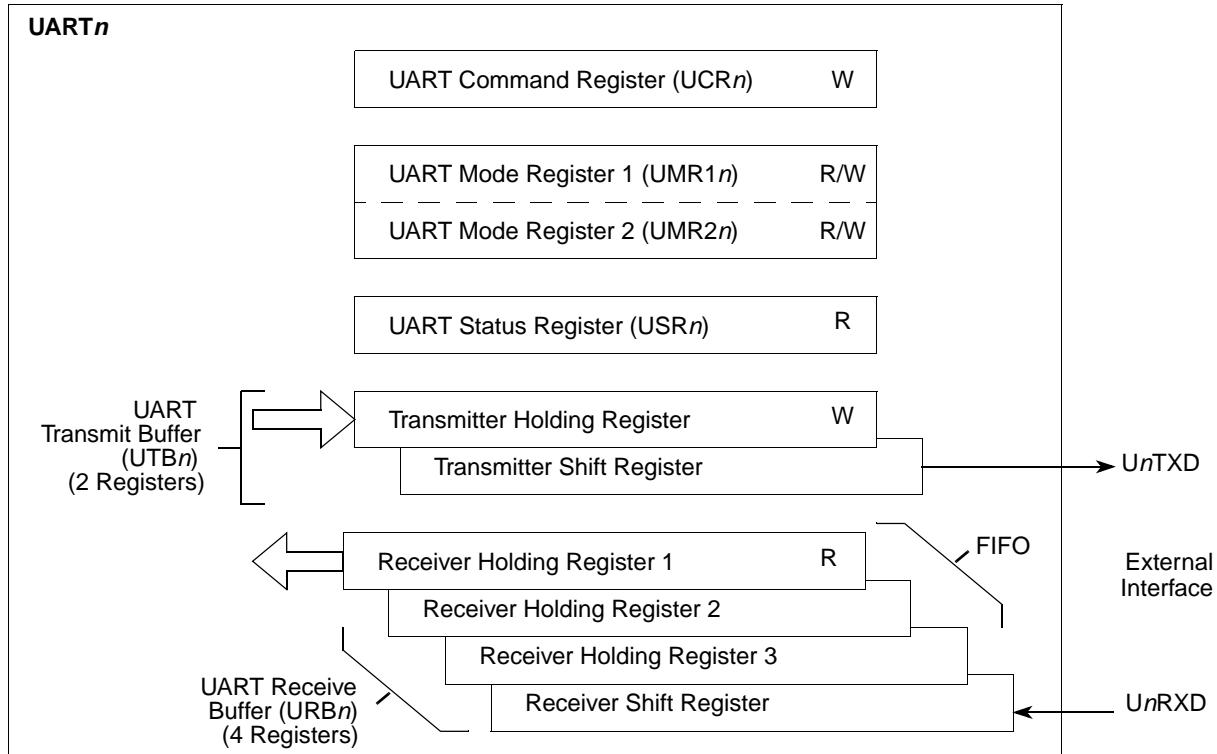


Figure 41-18. Transmitter and Receiver Functional Diagram

### 41.4.2.1 Transmitter

The transmitter is enabled through the UART command register (UCRn). When it is ready to accept a character, UART sets USRn[TXRDY]. The transmitter converts parallel data from the CPU to a serial bit stream on UnTXD. It automatically sends a start bit followed by the programmed number of data bits, an optional parity bit, and the programmed number of stop bits. The lsb is sent first. Data is shifted from the transmitter output on the falling edge of the clock source.

After the stop bits are sent, if no new character is in the transmitter holding register, the UnTXD output remains high (mark condition) and the transmitter empty bit (USRn[TXEMP]) is set. Transmission resumes and TXEMP is cleared when the CPU loads a new character into the UART transmit buffer (UTBn). If the transmitter receives a disable command, it continues until any character in the transmitter shift register is completely sent.

If the transmitter is reset through a software command, operation stops immediately (see [Section 41.3.5, “UART Command Registers \(UCRn\)”](#)). The transmitter is reenabled through the UCRn to resume operation after a disable or software reset.

If the clear-to-send operation is enabled,  $\overline{UnCTS}$  must be asserted for the character to be transmitted. If  $\overline{UnCTS}$  is negated in the middle of a transmission, the character in the shift register is sent and  $UnTXD$  remains in mark state until  $\overline{UnCTS}$  is reasserted. If transmitter is forced to send a continuous low condition by issuing a SEND BREAK command, transmitter ignores the state of  $\overline{UnCTS}$ .

If the transmitter is programmed to automatically negate  $\overline{UnRTS}$  when a message transmission completes,  $\overline{UnRTS}$  must be asserted manually before a message is sent. In applications in which the transmitter is disabled after transmission is complete and  $\overline{UnRTS}$  is appropriately programmed,  $\overline{UnRTS}$  is negated one bit time after the character in the shift register is completely transmitted. The transmitter must be manually reenabled by reasserting  $\overline{UnRTS}$  before the next message is sent.

Figure 41-19 shows the functional timing information for the transmitter.

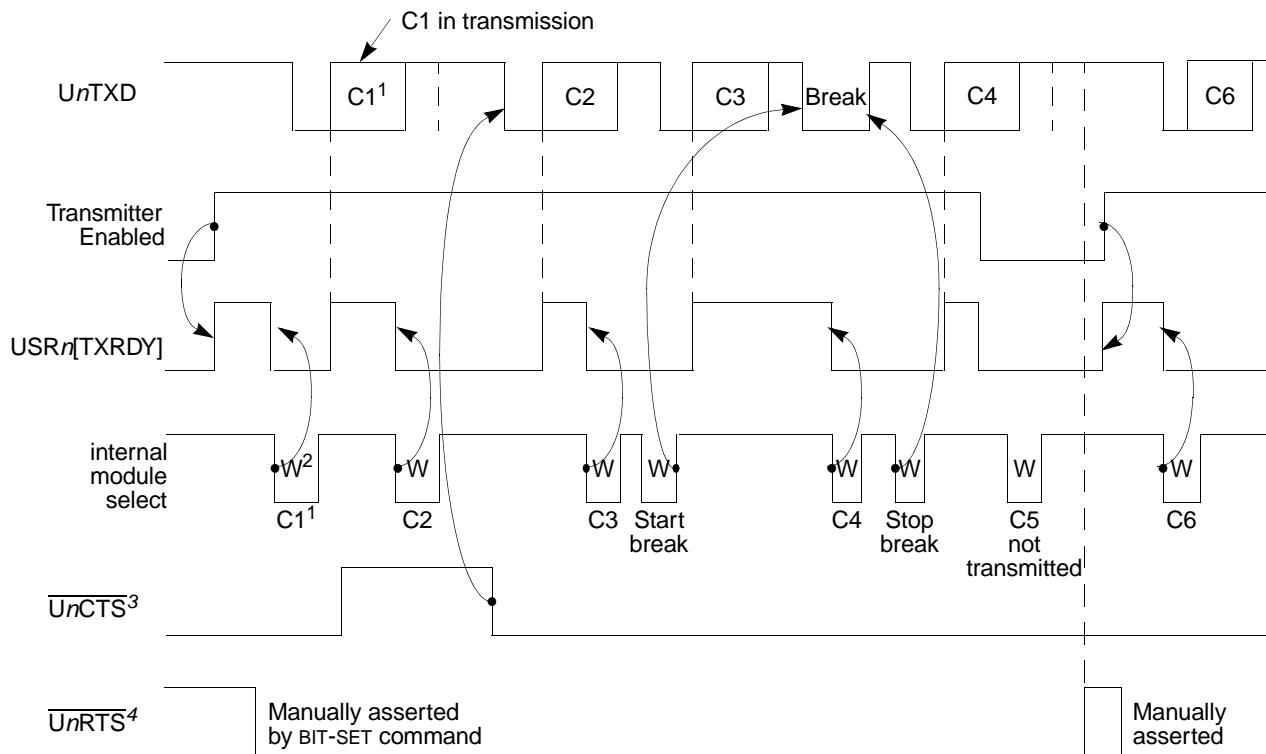


Figure 41-19. Transmitter Timing Diagram

#### 41.4.2.2 Receiver

The receiver is enabled through its UCR<sub>n</sub>, as described in Section 41.3.5, “UART Command Registers (UCR<sub>n</sub>).”

When the receiver detects a high-to-low (mark-to-space) transition of the start bit on  $UnRXD$ , the state of  $UnRXD$  is sampled eight times on the edge of the bit time clock starting one-half clock after the transition

(asynchronous operation) or at the next rising edge of the bit time clock (synchronous operation). If *UnRXD* is sampled high, start bit is invalid and the search for the valid start bit begins again.

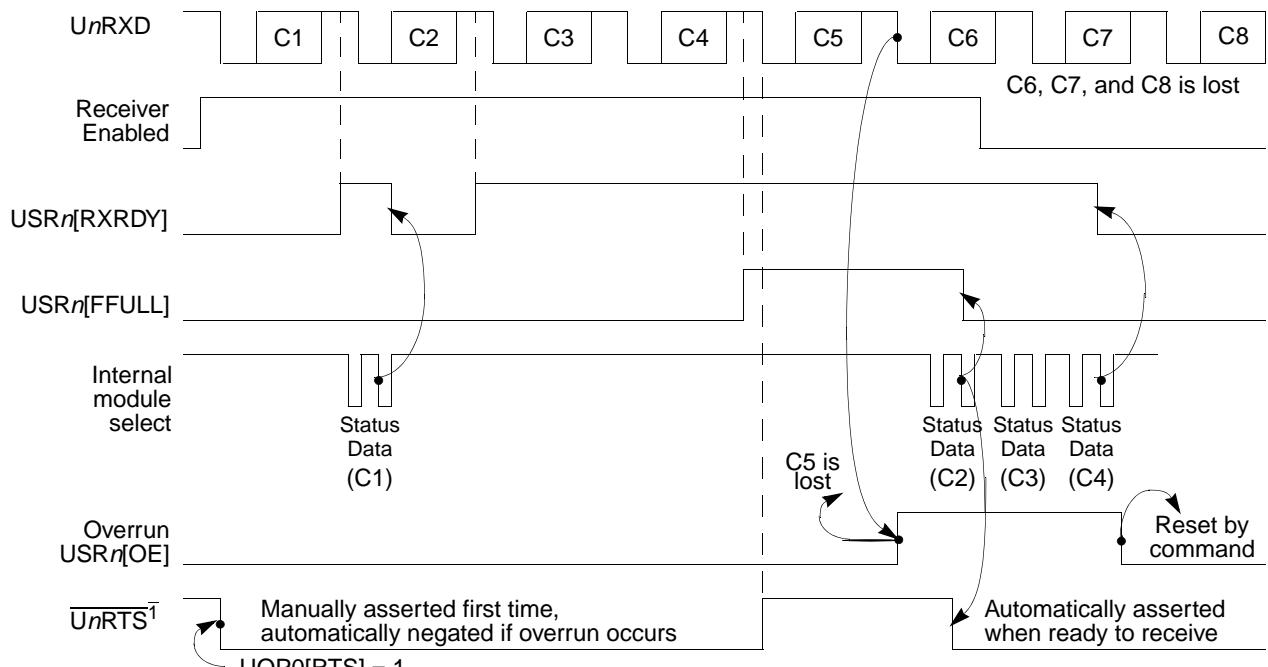
If *UnRXD* remains low, a valid start bit is assumed. The receiver continues sampling the input at one-bit time intervals at the theoretical center of the bit until the proper number of data bits and parity, if any, is assembled and one stop bit is detected. Data on the *UnRXD* input is sampled on the rising edge of the programmed clock source. The lsb is received first. The data then transfers to a receiver holding register and *USRn[RXRDY]* is set. If the character is less than 8 bits, the most significant unused bits in the receiver holding register are cleared.

After the stop bit is detected, receiver immediately looks for the next start bit. However, if a non-zero character is received without a stop bit (framing error) and *UnRXD* remains low for one-half of the bit period after the stop bit is sampled, receiver operates as if a new start bit were detected. Parity error, framing error, overrun error, and received break conditions set the respective PE, FE, OE, and RB error and break flags in the *USRn* at the received character boundary. They are valid only if *USRn[RXRDY]* is set.

If a break condition is detected (*UnRXD* is low for the entire character including the stop bit), a character of all 0s loads into the receiver holding register and *USRn[RB,RXRDY]* are set. *UnRXD* must return to a high condition for at least one-half bit time before a search for the next start bit begins.

The receiver detects the beginning of a break in the middle of a character if the break persists through the next character time. The receiver places the damaged character in the Rx FIFO and sets the corresponding *USRn* error bits and *USRn[RXRDY]*. Then, if the break lasts until the next character time, the receiver places an all-zero character into the Rx FIFO and sets *USRn[RB,RXRDY]*.

[Figure 41-20](#) shows receiver functional timing.



<sup>1</sup> UMR2n[RXRTS] = 1

Figure 41-20. Receiver Timing Diagram

#### 41.4.2.3 FIFO

The FIFO is used in the UART's receive buffer logic. The FIFO consists of three receiver holding registers. The receive buffer consists of the FIFO and a receiver shift register connected to the UnRXD (see Figure 41-18). Data is assembled in the receiver shift register and loaded into the top empty receiver holding register position of the FIFO. Therefore, data flowing from the receiver to the CPU is quadruple-buffered.

In addition to the data byte, three status bits—parity error (PE), framing error (FE), and received break (RB)—are appended to each data character in the FIFO; overrun error (OE) is not appended. By programming the ERR bit in the UART's mode register (UMR1n), status is provided in character or block modes.

USRn[RXRDY] is set when at least one character is available to be read by the CPU. A read of the receive buffer produces an output of data from the top of the FIFO. After the read cycle, the data at the top of the FIFO and its associated status bits are popped and the receiver shift register can add new data at the bottom of the FIFO. The FIFO-full status bit (FFULL) is set if all three positions are filled with data. The RXRDY or FFULL bit can be selected to cause an interrupt and TXRDY or RXRDY can be used to generate a DMA request.

The two error modes are selected by UMR1n[ERR]:

- In character mode (UMR1n[ERR] = 0), status is given in the USRn for the character at the top of the FIFO.

- In block mode, the  $USR_n$  shows a logical OR of all characters reaching the top of the FIFO since the last RESET ERROR STATUS command. Status is updated as characters reach the top of the FIFO. Block mode offers a data-reception speed advantage where the software overhead of error-checking each character cannot be tolerated. However, errors are not detected until the check is performed at the end of an entire message—the faulting character is not identified.

In either mode, reading the  $USR_n$  does not affect the FIFO. The FIFO is popped only when the receive buffer is read. The  $USR_n$  should be read before reading the receive buffer. If all three receiver holding registers are full, a new character is held in the receiver shift register until space is available. However, if a second new character is received, the contents of the character in the receiver shift register is lost, the FIFOs are unaffected, and  $USR_n[OE]$  is set when the receiver detects the start bit of the new overrunning character.

To support flow control, the receiver can be programmed to automatically negate and assert  $\overline{UnRTS}$ , in which case the receiver automatically negates  $\overline{UnRTS}$  when a valid start bit is detected and the FIFO is full. The receiver asserts  $\overline{UnRTS}$  when a FIFO position becomes available; therefore, connecting  $\overline{UnRTS}$  to the  $\overline{UnCTS}$  input of the transmitting device can prevent overrun errors.

### NOTE

The receiver continues reading characters in the FIFO if the receiver is disabled. If the receiver is reset, the FIFO,  $\overline{UnRTS}$  control, all receiver status bits, interrupts, and DMA requests are reset. No more characters are received until the receiver is reenabled.

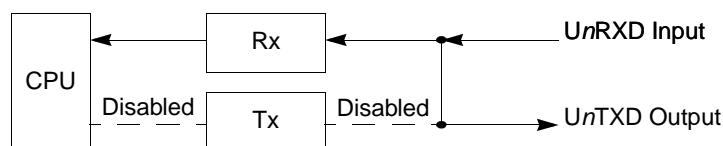
## 41.4.3 Looping Modes

The UART can be configured to operate in various looping modes. These modes are useful for local and remote system diagnostic functions. The modes are described in the following paragraphs and in [Section 41.3, “Memory Map/Register Definition.”](#)

The UART’s transmitter and receiver should be disabled when switching between modes. The selected mode is activated immediately upon mode selection, regardless of whether a character is being received or transmitted.

### 41.4.3.1 Automatic Echo Mode

In automatic echo mode, shown in [Figure 41-21](#), the UART automatically resends received data bit by bit. The local CPU-to-receiver communication continues normally, but the CPU-to-transmitter link is disabled. In this mode, received data is clocked on the receiver clock and re-sent on  $UnTXD$ . The receiver must be enabled, but the transmitter need not be.

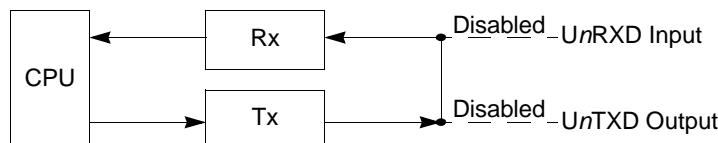


**Figure 41-21. Automatic Echo**

Because the transmitter is inactive,  $USRn[TXEMP,TXRDY]$  is inactive and data is sent as it is received. Received parity is checked but not recalculated for transmission. Character framing is also checked, but stop bits are sent as they are received. A received break is echoed as received until the next valid start bit is detected.

#### 41.4.3.2 Local Loopback Mode

[Figure 41-22](#) shows how  $UnTXD$  and  $UnRXD$  are internally connected in local loopback mode. This mode is for testing the operation of a UART by sending data to the transmitter and checking data assembled by the receiver to ensure proper operations.



**Figure 41-22. Local Loopback**

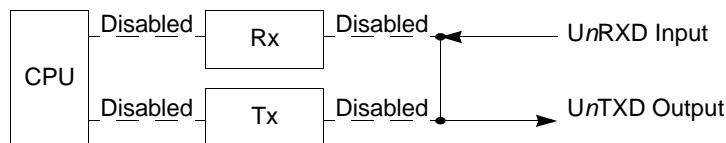
Features of this local loopback mode are:

- Transmitter and CPU-to-receiver communications continue normally in this mode.
- $UnRXD$  input data is ignored.
- $UnTXD$  is held marking.
- The receiver is clocked by the transmitter clock. The transmitter must be enabled, but the receiver need not be.

#### 41.4.3.3 Remote Loopback Mode

In remote loopback mode, shown in [Figure 41-23](#), the UART automatically transmits received data bit by bit on the  $UnTXD$  output. The local CPU-to-transmitter link is disabled. This mode is useful in testing receiver and transmitter operation of a remote UART. For this mode, transmitter uses the receiver clock.

Because the receiver is not active, received data cannot be read by the CPU and all status conditions are inactive. Received parity is not checked and is not recalculated for transmission. Stop bits are sent as they are received. A received break is echoed as received until next valid start bit is detected.

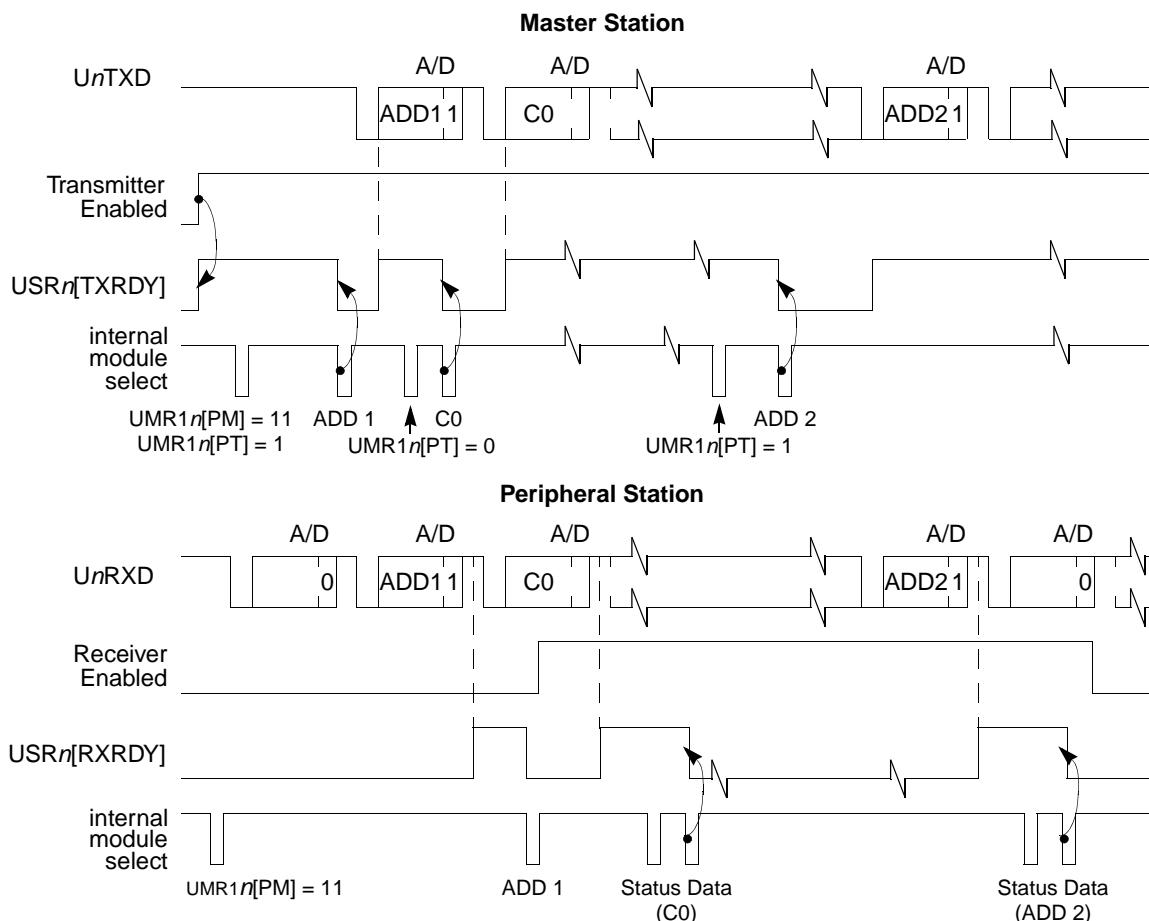


**Figure 41-23. Remote Loopback**

#### 41.4.4 Multidrop Mode

Setting  $UMR1n[PM]$  programs the UART to operate in a wake-up mode for multidrop or multiprocessor applications. In this mode, a master can transmit an address character followed by a block of data characters targeted for one of up to 256 slave stations.

Although slave stations have their receivers disabled, they continuously monitor the master's data stream. When the master sends an address character, the slave receiver notifies its respective CPU by setting  $\text{USR}_n[\text{RXRDY}]$  and generating an interrupt (if programmed to do so). Each slave station CPU then compares the received address to its station address and enables its receiver if it wishes to receive the subsequent data characters or block of data from the master station. Unaddressed slave stations continue monitoring the data stream. Data fields in the data stream are separated by an address character. After a slave receives a block of data, its CPU disables the receiver and repeats the process. Functional timing information for multidrop mode is shown in [Figure 41-24](#).



**Figure 41-24. Multidrop Mode Timing Diagram**

A character sent from the master station consists of a start bit, a programmed number of data bits, an address/data (A/D) bit flag, and a programmed number of stop bits. A/D equals 1 indicates an address character; A/D equals 0 indicates a data character. The polarity of A/D is selected through UMR1n[PT]. UMR1n should be programmed before enabling the transmitter and loading the corresponding data bits into the transmit buffer.

In multidrop mode, the receiver continuously monitors the received data stream, regardless of whether it is enabled or disabled. If the receiver is disabled, it sets the RXRDY bit and loads the character into the receiver holding register FIFO provided the received A/D bit is a 1 (address tag). The character is

discarded if the received A/D bit is 0 (data tag). If the receiver is enabled, all received characters are transferred to the CPU through the receiver holding register during read operations.

In either case, data bits load into the data portion of the FIFO while the A/D bit loads into the status portion of the FIFO normally used for a parity error (USR $n$ [PE]).

Framing error, overrun error, and break detection operate normally. The A/D bit takes the place of the parity bit; therefore, parity is neither calculated nor checked. Messages in this mode may continue containing error detection and correction information. If 8-bit characters are not required, one way to provide error detection is to use software to calculate parity and append it to the 5-, 6-, or 7-bit character.

#### 41.4.5 Single-Wire Mode with Polarity Control

Single-wire mode is supported on this device by using registers in [Chapter 15, “Pin-Multiplexing and Control.”](#)

- URTS\_POL and UCTS\_POL registers — Control the polarity of the UnRTS and UnCTS outputs of UART0–2. This allows the UART interface to connect to serial transceivers (e.g. RS232, RS485, and RS422) that require active low enables for direction control.
- UTXD\_WOM register — Enables single-wire mode for each of the nine UART modules. In single-wire mode the UnRXD signal is not required, and the UnTXD signal is used for transmit and receive for half duplex communication. This frees the UnRXD pin to be used as an alternate function or GPIO. The receive data path for each UART is internally connected to the UnTXD pin.
- URXD\_WOM register — Controls whether the internal UnRXD connection to UnTXD pin is automatic (controlled by the RTS signal) or manual (bit controlled).

##### NOTE

Enabling single wire mode (UTXD\_WOM) also enables wired-OR mode on the UnTXD pins, enabling them for peer-to-peer multidrop mode. The following figure depicts an example network in single-wire mode.

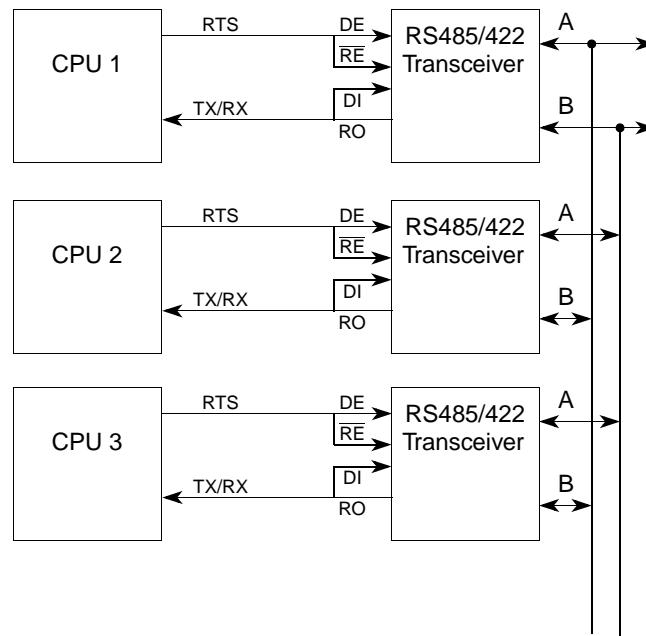


Figure 41-25. Single-Wire Mode

## 41.4.6 Bus Operation

This section describes bus operation during read, write, and interrupt acknowledge cycles to the UART module.

### 41.4.6.1 Read Cycles

The UART module responds to reads with byte data. Reserved registers return zeros.

### 41.4.6.2 Write Cycles

The UART module accepts write data as bytes only. Write cycles to read-only or reserved registers complete normally without an error termination, but data is ignored.

## 41.5 Initialization/Application Information

The software flowchart, [Figure 41-26](#), consists of:

- UART module initialization—These routines consist of SINIT and CHCHK (See Sheet 1 p. 41-31 and Sheet 2 p. 41-32). Before SINIT is called at system initialization, the calling routine allocates 2 words on the system FIFO. On return to the calling routine, SINIT passes UART status data on the FIFO. If SINIT finds no errors, the transmitter and receiver are enabled. SINIT calls CHCHK to perform the checks. When called, SINIT places the UART in local loopback mode and checks for the following errors:
  - Transmitter never ready
  - Receiver never ready

- Parity error
- Incorrect character received
- I/O driver routine—This routine (See Sheet 4 p. 41-34 and Sheet 5 p. 41-35) consists of INCH, the terminal input character routine which gets a character from the receiver, and OUTCH, which sends a character to the transmitter.
- Interrupt handling—This consists of SIRQ (See Sheet 4 p. 41-34), which is executed after the UART module generates an interrupt caused by a change-in-break (beginning of a break). SIRQ then clears the interrupt source, waits for the next change-in-break interrupt (end of break), clears the interrupt source again, then returns from exception processing to the system monitor.

## 41.5.1 Interrupt and DMA Request Initialization

### 41.5.1.1 Setting up the UART to Generate Core Interrupts

The list below provides steps to properly initialize the UART to generate an interrupt request to the processor's interrupt controller. See [Section 17.2.9.1, “Interrupt Sources,”](#) for details on interrupt assignments for the UART modules.

1. Initialize the appropriate ICR $x$  register in the interrupt controller.
2. Unmask appropriate bits in IMR in the interrupt controller.
3. Unmask appropriate bits in the core's status register (SR) to enable interrupts.
4. If TXRDY or RXRDY generates interrupt requests, verify that the corresponding UART DMA channels are not enabled.
5. Initialize interrupts in the UART, see [Table 41-14](#).

**Table 41-14. UART Interrupts**

Register	Bit	Interrupt
UMR $1n$	6	RxIRQ
UIMR $n$	7	Change of State (COS)
UIMR $n$	2	Delta Break
UIMR $n$	1	RxFIFO Full
UIMR $n$	0	TXRDY

### 41.5.1.2 Setting up the UART to Request DMA Service

The UART is capable of generating two internal DMA request signals: transmit and receive.

The transmit DMA request signal is asserted when the TXRDY (transmitter ready) in the UART interrupt status register (UISR $n$ [TXRDY]) is set. When the transmit DMA request signal is asserted, the DMA can initiate a data copy, reading the next character transmitted from memory and writing it into the UART transmit buffer (UTB $n$ ). This allows the DMA channel to stream data from memory to the UART for transmission without processor intervention. After the entire message has been moved into the UART, the DMA would typically generate an end-of-data-transfer interrupt request to the CPU. The resulting

interrupt service routine (ISR) could query the UART programming model to determine the end-of-transmission status.

Similarly, the receive DMA request signal is asserted when the FIFO full or receive ready (FFULL/RXRDY) flag in the interrupt status register (UISR $n$ [FFULL/RXRDY]) is set. When the receive DMA request signal is asserted, the DMA can initiate a data move, reading the appropriate characters from the UART receive buffer (URB $n$ ) and storing them in memory. This allows the DMA channel to stream data from the UART receive buffer into memory without processor intervention. After the entire message has been moved from the UART, the DMA would typically generate an end-of-data-transfer interrupt request to the CPU. The resulting interrupt service routine (ISR) should query the UART programming model to determine the end-of-transmission status. In typical applications, the receive DMA request should be configured to use RXRDY directly (and not FFULL) to remove any complications related to retrieving the final characters from the FIFO buffer.

The implementation described in this section allows independent DMA processing of transmit and receive data while continuing to support interrupt notification to the processor for  $\overline{CTS}$  change-of-state and delta break error managing.

[Table 41-15](#) shows the DMA requests.

**Table 41-15. UART DMA Requests**

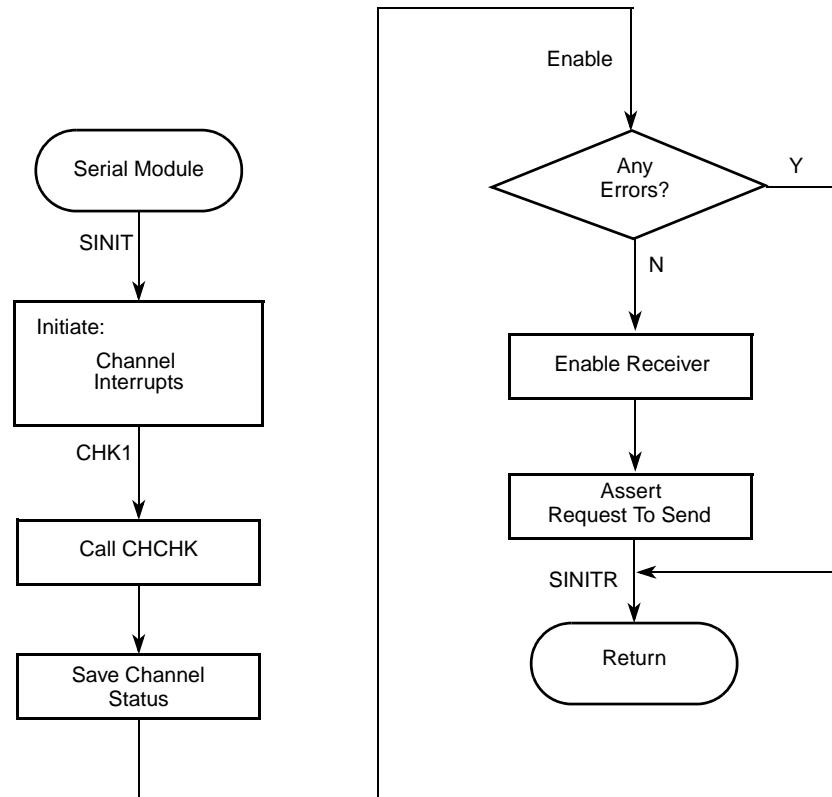
Register	Bit	DMA Request
UISR $n$	1	Receive DMA request
UISR $n$	0	Transmit DMA request

## 41.5.2 UART Module Initialization Sequence

The following shows the UART module initialization sequence.

1. UCR $n$ :
  - a) Reset the receiver and transmitter.
  - b) Reset the mode pointer (MISC[2–0] = 0b001).
2. UIMR $n$ : Enable the desired interrupt sources.
3. UACR $n$ : Initialize the input enable control (IEC bit).
4. UCSR $n$ : Select the receiver and transmitter clock. Use timer as source if required.
5. UMR1 $n$ :
  - a) If preferred, program operation of receiver ready-to-send (RXRTS bit).
  - a) Select receiver-ready or FIFO-full notification (RXRDY/FFULL bit).
  - b) Select character or block error mode (ERR bit).
  - c) Select parity mode and type (PM and PT bits).
  - d) Select number of bits per character (B/Cx bits).
6. UMR2 $n$ :
  - a) Select the mode of operation (CM bits).
  - b) If preferred, program operation of transmitter ready-to-send (TXRTS).

- c) If preferred, program operation of clear-to-send (TXCTS bit).
  - d) Select stop-bit length (SB bits).
7. UCRn: Enable transmitter and/or receiver.



**Figure 41-26. UART Mode Programming Flowchart (Sheet 1 of 5)**

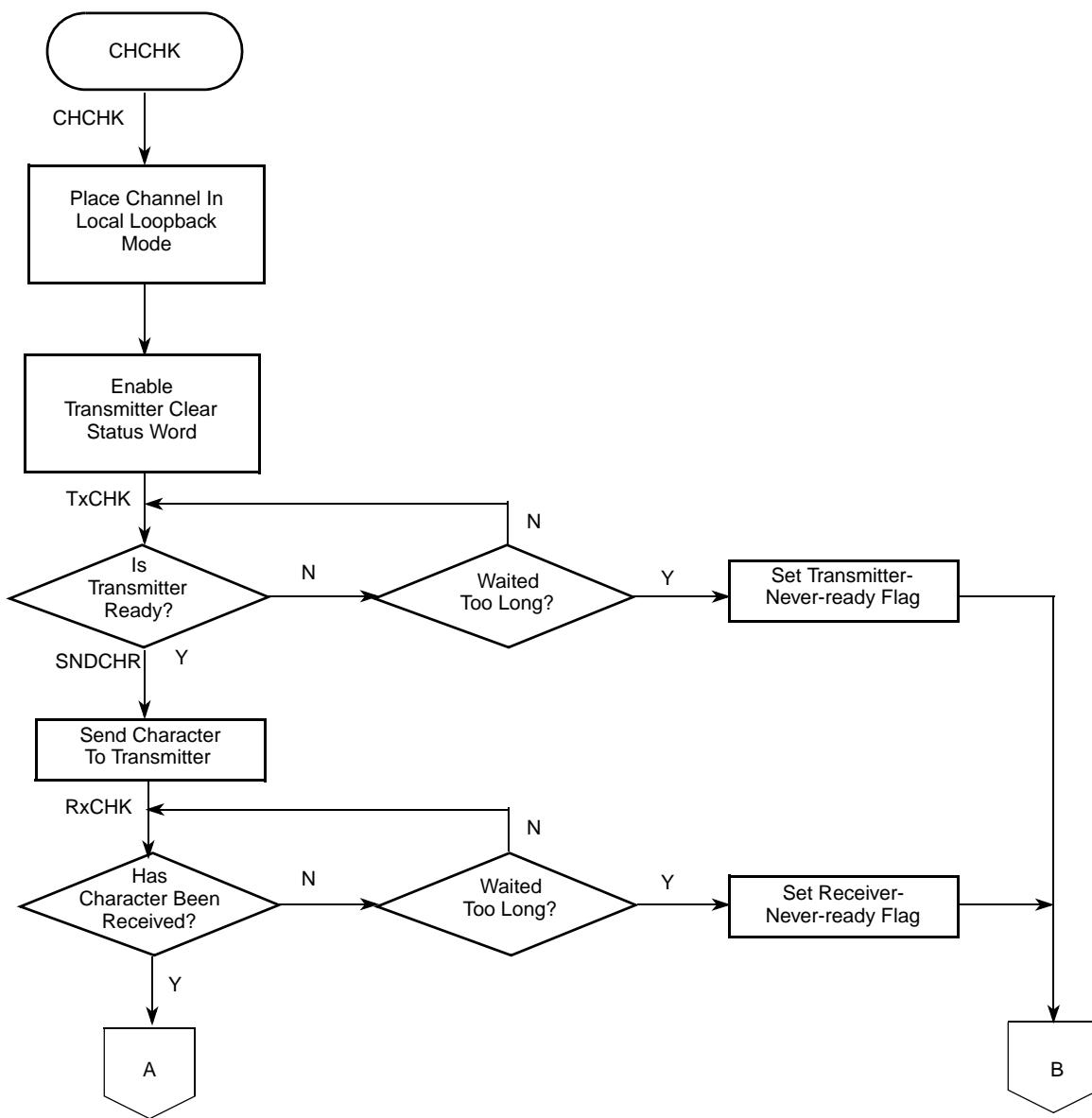


Figure 41-26. UART Mode Programming Flowchart (Sheet 2 of 5)

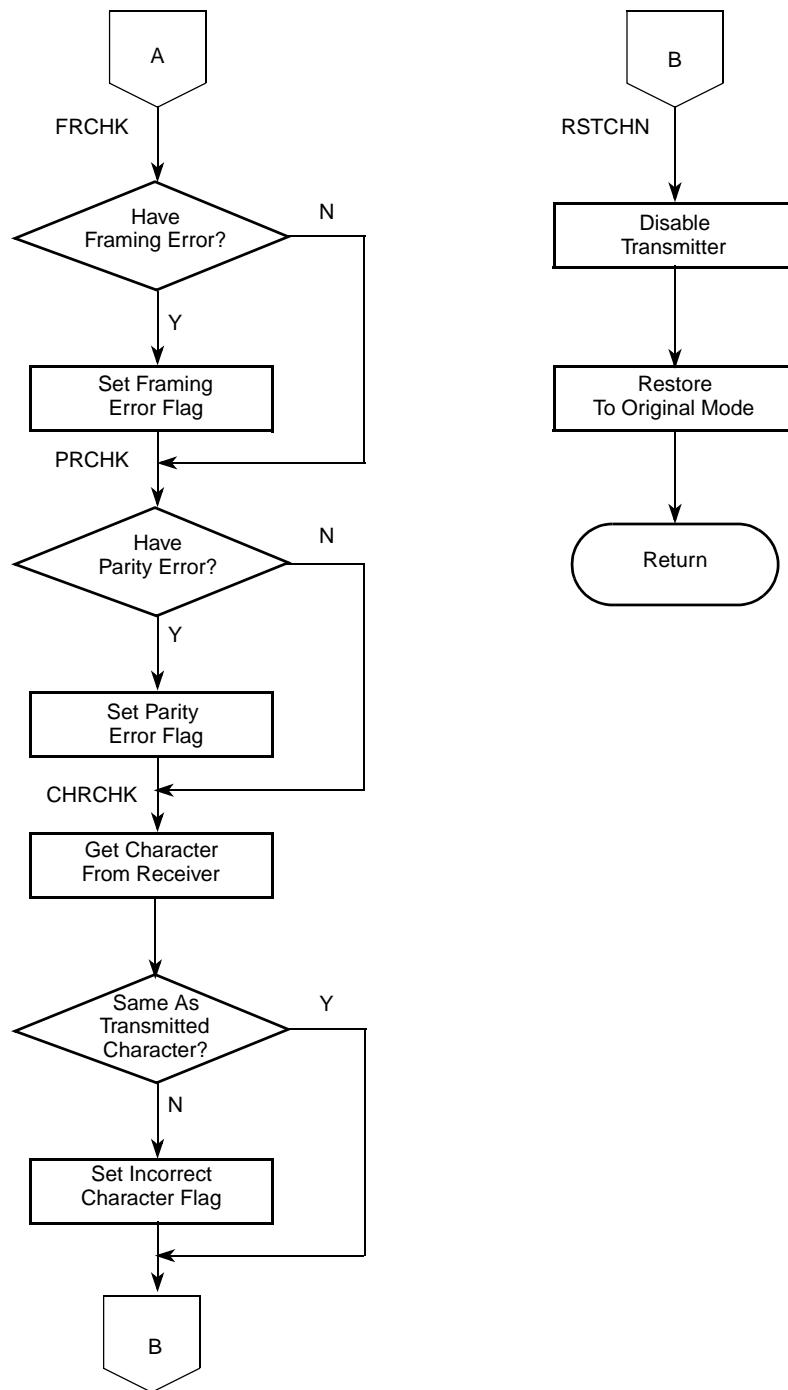


Figure 41-26. UART Mode Programming Flowchart (Sheet 3 of 5)

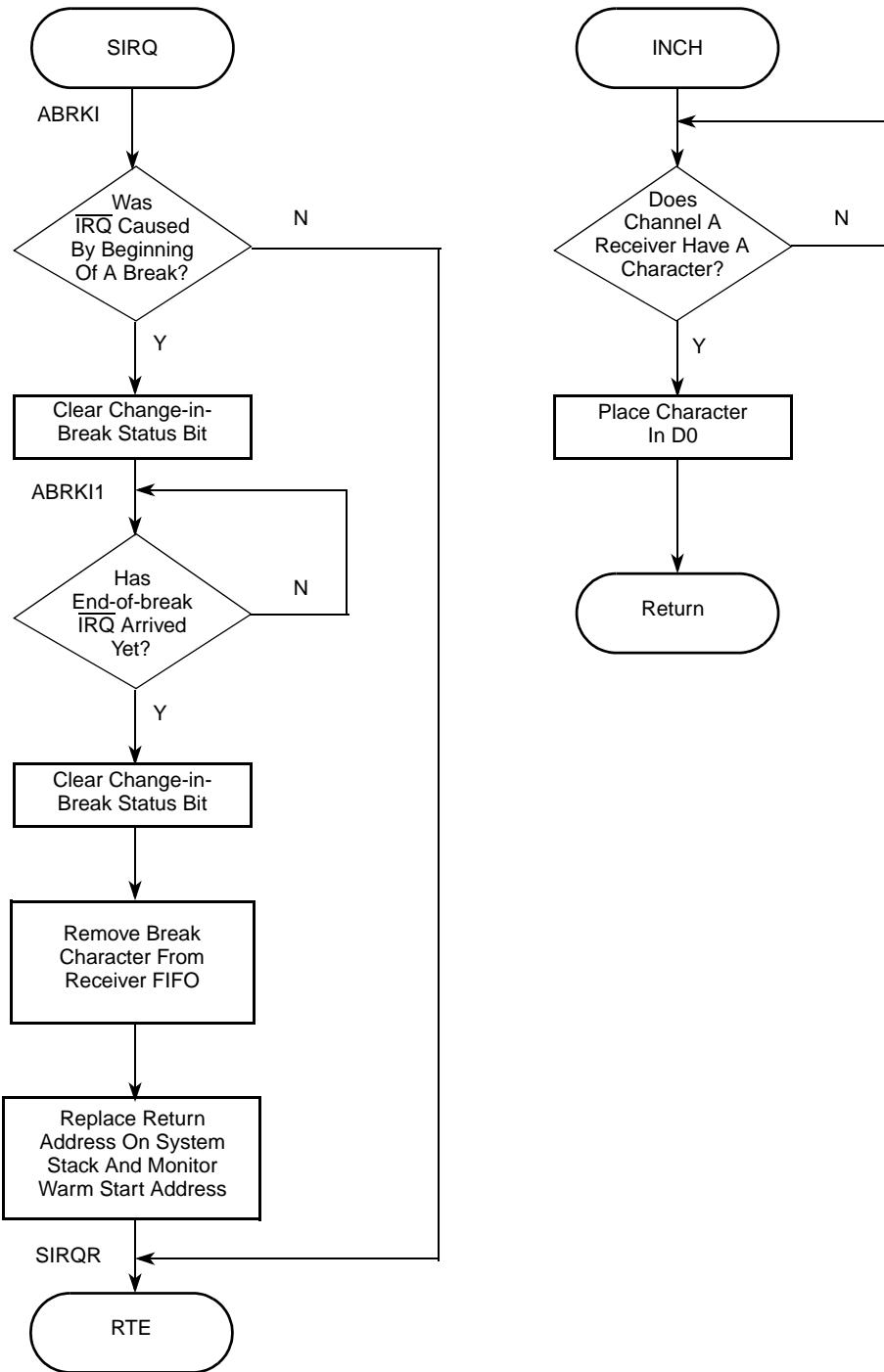


Figure 41-26. UART Mode Programming Flowchart (Sheet 4 of 5)

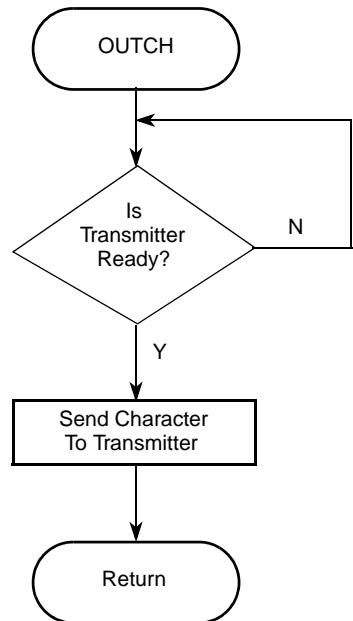


Figure 41-26. UART Mode Programming Flowchart (Sheet 5 of 5)



# Chapter 42

## I<sup>2</sup>C Interface

### 42.1 Introduction

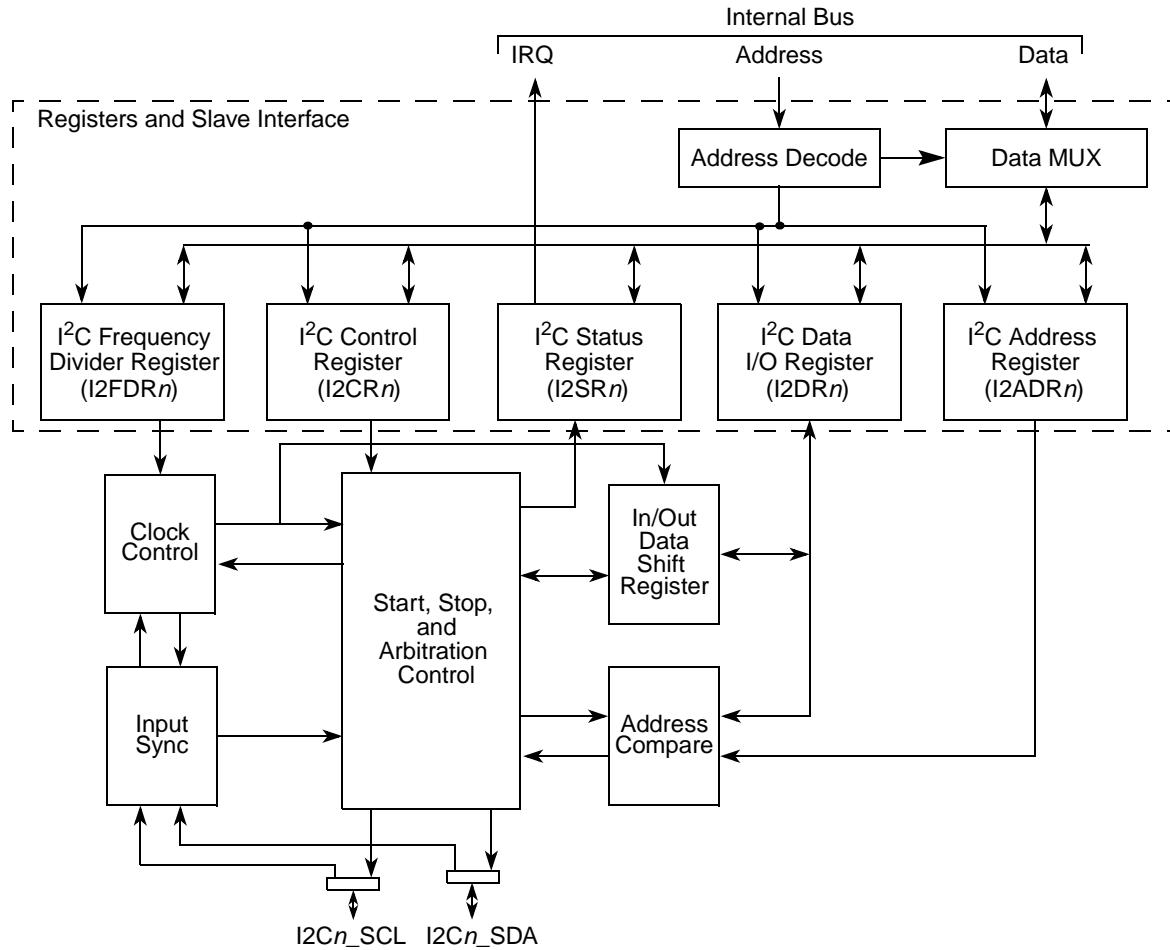
This chapter describes the I<sup>2</sup>C module, clock synchronization, and I<sup>2</sup>C programming model registers. It also provides extensive programming examples.

#### NOTE

This device contains six I<sup>2</sup>C modules, I<sup>2</sup>C0–5. The designation ‘n’, with  $n = 0\text{--}5$ , is used throughout this chapter to refer to registers and signals associated with the six identical I<sup>2</sup>C modules.

## 42.1.1 Block Diagram

Figure 42-1 is a I<sup>2</sup>C module block diagram, illustrating the interaction of the registers described in Section 42.2, “Memory Map/Register Definition”.



**Figure 42-1. I<sup>2</sup>C Module Block Diagram**

## 42.1.2 Overview

I<sup>2</sup>C is a two-wire, bidirectional serial bus that provides a simple, efficient method of data exchange, minimizing the interconnection between devices. This bus is suitable for applications that require occasional communication between many devices over a short distance. The flexible I<sup>2</sup>C bus allows additional devices to connect to the bus for expansion and system development.

The interface operates up to 100 Kbps with maximum bus loading and timing. The device is capable of operating at higher baud rates, up to a maximum of the internal bus clock divided by 20, with reduced bus loading. The maximum communication length and the number of devices connected are limited by a maximum bus capacitance of 400 pF.

The I<sup>2</sup>C system is a true multiple-master bus; it uses arbitration and collision detection to prevent data corruption in the event that multiple devices attempt to control the bus simultaneously. This feature

supports complex applications with multiprocessor control and can be used for rapid testing and alignment of end products through external connections to an assembly-line computer.

#### NOTE

The I<sup>2</sup>C module is compatible with the Philips I<sup>2</sup>C bus protocol. For information on system configuration, protocol, and restrictions, see *The I<sup>2</sup>C Bus Specification, Version 2.1*.

#### NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to [Chapter 15, “Pin-Multiplexing and Control”](#)) prior to configuring the I<sup>2</sup>C module.

### 42.1.3 Features

The I<sup>2</sup>C module has these key features:

- Compatibility with I<sup>2</sup>C bus standard version 2.1
- Multiple-master operation
- Software-programmable for one of 50 different serial clock frequencies
- Software-selectable acknowledge bit
- Interrupt-driven, byte-by-byte data transfer
- Arbitration-lost interrupt with automatic mode switching from master to slave
- Calling address identification interrupt
- START and STOP signal generation/detection
- Repeated START signal generation
- Acknowledge bit generation/detection
- Bus-busy detection

## 42.2 Memory Map/Register Definition

The below table lists the configuration registers used in the I<sup>2</sup>C interfaces.

**Table 42-1. I<sup>2</sup>C Module Memory Map**

Address	Register	Access	Reset Value	Section/Page
I <sup>2</sup> C0 I <sup>2</sup> C1 ... I <sup>2</sup> C5				
0xFC05_8000 0xFC03_8000 0xEC01_0000 0xEC01_4000 0xEC01_8000 0xEC01_C000	I <sup>2</sup> C Address Register (I2ADR $n$ )	R/W	0x00	<a href="#">42.2.1/42-5</a>
0xFC05_8004 0xFC03_8004 0xEC01_0004 0xEC01_4004 0xEC01_8004 0xEC01_C004	I <sup>2</sup> C Frequency Divider Register (I2FDR $n$ )	R/W	0x00	<a href="#">42.2.2/42-5</a>
0xFC05_8008 0xFC03_8008 0xEC01_0008 0xEC01_4008 0xEC01_8008 0xEC01_C008	I <sup>2</sup> C Control Register (I2CR $n$ )	R/W	0x00	<a href="#">42.2.3/42-6</a>
0xFC05_800C 0xFC03_800C 0xEC01_000C 0xEC01_400C 0xEC01_800C 0xEC01_C00C	I <sup>2</sup> C Status Register (I2SR $n$ )	R/W	0x81	<a href="#">42.2.4/42-7</a>
0xFC05_8010 0xFC03_8010 0xEC01_0010 0xEC01_4010 0xEC01_8010 0xEC01_C010 0xEC02_0010 0xEC02_4010	I <sup>2</sup> C Data I/O Register (I2DR $n$ )	R/W	0x00	<a href="#">42.2.5/42-8</a>

## 42.2.1 I<sup>2</sup>C Address Register (I2ADR $n$ )

I2ADR $n$  holds the address the I<sup>2</sup>C responds to when addressed as a slave. It is not the address sent on the bus during the address transfer when the module is performing a master transfer.

Address:	0xFC05_8000 (I2ADR0)	0xEC01_4000 (I2ADR3)	Access: User read/write			
	0xFC03_8000 (I2ADR1)	0xEC01_8000 (I2ADR4)				
	0xEC01_0000 (I2ADR2)	0xEC01_C000 (I2ADR5)				
	7      6      5      4	3      2      1      0				
R	ADR					
W						
Reset:	0      0      0      0	0      0      0      0	0      0      0      0			

Figure 42-2. I<sup>2</sup>C Address Register (I2ADR $n$ )

Table 42-2. I2ADR $n$  Field Descriptions

Field	Description
7–1 ADR	Slave address. Contains the specific slave address to be used by the I <sup>2</sup> C module. Slave mode is the default I <sup>2</sup> C mode for an address match on the bus.
0	Reserved, must be cleared.

## 42.2.2 I<sup>2</sup>C Frequency Divider Register (I2FDR $n$ )

The I2FDR $n$ , shown in Figure 42-3, provides a programmable prescaler to configure the I<sup>2</sup>C clock for bit-rate selection.

Address:	0xFC05_8004 (I2FDR0)	0xEC01_4004 (I2FDR3)	Access: User read/write	
	0xFC03_8004 (I2FDR1)	0xEC01_8004 (I2FDR4)		
	0xEC01_0004 (I2FDR2)	0xEC01_C004 (I2FDR5)		
	7      6      5      4	3      2      1      0		
R	0      0	IC		
W				
Reset:	0      0      0      0	0      0      0      0	0      0      0      0	

Figure 42-3. I<sup>2</sup>C Frequency Divider Register (I2FDR $n$ )

**Table 42-3. I2FDRn Field Descriptions**

Field	Description																																																																																																																																									
7–6	Reserved, must be cleared.																																																																																																																																									
5–0 IC	I <sup>2</sup> C clock rate. Prescales the clock for bit-rate selection. The serial bit clock frequency is equal to the internal bus clock divided by the divider shown below. Due to potentially slow I2Cn_SCL and I2Cn_SDA rise and fall times, bus signals are sampled at the prescaler frequency.	<table border="1"> <thead> <tr> <th>IC</th> <th>Divider</th> </tr> </thead> <tbody> <tr><td>0x00</td><td>28</td></tr> <tr><td>0x01</td><td>30</td></tr> <tr><td>0x02</td><td>34</td></tr> <tr><td>0x03</td><td>40</td></tr> <tr><td>0x04</td><td>44</td></tr> <tr><td>0x05</td><td>48</td></tr> <tr><td>0x06</td><td>56</td></tr> <tr><td>0x07</td><td>68</td></tr> <tr><td>0x08</td><td>80</td></tr> <tr><td>0x09</td><td>88</td></tr> <tr><td>0x0A</td><td>104</td></tr> <tr><td>0x0B</td><td>128</td></tr> <tr><td>0x0C</td><td>144</td></tr> <tr><td>0x0D</td><td>160</td></tr> <tr><td>0x0E</td><td>192</td></tr> <tr><td>0x0F</td><td>240</td></tr> </tbody> </table> <table border="1"> <thead> <tr> <th>IC</th> <th>Divider</th> </tr> </thead> <tbody> <tr><td>0x10</td><td>288</td></tr> <tr><td>0x11</td><td>320</td></tr> <tr><td>0x12</td><td>384</td></tr> <tr><td>0x13</td><td>480</td></tr> <tr><td>0x14</td><td>576</td></tr> <tr><td>0x15</td><td>640</td></tr> <tr><td>0x16</td><td>768</td></tr> <tr><td>0x17</td><td>960</td></tr> <tr><td>0x18</td><td>1152</td></tr> <tr><td>0x19</td><td>1280</td></tr> <tr><td>0x1A</td><td>1536</td></tr> <tr><td>0x1B</td><td>1920</td></tr> <tr><td>0x1C</td><td>2304</td></tr> <tr><td>0x1D</td><td>2560</td></tr> <tr><td>0x1E</td><td>3072</td></tr> <tr><td>0x1F</td><td>3840</td></tr> </tbody> </table> <table border="1"> <thead> <tr> <th>IC</th> <th>Divider</th> </tr> </thead> <tbody> <tr><td>0x20</td><td>20</td></tr> <tr><td>0x21</td><td>22</td></tr> <tr><td>0x22</td><td>24</td></tr> <tr><td>0x23</td><td>26</td></tr> <tr><td>0x24</td><td>28</td></tr> <tr><td>0x25</td><td>32</td></tr> <tr><td>0x26</td><td>36</td></tr> <tr><td>0x27</td><td>40</td></tr> <tr><td>0x28</td><td>48</td></tr> <tr><td>0x29</td><td>56</td></tr> <tr><td>0x2A</td><td>64</td></tr> <tr><td>0x2B</td><td>72</td></tr> <tr><td>0x2C</td><td>80</td></tr> <tr><td>0x2D</td><td>96</td></tr> <tr><td>0x2E</td><td>112</td></tr> <tr><td>0x2F</td><td>128</td></tr> </tbody> </table> <table border="1"> <thead> <tr> <th>IC</th> <th>Divider</th> </tr> </thead> <tbody> <tr><td>0x30</td><td>160</td></tr> <tr><td>0x31</td><td>192</td></tr> <tr><td>0x32</td><td>224</td></tr> <tr><td>0x33</td><td>256</td></tr> <tr><td>0x34</td><td>320</td></tr> <tr><td>0x35</td><td>384</td></tr> <tr><td>0x36</td><td>448</td></tr> <tr><td>0x37</td><td>512</td></tr> <tr><td>0x38</td><td>640</td></tr> <tr><td>0x39</td><td>768</td></tr> <tr><td>0x3A</td><td>896</td></tr> <tr><td>0x3B</td><td>1024</td></tr> <tr><td>0x3C</td><td>1280</td></tr> <tr><td>0x3D</td><td>1536</td></tr> <tr><td>0x3E</td><td>1792</td></tr> <tr><td>0x3F</td><td>2048</td></tr> </tbody> </table>	IC	Divider	0x00	28	0x01	30	0x02	34	0x03	40	0x04	44	0x05	48	0x06	56	0x07	68	0x08	80	0x09	88	0x0A	104	0x0B	128	0x0C	144	0x0D	160	0x0E	192	0x0F	240	IC	Divider	0x10	288	0x11	320	0x12	384	0x13	480	0x14	576	0x15	640	0x16	768	0x17	960	0x18	1152	0x19	1280	0x1A	1536	0x1B	1920	0x1C	2304	0x1D	2560	0x1E	3072	0x1F	3840	IC	Divider	0x20	20	0x21	22	0x22	24	0x23	26	0x24	28	0x25	32	0x26	36	0x27	40	0x28	48	0x29	56	0x2A	64	0x2B	72	0x2C	80	0x2D	96	0x2E	112	0x2F	128	IC	Divider	0x30	160	0x31	192	0x32	224	0x33	256	0x34	320	0x35	384	0x36	448	0x37	512	0x38	640	0x39	768	0x3A	896	0x3B	1024	0x3C	1280	0x3D	1536	0x3E	1792	0x3F	2048
IC	Divider																																																																																																																																									
0x00	28																																																																																																																																									
0x01	30																																																																																																																																									
0x02	34																																																																																																																																									
0x03	40																																																																																																																																									
0x04	44																																																																																																																																									
0x05	48																																																																																																																																									
0x06	56																																																																																																																																									
0x07	68																																																																																																																																									
0x08	80																																																																																																																																									
0x09	88																																																																																																																																									
0x0A	104																																																																																																																																									
0x0B	128																																																																																																																																									
0x0C	144																																																																																																																																									
0x0D	160																																																																																																																																									
0x0E	192																																																																																																																																									
0x0F	240																																																																																																																																									
IC	Divider																																																																																																																																									
0x10	288																																																																																																																																									
0x11	320																																																																																																																																									
0x12	384																																																																																																																																									
0x13	480																																																																																																																																									
0x14	576																																																																																																																																									
0x15	640																																																																																																																																									
0x16	768																																																																																																																																									
0x17	960																																																																																																																																									
0x18	1152																																																																																																																																									
0x19	1280																																																																																																																																									
0x1A	1536																																																																																																																																									
0x1B	1920																																																																																																																																									
0x1C	2304																																																																																																																																									
0x1D	2560																																																																																																																																									
0x1E	3072																																																																																																																																									
0x1F	3840																																																																																																																																									
IC	Divider																																																																																																																																									
0x20	20																																																																																																																																									
0x21	22																																																																																																																																									
0x22	24																																																																																																																																									
0x23	26																																																																																																																																									
0x24	28																																																																																																																																									
0x25	32																																																																																																																																									
0x26	36																																																																																																																																									
0x27	40																																																																																																																																									
0x28	48																																																																																																																																									
0x29	56																																																																																																																																									
0x2A	64																																																																																																																																									
0x2B	72																																																																																																																																									
0x2C	80																																																																																																																																									
0x2D	96																																																																																																																																									
0x2E	112																																																																																																																																									
0x2F	128																																																																																																																																									
IC	Divider																																																																																																																																									
0x30	160																																																																																																																																									
0x31	192																																																																																																																																									
0x32	224																																																																																																																																									
0x33	256																																																																																																																																									
0x34	320																																																																																																																																									
0x35	384																																																																																																																																									
0x36	448																																																																																																																																									
0x37	512																																																																																																																																									
0x38	640																																																																																																																																									
0x39	768																																																																																																																																									
0x3A	896																																																																																																																																									
0x3B	1024																																																																																																																																									
0x3C	1280																																																																																																																																									
0x3D	1536																																																																																																																																									
0x3E	1792																																																																																																																																									
0x3F	2048																																																																																																																																									

### 42.2.3 I<sup>2</sup>C Control Register (I2CRn)

I2CRn enables the I<sup>2</sup>C module and the I<sup>2</sup>C interrupt. It also contains bits that govern operation as a slave or a master.

Address:	0xFC05_8008 (I2CR0) 0xFC03_8008 (I2CR1) 0xEC01_0008 (I2CR2)	0xEC01_4008 (I2CR3) 0xEC01_8008 (I2CR4) 0xEC01_C008 (I2CR5)	Access:	User read/write																		
	<table border="1"> <tr> <td>R</td> <td>IEN</td> <td>IIEN</td> <td>MSTA</td> <td>MTX</td> <td>TXAK</td> <td>RSTA</td> <td>0</td> <td>DMAEN</td> </tr> <tr> <td>W</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>	R	IEN	IIEN	MSTA	MTX	TXAK	RSTA	0	DMAEN	W	0	0	0	0	0	0	0	0			
R	IEN	IIEN	MSTA	MTX	TXAK	RSTA	0	DMAEN														
W	0	0	0	0	0	0	0	0														

**Figure 42-4. I<sup>2</sup>C Control Register (I2CRn)**

**Table 42-4. I<sup>2</sup>CRn Field Descriptions**

Field	Description
7 IEN	I <sup>2</sup> C enable. Controls the software reset of the entire I <sup>2</sup> C module. If the module is enabled in the middle of a byte transfer, slave mode ignores the current bus transfer and starts operating when the next START condition is detected. Master mode is not aware that the bus is busy; initiating a start cycle may corrupt the current bus cycle, ultimately causing the current master or the I <sup>2</sup> C module to lose arbitration, after which bus operation returns to normal. 0 The I <sup>2</sup> C module is disabled, but registers can be accessed. 1 The I <sup>2</sup> C module is enabled. This bit must be set before any other I <sup>2</sup> CRn bits have any effect.
6 IIEN	I <sup>2</sup> C interrupt enable. If DMAEN is cleared, then this bit enables an interrupt request. 0 I <sup>2</sup> C module interrupts are disabled, but currently pending interrupt condition is not cleared. 1 If DMAEN is cleared, I <sup>2</sup> C module interrupts are enabled. An I <sup>2</sup> C interrupt occurs if I <sup>2</sup> SRn[IIF] is also set. If DMAEN is set, a DMA request is generated instead.
5 MSTA	Master/slave mode select bit. If the master loses arbitration, MSTA is cleared without generating a STOP signal. 0 Slave mode. Changing MSTA from 1 to 0 generates a STOP and selects slave mode. 1 Master mode. Changing MSTA from 0 to 1 signals a START on the bus and selects master mode.
4 MTX	Transmit/receive mode select bit. Selects the direction of master and slave transfers. 0 Receive 1 Transmit. When the device is addressed as a slave, software must set MTX according to I <sup>2</sup> SRn[SRW]. In master mode, MTX must be set according to the type of transfer required. Therefore, when the MCU addresses a slave device, MTX is always 1.
3 TXAK	Transmit acknowledge enable. Specifies the value driven onto I <sup>2</sup> Cn_SDA during acknowledge cycles for master and slave receivers. Writing TXAK applies only when the I <sup>2</sup> C bus is a receiver. 0 An acknowledge signal is sent to the bus at the ninth clock bit after receiving one byte of data. 1 No acknowledge signal response is sent (acknowledge bit = 1).
2 RSTA	Repeat start. Always read as 0. Attempting a repeat start without bus mastership causes loss of arbitration. 0 No repeat start 1 Generates a repeated START condition.
1	Reserved, must be cleared.
0 DMAEN	DMA request enable. Enables the I <sup>2</sup> C DMA request if I <sup>2</sup> CR[IIEN] is set. When a single-byte master-mode transfer completes a DMA request is sent to the DMA controller. 0 Disable DMA request. Interrupt requests are used instead, if IIEN is set. 1 If IIEN is set, enable DMA request

#### 42.2.4 I<sup>2</sup>C Status Register (I<sup>2</sup>SRn)

I<sup>2</sup>SRn contains bits that indicate transaction direction and status.

Address: 0xFC05_800C (I <sup>2</sup> SR0)	0xEC01_400C (I <sup>2</sup> SR3)	Access: User read/write																																																		
0xFC03_800C (I <sup>2</sup> SR1)	0xEC01_800C (I <sup>2</sup> SR4)																																																			
0xEC01_000C (I <sup>2</sup> SR2)	0xEC01_C00C (I <sup>2</sup> SR5)																																																			
<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr> <td>R</td><td>ICF</td><td>IAAS</td><td>IBB</td><td rowspan="2">IAL</td><td>0</td><td>SRW</td><td rowspan="2">IIF</td><td rowspan="2">RXAK</td></tr> <tr> <td>W</td><td></td><td></td><td></td><td></td><td>0</td><td>0</td><td></td><td></td></tr> </table>									R	ICF	IAAS	IBB	IAL	0	SRW	IIF	RXAK	W					0	0			<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr> <td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> <tr> <td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr> </table>																									
R	ICF	IAAS	IBB	IAL	0	SRW	IIF	RXAK																																												
W						0			0																																											
Reset:	1	0	0	0	0	0	0	1																																												

**Figure 42-5. I<sup>2</sup>C Status Register (I<sup>2</sup>SRn)**

**Table 42-5. I2SRn Field Descriptions**

Field	Description
7 ICF	I <sup>2</sup> C Data transferring bit. While one byte of data is transferred, ICF is cleared. 0 Transfer in progress 1 Transfer complete. Set by falling edge of ninth clock of a byte transfer.
6 IAAS	I <sup>2</sup> C addressed as a slave bit. The CPU is interrupted if I2CRn[IIEN] is set and I2CR[DMAEN] is cleared. Next, the CPU must check SRW and set its TX/RX mode accordingly. Writing to I2CRn clears this bit. 0 Not addressed. 1 Addressed as a slave. Set when its own address (IADR) matches the calling address.
5 IBB	I <sup>2</sup> C bus busy bit. Indicates the status of the bus. 0 Bus is idle. If a STOP signal is detected, IBB is cleared. 1 Bus is busy. When START is detected, IBB is set.
4 IAL	I <sup>2</sup> C arbitration lost. Set by hardware in the following circumstances. (IAL must be cleared by software by writing zero to it.) <ul style="list-style-type: none"><li>• I2Cn_SDA sampled low when the master drives high during an address or data-transmit cycle.</li><li>• I2Cn_SDA sampled low when the master drives high during the acknowledge bit of a data-receive cycle.</li><li>• A start cycle is attempted when the bus is busy.</li><li>• A repeated start cycle is requested in slave mode.</li><li>• A stop condition is detected when the master did not request it.</li></ul>
3	Reserved, must be cleared.
2 SRW	Slave read/write. When IAAS is set, SRW indicates the value of the R/W command bit of the calling address sent from the master. SRW is valid only when a complete transfer has occurred, no other transfers have been initiated, and the I <sup>2</sup> C module is a slave and has an address match. 0 Slave receive, master writing to slave. 1 Slave transmit, master reading from slave.
1 IIF	I <sup>2</sup> C interrupt. Must be cleared by software by writing a 0 in the interrupt routine. 0 No I <sup>2</sup> C interrupt pending 1 An interrupt is pending, which causes a processor interrupt request (if IIEN = 1 and DMAEN = 0). Set when one of the following occurs: <ul style="list-style-type: none"><li>• Complete one byte transfer (set at the falling edge of the ninth clock)</li><li>• Reception of a calling address that matches its own specific address in slave-receive mode</li><li>• Arbitration lost</li></ul>
0 RXAK	Received acknowledge. The value of I2Cn_SDA during the acknowledge bit of a bus cycle. 0 An acknowledge signal was received after the completion of 8-bit data transmission on the bus 1 No acknowledge signal was detected at the ninth clock.

### 42.2.5 I<sup>2</sup>C Data I/O Register (I2DRn)

In master-receive mode, reading I2DRn allows a read to occur and for the next data byte to be received. In slave mode, the same function is available after the I<sup>2</sup>C has received its slave address.

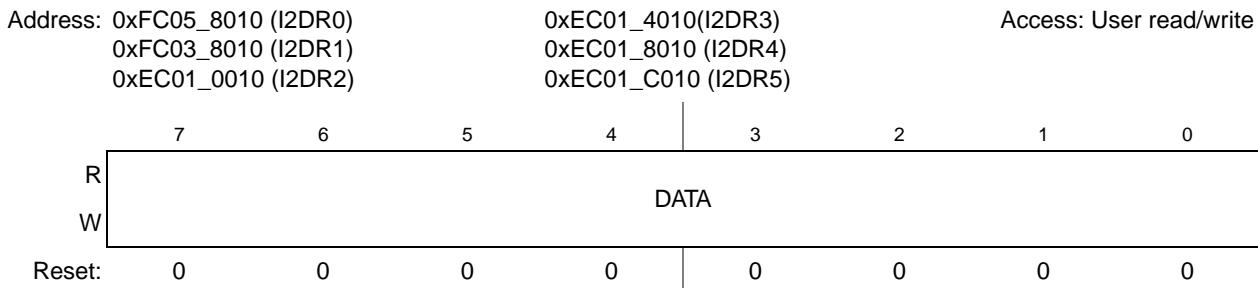
Figure 42-6. I<sup>2</sup>C Data I/O Register (I2DRn)

Table 42-6. I2DRn Field Description

Field	Description
7–0 DATA	<p>I<sup>2</sup>C data. When data is written to this register in master transmit mode, a data transfer is initiated. The most significant bit is sent first. In master receive mode, reading this register initiates the reception of the next byte of data. In slave mode, the same functions are available after an address match has occurred.</p> <p><b>Note:</b> In master transmit mode, the first byte of data written to I2DRn following assertion of I2CRn[MSTA] is used for the address transfer and should comprise the calling address (in position D7–D1) concatenated with the required R/W bit (in position D0). This bit (D0) is not automatically appended by the hardware, software must provide the appropriate R/W bit.</p> <p><b>Note:</b> I2CRn[MSTA] generates a start when a master does not already own the bus. I2CRn[RSTA] generates a start (restart) without the master first issuing a stop (i.e., the master already owns the bus). To start the read of data, a dummy read to this register starts the read process from the slave. The next read of the I2DRn register contains the actual data.</p>

## 42.3 Functional Description

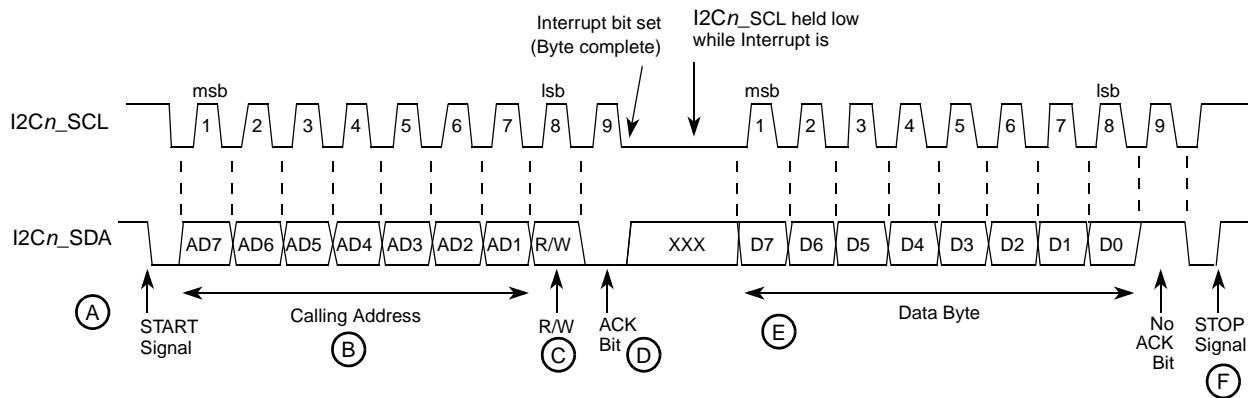
The I<sup>2</sup>C module uses a serial data line (I2C<sub>n</sub>\_SDA) and a serial clock line (I2C<sub>n</sub>\_SCL) for data transfer. For I<sup>2</sup>C compliance, all devices connected to these two signals must have open drain or open collector outputs. The logic AND function is exercised on both lines with external pull-up resistors.

Out of reset, the I<sup>2</sup>C default state is as a slave receiver. Therefore, when not programmed to be a master or responding to a slave transmit address, the I<sup>2</sup>C module should return to the default slave receiver state. See [Section 42.4.1, “Initialization Sequence,”](#) for exceptions.

Normally, a standard communication is composed of four parts: START signal, slave address transmission, data transfer, and STOP signal. These are discussed in the following sections.

### 42.3.1 START Signal

When no other device is bus master (I2C<sub>n</sub>\_SCL and I2C<sub>n</sub>\_SDA lines are at logic high), a device can initiate communication by sending a START signal (see A in [Figure 42-7](#)). A START signal is defined as a high-to-low transition of I2C<sub>n</sub>\_SDA while I2C<sub>n</sub>\_SCL is high. This signal denotes the beginning of a data transfer (each data transfer can be several bytes long) and awakens all slaves.

Figure 42-7. I<sup>2</sup>C Standard Communication Protocol

### 42.3.2 Slave Address Transmission

The master sends the slave address in the first byte after the START signal (B). After the seven-bit calling address, it sends the R/W bit (C), which tells the slave data transfer direction (0 equals write transfer, 1 equals read transfer).

Each slave must have a unique address. An I<sup>2</sup>C master must not transmit its own slave address; it cannot be master and slave at the same time.

The slave whose address matches that sent by the master pulls I<sup>2</sup>Cn\_SDA low at the ninth serial clock (D) to return an acknowledge bit.

### 42.3.3 Data Transfer

When successful slave addressing is achieved, data transfer can proceed (see E in Figure 42-7) on a byte-by-byte basis in the direction specified by the R/W bit sent by the calling master.

Data can be changed only while I<sup>2</sup>Cn\_SCL is low and must be held stable while I<sup>2</sup>Cn\_SCL is high, as Figure 42-7 shows. I<sup>2</sup>Cn\_SCL is pulsed once for each data bit, with the msb being sent first. The receiving device must acknowledge each byte by pulling I<sup>2</sup>Cn\_SDA low at the ninth clock; therefore, a data byte transfer takes nine clock pulses. See Figure 42-8.

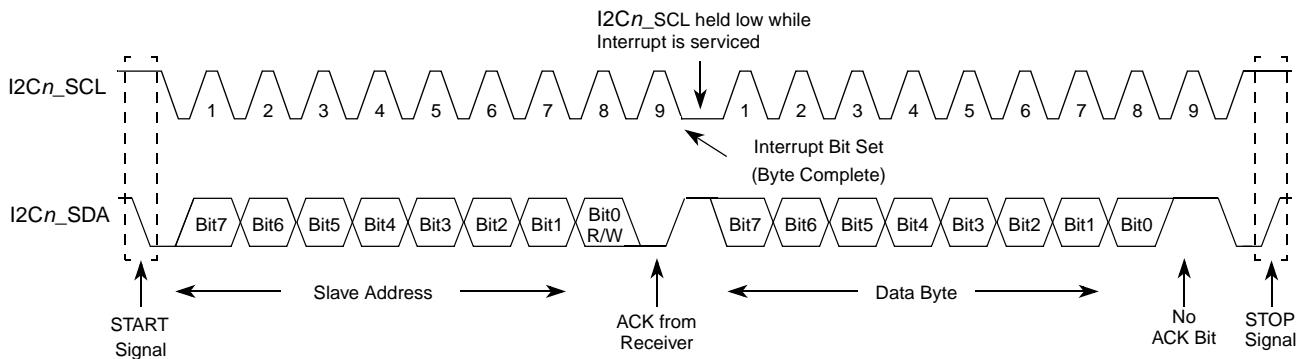
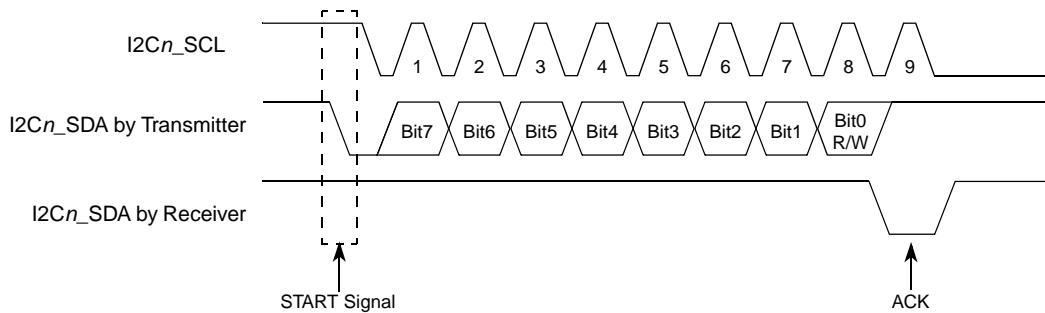


Figure 42-8. Data Transfer

### 42.3.4 Acknowledge

The transmitter releases the I<sub>2</sub>C<sub>n</sub>\_SDA line high during the acknowledge clock pulse as shown in Figure 42-9. The receiver pulls down the I<sub>2</sub>C<sub>n</sub>\_SDA line during the acknowledge clock pulse so that it remains stable low during the high period of the clock pulse.

If it does not acknowledge the master, the slave receiver must leave I<sub>2</sub>C<sub>n</sub>\_SDA high. The master can then generate a STOP signal to abort data transfer or generate a START signal (repeated start, shown in Figure 42-10 and discussed in Section 42.3.6, “Repeated START”) to start a new calling sequence.



**Figure 42-9. Acknowledgement by Receiver**

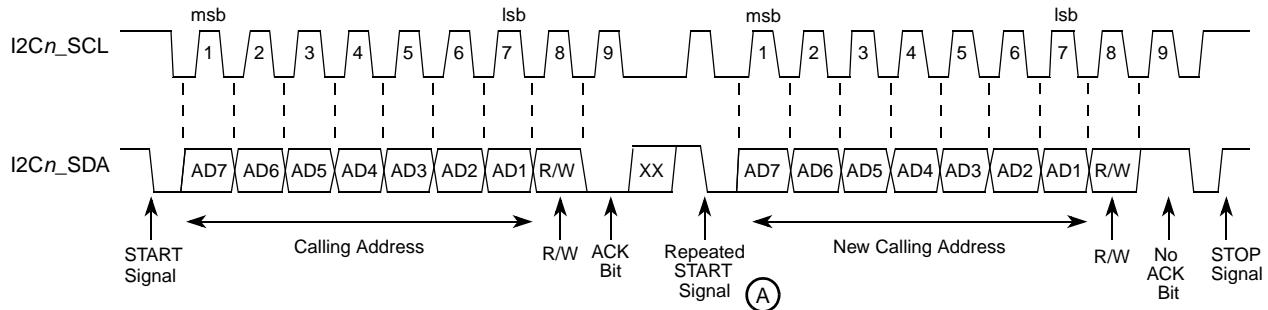
If the master receiver does not acknowledge the slave transmitter after a byte transmission, it means end-of-data to the slave. The slave releases I<sub>2</sub>C<sub>n</sub>\_SDA for the master to generate a STOP or START signal (Figure 42-9).

### 42.3.5 STOP Signal

The master can terminate communication by generating a STOP signal to free the bus. A STOP signal is defined as a low-to-high transition of I<sub>2</sub>C<sub>n</sub>\_SDA while I<sub>2</sub>C<sub>n</sub>\_SCL is at logical high (see F in Figure 42-7). The master can generate a STOP even if the slave has generated an acknowledgment, at which point the slave must release the bus. The master may also generate a START signal following a calling address, without first generating a STOP signal. Refer to Section 42.3.6, “Repeated START.”

### 42.3.6 Repeated START

A repeated START signal is a START signal generated without first generating a STOP signal to terminate the communication, as shown in Figure 42-10. The master uses a repeated START to communicate with another slave or with the same slave in a different mode (transmit/receive mode) without releasing the bus.



**Figure 42-10. Repeated START**

Various combinations of read/write formats are then possible:

- The first example in [Figure 42-11](#) is the case of master-transmitter transmitting to slave-receiver. The transfer direction is not changed.
  - The second example in [Figure 42-11](#) is the master reading the slave immediately after the first byte. At the moment of the first acknowledge, the master-transmitter becomes a master-receiver and the slave-receiver becomes slave-transmitter.
  - In the third example in [Figure 42-11](#), START condition and slave address are repeated using the repeated START signal. This is to communicate with same slave in a different mode without releasing the bus. The master transmits data to the slave first, and then the master reads data from slave by reversing the R/W bit.

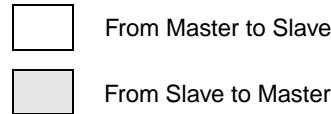
ST = Start

SP = Stop

A = Acknowledge (I2Cn\_SDA low)

$\overline{A}$  = Not Acknowledged (I<sub>2</sub>Cn SDA high)

Rept ST = Repeated Start



### Example 1:

R/W

ST	7bit Slave Address	0	A	Data	A	Data	$A/\bar{A}$	SP
----	--------------------	---	---	------	---	------	-------------	----

### Example 2:

R/W

ST	7bit Slave Address	1	A	Data	A	Data	$\bar{A}$	SP
----	--------------------	---	---	------	---	------	-----------	----

**Note:** No acknowledgement on the last byte

### Example 3:

R/W

Master Reads from Slave

Master Writes to Slave

**Figure 42-11. Data Transfer, Combined Format**

### 42.3.7 Clock Synchronization and Arbitration

I<sup>2</sup>C is a true multi-master bus that allows more than one master connected to it. If two or more master devices simultaneously request control of the bus, a clock synchronization procedure determines the bus clock. Because wire-AND logic is performed on the I<sub>2</sub>C<sub>n</sub>\_SCL line, a high-to-low transition on the I<sub>2</sub>C<sub>n</sub>\_SCL line affects all the devices connected on the bus. The devices start counting their low period and after a device's clock has gone low, it holds the I<sub>2</sub>C<sub>n</sub>\_SCL line low until the clock high state is reached. However, change of low to high in this device's clock may not change the state of the I<sub>2</sub>C<sub>n</sub>\_SCL line if another device clock remains within its low period. Therefore, synchronized clock I<sub>2</sub>C<sub>n</sub>\_SCL is held low by the device with the longest low period.

Devices with shorter low periods enter a high wait state during this time (see Figure 42-12). When all devices concerned have counted off their low period, the synchronized clock (I<sub>2</sub>C<sub>n</sub>\_SCL) line is released and pulled high. At this point, the device clocks and the I<sub>2</sub>C<sub>n</sub>\_SCL line are synchronized, and the devices start counting their high periods. The first device to complete its high period pulls the I<sub>2</sub>C<sub>n</sub>\_SCL line low again.

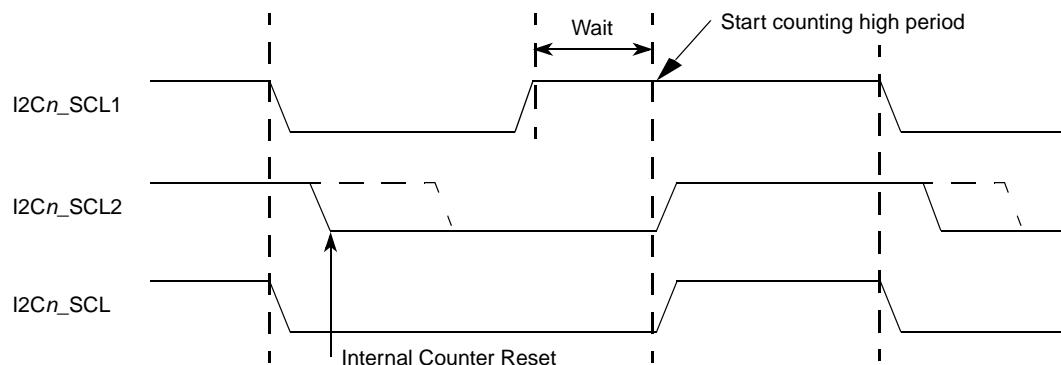


Figure 42-12. Clock Synchronization

A data arbitration procedure determines the relative priority of the contending masters. A bus master loses arbitration if it transmits logic 1 while another master transmits logic 0. The losing masters immediately switch over to slave receive mode and stop driving I<sub>2</sub>C<sub>n</sub>\_SDA output (see Figure 42-13). In this case, transition from master to slave mode does not generate a STOP condition. Meanwhile, hardware sets I<sub>2</sub>SR<sub>n</sub>[IAL] to indicate loss of arbitration.

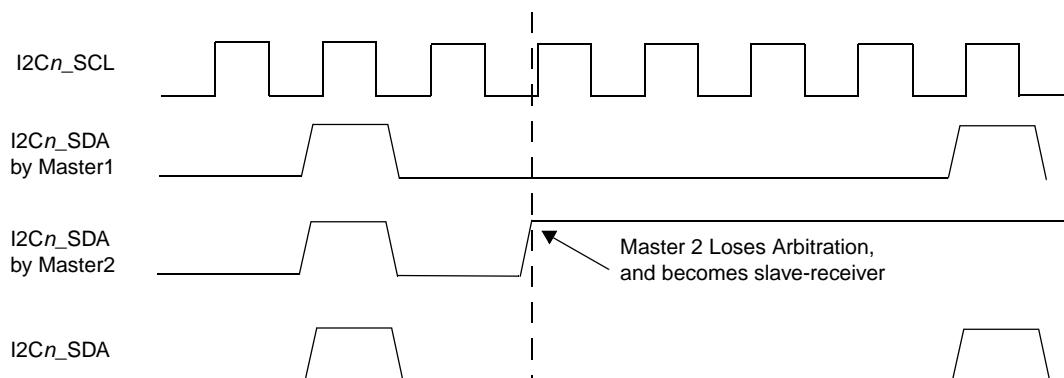


Figure 42-13. Arbitration Procedure

### 42.3.8 Handshaking and Clock Stretching

The clock synchronization mechanism can act as a handshake in data transfers. Slave devices can hold I<sup>2</sup>C<sub>n</sub>\_SCL low after completing one byte transfer. In such a case, the clock mechanism halts the bus clock and forces the master clock into wait states until the slave releases I<sup>2</sup>C<sub>n</sub>\_SCL.

Slaves may also slow down the transfer bit rate. After the master has driven I<sup>2</sup>C<sub>n</sub>\_SCL low, the slave can drive I<sup>2</sup>C<sub>n</sub>\_SCL low for the required period and then release it. If the slave I<sup>2</sup>C<sub>n</sub>\_SCL low period is longer than the master I<sup>2</sup>C<sub>n</sub>\_SCL low period, the resulting I<sup>2</sup>C<sub>n</sub>\_SCL bus signal low period is stretched.

## 42.4 Initialization/Application Information

The following examples show programming for initialization, signaling START, post-transfer software response, signaling STOP, and generating a repeated START.

### 42.4.1 Initialization Sequence

Before the interface can transfer serial data, registers must be initialized:

1. Set I<sup>2</sup>FDR<sub>n</sub>[IC] to obtain I<sup>2</sup>C<sub>n</sub>\_SCL frequency from the system bus clock. See [Section 42.2.2, “I<sup>2</sup>C Frequency Divider Register \(I<sup>2</sup>FDR<sub>n</sub>\).”](#)
2. Update the I<sup>2</sup>ADR<sub>n</sub> to define its slave address.
3. Set I<sup>2</sup>CR<sub>n</sub>[IEN] to enable the I<sup>2</sup>C bus interface system.
4. Modify the I<sup>2</sup>CR<sub>n</sub> to select or deselect master/slave mode, transmit/receive mode, and interrupt/DMA-enable or not.

#### NOTE

If I<sup>2</sup>SR<sub>n</sub>[IBB] is set when the I<sup>2</sup>C bus module is enabled, execute the following pseudocode sequence before proceeding with normal initialization code. This issues a STOP command to the slave device, placing it in idle state as if it were power-cycled on.

```
I2CRn = 0x0
I2CRn = 0xA0
dummy read of I2DRn
I2SRn = 0x0
I2CRn = 0x0
I2CRn = 0x80      ; re-enable
```

### 42.4.2 Generation of START

After completion of the initialization procedure, serial data can be transmitted by selecting the master transmitter mode. On a multiple-master bus system, I<sup>2</sup>SR<sub>n</sub>[IBB] must be tested to determine whether the serial bus is free. If the bus is free (IBB is cleared), the START signal and the first byte (the slave address) can be sent. The data written to the data register comprises the address of the desired slave and the lsb indicates the transfer direction.

The free time between a STOP and the next START condition is built into the hardware that generates the START cycle. Depending on the relative frequencies of the system clock and the I<sup>2</sup>C<sub>n</sub>\_SCL period, the

processor may need to wait until the I<sup>2</sup>C is busy after writing the calling address to the I2DR<sub>n</sub> before proceeding with the following instructions.

The following example signals START and transmits the first byte of data (slave address):

1. Check I2SR<sub>n</sub>[IBB]. If it is set, wait until it is clear.
2. After cleared, set to transmit mode by setting I2CR<sub>n</sub>[MTX].
3. Set master mode by setting I2CR<sub>n</sub>[MSTA]. This generates a START condition.
4. Transmit the calling address via the I2DR<sub>n</sub>.
5. Check I2SR<sub>n</sub>[IBB]. If it is clear, wait until it is set and go to step #1.

#### 42.4.3 Post-Transfer Software Response

Sending or receiving a byte sets the I2SR<sub>n</sub>[ICF], which indicates one byte communication is finished. I2SR<sub>n</sub>[IIF] is also set. An interrupt is generated if the interrupt function is enabled during initialization by setting I2CR<sub>n</sub>[IEN] and clearing I2CR[DMAEN]. Software must first clear I2SR<sub>n</sub>[IIF] in the interrupt routine. Reading from I2DR<sub>n</sub> in receive mode or writing to I2DR<sub>n</sub> in transmit mode can clear I2SR<sub>n</sub>[ICF].

Software can service the I<sup>2</sup>C I/O in the main program by monitoring the IIF bit if the interrupt function is disabled. Polling should monitor IIF rather than ICF, because that operation is different when arbitration is lost.

When an interrupt occurs at the end of the address cycle, the master is always in transmit mode; the address is sent. If master receive mode is required, I2CR<sub>n</sub>[MTX] should be toggled.

During slave-mode address cycles (I2SR<sub>n</sub>[IAAS] = 1), I2SR<sub>n</sub>[SRW] is read to determine the direction of the next transfer. MTX is programmed accordingly. For slave-mode data cycles (IAAS = 0), SRW is invalid. MTX should be read to determine the current transfer direction.

The following is an example of a software response by a master transmitter in the interrupt routine (see [Figure 42-14](#)).

1. Clear the I2CR<sub>n</sub>[IIF] flag.
2. Check if acknowledge has been received, I2SR<sub>n</sub>[RXAK].
3. If no ACK, end transmission. Else, transmit next byte of data via I2DR<sub>n</sub>.

#### 42.4.4 Generation of STOP

A data transfer ends when the master signals a STOP, which can occur after all data is sent, as in the following example.

1. Check if acknowledge has been received, I2SR<sub>n</sub>[RXAK]. If no ACK, end transmission and go to step #5.
2. Get value from transmitting counter, TXCNT. If no more data, go to step #5.
3. Transmit next byte of data via I2DR<sub>n</sub>.
4. Decrement TXCNT and go to step #1
5. Generate a stop condition by clearing I2CR<sub>n</sub>[MSTA].

For a master receiver to terminate a data transfer, it must inform the slave transmitter by not acknowledging the last data byte. This is done by setting I2CR $n$ [TXAK] before reading the next-to-last byte. Before the last byte is read, a STOP signal must be generated, as in the following example.

1. Decrement RXCNT.
2. If last byte (RXCNT = 0) go to step #4.
3. If next to last byte (RXCNT = 1), set I2CR $n$ [TXAK] to disable ACK and go to step #5.
4. This is last byte, so clear I2CR $n$ [MSTA] to generate a STOP signal.
5. Read data from I2DR $n$ .
6. If there is more data to be read (RXCNT ≠ 0), go to step #1 if desired.

#### 42.4.5 Generation of Repeated START

If the master wants the bus after the data transfer, it can signal another START followed by another slave address without signaling a STOP, as in the following example.

1. Generate a repeated START by setting I2CR $n$ [RSTA].
2. Transmit the calling address via I2DR $n$ .

#### 42.4.6 Slave Mode

In the slave interrupt service routine, software must poll the I2SR $n$ [IAAS] bit to determine if the controller has received its slave address. If IAAS is set, software must set the transmit/receive mode select bit (I2CR $n$ [MTX]) according to the I2SR $n$ [SRW]. Writing to I2CR $n$  clears IAAS automatically. The only time IAAS is read as set is from the interrupt at the end of the address cycle where an address match occurred; interrupts resulting from subsequent data transfers have IAAS cleared. A data transfer can now be initiated by writing information to I2DR $n$  for slave transmits, or read from I2DR $n$  in slave-receive mode. A dummy read of I2DR $n$  in slave/receive mode releases I2C $n$ \_SCL, allowing the master to send data.

In the slave transmitter routine, I2SR $n$ [RXAK] must be tested before sending the next byte of data. Setting RXAK means an end-of-data signal from the master receiver, after which software must switch it from transmitter to receiver mode. Reading I2DR $n$  releases I2C $n$ \_SCL so the master can generate a STOP signal.

#### 42.4.7 Arbitration Lost

If several devices try to engage the bus at the same time, one becomes master. Hardware immediately switches devices that lose arbitration to slave receive mode. Data output to I2C $n$ \_SDA stops, but I2C $n$ \_SCL continues generating until the end of the byte during which arbitration is lost. An interrupt occurs at the falling edge of the ninth clock of this transfer with I2SR $n$ [IAL] set and I2CR $n$ [MSTA] cleared.

If a non-master device tries to transmit or execute a START, hardware inhibits the transmission, clears MSTA without signaling a STOP, generates an interrupt to the CPU, and sets IAL to indicate a failed

attempt to engage the bus. When considering these cases, slave service routine should first test IAL and software should clear it if it is set.

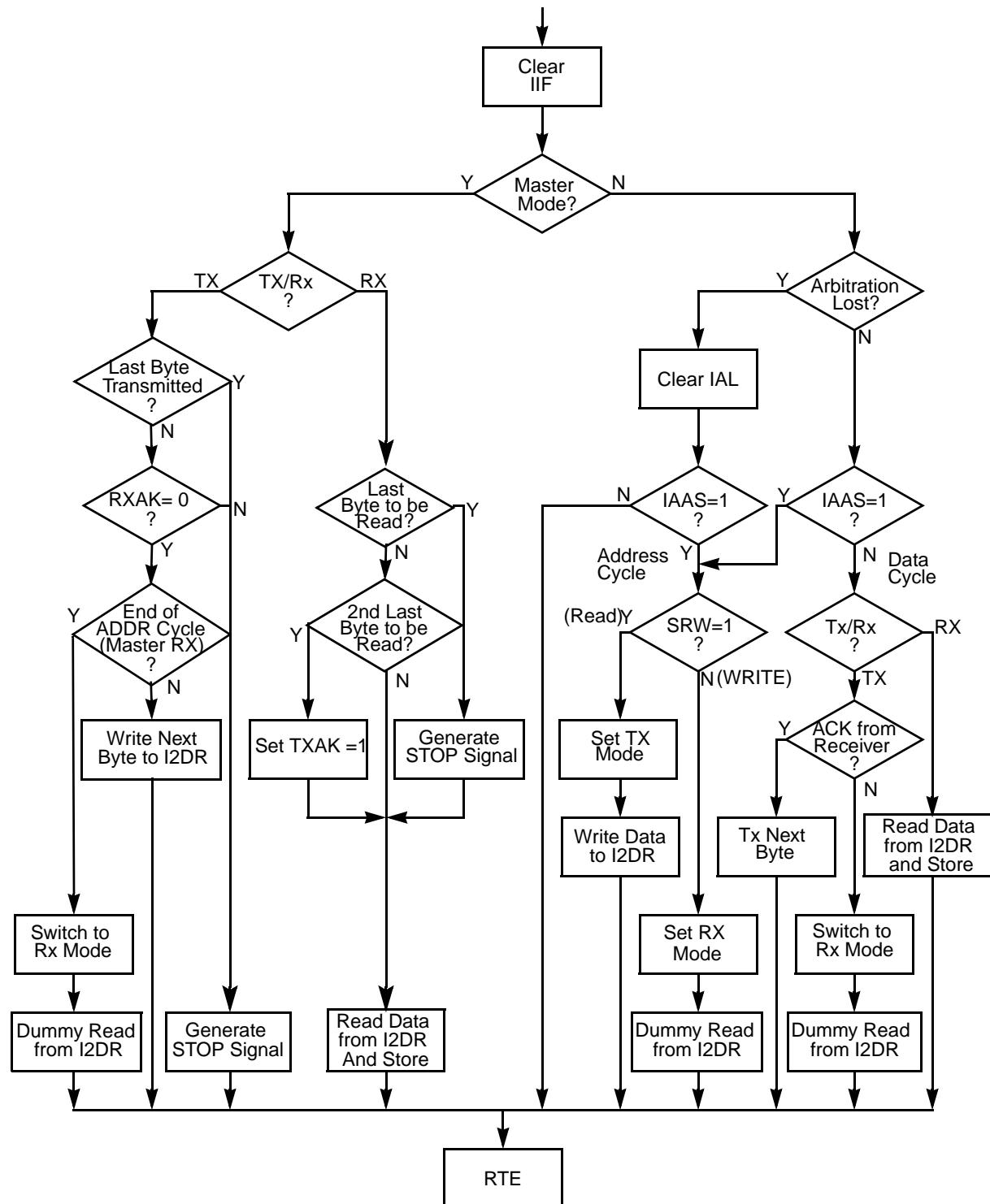


Figure 42-14. Flow-Chart of Typical I<sup>2</sup>C Interrupt Routine



# Chapter 43

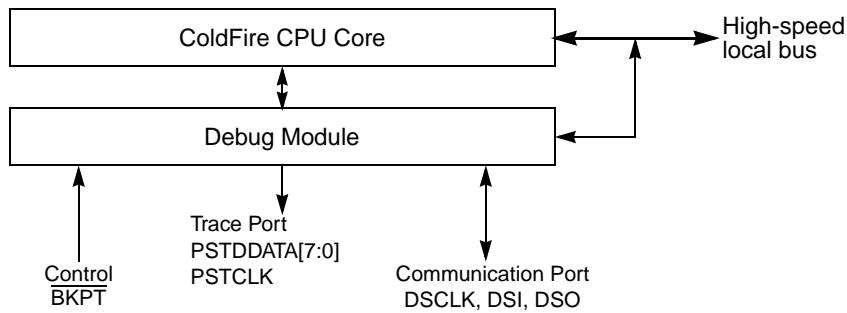
## Debug Module

### 43.1 Introduction

This chapter describes the revision D+ enhanced hardware debug module.

#### 43.1.1 Block Diagram

The debug module is shown in [Figure 43-1](#).



**Figure 43-1. Processor/Debug Module Interface**

#### 43.1.2 Overview

Debug support is divided into three areas:

- Real-time trace support—The ability to determine the dynamic execution path through an application is fundamental for debugging. The ColdFire solution implements an 8-bit parallel output bus that reports processor execution status and data to an external emulator system. See [Section 43.4.4, “Real-Time Trace Support”](#). This device also implements a trace buffer (PSTB) that records processor execution status and data, which can be subsequently accessed by the external emulator system to provide program and data trace information. See [Section 43.4.4, “Real-Time Trace Support”](#).
- Background debug mode (BDM)—Provides low-level debugging in the ColdFire processor complex. In BDM, the processor complex is halted and a variety of commands can be sent to the processor to access memory, registers, and peripherals. The external emulator uses a three-pin, serial, full-duplex channel. See [Section 43.4.1, “Background Debug Mode \(BDM\),”](#) and [Section 43.3, “Memory Map/Register Definition”](#).
- Real-time debug support—BDM requires the processor to be halted, which many real-time embedded applications cannot do. Debug interrupts let real-time systems execute a unique service routine that can quickly save the contents of key registers and variables and return the system to

normal operation. External development systems can access saved data, because the hardware supports concurrent operation of the processor and BDM-initiated commands. In addition, the option allows interrupts to occur. See [Section 43.4.2, “Real-Time Debug Support”](#).

The first version 2 ColdFire core devices implemented the original debug architecture, now called revision A. Based on feedback from customers and third-party developers, enhancements have been added to succeeding generations of ColdFire cores. For revision A, CSR[HRL] is 0. See [Section 43.3.2, “Configuration/Status Register \(CSR\)”](#).

Revision B (and B+) of the debug architecture offers more flexibility for configuring the hardware breakpoint trigger registers and removing the restrictions involving concurrent BDM processing while hardware breakpoint registers are active. Revision B+ adds three additional PC breakpoint registers. For revision B, CSR[HRL] is 1, and for revision B+, CSR[HRL] is 0x9.

Revision C of the debug architecture more than doubles the on-chip breakpoint registers and provides an ability to interrupt debug service routines. This revision also combines the PST and DDATA signals into PSTDDATA[7:0]. Because real-time trace information appears as a sequence of 4-bit values, there are no alignment restrictions. In other words, PST values and operands may appear on either nibble of PSTDDATA. For revision C, CSR[HRL] is 2.

The addition of the memory management unit (MMU) to the baseline architecture requires corresponding enhancements to the ColdFire debug functionality, resulting in revision D. For revision D, the revision level bit, CSR[HRL], is 3.

With software support, the MMU can provide a demand-paged, virtual address environment. To support debugging in this virtual environment, the debug enhancements are primarily related to the expansion of the virtual address to include the 8-bit address space identifier (ASID). Conceptually, the virtual address is expanded to a 40-bit value: the 8-bit ASID plus the 32-bit address.

The expansion of the virtual address affects two major debug functions:

- The ASID is optionally included in the specification of the hardware breakpoint registers. As an example, the four PC breakpoint registers are each expanded by 8 bits, so that a specific ASID value may be programmed as part of the breakpoint instruction address. Likewise, each operand address/data breakpoint register is expanded to include an ASID value. Finally, control registers define if and how the ASID is to be included in the breakpoint comparison trigger logic.
- The debug module implements the concept of ownership trace in which the ASID value may be optionally displayed as part of the real-time trace functionality. When enabled, real-time trace displays instruction addresses on every change-of-flow instruction that is not absolute or PC-relative. For Rev. D, this instruction address display optionally includes the contents of the ASID, thus providing the complete instruction virtual address on these instructions.
- Additionally when a SYNC\_PC serial BDM command is loaded from the external development system, the processor optionally displays the complete virtual instruction address, including the 8-bit ASID value.

In addition to these ASID-related changes, the MMU control registers are accessible by using serial BDM commands. The same BDM access capabilities are also provided for the EMAC programming model.

Finally, a serial BDM command is implemented (FORCE\_TA) to assist debugging when a software error generates an incorrect memory address that hangs the external bus. The BDM command attempts to break this condition by forcing a bus termination.

Revision D+ adds the ignore pending interrupt bit (CSR[IPI]) for debug revision D. (This bit is included in revision A, B, and B+). For revision D+, the revision level (CSR[HRL]) is 0xB.

To supplement the existing real-time PST/DDATA trace included as part of the classic ColdFire debug architecture, Revision D+PSTB implements an on-chip trace buffer that records processor execution status and instruction addresses. The external emulator system can access this buffer for program trace information. The PST trace buffer (PSTB) records compressed processor execution status and debug data in a 128-packet memory that is mapped into the debug register programming model so the trace information can be retrieved via the 3-pin serial BDM interface. The PSTB supports programmable start/stop recording conditions, as well as special continuous and PC-profiling modes of operation along with normal recording.

Support for the PSTB functionality adds two debug control registers (the extended configuration/status register (XCSR) and configuration/status register 2 (CSR2)), the read-only PSTB data values, and BDM commands to access these new registers. For Revision D+PSTB, CSR[HRL] is 0xF and CSR2[D1HRL] is 0xD.

The following table summarizes the various debug revisions.

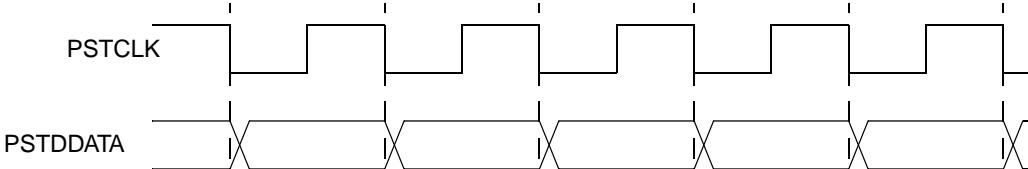
**Table 43-1. Debug Revision Summary**

Revision	CSR[HRL]	CSR2 [D1HRL]	Enhancements
A	0000	—	Initial debug revision
B	0001	—	BDM command execution does not affect hardware breakpoint logic Added BDM address attribute register (BAAR) <del>BKPT</del> configurable interrupt (CSR[BKD]) Level 1 and level 2 triggers on OR condition, in addition to AND SYNC_PC command to display the processor's current PC
B+	1001	—	3 additional PC breakpoint registers PBR1–3
C	0010	—	Combined PST and DDATA signals Adds breakpoint registers Supports normal interrupt request service during debug Redefinition of the PST values for the RTS instruction
D	0011	—	MMU enhancements to support ASID FORCE_TA command
D+	1011	—	Added CSR[IPI] for revision D
D+PSTB	1111	1101	Added PST buffer

## 43.2 Signal Descriptions

Table 43-2 describes debug module signals. All ColdFire debug signals are unidirectional and related to a rising edge of the processor core's clock signal. The standard 26-pin debug connector is shown in Section 43.4.6, “Freescale-Recommended BDM Pinout”.

**Table 43-2. Debug Module Signals**

Signal	Description
Development Serial Clock (DSCLK)	Internally synchronized input. (The logic level on DSCLK is validated if it has the same value on two consecutive rising bus clock edges.) Clocks the serial communication port to the debug module during packet transfers. Maximum frequency is 1/5 the processor status clock (PSTCLK). At the synchronized rising edge of DSCLK, the data input on DSI is sampled and DSO changes state.
Development Serial Input (DSI)	Internally synchronized input that provides data input for the serial communication port to the debug module after the DSCLK has been seen as high (logic 1).
Development Serial Output (DSO)	Provides serial output communication for debug module responses. DSO is registered internally. The output is delayed from the validation of DSCLK high.
Breakpoint ( $\overline{\text{BKPT}}$ )	Input requests a manual breakpoint. Assertion of $\overline{\text{BKPT}}$ puts the processor into a halted state after the current instruction completes. Halt status is reflected on processor status/debug data signals (PSTDDATA[7:0]) as multiple cycles of 0xF. If CSR[BKD] is set (disabling normal $\overline{\text{BKPT}}$ functionality), asserting $\overline{\text{BKPT}}$ generates a debug interrupt exception in the processor.
Processor Status Clock (PSTCLK)	Half-speed version of the processor clock. Its rising edge appears in the center of the two-processor-cycle window of valid PSTDDATA output. PSTCLK indicates when the development system should sample PSTDDATA values. The following figure shows PSTCLK timing with respect to PSTDDATA.  If real-time trace is not used, setting CSR[PCD] keeps PSTCLK and PSTDDATA outputs from toggling without disabling triggers. Non-quiescent operation can be reenabled by clearing CSR[PCD], although the external development systems must resynchronize with the PSTDDATA output. PSTCLK starts clocking only when the first non-zero PST value (0xC, 0xD, or 0xF) occurs during system reset exception processing. <a href="#">Table 43-35</a> describes PST values.
Processor Status/Debug Data (PSTDDATA[7:0])	These outputs, which change on the negative edge of PSTCLK, indicate processor status and captured address and data values and are discussed more thoroughly in <a href="#">Section 43.2.1, "Processor Status/Debug Data (PSTDDATA[7:0])"</a> .

### 43.2.1 Processor Status/Debug Data (PSTDDATA[7:0])

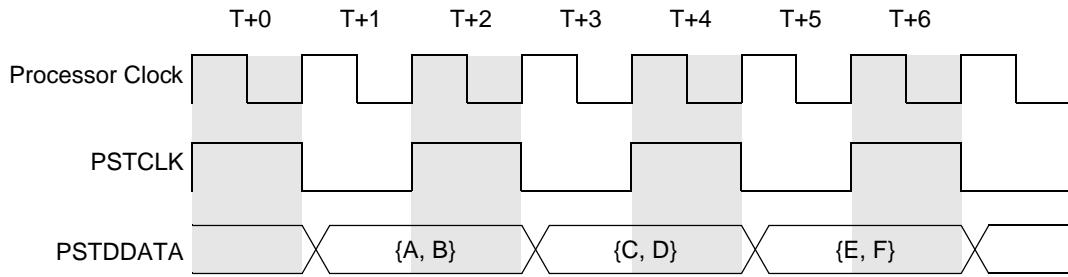
Processor status data outputs indicate processor status and captured address and data values. They operate at half the processor's frequency. Given that real-time trace information appears as a sequence of 4-bit data values, there are no alignment restrictions; that is, the processor status (PST) values and operands may appear on either nibble of PSTDDATA[7:0]. The upper nibble (PSTDDATA[7:4]) is the more significant and yields values first.

The CSR register controls capturing of data values to be presented on PSTDDATA. Executing the WDDATA instruction captures data that is displayed on PSTDDATA too. These signals are updated each processor cycle and display two values at a time for two processor clock cycles. [Table 43-3](#) shows the PSTDDATA output for the processor's sequential execution of single-cycle instructions (A, B, C, D...). Cycle counts are shown relative to processor frequency. These outputs indicate the current processor pipeline status and are not related to the current bus transfer.

**Table 43-3. PSTDDATA: Sequential Execution of Single-Cycle Instructions**

Cycles	PSTDDATA[7:0]
T+0, T+1	{PST for A, PST for B}
T+2, T+3	{PST for C, PST for D}
T+4, T+5	{PST for E, PST for F}

The signal timing for the example in [Table 43-3](#) is shown in [Figure 43-2](#).

**Figure 43-2. PSTDDATA: Single-Cycle Instruction Timing**

[Table 43-4](#) shows the case where a PSTDDATA module captures a memory operand on a simple load instruction: `move.l <mem>, Rx`.

**Table 43-4. PSTDDATA: Data Operand Captured**

Cycle	PSTDDATA[7:0]
T	{PST for move.l, PST marker for captured operand} = {0x1, 0xB}
T+1	{0x1, 0xB}
T+2	{Operand[3:0], Operand[7:4]}
T+3	{Operand[3:0], Operand[7:4]}
T+4	{Operand[11:8], Operand[15:12]}
T+5	{Operand[11:8], Operand[15:12]}
T+6	{Operand[19:16], Operand[23:20]}
T+7	{Operand[19:16], Operand[23:20]}
T+8	{Operand[27:24], Operand[31:28]}
T+9	{Operand[27:24], Operand[31:28]}
T+10	(PST for next instruction)
T+11	(PST for next instruction,...)

**NOTE**

A PST marker and its data display are sent contiguously. Except for this transmission, the IDLE status (0x0) can appear anytime. Again, given that real-time trace information appears as a sequence of 4-bit values, there are no alignment restrictions. That is, PST values and operands may appear on either nibble of PSTDDATA.

### 43.3 Memory Map/Register Definition

In addition to the existing BDM commands that provide access to the processor's registers and the memory subsystem, the debug module contain a number of registers to support the required functionality. These registers are also accessible from the processor's supervisor programming model by executing the WDEBUG instruction (write only). Therefore, the breakpoint hardware in debug module can be read or written by the external development system using the debug serial interface or written by the operating system running on the processor core. Software guarantees that accesses to these resources are serialized and logically consistent. Hardware provides a locking mechanism in CSR to allow external development system to disable any attempted writes by the processor to the breakpoint registers (setting CSR[IPW]). BDM commands must not be issued if the ColdFire processor is using the WDEBUG instruction to access debug module registers, or the resulting behavior is undefined. The DSCLK must be quiescent during operation of the WDEBUG command.

These registers, shown in [Table 43-5](#), are treated as 32-bit quantities, regardless of the number of implemented bits. These registers are also accessed through the BDM port by the commands, WDMREG and RDMREG, described in [Section 43.4.1.5, “BDM Command Set”](#). These commands contain a 5-bit field, DRc, that specifies the register, as shown in [Table 43-5](#).

**Table 43-5. Debug Module Memory Map**

DRc[4-0]	Register Name	Width (bits)	Access	Reset Value	Section/ Page
0x00	Configuration/status register (CSR)	32	R/W See Note	0x00F0_0000	<a href="#">43.3.2/43-8</a>
0x01	Extended configuration/status register (XCSR)	32	R/W See Note	0x0000_0000	<a href="#">43.3.3/43-11</a>
0x02	Configuration/status register 2 (CSR2)	32	R/W See Note	See section	<a href="#">43.3.4/43-13</a>
0x05	BDM address attribute register (BAAR)	32 <sup>1</sup>	W	0x05	<a href="#">43.3.5/43-15</a>
0x06	Address attribute trigger register (AATR)	32	W	0x0000_0005	<a href="#">43.3.6/43-16</a>
0x07	Trigger definition register (TDR)	32	W	0x0000_0000	<a href="#">43.3.7/43-18</a>
0x08	PC breakpoint register 0 (PBR0)	32	W	Undefined	<a href="#">43.3.8/43-21</a>
0x09	PC breakpoint mask register (PBMR)	32	W	Undefined	<a href="#">43.3.8/43-21</a>
0x0A	PC breakpoint ASID control (PBAC)	32	W	Undefined	<a href="#">43.3.9/43-22</a>
0x0C	Address breakpoint high register (ABHR)	32	W	Undefined	<a href="#">43.3.10/43-23</a>
0x0D	Address breakpoint low register (ABLR)	32	W	Undefined	<a href="#">43.3.10/43-23</a>

**Table 43-5. Debug Module Memory Map (continued)**

DRc[4–0]	Register Name	Width (bits)	Access	Reset Value	Section/Page
0x0E	Data breakpoint register (DBR)	32	W	Undefined	<a href="#">43.3.11/43-24</a>
0x0F	Data breakpoint mask register (DBMR)	32	W	Undefined	<a href="#">43.3.11/43-24</a>
0x14	PC breakpoint ASID register (PBASID)	32	W	Undefined	<a href="#">43.3.12/43-25</a>
0x16	Address attribute trigger register 1 (AATR1)	32	W	0x0005	<a href="#">43.3.6/43-16</a>
0x17	Extended trigger definition register (XTDR)	32	W	0x0000_0000	<a href="#">43.3.13/43-26</a>
0x18	PC breakpoint register 1 (PBR1)	32	W	See Section	<a href="#">43.3.8/43-21</a>
0x1A	PC breakpoint register 2 (PBR2)	32	W	See Section	<a href="#">43.3.8/43-21</a>
0x1B	PC breakpoint register 3 (PBR3)	32	W	See Section	<a href="#">43.3.8/43-21</a>
0x1C	Address high breakpoint register 1 (ABHR1)	32	W	Undefined	<a href="#">43.3.10/43-23</a>
0x1D	Address low breakpoint register 1 (ABLR1)	32	W	Undefined	<a href="#">43.3.10/43-23</a>
0x1E	Data breakpoint register 1 (DBR1)	32	W	Undefined	<a href="#">43.3.11/43-24</a>
0x1F	Data breakpoint mask register 1 (DBMR1)	32	W	Undefined	<a href="#">43.3.11/43-24</a>
0x20 + n; n = 0x0–0x17	Trace Buffer Longword n (TBLWn); n = 0–23	32	R (BDM) <sup>2</sup>	Undefined, Unaffected	<a href="#">43.3.14/43-30</a>
0x3F	Most recently sampled PC (PCRS)	32	R (BDM) <sup>3</sup>	Undefined, Unaffected	<a href="#">43.3.15/43-31</a>

<sup>1</sup> Each debug register is accessed as a 32-bit register; reserved fields are not used (don't care).

<sup>2</sup> The contents of the PST trace buffer is read-only from BDM (32 bits per access) using RDMREG commands.

<sup>3</sup> The contents of the PCRS is read-only from BDM (32 bits per access) using RDMREG commands.

## NOTE

Debug control registers can be written by the external development system or the CPU through the WDEBUG instruction. These control registers are write-only from the programming model and they can be written through the BDM port using the WDMREG command. In addition, the configuration/status registers (CSR, XCSR, and CSR2) can be read through the BDM port using the RDMREG command.

The ColdFire debug architecture supports a number of hardware breakpoint registers, that can be configured into single- or double-level triggers based on the PC or operand address ranges with an optional inclusion of specific data values. With the addition of the MMU capabilities, the breakpoint specifications must be expanded to optionally include the address space identifier (ASID) in these user-programmable virtual address triggers.

The core includes four PC breakpoint triggers and two sets of operand address breakpoint triggers, each with two independent address registers (to allow specification of a range) and a data breakpoint with masking capabilities. Core breakpoint triggers are accessible through the serial BDM interface or written through the supervisor programming model using the WDEBUG instruction.

Two ASID-related registers (PBAC and PBASID) are added for the PC breakpoint qualification, and two existing registers (AATR and AATTR1) are expanded for the address breakpoint qualification.

### 43.3.1 Shared Debug Resources

The debug module revision A implementation provides a common hardware structure for BDM and breakpoint functionality. Certain hardware structures are used for BDM and breakpoint purposes as shown in [Table 43-6](#).

**Table 43-6. Shared BDM/Breakpoint Hardware**

Register	BDM Function	Breakpoint Function
AATR	Bus attributes for all memory commands	Attributes for address breakpoint
ABHR	Address for all memory commands	Address for address breakpoint
DBR	Data for all BDM write commands	Data for data breakpoint

Therefore, loading a register to perform a specific function that shares hardware resources is destructive to the shared function. For example, if an operand address breakpoint is loaded into the debug module, a BDM command to access memory overwrites an address breakpoint in ABHR. If a data breakpoint is configured, a BDM write command overwrites the data breakpoint in DBR.

Revision B added hardware registers to eliminate these shared functions. The BAAR is used to specify bus attributes for BDM memory commands and has the same format as the LSB of the AATR. The registers containing the BDM memory address and the BDM data are not program visible.

### 43.3.2 Configuration/Status Register (CSR)

The CSR defines the debug configuration for the processor and memory subsystem and contains status information from the breakpoint logic. CSR is write-only from the programming model. It can be read from and written to through the BDM port. CSR is accessible in supervisor mode as debug control register 0x00 using the WDEBUG instruction and through the BDM port using the RDMREG and WDMREG commands.

DRc[4:0]: 0x00 (CSR)

Access: Supervisor write-only  
BDM read/write

R				W				R				W			
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BSTAT				FOF	TRG	HALT	BKPT	HRL				0	BKD	PCD	IPW
W															
Reset	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0
R				W				R				W			
MAP	TRC	EMU	DDC	UHE	BTB	0	NPL	IPI	SSM	OTE	0	0	0	0	0
W															
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

**Figure 43-3. Configuration/Status Register (CSR)**

**Table 43-7. CSR Field Descriptions**

Field	Description
31–28 BSTAT	Breakpoint Status. Provides read-only status (from the BDM port only) information concerning hardware breakpoints. Also output on PSTDDATA when it is not displaying PST or other processor data. BSTAT is cleared by a TDRor XTDR write or by a CSR read when a level-2 breakpoint is triggered or a level-1 breakpoint is triggered and the level-2 breakpoint is disabled. 0000 No breakpoints enabled 0001 Waiting for level-1 breakpoint 0010 Level-1 breakpoint triggered 0101 Waiting for level-2 breakpoint 0110 Level-2 breakpoint triggered Else Reserved
27 FOF	Fault-on-fault. If FOF is set, a catastrophic halt occurred and forced entry into BDM.
26 TRG	Hardware breakpoint trigger. If TRG is set, a hardware breakpoint halted the processor core and forced entry into BDM. Reset or the debug GO command clear TRG.
25 HALT	Processor halt. If HALT is set, the processor executed a HALT and forced entry into BDM. Reset or the debug GO command clear HALT.
24 BKPT	Breakpoint assert. If BKPT is set, $\overline{\text{BKPT}}$ was asserted, forcing the processor into BDM. Reset or the debug GO command clear BKPT.
23–20 HRL	Hardware revision level. Indicates, from the BDM port only, the level of debug module functionality. An emulator could use this information to identify the level of functionality supported. 0000 Revision A 0001 Revision B 0010 Revision C 0011 Revision D 1001 Revision B+ 1011 Revision D+ 1111 Revision D+PSTB (This is the value used for this device)
19	Reserved, must be cleared.
18 BKD	Breakpoint disable. Disables the normal $\overline{\text{BKPT}}$ input signal functionality, and allows the assertion of this pin to generate a debug interrupt. 0 Normal operation 1 $\overline{\text{BKPT}}$ is edge-sensitive: a high-to-low edge on $\overline{\text{BKPT}}$ signals a debug interrupt to the ColdFire core. The processor makes this interrupt request pending until the next sample point occurs, when the exception is initiated. In the ColdFire architecture, the interrupt sample point occurs once per instruction. There is no support for nesting debug interrupts.
17 PCD	PSTCLK disable. 0 PSTCLK is fully operational 1 Disables the generation of the PSTCLK and PSTDDATA output signals, and forces these signals to remain quiescent
16 IPW	Inhibit processor writes. Setting IPW inhibits processor-initiated writes to the debug module's programming model registers. Only commands from the external development system can modify IPW.
15 MAP	Force processor references in emulator mode. 0 All emulator-mode references are mapped into supervisor code and data spaces. 1 The processor maps all references while in emulator mode to a special address space, TT equals 10, TM equals 101 or 110. The internal SRAM and caches are disabled.

**Table 43-7. CSR Field Descriptions (continued)**

Field	Description
14 TRC	Force emulation mode on trace exception. 0 The processor enters supervisor mode 1 The processor enters emulator mode when a trace exception occurs
13 EMU	Force emulation mode. 0 Do not force emulator mode 1 The processor begins executing in emulator mode. See <a href="#">Section 43.4.2.2, “Emulator Mode”</a> .
12–11 DDC	Debug data control. Controls operand data capture for PSTDDATA, which displays the number of bytes defined by the operand reference size before the actual data; byte displays 8 bits, word displays 16 bits, and long displays 32 bits (one nibble at a time across multiple PSTCLK clock cycles). See <a href="#">Table 43-35</a> . 00 No operand data is displayed. 01 Capture all write data. 10 Capture all read data. 11 Capture all read and write data.
10 UHE	User halt enable. Selects the CPU privilege level required to execute the HALT instruction. 0 HALT is a supervisor-only instruction. 1 HALT is a supervisor/user instruction.
9–8 BTB	Branch target bytes. Defines the number of bytes of branch target address PSTDDATA displays. 00 0 bytes 01 Lower 2 bytes of the target address 10 Lower 3 bytes of the target address 11 Entire 4-byte target address See <a href="#">Section 43.4.4.2, “Begin Execution of Taken Branch (PST = 0x5)”</a> .
7	Reserved, must be cleared.
6 NPL	Non-pipelined mode. Determines whether the core operates in pipelined mode or not. 0 Pipelined mode 1 Non-pipelined mode. The processor effectively executes one instruction at a time with no overlap. This adds at least 5 cycles to the execution time of each instruction. Superscalar instruction dispatch is disabled when operating in this mode. Given an average execution latency of 1.6 cycles/instruction, throughput in non-pipeline mode would be 6.6 cycles/instruction, approximately 25% or less of pipelined performance. Regardless of the NPL state, a triggered PC breakpoint is always reported before the triggering instruction executes. In normal pipeline operation, occurrence of an address and/or data breakpoint trigger is imprecise. In non-pipeline mode, triggers are always reported before the next instruction begins execution and trigger reporting can be considered precise. An address or data breakpoint should always occur before the next instruction begins execution. Therefore, the occurrence of the address/data breakpoints should be guaranteed.
5 IPI	Ignore pending interrupts. 0 Core services any pending interrupt requests that were signalled while in single-step mode. 1 Core ignores any pending interrupt requests signalled while in single-instruction-step mode.
4 SSM	Single-Step Mode. Setting SSM puts the processor in single-step mode. 0 Normal mode. 1 Single-step mode. The processor halts after execution of each instruction. While halted, any BDM command can be executed. On receipt of the GO command, the processor executes the next instruction and halts again. This process continues until SSM is cleared.
3 OTE	Ownership-trace enable. Enables the display of the ASID on the PSTDDATA outputs upon entrance into user mode, a load of the ASID by a MOVEC instruction, or the execution of a BDM SYNC_PC command. 0 No ASID displayed 1 ASID displayed on PSTDDATA outputs

**Table 43-7. CSR Field Descriptions (continued)**

Field	Description
2	Reserved, must be cleared.
1 FDBG	Force the debug mode core output signal (to the on-chip peripherals). The debug mode output is logically defined as: Debug mode output = CSR[FDBG]   ( $\overline{\text{CSR[DBGH]}}$ and Core is halted) 0 Debug mode output is not forced asserted. 1 Debug mode output core output signal is asserted.
0 DBGH	Disable debug signal assertion during core halt. The debug mode output (to the on-chip peripherals) is logically defined as: Debug mode output = CSR[FDBG]   ( $\overline{\text{CSR[DBGH]}}$ and Core is halted) 0 Debug mode output is asserted when the core is halted. 1 Debug mode output is not asserted when the core is halted.

### 43.3.3 Extended Configuration/Status Register (XCSR)

The 32-bit XCSR is partitioned into two sections: the upper byte contains read-only status bits and the lower 24 bits contain fields related to generating automatic SYNC\_PC commands. The SYNC\_PC commands can periodically capture and display the current PC on the PST/DDATA output ports and/or in the PST trace buffer (if properly configured).

There are multiple ways to reference the XCSR. They are summarized in [Table 43-8](#).

**Table 43-8. XCSR Reference Summary**

Method	Reference Details
RDMREG	Reads XCSR[31–0] from the BDM interface.
WDMREG	Writes XCSR[31–0] from the BDM interface.
WDEBUG instruction	Writes XCSR[23–0] during the core's execution of WDEBUG instruction. This instruction is a privileged supervisor-mode instruction.

DRc: 0x01 (XCSR)

Access: Supervisor write-only  
BDM read/write

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

**Figure 43-4. Extended Configuration/Status Register (XCSR)**

Table 43-9. XCSR Field Descriptions

Field	Description																																				
31 CPUHALT	Indicates that the CPU is in the halt state. The CPU state may be running, stopped, or halted, which is determined by the CPUHALT and CPUSTOP bits as shown below.																																				
	<table border="1"> <thead> <tr> <th>XCSR [CPUHALT]</th> <th>XCSR [CPUSTOP]</th> <th>CPU State</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>Running</td> </tr> <tr> <td>0</td> <td>1</td> <td>Stopped</td> </tr> <tr> <td>1</td> <td>0</td> <td>Halted</td> </tr> </tbody> </table>	XCSR [CPUHALT]	XCSR [CPUSTOP]	CPU State	0	0	Running	0	1	Stopped	1	0	Halted																								
XCSR [CPUHALT]	XCSR [CPUSTOP]	CPU State																																			
0	0	Running																																			
0	1	Stopped																																			
1	0	Halted																																			
30 CPUSTOP	Indicates that the CPU is in the stop state. The CPU state may be running, stopped, or halted, which is determined by the CPUHALT and CPUSTOP bits as shown in the CPUHALT bit description.																																				
29–3	Reserved, must be cleared.																																				
2–1 APCSC	<p>Automatic PC synchronization control. Determines the periodic interval of PC address captures, if XCSR[APCENB] is set. When the selected interval is reached, a SYNC_PC command is sent to the ColdFire CPU. For more information on the SYNC_PC operation, see the APCENB description.</p> <p>The chosen frequency depends on CSR2[APCDIV16] as shown in the equation and table below:</p> $\text{PC address capture period} = \frac{2^{(\text{APCSC} + 1)} \times 1024}{\text{APCDIV16}} \quad \text{Eqn. 43-1}$ <table border="1"> <thead> <tr> <th>XCSR [APCENB]</th> <th>CSR2 [APCDIV16]</th> <th>XCSR [APCSC]</th> <th>SYNC_PC Interval</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>0</td> <td>00</td> <td>2048 cycles</td> </tr> <tr> <td>1</td> <td>0</td> <td>01</td> <td>4096 cycles</td> </tr> <tr> <td>1</td> <td>0</td> <td>10</td> <td>8192 cycles</td> </tr> <tr> <td>1</td> <td>0</td> <td>11</td> <td>16384 cycles</td> </tr> <tr> <td>1</td> <td>1</td> <td>00</td> <td>128 cycles</td> </tr> <tr> <td>1</td> <td>1</td> <td>01</td> <td>256 cycles</td> </tr> <tr> <td>1</td> <td>1</td> <td>10</td> <td>512 cycles</td> </tr> <tr> <td>1</td> <td>1</td> <td>11</td> <td>1024 cycles</td> </tr> </tbody> </table>	XCSR [APCENB]	CSR2 [APCDIV16]	XCSR [APCSC]	SYNC_PC Interval	1	0	00	2048 cycles	1	0	01	4096 cycles	1	0	10	8192 cycles	1	0	11	16384 cycles	1	1	00	128 cycles	1	1	01	256 cycles	1	1	10	512 cycles	1	1	11	1024 cycles
XCSR [APCENB]	CSR2 [APCDIV16]	XCSR [APCSC]	SYNC_PC Interval																																		
1	0	00	2048 cycles																																		
1	0	01	4096 cycles																																		
1	0	10	8192 cycles																																		
1	0	11	16384 cycles																																		
1	1	00	128 cycles																																		
1	1	01	256 cycles																																		
1	1	10	512 cycles																																		
1	1	11	1024 cycles																																		
0 APCENB	<p>Automatic PC synchronization enable. Enables the periodic output of the PC which can be used for PST/DDATA trace synchronization.</p> <p>As described in XCSR[APCSC], when the enabled periodic timer expires, a SYNC_PC command is sent to the ColdFire CPU which generates a forced instruction fetch of the next instruction. The PST/DDATA module captures the target address as defined by CSR[9:8]. This produces a PST sequence of the PST marker indicating a 2-, 3-, or 4-byte address, followed by the captured instruction address and optionally, a marker and ASID.</p> <p>0 Automatic PC synchronization disabled 1 Automatic PC synchronization enabled</p>																																				

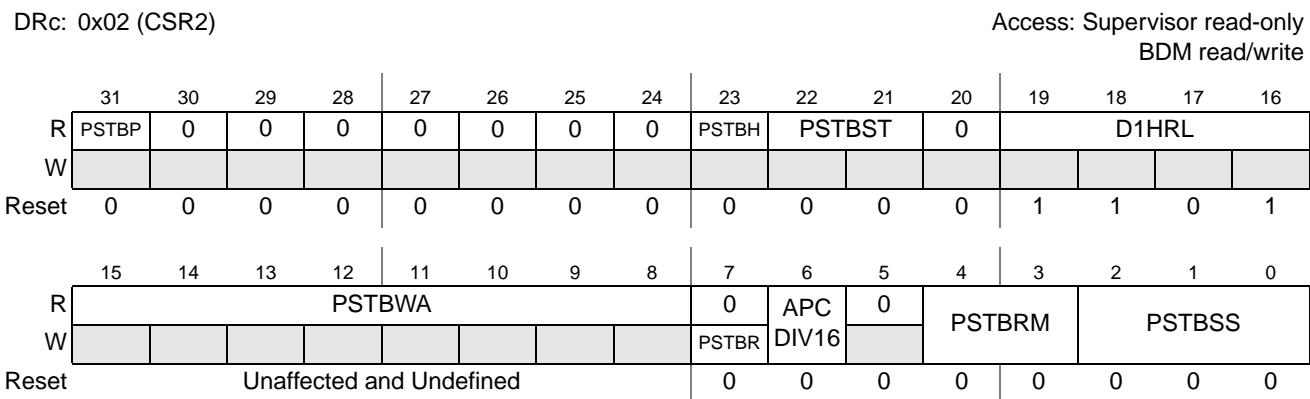
### 43.3.4 Configuration/Status Register 2 (CSR2)

The 32-bit CSR2 is partitioned into two sections: the upper byte contains read-only status bits and the lower 24 bits contain fields related to configuring the PST trace buffer (PSTB).

There are multiple ways to reference CSR2. They are summarized in [Table 43-10](#).

**Table 43-10. CSR2 Reference Summary**

Method	Reference Details
RDMREG	Reads CSR2[31–0] from the BDM interface.
WDMREG	Writes CSR2[23–0] from the BDM interface.
WDEBUG Instruction	Writes CSR2[23–0] during the core's execution of WDEBUG instruction. This instruction is a privileged supervisor-mode instruction.



**Figure 43-5. Configuration/Status Register 2 (CSR2)**

**Table 43-11. CSR2 Field Descriptions**

Field	Description
31 PSTBP	PST buffer stop. Signals if a PST buffer stop condition has been reached. 0 A PST trace buffer stop condition has not been reached 1 A PST trace buffer stop condition has been reached
30–24	Reserved, must be cleared.
23 PSTBH	PST trace buffer halt. Indicates if the processor is halted due to the PST trace buffer being full when recording in a continuous mode. 0 PST trace buffer not full 1 CPU halted due to PST trace buffer being full in continuous mode
22–21 PSTBST	PST trace buffer state. Indicates the current state of the PST trace buffer recording. 00 PSTB disabled 01 PSTB enabled and waiting for the start condition 10 PSTB enabled, recording and waiting for the stop condition 11 PSTB enabled, completed recording after the stop condition was reached
20	Reserved, must be cleared.

**Table 43-11. CSR2 Field Descriptions (continued)**

Field	Description						
19–16 D1HRL	Debug hardware revision level. Indicates the auxiliary hardware revision level of the debug module implemented in the ColdFire core. For this device, this field is 0xD, indicating a 128-entry trace buffer.						
15–8 PSTBWA	PST trace buffer write address. Indicates the current write address of the PST trace buffer. The most significant bit of this field is sticky; if set, it remains set until a PST/DDATA reset event occurs. As the ColdFire core inserts PST and DDATA packets into the trace buffer, this field is incremented. The value of the write address defines the next location in the PST trace buffer to be loaded. In other words, the contents of PSTB[PSTBWA-1] is the last valid entry in the trace buffer. The msb of this field can be used to determine if the entire PST trace buffer has been loaded with valid data. <table border="1" data-bbox="518 587 1230 770"> <thead> <tr> <th>PSTBWA[7]</th><th>PSTB Valid Data Locations (Oldest to Newest)</th></tr> </thead> <tbody> <tr> <td>0</td><td>0, 1, ... PSTBWA-1</td></tr> <tr> <td>1</td><td>PSTBWA, PSTBWA+1,..., 0, 1, PSTBWA-1</td></tr> </tbody> </table> <p>The PSTBWA is unaffected when a buffer stop condition has been reached, the buffer is disabled, or a system reset occurs. This allows the contents of the PST trace buffer to be retrieved after these events to assist in debug.</p>	PSTBWA[7]	PSTB Valid Data Locations (Oldest to Newest)	0	0, 1, ... PSTBWA-1	1	PSTBWA, PSTBWA+1,..., 0, 1, PSTBWA-1
PSTBWA[7]	PSTB Valid Data Locations (Oldest to Newest)						
0	0, 1, ... PSTBWA-1						
1	PSTBWA, PSTBWA+1,..., 0, 1, PSTBWA-1						
7 PSTBR	PST trace buffer reset. Generates a reset of the PST trace buffer logic, which clears PSTBWA and PSTBST. The same resources are reset when a disabled trace buffer becomes enabled and upon the receipt of a BDM GO command when operating in continuous trace mode. These reset events also clear any accumulation of PSTs. This bit always reads as a zero. 0 Do not force a PST trace buffer reset 1 Force a PST trace buffer reset						
6 APCDIV16	Automatic PC synchronization divide cycle counts by 16. This bit divides the cycle counts for automatic SYNC_PC command insertion by 16. See the APCSC and APCENB field descriptions.						
5	Reserved, must be cleared.						

**Table 43-11. CSR2 Field Descriptions (continued)**

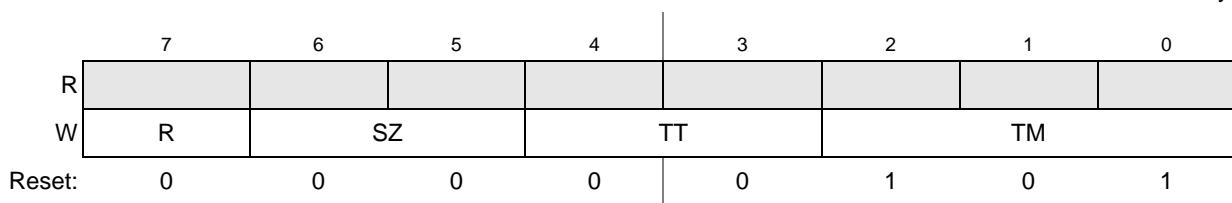
Field	Description
4–3 PSTBRM	PST trace buffer recording mode. Defines the trace buffer recording mode. The start and stop recording conditions are defined by the PSTBSS field. 00 Normal recording mode 01 Continuous, normal recording 10 PC profile recording 11 Continuous PC profile recording
2–0 PSTBSS	PST trace buffer start/stop definition. Specifies the start and stop conditions for PST trace buffer recording. In certain cases, the start and stop conditions are defined by the breakpoint registers. The remaining breakpoint registers are available for trigger configurations.

PSTBSS	Start Condition	Stop Condition
000	Trace buffer disabled, no recording	
001	Unconditional recording	
010	ABxR{& DBR/DBMR}	PBR0/PBMR
011		PBR1
100	PBR0/PBMR	ABxR{& DBR/DBMR}
101		PBR1
110	PBR1	ABxR{& DBR/DBMR}
111		PBR0/PBMR

### 43.3.5 BDM Address Attribute Register (BAAR)

The BAAR register defines the address space for memory-referencing BDM commands. BAAR[R, SZ] are loaded directly from the BDM command, while the low-order 5 bits can be programmed from the external development system. To maintain compatibility with revision A, BAAR is loaded any time the AATR is written. The BAAR is initialized to a value of 0x05, setting supervisor data as the default address space.

DRc[4:0]: 0x05 (BAAR)

Access: Supervisor write-only  
BDM write-only**Figure 43-6. BDM Address Attribute Register (BAAR)**

**Table 43-12. BAAR Field Descriptions**

Field	Description
7 R	Read/Write. 0 Write 1 Read
6–5 SZ	Size. 00 Longword 01 Byte 10 Word 11 Reserved
4–3 TT	Transfer Type. See the TT definition in the AATR description, <a href="#">Section 43.3.6, “Address Attribute Trigger Registers (AATR, AATR1)”</a> .
2–0 TM	Transfer Modifier. See the TM definition in the AATR description, <a href="#">Section 43.3.6, “Address Attribute Trigger Registers (AATR, AATR1)”</a> .

### 43.3.6 Address Attribute Trigger Registers (AATR, AATR1)

The AATR and AATR1 define address attributes and a mask to be matched in the trigger. The register value is compared with address attribute signals from the processor's local high-speed bus, as defined by the setting of the trigger definition register (TDR) for AATR and the extended trigger definition register (XTDR) for AATR1. AATR $n$  is accessible in supervisor mode as debug control register 0x06 using the WDEBUG instruction and through the BDM port using the WDMREG command.

This register is expanded to include an optional ASID specification and a control bit that enables the use of the ASID field.

DRc[4:0]: 0x06 (AATR) 0x16 (AATR1)																Access: Supervisor write-only BDM write-only							
R																							
W																AATRASID							
Reset																							
R																ASID CTRL							
W																AATRASID							
Reset																							
R																TMM							
W																TTM							
Reset																RM SZM TTM TMM R SZ TT TM							

**Figure 43-7. Address Attribute Trigger Registers (AATR, AATR1)**

**Table 43-13. AATR $n$  Field Descriptions**

Field	Description
31–25	Reserved, must be cleared.
24 ASIDCTRL	ABLR/ABHR/AATR address breakpoint ASID enable. Corresponds to the ASID control enable for the address breakpoint defined in ABLR, ABHR, and AATR. 0 Disable ASID qualifier (reset default) 1 Enable ASID qualifier
23–16 AATRASID	ABLR/ABHR/AATR ASID. Corresponds to the ASID to be included in the address breakpoint specified by ABLR, ABHR, and AATR.
15 RM	Read/write Mask. Setting RM masks R in address comparisons.
14–13 SZM	Size Mask. Setting an SZM bit masks the corresponding SZ bit in address comparisons.
12–11 TTM	Transfer Type Mask. Setting a TTM bit masks the corresponding TT bit in address comparisons.
10–8 TMM	Transfer Modifier Mask. Setting a TMM bit masks the corresponding TM bit in address comparisons.
7 R	Read/Write. R is compared with the R/W signal of the processor's local bus.
6–5 SZ	Size. Compared to the processor's local bus size signals. 00 Longword 01 Byte 10 Word 11 Reserved

**Table 43-13. AATR<sub>n</sub> Field Descriptions (continued)**

Field	Description																													
4–3 TT	<p>Transfer Type. Compared with the local bus transfer type signals.</p> <p>00 Normal processor access 01 Reserved 10 Emulator mode access 11 Reserved</p> <p>These bits also define the TT encoding for BDM memory commands. In this case, the 01 encoding indicates an external or DMA access (for backward compatibility). These bits affect the TM bits.</p>																													
2–0 TM	<p>Transfer Modifier. Compared with the local bus transfer modifier signals, which give supplemental information for each transfer type. These bits also define the TM encoding for BDM memory commands (for backward compatibility).</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;">TM</th> <th style="text-align: center;">TT=00 (normal mode)</th> <th style="text-align: center;">TT=10 (emulator mode)</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">000</td> <td>Data and instruction cache line push</td> <td>Reserved</td> </tr> <tr> <td style="text-align: center;">001</td> <td>User data access</td> <td>Reserved</td> </tr> <tr> <td style="text-align: center;">010</td> <td>User code access</td> <td>Reserved</td> </tr> <tr> <td style="text-align: center;">011</td> <td>Instruction cache invalidate</td> <td>Reserved</td> </tr> <tr> <td style="text-align: center;">100</td> <td>Data cache push/ Instruction cache invalidate</td> <td>Reserved</td> </tr> <tr> <td style="text-align: center;">101</td> <td>Supervisor data access</td> <td>Emulator mode access</td> </tr> <tr> <td style="text-align: center;">110</td> <td>Supervisor code access</td> <td>Emulator code access</td> </tr> <tr> <td style="text-align: center;">111</td> <td>INTOUCH instruction access</td> <td>Reserved</td> </tr> </tbody> </table>			TM	TT=00 (normal mode)	TT=10 (emulator mode)	000	Data and instruction cache line push	Reserved	001	User data access	Reserved	010	User code access	Reserved	011	Instruction cache invalidate	Reserved	100	Data cache push/ Instruction cache invalidate	Reserved	101	Supervisor data access	Emulator mode access	110	Supervisor code access	Emulator code access	111	INTOUCH instruction access	Reserved
TM	TT=00 (normal mode)	TT=10 (emulator mode)																												
000	Data and instruction cache line push	Reserved																												
001	User data access	Reserved																												
010	User code access	Reserved																												
011	Instruction cache invalidate	Reserved																												
100	Data cache push/ Instruction cache invalidate	Reserved																												
101	Supervisor data access	Emulator mode access																												
110	Supervisor code access	Emulator code access																												
111	INTOUCH instruction access	Reserved																												

### 43.3.7 Trigger Definition Register (TDR)

The TDR configures the operation of the hardware breakpoint logic corresponding with the ABHR/ABLR/AATR, PBR/PBR1/PBR2/PBR3/PBMR, and DBR/DBMR registers within the debug module. TDR controls the actions taken under the defined conditions with the XTDR and its associated debug registers. Breakpoint logic may be configured as one- or two-level triggers. TDR[31–16] or XTDR[31–16] bits define second-level triggers, and bits 15–0 define first-level triggers.

#### NOTE

The debug module has no hardware interlocks to prevent spurious breakpoint triggers while the breakpoint registers are being loaded. Disable TDR and XTDR (by clearing TDR[29,13] and XTDR[29,13]) before defining triggers.

A write to TDR clears the CSR trigger status bits, CSR[BSTAT]. TDR is accessible in supervisor mode as debug control register 0x07 using the WDEBUG instruction and through the BDM port using the WDMREG command.

DRc[4:0]: 0x07 (TDR)

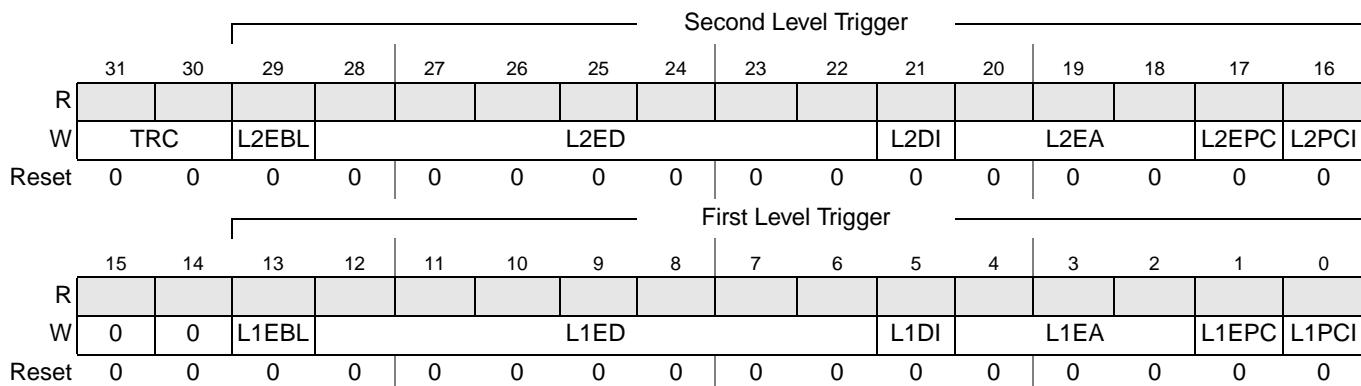
Access: Supervisor write-only  
BDM write-only

Figure 43-8. Trigger Definition Register (TDR)

Table 43-14. TDR Field Descriptions

Field	Description																	
31–30 TRC	Trigger Response Control. Determines how the processor responds to a completed trigger condition. The trigger response is always displayed on PSTDDATA. 00 Display on PSTDDATA only 01 Processor halt 10 Debug interrupt 11 Reserved																	
29 L2EBL	Enable Level 2 Breakpoint. Global enable for the breakpoint trigger. 0 Disables all level 2 breakpoints 1 Enables all level 2 breakpoint triggers																	
28–22 L2ED	Enable Level 2 Data Breakpoint. Setting an L2ED bit enables the corresponding data breakpoint condition based on the size and placement on the processor's local data bus. Clearing all ED bits disables data breakpoints.																	
	<table border="1"> <thead> <tr> <th>TDR Bit</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>28</td> <td>Data longword. Entire processor's local data bus.</td> </tr> <tr> <td>27</td> <td>Lower data word.</td> </tr> <tr> <td>26</td> <td>Upper data word.</td> </tr> <tr> <td>25</td> <td>Lower lower data byte. Low-order byte of the low-order word.</td> </tr> <tr> <td>24</td> <td>Lower middle data byte. High-order byte of the low-order word.</td> </tr> <tr> <td>23</td> <td>Upper middle data byte. Low-order byte of the high-order word.</td> </tr> <tr> <td>22</td> <td>Upper upper data byte. High-order byte of the high-order word.</td> </tr> </tbody> </table>		TDR Bit	Description	28	Data longword. Entire processor's local data bus.	27	Lower data word.	26	Upper data word.	25	Lower lower data byte. Low-order byte of the low-order word.	24	Lower middle data byte. High-order byte of the low-order word.	23	Upper middle data byte. Low-order byte of the high-order word.	22	Upper upper data byte. High-order byte of the high-order word.
TDR Bit	Description																	
28	Data longword. Entire processor's local data bus.																	
27	Lower data word.																	
26	Upper data word.																	
25	Lower lower data byte. Low-order byte of the low-order word.																	
24	Lower middle data byte. High-order byte of the low-order word.																	
23	Upper middle data byte. Low-order byte of the high-order word.																	
22	Upper upper data byte. High-order byte of the high-order word.																	
21 L2DI	Level 2 Data Breakpoint Invert. Inverts the logical sense of all the data breakpoint comparators. This can develop a trigger based on the occurrence of a data value other than the DBR contents. 0 No inversion 1 Invert data breakpoint comparators.																	

**Table 43-14. TDR Field Descriptions (continued)**

Field	Description																
20–18 L2EA	<p>Enable Level 2 Address Breakpoint. Setting an L2EA bit enables the corresponding address breakpoint. Clearing all three bits disables the breakpoint.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;">TDR Bit</th><th style="text-align: center;">Description</th></tr> </thead> <tbody> <tr> <td style="text-align: center;">20</td><td>Address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.</td></tr> <tr> <td style="text-align: center;">19</td><td>Address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.</td></tr> <tr> <td style="text-align: center;">18</td><td>Address breakpoint low. The breakpoint is based on the address in the ABLR.</td></tr> </tbody> </table>	TDR Bit	Description	20	Address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.	19	Address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.	18	Address breakpoint low. The breakpoint is based on the address in the ABLR.								
TDR Bit	Description																
20	Address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.																
19	Address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.																
18	Address breakpoint low. The breakpoint is based on the address in the ABLR.																
17 L2EPC	<p>Enable Level 2 PC Breakpoint.            0 Disable PC breakpoint            1 Enable PC breakpoint where the trigger is defined by the logical summation of:</p> $(PBR0 \text{ and } \overline{PBMR}) \mid PBR1 \mid PBR2 \mid PBR3$ <span style="float: right;"><i>Eqn. 43-2</i></span>																
16 L2PCI	<p>Level 2 PC Breakpoint Invert.            0 The PC breakpoint is defined within the region defined by PBR<math>n</math> and PBMR.            1 The PC breakpoint is defined outside the region defined by PBR<math>n</math> and PBMR.</p>																
15–14	Reserved, must be cleared.																
13 L1EBL	<p>Enable Level 1 Breakpoint. Global enable for the breakpoint trigger.            0 Disables all level 1 breakpoints            1 Enables all level 1 breakpoint triggers</p>																
12–6 L1ED	<p>Enable Level 1 Data Breakpoint. Setting an L1ED bit enables the corresponding data breakpoint condition based on the size and placement on the processor's local data bus. Clearing all L1ED bits disables data breakpoints.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th style="text-align: center;">TDR Bit</th><th style="text-align: center;">Description</th></tr> </thead> <tbody> <tr> <td style="text-align: center;">12</td><td>Data longword. Entire processor's local data bus.</td></tr> <tr> <td style="text-align: center;">11</td><td>Lower data word.</td></tr> <tr> <td style="text-align: center;">10</td><td>Upper data word.</td></tr> <tr> <td style="text-align: center;">9</td><td>Lower lower data byte. Low-order byte of the low-order word.</td></tr> <tr> <td style="text-align: center;">8</td><td>Lower middle data byte. High-order byte of the low-order word.</td></tr> <tr> <td style="text-align: center;">7</td><td>Upper middle data byte. Low-order byte of the high-order word.</td></tr> <tr> <td style="text-align: center;">6</td><td>Upper upper data byte. High-order byte of the high-order word.</td></tr> </tbody> </table>	TDR Bit	Description	12	Data longword. Entire processor's local data bus.	11	Lower data word.	10	Upper data word.	9	Lower lower data byte. Low-order byte of the low-order word.	8	Lower middle data byte. High-order byte of the low-order word.	7	Upper middle data byte. Low-order byte of the high-order word.	6	Upper upper data byte. High-order byte of the high-order word.
TDR Bit	Description																
12	Data longword. Entire processor's local data bus.																
11	Lower data word.																
10	Upper data word.																
9	Lower lower data byte. Low-order byte of the low-order word.																
8	Lower middle data byte. High-order byte of the low-order word.																
7	Upper middle data byte. Low-order byte of the high-order word.																
6	Upper upper data byte. High-order byte of the high-order word.																
5 L1DI	<p>Level 1 Data Breakpoint Invert. Inverts the logical sense of all the data breakpoint comparators. This can develop a trigger based on the occurrence of a data value other than the DBR contents.            0 No inversion            1 Invert data breakpoint comparators.</p>																

**Table 43-14. TDR Field Descriptions (continued)**

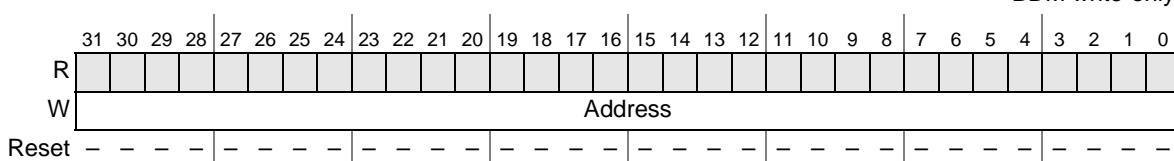
Field	Description	
4–2 L1EA	Enable Level 1 Address Breakpoint. Setting an L1EA bit enables the corresponding address breakpoint. Clearing all three bits disables the address breakpoint.	
	<b>TDR Bit</b>	<b>Description</b>
	4	Enable address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.
	3	Enable address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.
	2	Enable address breakpoint low. The breakpoint is based on the address in the ABLR.
1 L1EPC	Enable Level 1 PC breakpoint. 0 Disable PC breakpoint 1 Enable PC breakpoint	
0 L1PCI	Level 1 PC Breakpoint Invert. 0 The PC breakpoint is defined within the region defined by PBR $n$ and PBMR. 1 The PC breakpoint is defined outside the region defined by PBR $n$ and PBMR.	

### 43.3.8 Program Counter Breakpoint/Mask Registers (PBR0–3, PBMR)

The PBR $n$  registers define an instruction address for use as part of the trigger. These registers' contents are compared with the processor's program counter register when the appropriate valid bit is set (for PBR1–3) and TDR or XTDR are configured appropriately. PBR0 bits are masked by setting corresponding PBMR bits (PBMR has no effect on PBR1–3). Results are compared with the processor's program counter register, as defined in TDR or XTDR. Breakpoint registers, PBR1–3, have no masking associated with them. The contents of the breakpoint registers are compared with the processor's program counter register when TDR is configured appropriately.

The PC breakpoint registers are accessible in supervisor mode using the WDEBUG instruction and through the BDM port using the WDMREG command using values shown in [Section 43.4.1.5, “BDM Command Set”](#).

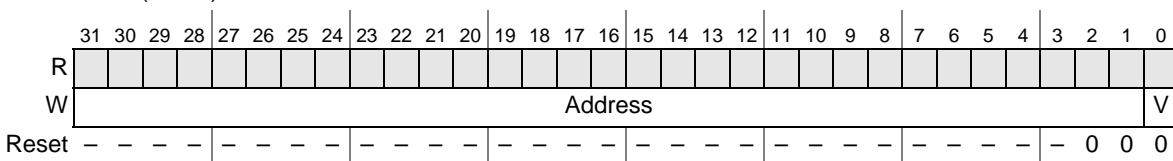
DRc[4:0]: 0x08 (PBR0)

Access: Supervisor write-only  
BDM write-only**Figure 43-9. PC Breakpoint Register (PBR0)**

**Table 43-15. PBR0 Field Descriptions**

Field	Description
31–0 Address	PC Breakpoint Address. The address to be compared with the PC as a breakpoint trigger. <b>Note:</b> PBR0[0] should always be loaded with a 0.

DRc[4:0]: 0x18 (PBR1)  
0x1A (PBR2)  
0x1B (PBR3) Access: Supervisor write-only  
BDM write-only



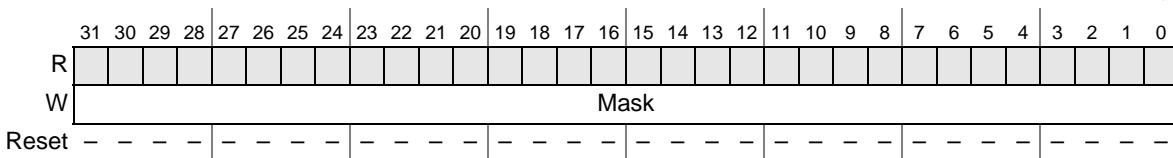
**Figure 43-10. PC Breakpoint Register  $n$  (PBR $n$ )**

**Table 43-16. PBR*n* Field Descriptions**

Field	Description
31–1 Address	PC Breakpoint Address. The 31-bit address to be compared with the PC as a breakpoint trigger.
0 V	Valid Bit. This bit must be set for the PC breakpoint to occur at the address specified in the Address field. 0 PBR is disabled. 1 PBR is enabled.

Figure 43-11 shows PBMR. PBMR is accessible in supervisor mode using the WDEBUG instruction and via the BDM port using the WDMREG command. PBMR only masks PBR0.

DRc[4:0]: 0x09 (PBMR) Access: Supervisor write-only  
BDM write-only



**Figure 43-11. PC Breakpoint Mask Register (PBMR)**

**Table 43-17. PBMR Field Descriptions**

Field	Description
31–0 Mask	PC Breakpoint Mask. 0 The corresponding PBR0 bit is compared to the appropriate PC bit. 1 The corresponding PBR0 bit is ignored.

### 43.3.9 PC Breakpoint ASID Control Register (PBAC)

The PBAC register configures the breakpoint qualification for each PC breakpoint register (PBR $n$ ). Four bits are dedicated for each breakpoint register and specify how the ASID is used in PC breakpoint qualification.

The four-bit field correspond directly to the PBRn registers and are functionally identical. They enable or disable ASID, supervisor mode, and user mode breakpoint qualification. Reset clears these fields, disabling qualifications, and defaulting to the revision C debug module functionality.

**Figure 43-12. PC Breakpoint ASID Control Register (PBAC)**

**Table 43-18. PBAC Field Descriptions**

Field	Description				
31–16	Reserved, must be cleared.				
15–12 PBR3AC 11–8 PBR2AC 7–4 PBR1AC 3–0 PBR0AC	PBR $n$ ASID control. Corresponds to the ASID control associated with PBR $n$ . Determines whether the ASID is included in the PC breakpoint comparison and whether the operating mode (supervisor or user) is included in the comparison logic.				
<b>PBR<math>n</math>AC[3] Not Used</b>	<b>PBR<math>n</math>AC[2] ASID Included</b>	<b>PBR<math>n</math>AC[1] Mode Qualification</b>	<b>PBR<math>n</math>AC[0] User or Supervisor</b>	<b>Description</b>	
	x	0	0	x	No ASID nor mode qualification
	x	0	1	0	No ASID qualification; user mode qualification enabled
	x	0	1	1	No ASID qualification; supervisor mode qualification enabled
	x	1	0	x	ASID qualification enabled; no mode qualification
	x	1	1	0	ASID and user mode qualification enabled
	x	1	1	1	ASID and supervisor mode qualification enabled

### 43.3.10 Address Breakpoint Registers (ABLR/ABLR1, ABHR/ABHR1)

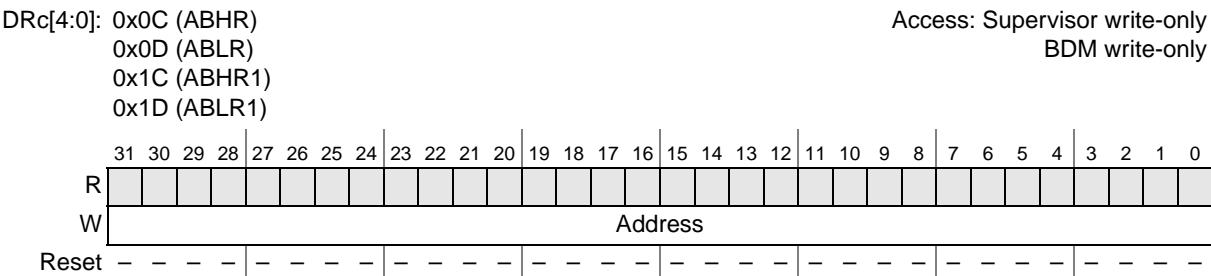
The ABLR, ABLR1, ABHR and ABHR1 define regions in the processor's data address space that can act as part of the trigger. These register values are compared with the address for each transfer on the processor's high-speed local bus. The trigger definition register (TDR) identifies the trigger as one of three cases:

- Identically the value in ABLR
  - Inside the range bound by ABLR and ABHR inclusive
  - Outside that same range

XTDR determines the same for ABLR1 and ABHR1.

## Debug Module

ABLR, ABLR1, ABHR and ABHR1 are accessible in supervisor mode using the WDEBUG instruction and via the BDM port using the WDMREG command.



**Figure 43-13. Address Breakpoint Registers (ABLR, ABHR, ABLR1, ABHR1)**

**Table 43-19. ABLR and ABLR1 Field Description**

Field	Description
31–0 Address	Low Address. Holds the 32-bit address marking the lower bound of the address breakpoint range. Breakpoints for specific single addresses are programmed into ABLR or ABLR1.

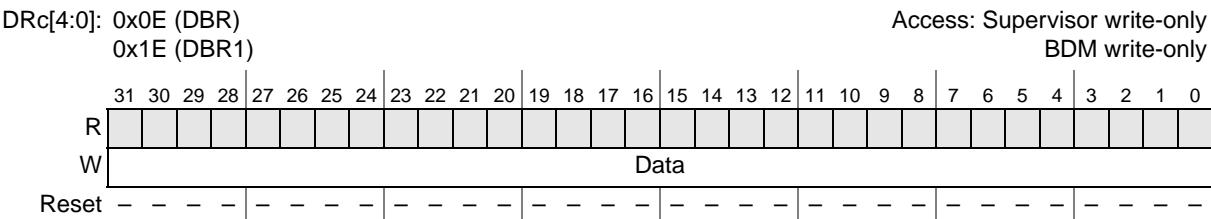
**Table 43-20. ABHR and ABHR1 Field Description**

Field	Description
31–0 Address	High Address. Holds the 32-bit address marking the upper bound of the address breakpoint range.

### 43.3.11 Data Breakpoint and Mask Registers (DBR/DBR1, DBMR/DBMR1)

The data breakpoint registers (DBR/DBR1), specify data patterns used as part of the trigger into debug mode. DBRn bits are masked by setting corresponding DBMR bits, as defined in TDR.

DBR, DBR1, DBMR, and DBMR1 are accessible in supervisor mode using the WDEBUG instruction and through the BDM port using the WDMREG command.



**Figure 43-14. Data Breakpoint Registers (DBR, DBR1)**

**Table 43-21. DBR, DBR1 Field Descriptions**

Field	Description
31–0 Data	Data Breakpoint Value. Contains the value to be compared with the data value from the processor's local bus as a breakpoint trigger.

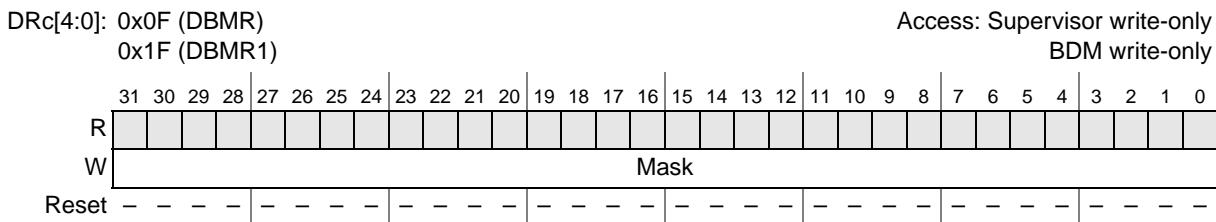


Figure 43-15. Data Breakpoint Mask Registers (DBMR, DBMR1)

Table 43-22. DBMR, DBMR1 Field Descriptions

Field	Description
31–0 Mask	Data Breakpoint Mask. The 32-bit mask for the data breakpoint trigger. Clearing a DBMR bit allows the corresponding DBR bit to be compared to the appropriate bit of the processor's local data bus. Setting a DBMR bit causes that bit to be ignored.

The DBRs support aligned and misaligned references. [Table 43-23](#) shows relationships between processor address, access size, and location within the 32-bit data bus.

Table 43-23. Address, Access Size, and Operand Data Location

Address[1:0]	Access Size	Operand Location
00	Byte	D[31:24]
01	Byte	D[23:16]
10	Byte	D[15:8]
11	Byte	D[7:0]
0x	Word	D[31:16]
1x	Word	D[15:0]
xx	Longword	D[31:0]

### 43.3.12 PC Breakpoint ASID Register (PBASID)

Each PC breakpoint register (PBR0–3) specifies an instruction address that can be used to trigger a breakpoint. To support debugging in a virtual environment, an ASID can optionally be associated with the instruction address in the PC breakpoint registers. The optional specification of an ASID value is made using PBASID and its exact inclusion within the breakpoint specification defined by the PBAC.

PBASID contains one 8-bit ASID values for each PC breakpoint register, as described in [Table 43-24](#), which allows each PC breakpoint register to be associated with a unique virtual address and process.

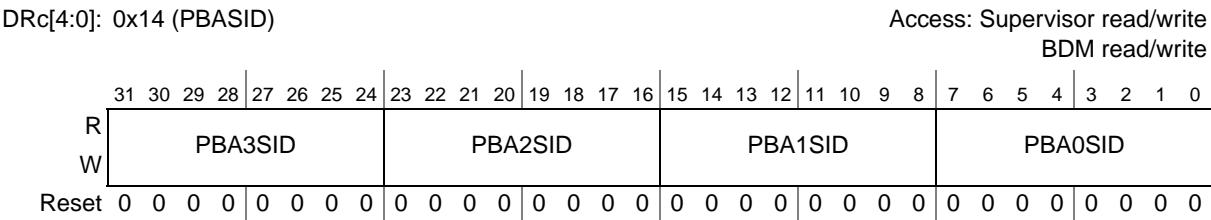


Figure 43-16. PC Breakpoint ASID Register (PBASID)

Table 43-24. PBASID Field Descriptions

Field	Description
31–24 PBR3ASID	Corresponds to the ASID associated with PBR3.
23–16 PBR2ASID	Corresponds to the ASID associated with PBR2.
15–8 PBR1ASID	Corresponds to the ASID associated with PBR1.
7–0 PBR0ASID	Corresponds to the ASID associated with PBR0.

### 43.3.13 Extended Trigger Definition Register (XTDR)

The XTDR configures the operation of the hardware breakpoint logic that corresponds with the ABHR1/ABLR1/AATTR1, DBR1/DBMR1 registers within the debug module. In conjunction with the TDR and its associated debug registers, XTDR controls the actions taken under the defined conditions. Breakpoint logic may be configured as a one- or two-level triggers. TDR[31–16] or XTDR[31–16] bits define second-level triggers, and bits 15–0 define first-level triggers.

#### NOTE

The debug module has no hardware interlocks; so to prevent spurious breakpoint triggers while the breakpoint registers are being loaded, disable TDR and XTDR (by clearing TDR[29,13] and XTDR[29,13]) before defining triggers.

A write to XTDR clears the CSR trigger status bits, CSR[BSTAT]. XTDR is accessible in supervisor mode using the WDEBUG instruction and through the BDM port using the WDMREG command.

[Section 43.3.13.1, “Resulting Set of Possible Trigger Combinations,”](#) describes how to handle multiple breakpoint conditions.

DRc[4:0]: 0x17 (XTDR)

Access: Supervisor write-only  
BDM write-only

Second Level Trigger																
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
W	0	0	L2EBL	L2ED						L2DI	L2EA			0	0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
First Level Trigger																
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	0	0	L1EBL	L1ED						L1DI	L1EA			0	0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 43-17. Extended Trigger Definition Register (XTDR)

Table 43-25. XTDR Field Descriptions

Field	Description																	
31–30	Reserved, must be cleared.																	
29 L2EBL	Enable level 2 breakpoint. Global enable for the breakpoint trigger. 0 Disables all level 2 breakpoints 1 Enables all level 2 breakpoint triggers																	
28–22 L2ED	Enable level 2 data breakpoint. Setting an L2ED bit enables the corresponding data breakpoint condition based on the size and placement on the processor's local data bus. Clearing all ED bits disables data breakpoints.																	
	<table border="1"> <thead> <tr> <th>TDR Bit</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>28</td> <td>Data longword. Entire processor's local data bus.</td> </tr> <tr> <td>27</td> <td>Lower data word.</td> </tr> <tr> <td>26</td> <td>Upper data word.</td> </tr> <tr> <td>25</td> <td>Lower lower data byte. Low-order byte of the low-order word.</td> </tr> <tr> <td>24</td> <td>Lower middle data byte. High-order byte of the low-order word.</td> </tr> <tr> <td>23</td> <td>Upper middle data byte. Low-order byte of the high-order word.</td> </tr> <tr> <td>22</td> <td>Upper upper data byte. High-order byte of the high-order word.</td> </tr> </tbody> </table>		TDR Bit	Description	28	Data longword. Entire processor's local data bus.	27	Lower data word.	26	Upper data word.	25	Lower lower data byte. Low-order byte of the low-order word.	24	Lower middle data byte. High-order byte of the low-order word.	23	Upper middle data byte. Low-order byte of the high-order word.	22	Upper upper data byte. High-order byte of the high-order word.
TDR Bit	Description																	
28	Data longword. Entire processor's local data bus.																	
27	Lower data word.																	
26	Upper data word.																	
25	Lower lower data byte. Low-order byte of the low-order word.																	
24	Lower middle data byte. High-order byte of the low-order word.																	
23	Upper middle data byte. Low-order byte of the high-order word.																	
22	Upper upper data byte. High-order byte of the high-order word.																	
21 L2DI	Level 2 data breakpoint invert. Inverts the logical sense of all the data breakpoint comparators. This can develop a trigger based on the occurrence of a data value other than the DBR contents. 0 No inversion 1 Invert data breakpoint comparators.																	

**Table 43-25. XTDR Field Descriptions (continued)**

<b>Field</b>	<b>Description</b>																	
20–18 L2EA	Enable level 2 address breakpoint. Setting an L2EA bit enables the corresponding address breakpoint. Clearing all three bits disables the breakpoint.																	
	<table border="1"> <thead> <tr> <th><b>TDR Bit</b></th><th><b>Description</b></th></tr> </thead> <tbody> <tr> <td>20</td><td>Address breakpoint inverted. Breakpoint is based outside the range between ABLR1 and ABHR1.</td></tr> <tr> <td>19</td><td>Address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR1 and ABHR1.</td></tr> <tr> <td>18</td><td>Address breakpoint low. The breakpoint is based on the address in the ABLR1.</td></tr> </tbody> </table>		<b>TDR Bit</b>	<b>Description</b>	20	Address breakpoint inverted. Breakpoint is based outside the range between ABLR1 and ABHR1.	19	Address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR1 and ABHR1.	18	Address breakpoint low. The breakpoint is based on the address in the ABLR1.								
<b>TDR Bit</b>	<b>Description</b>																	
20	Address breakpoint inverted. Breakpoint is based outside the range between ABLR1 and ABHR1.																	
19	Address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR1 and ABHR1.																	
18	Address breakpoint low. The breakpoint is based on the address in the ABLR1.																	
17–14	Reserved, must be cleared.																	
13 L1EBL	Enable level 1 breakpoint. Global enable for the breakpoint trigger. 0 Disables all level 1 breakpoints 1 Enables all level 1 breakpoint triggers																	
12–6 L1ED	Enable level 1 data breakpoint. Setting an L1ED bit enables the corresponding data breakpoint condition based on the size and placement on the processor's local data bus. Clearing all L1ED bits disables data breakpoints.																	
	<table border="1"> <thead> <tr> <th><b>TDR Bit</b></th><th><b>Description</b></th></tr> </thead> <tbody> <tr> <td>12</td><td>Data longword. Entire processor's local data bus.</td></tr> <tr> <td>11</td><td>Lower data word.</td></tr> <tr> <td>10</td><td>Upper data word.</td></tr> <tr> <td>9</td><td>Lower lower data byte. Low-order byte of the low-order word.</td></tr> <tr> <td>8</td><td>Lower middle data byte. High-order byte of the low-order word.</td></tr> <tr> <td>7</td><td>Upper middle data byte. Low-order byte of the high-order word.</td></tr> <tr> <td>6</td><td>Upper upper data byte. High-order byte of the high-order word.</td></tr> </tbody> </table>		<b>TDR Bit</b>	<b>Description</b>	12	Data longword. Entire processor's local data bus.	11	Lower data word.	10	Upper data word.	9	Lower lower data byte. Low-order byte of the low-order word.	8	Lower middle data byte. High-order byte of the low-order word.	7	Upper middle data byte. Low-order byte of the high-order word.	6	Upper upper data byte. High-order byte of the high-order word.
<b>TDR Bit</b>	<b>Description</b>																	
12	Data longword. Entire processor's local data bus.																	
11	Lower data word.																	
10	Upper data word.																	
9	Lower lower data byte. Low-order byte of the low-order word.																	
8	Lower middle data byte. High-order byte of the low-order word.																	
7	Upper middle data byte. Low-order byte of the high-order word.																	
6	Upper upper data byte. High-order byte of the high-order word.																	
5 L1DI	Level 1 data breakpoint invert. Inverts the logical sense of all the data breakpoint comparators. This can develop a trigger based on the occurrence of a data value other than the DBR contents. 0 No inversion 1 Invert data breakpoint comparators.																	

**Table 43-25. XTDR Field Descriptions (continued)**

Field	Description									
4–2 L1EA	Enable level 1 address breakpoint. Setting an L1EA bit enables the corresponding address breakpoint. Clearing all three bits disables the address breakpoint. <table border="1" style="margin-left: 20px;"> <thead> <tr> <th style="text-align: center;">TDR Bit</th> <th style="text-align: center;">Description</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">4</td> <td>Enable address breakpoint inverted. Breakpoint is based outside the range between ABLR1 and ABHR1.</td> </tr> <tr> <td style="text-align: center;">3</td> <td>Enable address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR1 and ABHR1.</td> </tr> <tr> <td style="text-align: center;">2</td> <td>Enable address breakpoint low. The breakpoint is based on the address in the ABLR1.</td> </tr> </tbody> </table>		TDR Bit	Description	4	Enable address breakpoint inverted. Breakpoint is based outside the range between ABLR1 and ABHR1.	3	Enable address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR1 and ABHR1.	2	Enable address breakpoint low. The breakpoint is based on the address in the ABLR1.
TDR Bit	Description									
4	Enable address breakpoint inverted. Breakpoint is based outside the range between ABLR1 and ABHR1.									
3	Enable address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR1 and ABHR1.									
2	Enable address breakpoint low. The breakpoint is based on the address in the ABLR1.									
1–0	Reserved, must be cleared.									

### 43.3.13.1 Resulting Set of Possible Trigger Combinations

The resulting set of possible breakpoint trigger combinations consist of the following options where || denotes logical OR, && denotes logical AND, and {} denotes an optional additional trigger term:

One-level triggers of the form:

```

if      (PC_breakpoint)
if      (PC_breakpoint    || Address_breakpoint{&& Data_breakpoint})
if      (PC_breakpoint    || Address_breakpoint{&& Data_breakpoint}
           || Address1_breakpoint {&& Data1_breakpoint})

if (Address_breakpoint  {&& Data_breakpoint})
if ((Address_breakpoint {&& Data_breakpoint}
           || (Address1_breakpoint{&& Data1_breakpoint})))

if      (Address1_breakpoint {&& Data1_breakpoint})

```

Two-level triggers of the form:

```

if      (PC_breakpoint)
      then if          (Address_breakpoint{&& Data_breakpoint})

if      (PC_breakpoint)
      then if          (Address_breakpoint{&& Data_breakpoint}
                         || Address1_breakpoint{&& Data1_breakpoint})

if      (PC_breakpoint)
      then if          (Address1_breakpoint{&& Data1_breakpoint})

if      (Address_breakpoint {&& Data_breakpoint})
      then if          (Address1_breakpoint{&& Data1_breakpoint})

if      (Address1_breakpoint {&& Data1_breakpoint})
      then if          (Address_breakpoint{&& Data_breakpoint})

if      (Address_breakpoint {&& Data_breakpoint})

```

## Debug Module

```

        then if          (PC_breakpoint)

if      (Address1_breakpoint      {&&  Data1_breakpoint})
then if          (PC_breakpoint)

if      (Address_breakpoint      {&&  Data_breakpoint})
then if          (PC_breakpoint
||           Address1_breakpoint{&&  Data1_breakpoint})

if      (Address1_breakpoint      {&&  Data1_breakpoint})
then if          (PC_breakpoint||  Address_breakpoint{&&  Data_breakpoint})

```

In this example, PC\_breakpoint is the logical summation of the PBR0/PBMR, PBR1, PBR2, and PBR3 breakpoint registers; Address\_breakpoint is a function of ABHR, ABLR, and AATR; Data\_breakpoint is a function of DBR and DBMR; Address1\_breakpoint is a function of ABHR1, ABLR1, and AATR1; and Data1\_breakpoint is a function of DBR1 and DBMR1. In all cases, the data breakpoints can be included with an address breakpoint to further qualify a trigger event as an option.

### 43.3.14 PST Trace Buffer Longwords (TBLW0–23)

The contents of the PST trace buffer are mapped into the debug register programming model at consecutive addresses beginning at 0x20. Use the RDMREG BDM command to read 32 bits of captured PST/DDATA values from the trace buffer at the specified address. The PST trace buffer contains 128 six-bit entries, packed consecutively into 24 longword locations (TBLW0–23). See [Figure 43-18](#) for an illustration of how the trace buffer entries (TB0–127) are packed.

The write pointer for the trace buffer is available as CSR2[PSTBWA]. Using this pointer, it is possible to determine the oldest-to-newest entries in the trace buffer.

CRN	TBLW <sub>n</sub>	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
0x20	TBLW0																																
0x21	TBLW1																																
0x22	TBLW2																																
0x23	TBLW3																																
0x24	TBLW4																																
0x25	TBLW5																																
...	...																																
0x35	TBLW21																																
0x36	TBLW22																																
0x37	TBLW23																																

**Figure 43-18. PST Trace Buffer Entries and Locations**

Each 6-bit entry includes a 1- or 2-bit prefix so the type of the entry can easily be determined when post-processing the PSTB.

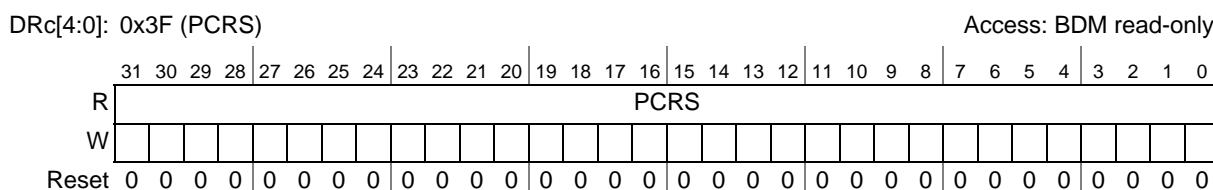
	5	4	3	2	1	0
PSTB[PST]	0		PST[4:0]			
Data PSTB[DDATA]	1	$\overline{R/W}$		Data[3:0]		
Address PSTB[DDATA]	1	0		Address[3:0] <sup>1</sup>		
Reset:	—	—	—	—	—	—

1 Depending on which nibble is displayed (as determined by CSR[9:8]), address[3:0] sequentially displays four bits of the real CPU address [16:1], [24:1], or [0,31:1].

**Figure 43-19. PST/DDATA Trace Buffer Entry Format**

### 43.3.15 Most Recently Sampled PC (PCRS)

In PC profiling recording mode, a BDM read at address 0x3F (PCRS) returns the most recently sampled PC. Each time the PC is sampled and loaded into the PSTB, the full 32-bit PC is also loaded into PCRS. You can use the RDMREG BDM command (see [Section 43.4.1.5.15, “Read Debug Module Register \(rdmreg\)”](#)) at any time including in halt mode. This provides real-time PC profiling that is minimally intrusive to the system.



**Figure 43-20. Most Recently Sampled PC (PCRS)**

**Table 43-26. PCRS Field Descriptions**

Field	Description
31–0 PCRS	Most recently sampled PC. Available only through the RDMREG command.

## 43.4 Functional Description

#### **43.4.1 Background Debug Mode (BDM)**

The ColdFire family implements a low-level system debugger in the microprocessor in a dedicated hardware module. Communication with the development system is managed through a dedicated, high-speed serial command interface. Although some BDM operations, such as CPU register accesses, require the CPU to be halted, other BDM commands, such as memory accesses, can be executed while the processor is running.

BDM is useful because:

- In-circuit emulation is not needed, so physical and electrical characteristics of the system are not affected.
- BDM is always available for debugging the system and provides a communication link for upgrading firmware in existing systems.
- Provides high-speed cache downloading (500 Kbytes/sec), especially useful for flash programming
- Provides absolute control of the processor, and thus the system. This feature allows quick hardware debugging with the same tool set used for firmware development.

#### 43.4.1.1 CPU Halt

Although most BDM operations can occur in parallel with CPU operations, unrestricted BDM operation requires the CPU to be halted. The sources that can cause the CPU to halt are listed below in order of priority:

1. A catastrophic fault-on-fault condition automatically halts the processor.
2. A hardware breakpoint trigger can generate a pending halt condition similar to the assertion of  **$\overline{BKPT}$** . This type of halt is always first marked as pending in the processor, which samples for pending halt and interrupt conditions once per instruction. When a pending condition is asserted, the processor halts execution at the next sample point. See [Section 43.4.2.1, “Theory of Operation”](#).
3. The execution of a HALT instruction immediately suspends execution. Attempting to execute HALT in user mode while CSR[UHE] is cleared generates a privilege violation exception. If CSR[UHE] is set, HALT can be executed in user mode. After HALT executes, the processor can be restarted by serial shifting a GO command into the debug module. Execution continues at the instruction after HALT.
4. The assertion of the  **$\overline{BKPT}$**  input is treated as a pseudo-interrupt; asserting  **$\overline{BKPT}$**  creates a pending halt postponed until the processor core samples for halts/interrupts. The processor samples for these conditions once during the execution of each instruction; if a pending halt is detected, the processor suspends execution and enters the halted state.

There are two special cases involving the assertion of  **$\overline{BKPT}$** :

- After the system reset signal is negated, the processor waits for 16 processor clock cycles before beginning reset exception processing. If the  **$\overline{BKPT}$**  input is asserted within eight cycles after RESET is negated, the processor enters the halt state, signaling halt status (0xF) on the PSTDDATA outputs. While the processor is in this state, all resources accessible through the debug module can be referenced. This is the only chance to force the processor into emulation mode through CSR[EMU].
- After system initialization, the processor’s response to the GO command depends on the set of BDM commands performed while it is halted for a breakpoint. Specifically, if the PC register was loaded, the GO command causes the processor to exit halted state and pass control to the instruction address in the PC, bypassing normal reset exception processing. If the PC was not loaded, the GO command causes the processor to exit halted state and continue reset exception processing.

- The ColdFire architecture also manages a special case of  $\overline{\text{BKPT}}$  asserted while the processor is stopped by execution of the STOP instruction. For this case, the processor exits the stopped mode and enters the halted state, at which point all BDM commands may be exercised. When restarted, the processor continues by executing the next sequential instruction, which follows the STOP opcode.

The CSR[27–24] bits indicate the halt source, showing the highest priority source for multiple halt conditions. Debug module revisions A and B clear CSR[27–24] upon a read of the CSR, but revision C and D (in V4) do not. The debug GO command clears CSR[26–24].

HALT can be recognized by counting 0xFF occurrences on PSTDDATA. The count is necessary to determine between a possible data output value of 0xFF and the HALT condition. Because data always follows a marker (0x8, 0x9, 0xA, or 0xB), PSTDDATA can display no more than four data 0xFFs. Two such scenarios exist:

- A B marker occurs on the left nibble of PSTDDATA with the data of 0xFF following:

```
PSTDDATA[7:0]
0xBF
0xFF
0xFF
0xFF
0xFF
0xFX (X indicates that the next PST value is guaranteed to not be 0xF)
```

- A B marker occurs on the right nibble of PSTDDATA with the data of 0xFF following:

```
PSTDDATA[7:0]
0xYB
0xFF
0xFF
0xFF
0xFF
0xFF
0XY (X indicates that the PST value is guaranteed to not be 0xF, and Y indicates a PSTDDATA value that doesn't affect the 0xFF count).
```

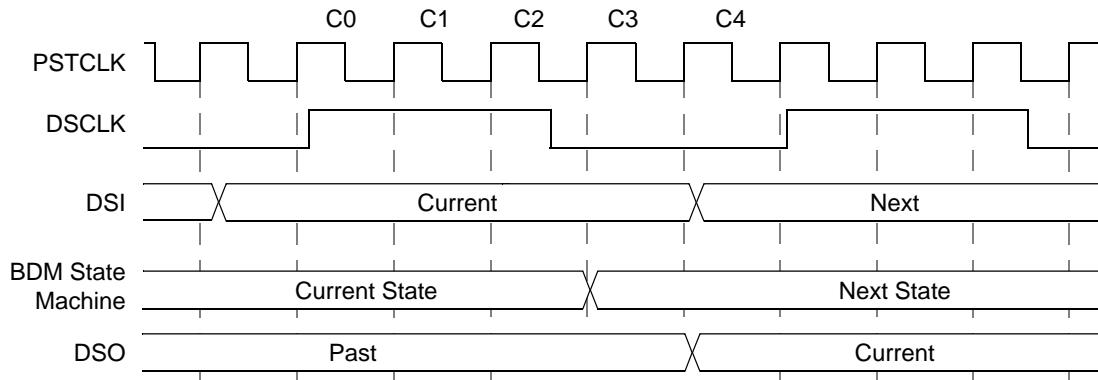
Thus, a count of nine or more sequential 0xF PSTDDATA nibbles or five or more sequential 0xFF PSTDDATA bytes signifies the HALT condition.

#### 43.4.1.2 BDM Serial Interface

When the CPU is halted and PSTDDATA reflects the halt status, the development system can send unrestricted commands to the debug module. The debug module implements a synchronous serial protocol using two inputs (DSCLK and DSI) and one output (DSO), where DSO is specified as a delay relative to the rising edge of the processor clock. See [Table 43-2](#). The development system serves as the serial communication channel master and must generate DSCLK.

The serial channel operates at a frequency from DC to 1/5 of the PSTCLK frequency. The channel uses full-duplex mode, where data is sent and received simultaneously by master and slave devices. The transmission consists of 17-bit packets composed of a status/control bit and a 16-bit data word. As shown

in Figure 43-21, all state transitions are enabled on a rising edge of the PSTCLK clock when DSCLK is high; DS1 is sampled and DSO is driven.



**Figure 43-21. Maximum BDM Serial Interface Timing**

DSCLK and DS1 are synchronized inputs. DSCLK acts as a pseudo clock enable and is sampled, along with DS1, on the rising edge of PSTCLK. DSO is delayed from the DSCLK-enabled PSTCLK rising edge (registered after a BDM state machine state change). All events in the debug module's serial state machine are based on the PSTCLK rising edge. DSCLK must also be sampled low (on a positive edge of PSTCLK) between each bit exchange. The msb is sent first. Because DSO changes state based on an internally recognized rising edge of DSCLK, DSO cannot be used to indicate the start of a serial transfer. The development system must count clock cycles in a given transfer. C0–C4 are described as:

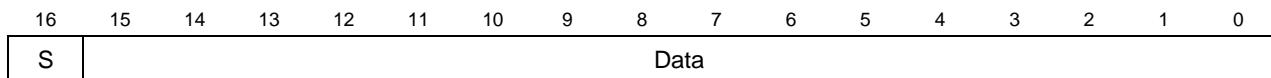
- C0: Set the state of the DS1 bit
- C1: First synchronization cycle for DS1 (DSCLK is high)
- C2: Second synchronization cycle for DS1 (DSCLK is high)
- C3: BDM state machine changes state depending upon DS1 and whether the entire input data transfer has been transmitted
- C4: DSO changes to next value

#### NOTE

A not-ready response can be ignored except during a memory-referencing cycle. Otherwise, the debug module can accept a new serial transfer after 32 processor clock periods.

#### 43.4.1.3 Receive Packet Format

The basic receive packet consists of 16 data bits and 1 status bit



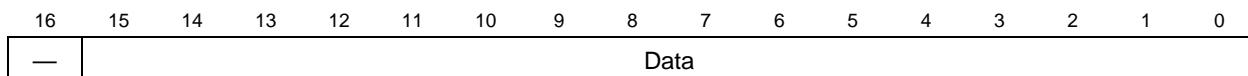
**Figure 43-22. Receive BDM Packet**

**Table 43-27. Receive BDM Packet Field Description**

Field	Description																				
16 S	<p>Status. Indicates the status of CPU-generated messages listed below. The not-ready response can be ignored unless a memory-referencing cycle is in progress. Otherwise, the debug module can accept a new serial transfer after 32 processor clock periods.</p> <table border="1"> <thead> <tr> <th>S</th> <th>Data</th> <th>Message</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>xxxx</td> <td>Valid data transfer</td> </tr> <tr> <td>0</td> <td>FFFF</td> <td>Status OK</td> </tr> <tr> <td>1</td> <td>0000</td> <td>Not ready with response; come again</td> </tr> <tr> <td>1</td> <td>0001</td> <td>Error-Terminated bus cycle; data invalid</td> </tr> <tr> <td>1</td> <td>FFFF</td> <td>Illegal Command</td> </tr> </tbody> </table>			S	Data	Message	0	xxxx	Valid data transfer	0	FFFF	Status OK	1	0000	Not ready with response; come again	1	0001	Error-Terminated bus cycle; data invalid	1	FFFF	Illegal Command
S	Data	Message																			
0	xxxx	Valid data transfer																			
0	FFFF	Status OK																			
1	0000	Not ready with response; come again																			
1	0001	Error-Terminated bus cycle; data invalid																			
1	FFFF	Illegal Command																			
15–0 Data	Data. Contains the message to be sent from the debug module to the development system. The response message is always a single word, with the data field encoded as shown above.																				

**43.4.1.3.1 Transmit Packet Format**

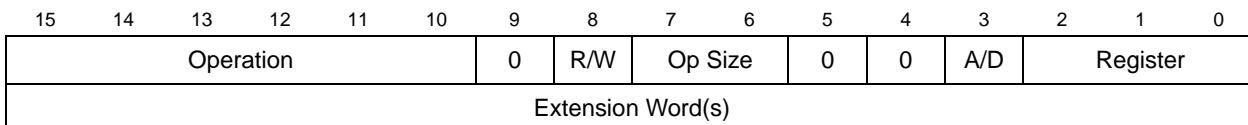
The basic transmit packet consists of 16 data bits and 1 reserved bit.

**Figure 43-23. Transmit BDM Packet****Table 43-28. Transmit BDM Packet Field Description**

Field	Description		
16	Reserved, must be cleared.		
15–0 Data	Data bits 15–0. Contains the data to be sent from the development system to the debug module.		

**43.4.1.3.2 BDM Command Format**

All ColdFire family BDM commands include a 16-bit operation word followed by an optional set of one or more extension words.

**Figure 43-24. BDM Command Format**

**Table 43-29. BDM Field Descriptions**

<b>Field</b>	<b>Description</b>															
15–10 Operation	Specifies the command. These values are listed in <a href="#">Table 43-30</a> .															
9	Reserved, must be cleared.															
8 R/W	Direction of operand transfer. 0 Data is written to the CPU or to memory from the development system. 1 The transfer is from the CPU to the development system.															
7–6 Op Size	Operand Data Size for Sized Operations. Addresses are expressed as 32-bit absolute values. A command performing a byte-sized memory read leaves the upper 8 bits of the response data undefined. Referenced data is returned in the lower 8 bits of the response.  <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th></th> <th><b>Operand Size</b></th> <th><b>Bit Values</b></th> </tr> </thead> <tbody> <tr> <td>00</td> <td>Byte</td> <td>8 bits</td> </tr> <tr> <td>01</td> <td>Word</td> <td>16 bits</td> </tr> <tr> <td>10</td> <td>Longword</td> <td>32 bits</td> </tr> <tr> <td>11</td> <td>Reserved</td> <td>—</td> </tr> </tbody> </table>		<b>Operand Size</b>	<b>Bit Values</b>	00	Byte	8 bits	01	Word	16 bits	10	Longword	32 bits	11	Reserved	—
	<b>Operand Size</b>	<b>Bit Values</b>														
00	Byte	8 bits														
01	Word	16 bits														
10	Longword	32 bits														
11	Reserved	—														
5–4	Reserved, must be cleared.															
3 A/D Register	Address/Data. Determines whether the register field specifies a data or address register. 0 Data register. 1 Address register.															
2–0 Register	Contains the register number in commands that operate on processor registers. See <a href="#">Table 43-31</a> .															

#### 43.4.1.3.3 Extension Words as Required

Some commands require extension words for addresses and/or immediate data. Addresses require two extension words because only absolute long addressing is permitted. Longword accesses are forcibly longword-aligned and word accesses are forcibly word-aligned. Immediate data can be 1 or 2 words long. Byte and word data each requires a single extension word, while longword data requires two extension words.

Operands and addresses are transferred most-significant word first. In the following descriptions of the BDM command set, the optional set of extension words is defined as address, data, or operand data.

#### 43.4.1.4 Command Sequence Diagrams

The command sequence diagram in [Figure 43-25](#) shows serial bus traffic for commands. Each bubble represents a 17-bit bus transfer. The top half of each bubble indicates the data the development system sends to the debug module; the bottom half indicates the debug module's response to the previous development system commands. Command and result transactions overlap to minimize latency.

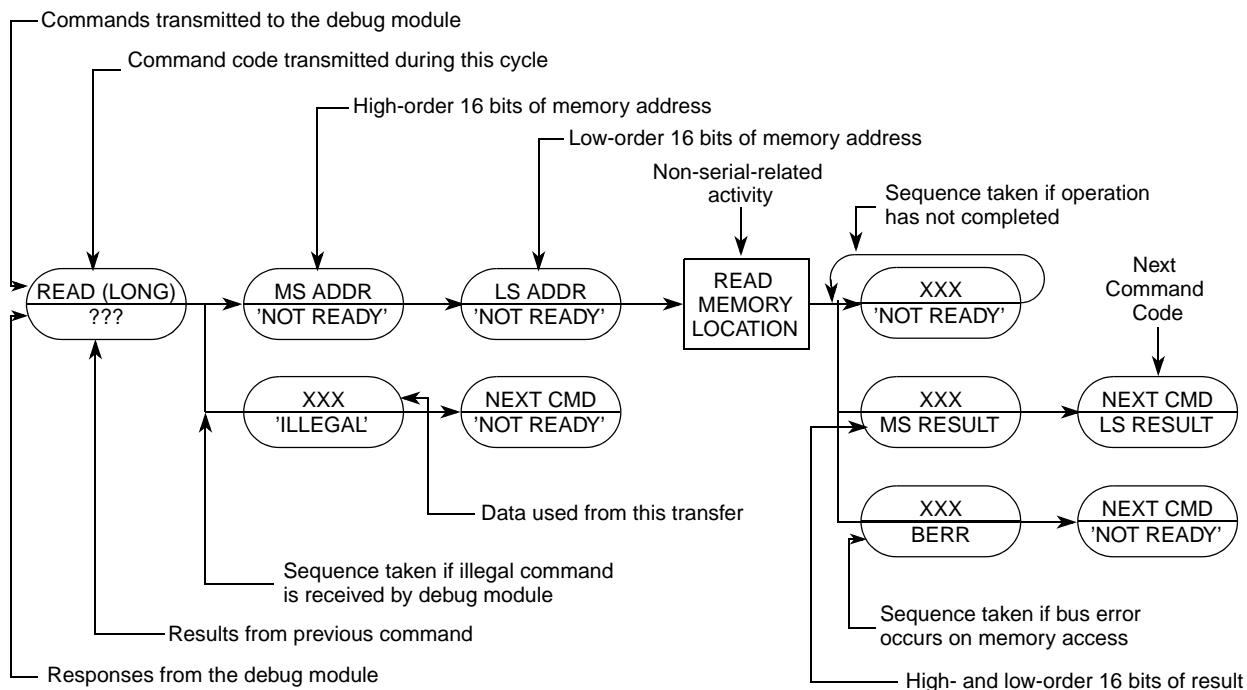


Figure 43-25. Command Sequence Diagram

The sequence is as follows:

- In cycle 1, the development system command is issued (READ in this example). The debug module responds with the low-order results of the previous command or a command complete status of the previous command, if no results are required.
- In cycle 2, the development system supplies the high-order 16 address bits. The debug module returns a not-ready response unless the received command is decoded as unimplemented, which is indicated by the illegal command encoding. If this occurs, the development system should retransmit the command.

#### NOTE

A not-ready response can be ignored except during a memory-referencing cycle. Otherwise, the debug module can accept a new serial transfer after 32 processor clock periods.

- In cycle 3, the development system supplies the low-order 16 address bits. The debug module always returns a not-ready response.
- At the completion of cycle 3, the debug module initiates a memory read operation. Any serial transfers that begin during a memory access return a not-ready response.
- Results are returned in the two serial transfer cycles after the memory access completes. For any command performing a byte-sized memory read operation, the upper 8 bits of the response data are undefined and the referenced data is returned in the lower 8 bits. The next command's opcode is sent to the debug module during the final transfer. If a bus error terminates a memory or register access, error status (S = 1, DATA = 0x0001) returns instead of result data.

### 43.4.1.5 BDM Command Set

[Table 43-30](#) summarizes the BDM command set. Subsequent sections contain detailed descriptions of each command. Issuing a BDM command when the processor is accessing debug module registers using the WDEBUG instruction causes undefined behavior. See [Table 43-31](#) for register address encodings.

**Table 43-30. BDM Command Summary**

Command	Mnemonic	Description	CPU State <sup>1</sup>	Section/Page	Command (Hex)
Read A/D register	RAREG/RDREG	Read the selected address or data register and return the results through the serial interface.	Halted	<a href="#">43.4.1.5.1/43-39</a>	0x218 {A/D, Reg[2:0]}
Write A/D register	WAREG/WDREG	Write the data operand to the specified address or data register.	Halted	<a href="#">43.4.1.5.2/43-39</a>	0x208 {A/D, Reg[2:0]}
Read memory location	READ	Read the data at the memory location specified by the longword address.	Steal	<a href="#">43.4.1.5.3/43-40</a>	0x1900—byte 0x1940—word 0x1980—lword
Write memory location	WRITE	Write the operand data to the memory location specified by the longword address.	Steal	<a href="#">43.4.1.5.4/43-41</a>	0x1800—byte 0x1840—word 0x1880—lword
Dump memory block	DUMP	Used with READ to dump large blocks of memory. An initial READ executes to set up the starting address of the block and to retrieve the first result. A DUMP command retrieves subsequent operands.	Steal	<a href="#">43.4.1.5.5/43-43</a>	0x1D00—byte 0x1D40—word 0x1D80—lword
Fill memory block	FILL	Used with WRITE to fill large blocks of memory. An initial WRITE executes to set up the starting address of the block and to supply the first operand. A FILL command writes subsequent operands.	Steal	<a href="#">43.4.1.5.6/43-45</a>	0x1C00—byte 0x1C40—word 0x1C80—lword
Resume execution	GO	The pipeline is flushed and refilled before resuming instruction execution at the current PC.	Halted	<a href="#">43.4.1.5.7/43-46</a>	0x0C00
No operation	NOP	Perform no operation; may be used as a null command.	Parallel	<a href="#">43.4.1.5.8/43-47</a>	0x0000
Output the current PC	SYNC_PC	Capture the current PC and display it on the PSTDDATA outputs.	Parallel	<a href="#">43.4.1.5.9/43-47</a>	0x0001
Read control register	RCREG	Read the system control register.	Halted	<a href="#">43.4.1.5.11/43-49</a>	0x2980
Write control register	WCREG	Write the operand data to the system control register.	Halted	<a href="#">43.4.1.5.14/43-52</a>	0x2880
Read debug module register	RDMREG	Read the debug module register.	Parallel	<a href="#">43.4.1.5.15/43-53</a>	0x2D {0x4 <sup>2</sup> DRc[4:0]}
Write debug module register	WDMREG	Write the operand data to the debug module register.	Parallel	<a href="#">43.4.1.5.16/43-54</a>	0x2C {0x4 <sup>2</sup> DRc[4:0]}

<sup>1</sup> General command effect and/or requirements on CPU operation:

- Halted: The CPU must be halted to perform this command.
- Steal: Command generates bus cycles that can be interleaved with bus accesses.
- Parallel: Command is executed in parallel with CPU activity.

<sup>2</sup> 0x4 is a three-bit field.

Freescale reserves unassigned command opcodes. All unused command formats within any revision level perform a NOP and return the illegal command response.

The following sections describe the commands summarized in [Table 43-30](#).

### NOTE

The BDM status bit (S) is 0 for normally completed commands. S is set for illegal commands, not-ready responses, and transfers with bus-errors.

[Section 43.4.1.2, “BDM Serial Interface,”](#) describes the receive packet format.

#### 43.4.1.5.1 Read A/D Register (RAREG/RDREG)

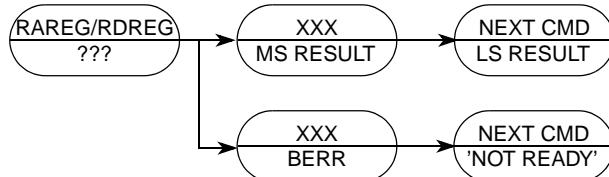
Read the selected address or data register and return the 32-bit result. A bus error response is returned if the CPU core is not halted.

Command/Result Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Command				0x2				0x1				0x8		A/D		Register
Result									D[31:16]							
									D[15:0]							

**Figure 43-26. RAREG/RDREG Command Format**

Command Sequence:



**Figure 43-27. RAREG/RDREG Command Sequence**

Operand Data: None

Result Data: The contents of the selected register are returned as a longword value, most-significant word first.

#### 43.4.1.5.2 Write A/D Register (WAREG/WDREG)

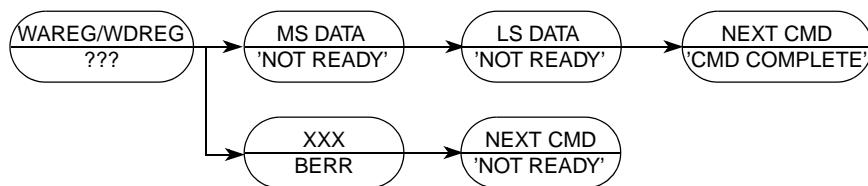
The operand longword data is written to the specified address or data register. A write alters all 32 register bits. A bus error response is returned if the CPU core is not halted.

Command Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x2				0x0				0x8		A/D		Register			
D[31:16]															
D[15:0]															

**Figure 43-28. WAREG/WDREG Command Format**

Command Sequence:

**Figure 43-29. WAREG/WDREG Command Sequence**

Operand Data:

Longword data is written into the specified address or data register. The data is supplied most-significant word first.

Result Data:

Command complete status is indicated by returning 0xFFFF (with S cleared) when the register write is complete.

#### 43.4.1.5.3 Read Memory Location (READ)

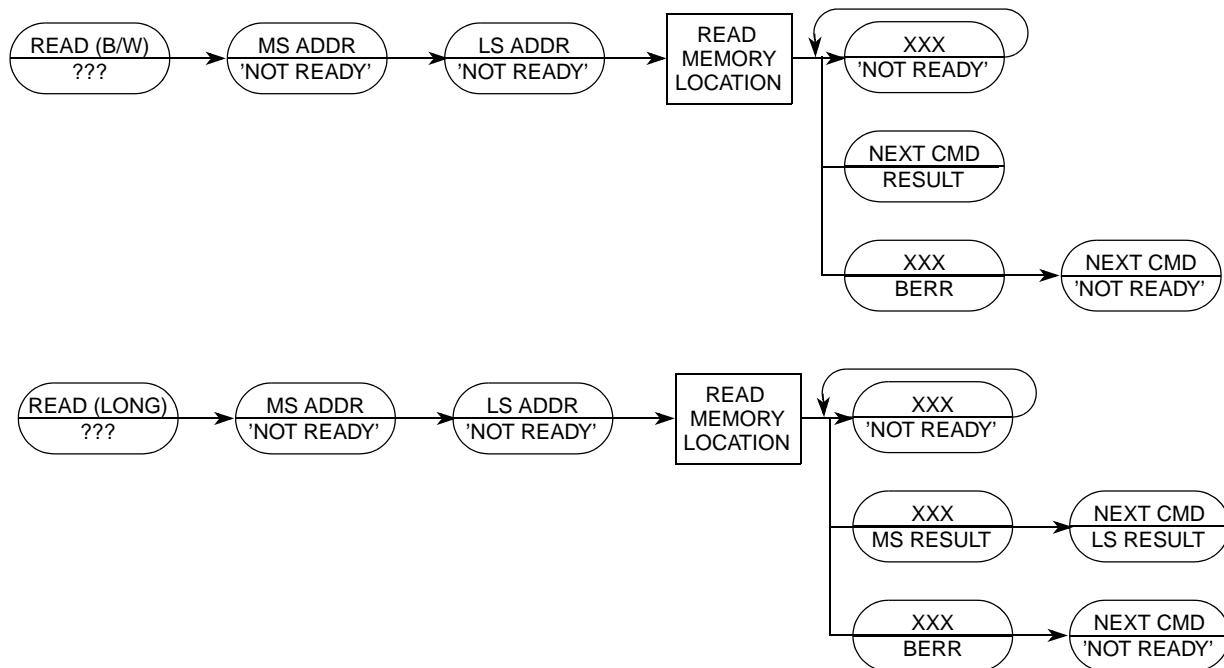
Read data at the longword address. Address space is defined by BAAR[TT,TM]. Hardware forces low-order address bits to 0s for word and longword accesses to ensure that word addresses are word-aligned and longword addresses are longword-aligned.

Command/Result Formats:

Byte	Command	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		0x1			0x9			0x0								0x0	
A[31:16]																	
A[15:0]																	
Word	Command	X	X	X	X	X	X	X	X					D[7:0]			
		0x1			0x9			0x4						0x0			
A[31:16]																	
A[15:0]																	
Longword	Command	D[15:0]															
		0x1			0x9			0x8								0x0	
A[31:16]																	
A[15:0]																	
	Result	D[31:16]															
D[15:0]																	

**Figure 43-30. READ Command/Result Formats**

Command Sequence:



**Figure 43-31. READ Command Sequence**

Operand Data:

The only operand is the longword address of the requested location.

Result Data:

Word results return 16 bits of data; longword results return 32. Bytes are returned in the LSB of a word result; the upper byte is undefined. 0x0001 (S = 1) is returned if a bus error occurs.

#### 43.4.1.5.4 Write Memory Location (WRITE)

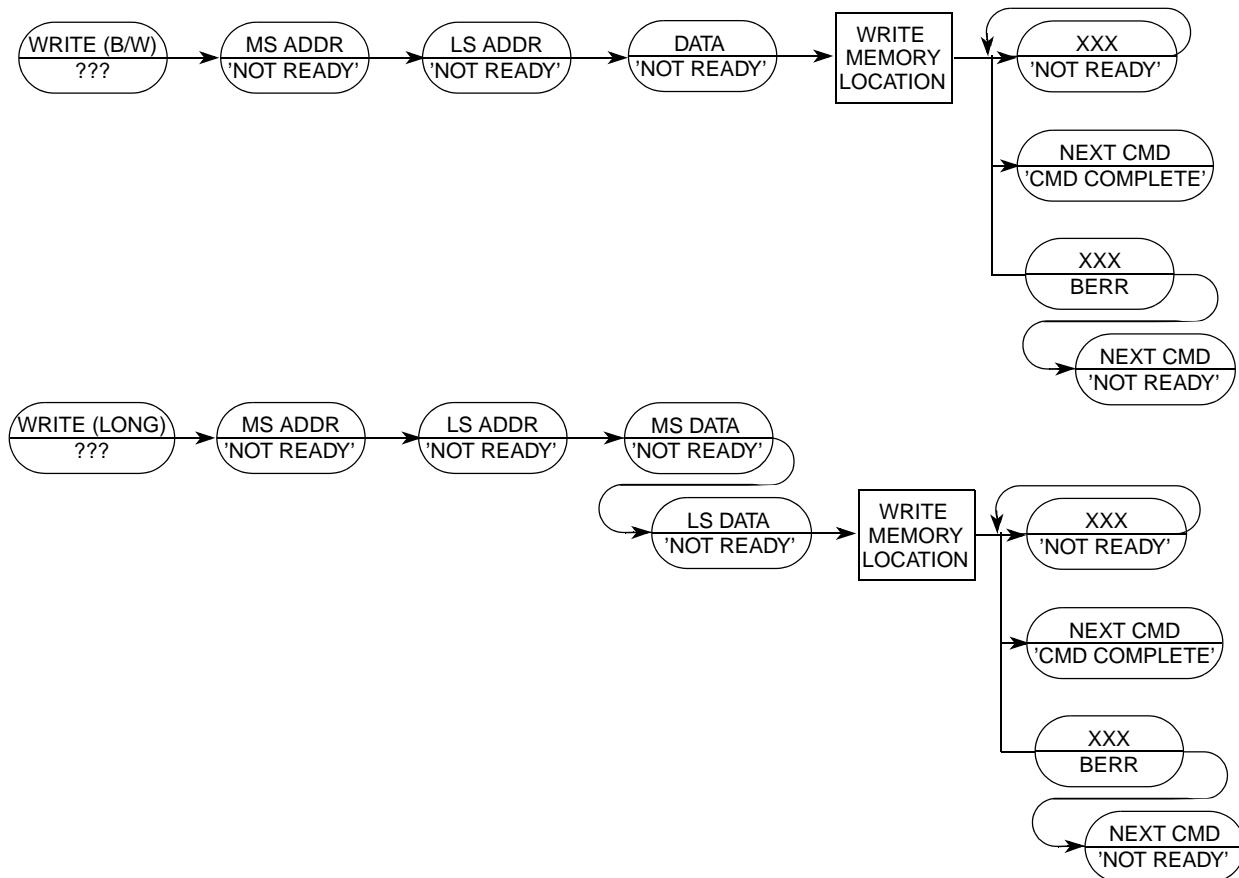
Write data to the memory location specified by the longword address. BAAR[TT,TM] defines address space. Hardware forces low-order address bits to 0s for word and longword accesses to ensure that word addresses are word-aligned and longword addresses are longword-aligned.

Command Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Byte	0x1							0x8				0x0			0x0	
	A[31:16]															
	A[15:0]															
	X	X	X	X	X	X	X	X							D[7:0]	
Word	0x1							0x8				0x4			0x0	
	A[31:16]															
	A[15:0]															
	D[15:0]															
Longword	0x1							0x8				0x8			0x0	
	A[31:16]															
	A[15:0]															
	D[31:16]															
	D[15:0]															

Figure 43-32. WRITE Command Format

Command Sequence:



**Figure 43-33. WRITE Command Sequence**

**Operand Data:**

This two-operand instruction requires a longword absolute address that specifies a location the data operand is written. Byte data is sent as a 16-bit word, justified in the LSB; 16- and 32-bit operands are sent as 16 and 32 bits, respectively.

**Result Data:**

Command complete status is indicated by returning 0xFFFF (with S cleared) when the register write is complete. A value of 0x0001 (with S set) is returned if a bus error occurs.

#### 43.4.1.5.5 Dump Memory Block (DUMP)

DUMP is used with the READ command to access large blocks of memory. An initial READ is executed to set up the starting address of the block and to retrieve the first result. If an initial READ is not executed before the first DUMP, an illegal command response is returned. The DUMP command retrieves subsequent operands. The initial address increments by the operand size (1, 2, or 4) and saves in a temporary register. Subsequent DUMP commands use this address, perform the memory read, increment it by the current operand size, and store the updated address in the temporary register.

**NOTE**

DUMP does not check for a valid address; it is a valid command only when preceded by NOP, READ, or another DUMP command. Otherwise, an illegal command response is returned. NOP can be used for intercommand padding without corrupting the address pointer.

The size field is examined each time a DUMP command is processed, allowing the operand size to be dynamically altered.

Command/Result Formats:

Byte	Command	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
	Result	X	X	X	X	X	X	X	X							D[7:0]				
Word	Command	0x1				0xD				0x0				0x0						
	Result	D[15:0]																		
Longword	Command	0x1				0xD				0x8				0x0						
	Result	D[31:16]																		
		D[15:0]																		

Figure 43-34. DUMP Command/Result Formats

Command Sequence:

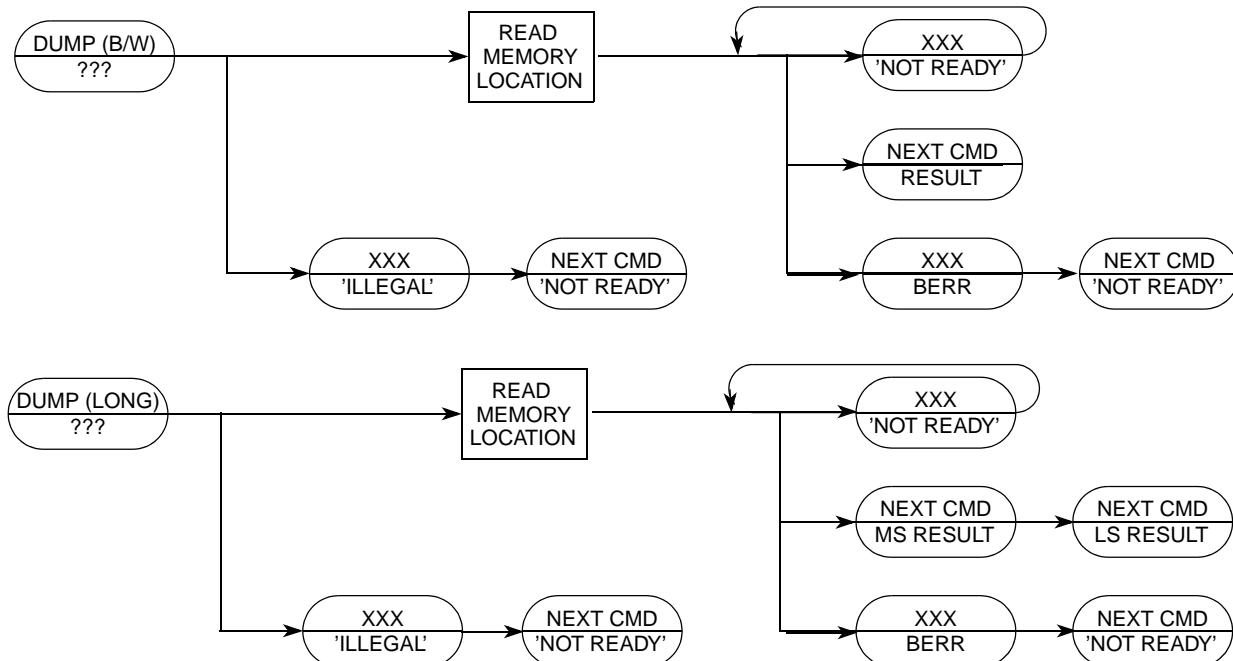


Figure 43-35. DUMP Command Sequence

Operand Data: None

**Result Data:**

Requested data is returned as a word or longword. Byte data is returned in the least-significant byte of a word result. Word results return 16 bits of significant data; longword results return 32 bits. A value of 0x0001 (with S set) is returned if a bus error occurs.

**43.4.1.5.6 Fill Memory Block (FILL)**

A FILL command is used with the WRITE command to access large blocks of memory. An initial WRITE is executed to set up the starting address of the block and to supply the first operand. The FILL command writes subsequent operands. The initial address increments by the operand size (1, 2, or 4) and saves in a temporary register after the memory write. Subsequent FILL commands use this address, perform the write, increment it by the current operand size, and store the updated address in the temporary register.

If an initial WRITE is not executed preceding the first FILL command, the illegal command response is returned.

**NOTE**

The FILL command does not check for a valid address: FILL is a valid command only when preceded by another FILL, a NOP, or a WRITE command. Otherwise, an illegal command response is returned. The NOP command can be used for intercommand padding without corrupting the address pointer.

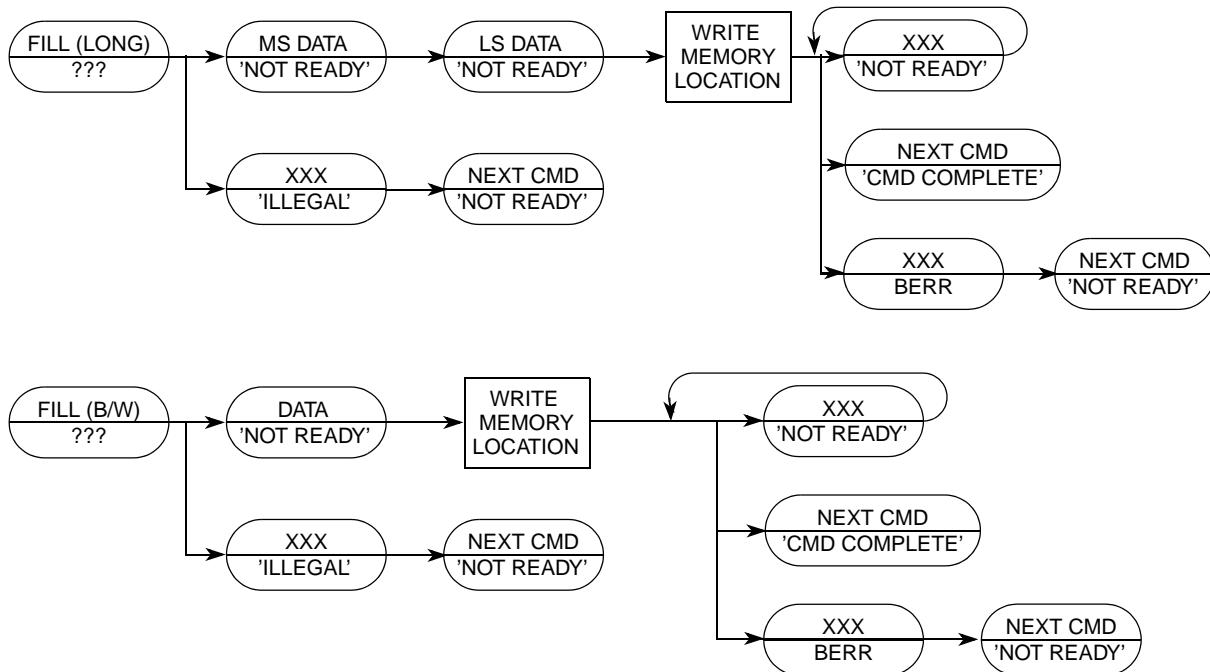
The size field is examined each time a FILL command is processed, allowing the operand size to be altered dynamically.

Command Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0				
Byte	0x1				0xC				0x0				0x0							
	X	X	X	X	X	X	X	X					D[7:0]							
Word	0x1				0xC				0x4				0x0							
	D[15:0]																			
Longword	0x1				0xC				0x8				0x0							
	D[31:16]																			
	D[15:0]																			

Figure 43-36. FILL Command Format

Command Sequence:



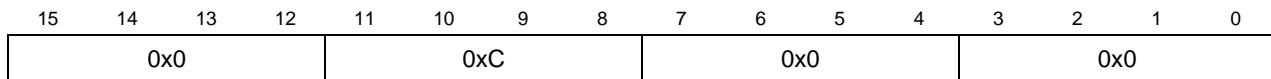
**Figure 43-37. FILL Command Sequence**

**Operand Data:** A single operand is data to be written to the memory location. Byte data is sent as a 16-bit word, justified in the least-significant byte; 16- and 32-bit operands are sent as 16 and 32 bits, respectively.

**Result Data:** Command complete status (0xFFFF) is returned when the register write is complete. A value of 0x0001 (with S set) is returned if a bus error occurs.

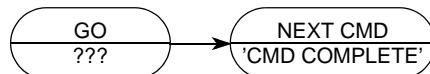
#### 43.4.1.5.7 Resume Execution (GO)

The pipeline is flushed and refilled before normal instruction execution resumes. Prefetching begins at the current address in the PC and at the current privilege level. If any register (such as the PC or SR) is altered by a BDM command while the processor is halted, the updated value is used when prefetching resumes. If a GO command issues and the CPU is not halted, the command is ignored.



**Figure 43-38. Go Command Format**

Command Sequence:



**Figure 43-39. Go Command Sequence**

Operand Data:

None

Result Data:

The command-complete response (0xFFFF) is returned during the next shift operation.

#### 43.4.1.5.8 No Operation (NOP)

NOP performs no operation and may be used as a null command where required.

Command Formats:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0					0x0			0x0			0x0			0x0	

Figure 43-40. NOP Command Format

Command Sequence:

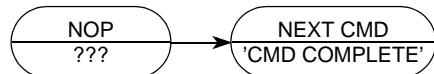


Figure 43-41. NOP Command Sequence

Operand Data:

None

Result Data:

The command-complete response, 0xFFFF (with S cleared), is returned during the next shift operation.

#### 43.4.1.5.9 Synchronize PC to the PSTDDATA Lines (SYNC\_PC)

The SYNC\_PC command captures the current PC and displays it on the PSTDDATA outputs. After the debug module receives the command, it sends a signal to the ColdFire processor that the current PC must be displayed. The processor then forces an instruction fetch at the next PC with the address being captured in the DDATA logic under control of the CSR[BTB] bits. The specific sequence of PSTDDATA values is defined below:

1. Debug signals a SYNC\_PC command is pending.
2. CPU completes the current instruction.
3. CPU forces an instruction fetch to the next PC, generates a PST equaling 0x5 value indicating a taken branch and signals the capture of DDATA.
4. The instruction address corresponding to the PC is captured.
5. The PST marker (0x9–0xB) is generated and displayed as defined by the CSR[BTB] bit followed by the captured PC address.

If the option to display ASID is enabled (CSR[OTE] = 1), the 8-bit ASID follows the address. That is, the PSTDDATA sequence is {0x5, Marker, Instruction Address, 0x8, ASID}, where the 0x8 is the marker for the ASID.

The SYNC\_PC command can be used to dynamically access the PC for performance monitoring. The execution of this command is considerably less obtrusive to the real-time operation of an application than a HALT-CPU/READ-PC/RESUME command sequence.

Command Formats:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0				0x0			0x0						0x1		

Figure 43-42. SYNC\_PC Command Format

Command Sequence:

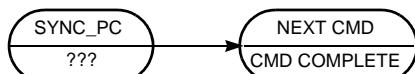


Figure 43-43. SYNC\_PC Command Sequence

Operand Data: None

Result Data: Command complete status (0xFFFF) is returned when the register write is complete.

#### 43.4.1.5.10 Force Transfer Acknowledge (FORCE\_TA)

Debug revision D logic implements the new FORCE\_TA serial BDM command to resolve a hung bus condition. In some system designs, references to certain unmapped memory addresses may cause the external bus to hang with no transfer acknowledge generated by any bus responders. The FORCE\_TA forces generation of a transfer acknowledge signal.

There are two scenarios of interest: one caused by a processor access and the other caused by a BDM access. The following sequences identify the operations needed to break the hung bus condition:

- Bus hang caused by processor or external or internal alternate master:
  - Assert the breakpoint input to force a processor core halt.
  - If the bus hang was caused by a processor access, send in FORCE\_TA commands until the processor is halted, as signaled by PST = 0xF. Due to pipeline and store buffer depths, many memory accesses may be queued up behind the access causing the bus hang. Repeated FORCE\_TA commands eventually allow processing of all these pending accesses. As soon as the processor is halted, the system reaches a quiescent, controllable state.
  - If the hang was caused by another master, such as a DMA channel, the processor can halt immediately. In this case as well, multiple assertions of the FORCE\_TA command may be required to terminate the alternate master's errant access.
- Bus hang caused by BDM access:
  - It is assumed the processor is already halted at the time of the errant BDM access. To resolve the hung bus, it is necessary to process four or more FORCE\_TA commands, because the BDM command may have initiated a cache line access that fetches 4 longwords, each needing a unique transfer acknowledge.

Formats:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x0	0x0				0x0	0x0				0x2					

Figure 43-44. FORCE\_TA Command

Command Sequence:

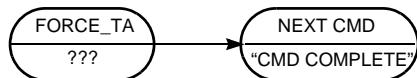


Figure 43-45. FORCE\_TA Command Sequence

Operand Data: None

Result Data: The command complete response, 0xFFFF (with the status bit cleared), is returned during the next shift operation. This response indicates the FORCE\_TA command was processed correctly and does not necessarily reflect the status of any internal bus.

#### 43.4.1.5.11 Read Control Register (RCREG)

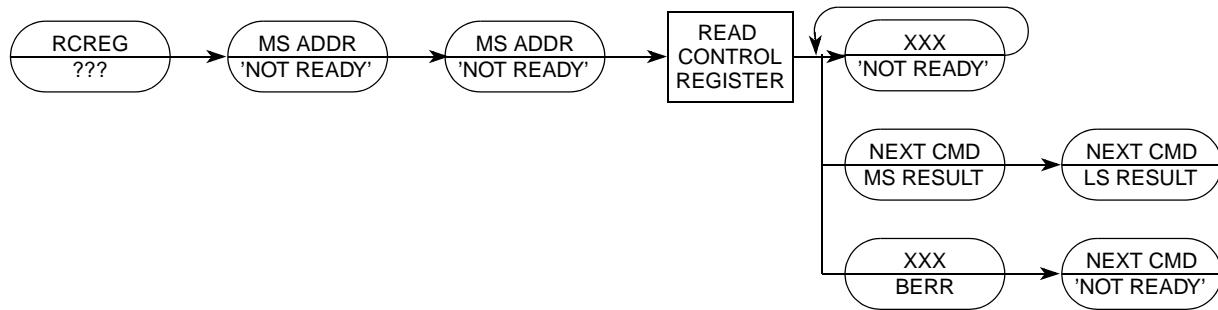
Read the selected control register and return the 32-bit result. Accesses to the processor/memory control registers are always 32 bits wide, regardless of register width. The second and third words of the command form a 32-bit address, which the debug module uses to generate a special bus cycle to access the specified control register. The 12-bit Rc field is the same the processor's MOVEC instruction uses.

Command/Result Formats:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0											
Command	0x2	0x9				0x8	0x0																			
	0x0	0x0				0x0	0x0				0x0															
	0x0					Rc																				
Result	D[31:16]																									
	D[15:0]																									

Figure 43-46. RCREG Command/Result Formats

Command Sequence:



**Figure 43-47. RCREG Command Sequence**

Operand Data: The only operand is the 32-bit Rc control register select field.

Result Data: Control register contents are returned as a longword, most-significant word first. The implemented portion of registers smaller than 32 bits is guaranteed correct; other bits are undefined.

Rc encoding: See [Table 43-31](#).

**Table 43-31. Control Register Map**

Rc	Register Definition
0x002	Cache Control Register (CACR)
0x003	Address Space Identifier (ASID)
0x004	Access Control Register (ACR0)
0x005	Access Control Register (ACR1)
0x006	Access Control Register (ACR2)
0x007	Access Control Register (ACR3)
0x008	MMU Base Address Register (MMUBAR)
0x009	GPIO Base Address Register (RGPIOBAR) <sup>1</sup>
0x(0,1)80 – 0x(0,1)87	Data Registers 0–7 (0 = load, 1 = store)
0x(0,1)88 – 0x(0,1)8F	Address Registers 0–7 (0 = load, 1 = store) (A7 is user stack pointer)
0x800	Other Stack Pointer (OTHER_A7)
0x801	Vector Base Register (VBR)
0x804	MAC Status Register (MACSR)
0x805	MAC Mask Register (MASK)
0x806	MAC Accumulator 0 (ACC0)
0x807	MAC Accumulator 0,1 Extension Bytes (ACCEXT01)
0x808	MAC Accumulator 2,3 Extension Bytes (ACCEXT23)
0x809	MAC Accumulator 1 (ACC1)
0x80A	MAC Accumulator 2 (ACC2)

**Table 43-31. Control Register Map (continued)**

Rc	Register Definition
0x80B	MAC Accumulator 3 (ACC3)
0x80E	Status Register (SR)
0x80F	Program Register (PC)
0xC05	RAM Base Address Register (RAMBAR)

<sup>1</sup> If an RGPIO module is available on this device.

#### 43.4.1.5.12 BDM Accesses of the Stack Pointer Registers (A7: SSP and USP)

The ColdFire core supports two unique stack pointer (A7) registers: the supervisor stack pointer (SSP) and the user stack pointer (USP). The hardware implementation of these two programmable-visible 32-bit registers does not uniquely identify one as the SSP and the other as the USP. Rather, the hardware uses one 32-bit register as the currently-active A7; the other is named the OTHER\_A7. Therefore, the contents of the two hardware registers is a function of the operating mode of the processor:

```
if SR[S] = 1
    then      A7 = Supervisor Stack Pointer
              OTHER_A7 = User Stack Pointer
    else      A7 = User Stack Pointer
              OTHER_A7 = Supervisor Stack Pointer
```

The BDM programming model supports reads and writes to A7 and OTHER\_A7 directly. It is the responsibility of the external development system to determine the mapping of A7 and OTHER\_A7 to the two program-visible definitions (supervisor and user stack pointers), based on the SR[S] bit.

#### 43.4.1.5.13 BDM Accesses of the EMAC Registers

The presence of rounding logic in the output datapath of the EMAC requires special care for BDM-initiated reads and writes of its programming model. In particular, any result rounding modes must be disabled during the read/write process so the exact bit-wise EMAC register contents are accessed.

For example, a BDM read of an accumulator (ACCx) must be preceded by two commands accessing the MAC status register, as shown in the following sequence:

```
BdmReadACCx (
    rcreg  macsr;           // read current macsr contents and save
    wcreg  #0,macsr;        // disable all rounding modes
    rcreg  ACCx;            // read the desired accumulator
    wcreg  #saved_data,macsr; // restore the original macsr
)
```

Likewise, to write an accumulator register, the following BDM sequence is needed:

```
BdmWriteACCx (
    rcreg  macsr;           // read current macsr contents and save
    wcreg  #0,macsr;        // disable all rounding modes
    wcreg  #data,ACCx;       // write the desired accumulator
    wcreg  #saved_data,macsr; // restore the original macsr
)
```

Additionally, writes to the accumulator extension registers must be performed after the corresponding accumulators are updated because a write to any accumulator alters the corresponding extension register contents.

For more information on saving and restoring the complete EMAC programming model, see [Section 5.3.1.2, “Saving and Restoring the EMAC Programming Model.”](#)

#### 43.4.1.5.14 Write Control Register (WCREG)

The operand (longword) data is written to the specified control register. The write alters all 32 register bits. See the RCREG instruction description for the Rc encoding and for additional notes on writes to the A7 stack pointers and the EMAC programming model.

Command/Result Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Command	0x2				0x8			0x8				0x0				
	0x0				0x0			0x0				0x0				
	0x0											Rc				
Result								D[31:16]								
								D[15:0]								

Figure 43-48. WCREG Command/Result Formats

Command Sequence:

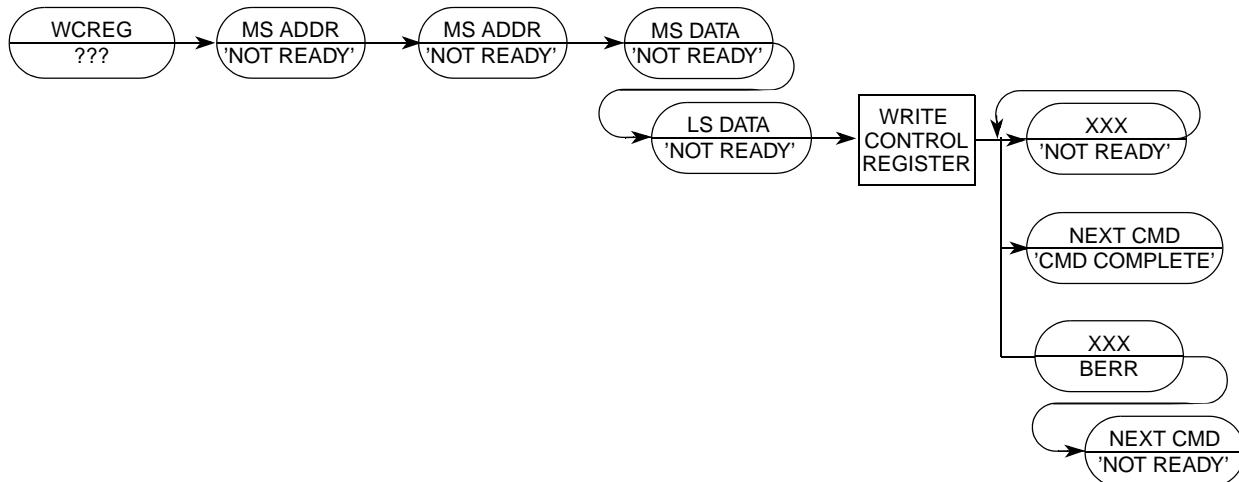


Figure 43-49. WCREG Command Sequence

Operand Data: This instruction requires two longword operands. The first selects the register to the operand data writes to; the second contains the data.

Result Data: Successful write operations return 0xFFFF. Bus errors on the write cycle are indicated by the setting of bit 16 in the status message and by a data pattern of 0x0001.

#### 43.4.1.5.15 Read Debug Module Register (RDMREG)

Read the selected debug module register and return the 32-bit result. The only valid register selection for the RDMREG command is CSR (DRc=0x00), XCSR (DRc=0x01), CSR2 (DRc=0x02), the trace buffer contents (which are TBLW0–23 (DRc=0x20–0x37)), and PCRS (the most recently sampled PC when operating in the PC profile recording mode).

To configure PC profiling recording mode set the following:

- CSR2[PSTBRM] = 0x2
- XCSR[APCENB] = 1
- CSR[BTB] = non-zero
- XCSR[APCSC] = periodic interval of PC address captures of choice.

In this mode, a read at address 0x3F (PCRS) returns the most recently sampled PC. Each time the PC is sampled and loaded into the PSTB, the full 32-bit PC is also loaded into PCRS. You can use the RDMREG BDM command at any time including in halt mode. This provides real-time PC profiling that is minimally intrusive to the system.

Command/Result Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0												
Command	0x2		0xD		1	0	DRc																					
Result	D[31:16]																											
	D[15:0]																											

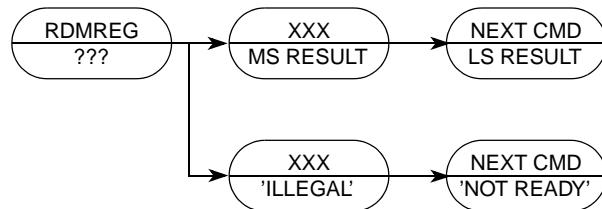
Figure 43-50. RDMREG Command/Result Formats

Table 43-32 shows the definition of DRc encoding.

Table 43-32. Definition of DRc Encoding—Read

DRc[5:0]	Debug Register Definition	Mnemonic
0x00	Configuration/Status	CSR
0x01	Extended configuration/status register	XCSR
0x02	Configuration/status register 2	CSR2
0x20–0x37	PSTB trace buffer longwords	TBLW0–23
0x3F	Most recently sampled PC	PCRS

Command Sequence:



**Figure 43-51. RDMREG Command Sequence**

Operand Data: None

Result Data: The contents of the selected debug register are returned as a longword value. The data is returned most-significant word first.

#### 43.4.1.5.16 Write Debug Module Register (WDMREG)

The operand (longword) data is written to the specified debug module register. All 32 bits of the register are altered by the write. DSCLK must be inactive while the debug module register writes from the CPU accesses are performed using the WDEBUG instruction.

Command Format:

**Figure 43-52. WDMREG BDM Command Format**

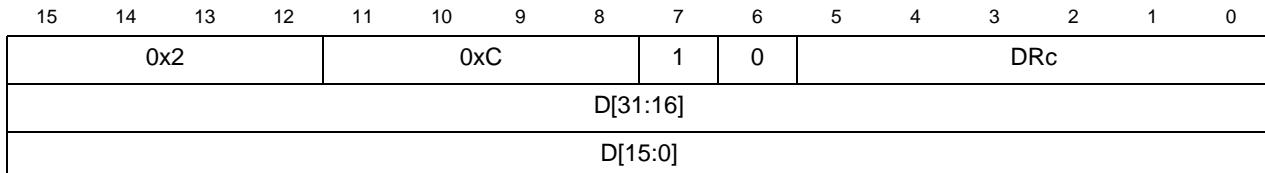
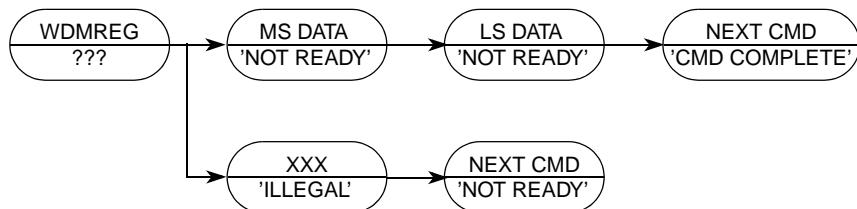


Table 43-5 shows the definition of the DRc write encoding.

Command Sequence:



**Figure 43-53. WDMREG Command Sequence**

Operand Data: Longword data is written into the specified debug register. The data is supplied most-significant word first.

Result Data: Command complete status (0xFFFF) is returned when register write is complete.

## 43.4.2 Real-Time Debug Support

The ColdFire family provides support debugging real-time applications. For these types of embedded systems, the processor must continue to operate during debug. The foundation of this area of debug support is that while the processor cannot be halted to allow debugging, the system can generally tolerate the small intrusions of the BDM inserting instructions into the pipeline with minimal effect on real-time operation.

The debug module provides four types of breakpoints: PC with mask, PC without mask, operand address range, and data with mask. These breakpoints can be configured into one- or two-level triggers with the exact trigger response also programmable. The debug module programming model can be written from the external development system using the debug serial interface or from the processor's supervisor programming model using the WDEBUG instruction. Only CSR is readable using the external development system.

### 43.4.2.1 Theory of Operation

Breakpoint hardware can be configured through TDR[TCR] to respond to triggers by displaying PSTDDATA, initiating a processor halt, or generating a debug interrupt. As shown in [Table 43-33](#), when a breakpoint is triggered, an indication (CSR[BSTAT]) is provided on the PSTDDATA output port of the DDATA information when it is not displaying captured processor status, operands, or branch addresses. See [Section 43.4.4.3, “Processor Stopped or Breakpoint State Change \(PST = 0xE\).”](#)

**Table 43-33. PSTDDATA Nibble/CSR[BSTAT] Breakpoint Response**

PSTDDATA Nibble <sup>1</sup>	CSR[BSTAT] <sup>1</sup>	Breakpoint Status
0000	0000	No breakpoints enabled
0010	0001	Waiting for level-1 breakpoint
0100	0010	Level-1 breakpoint triggered
1010	0101	Waiting for level-2 breakpoint
1100	0110	Level-2 breakpoint triggered

<sup>1</sup> Encodings not shown are reserved for future use.

The breakpoint status is also posted in the CSR. CSR[BSTAT] is cleared by a CSR read when a level-2 breakpoint is triggered or a level-1 breakpoint is triggered and a level-2 breakpoint is not enabled. Status is also cleared by writing to either TDR or XTDR to disable trigger options.

BDM instructions use the appropriate registers to load and configure breakpoints. As the system operates, a breakpoint trigger generates the response defined in TDR.

PC breakpoints are treated in a precise manner—exception recognition and processing are initiated before the excepting instruction executes. All other breakpoint events are recognized on the processor's local bus, but are made pending to the processor and sampled like other interrupt conditions. As a result, these interrupts are imprecise.

In systems that tolerate the processor being halted, a BDM-entry can be used. With TDR[TRC] equals 01, a breakpoint trigger causes the core to halt (PST = 0xF).

If the processor core cannot be halted, the debug interrupt can be used. With this configuration, TDR[TRC] equals 10, breakpoint trigger becomes a debug interrupt to the processor, which is treated higher than the nonmaskable level-7 interrupt request. As with all interrupts, it is made pending until the processor reaches a sample point, which occurs once per instruction. Again, the hardware forces the PC breakpoint to occur before the targeted instruction executes and is precise. This is possible because the PC breakpoint is enabled when interrupt sampling occurs. For address and data breakpoints, reporting is considered imprecise, because several instructions may execute after the triggering address or data is detected.

As soon as the debug interrupt is recognized, the processor aborts execution and initiates exception processing. This event is signaled externally by the assertion of a unique PST value (PST = 0xD) for multiple cycles. The core enters emulator mode when exception processing begins. After the standard 8-byte exception stack is created, the processor fetches a unique exception vector from the vector table. [Table 43-34](#) describes the two unique entries that distinguish PC breakpoints from other trigger events.

**Table 43-34. Exception Vector Assignments**

Vector Number	Vector Offset (Hex)	Stacked Program Counter	Assignment
12	0x030	Next	Non-PC-breakpoint debug interrupt
13	0x034	Next	PC-breakpoint debug interrupt

Refer to the *ColdFire Programmer's Reference Manual* for more information.

In the case of a two-level trigger, the last breakpoint event determines the exception vector; however, if the second-level trigger is PC || Address {&& Data} (as shown in the last condition in the code example in [Section 43.3.13.1, ‘Resulting Set of Possible Trigger Combinations’](#)), the vector taken is determined by the first condition that occurs after the first-level trigger: vector 13 if PC occurs first or vector 12 if Address {&& Data} occurs first. If both occur simultaneously, the non-PC-breakpoint debug interrupt is taken (vector number 12).

Execution continues at the instruction address in the vector corresponding to the debug interrupt. The debug interrupt handler can use supervisor instructions to save the necessary context, such as the state of all program-visible registers into a reserved memory area.

During a debug interrupt service routine, all normal interrupt requests are evaluated and sampled once per instruction. If any exception occurs, the processor responds as follows:

1. It saves a copy of the current value of the emulator mode state bit and then exits emulator mode by clearing the actual state.
2. The fault status field (FS) in the next exception stack frame is set to 0010 to indicate the processor was in emulator mode when the interrupt occurred. See [Section 3.3.3.1, ‘Exception Stack Frame Definition.’](#)
3. It passes control to the appropriate exception handler.
4. It executes an RTE instruction when the exception handler finishes. During the processing of the RTE, FS is reloaded from the system stack. If this bit field is set to 0010, the processor sets the emulator mode state and resumes execution of the original debug interrupt service routine. This is signaled externally by the generation of the PST value that originally identified the debug interrupt exception, that is, PST = 0xD.

Fault status encodings are listed in [Table 3-7](#). The implementation of this debug interrupt handling fully supports the servicing of a number of normal interrupt requests during a debug interrupt service routine.

The emulator mode state bit is essentially changed to be a program-visible value, stored into memory during exception stack frame creation, and loaded from memory by the RTE instruction.

When debug interrupt operations complete, the RTE instruction executes and the processor exits emulator mode. After the debug interrupt handler completes execution, the external development system can use BDM commands to read the reserved memory locations.

In revisions B/B+ and C, the hardware inhibits generation of another debug interrupt during the first instruction after the RTE exits emulator mode. This behavior is consistent with the logic involving trace mode where the first instruction executes before another trace exception is generated. Thus, all hardware breakpoints are disabled until the first instruction after the RTE completes execution, regardless of the programmed trigger response.

#### 43.4.2.2 Emulator Mode

Emulator mode facilitates non-intrusive emulator functionality. This mode can be entered in three different ways:

- Setting CSR[EMU] forces the processor into emulator mode. EMU is examined only if  $\overline{\text{RSTI}}$  is negated and the processor begins reset exception processing. It can be set while the processor is halted before reset exception processing begins. See [Section 43.4.1.1, “CPU Halt”](#).
- A debug interrupt always puts the processor in emulation mode when debug interrupt exception processing begins.
- Setting CSR[TRC] forces the processor into emulation mode when trace exception processing begins.

While operating in emulation mode, the processor exhibits the following properties:

- Unmasked interrupt requests are serviced. The resulting interrupt exception stack frame has FS set appropriately (0010) to indicate the interrupt occurred while in emulator mode.
- If CSR[MAP] is set, all caching of memory and the SRAM module are disabled. All memory accesses are forced into a specially mapped address space signaled by TT equals 0x2, TM equals 0x5, or 0x6. This includes stack frame writes and vector fetch for the exception that forced entry into this mode.

The RTE instruction exits emulation mode. The processor status output port provides a unique encoding for emulator mode entry (0xD) and exit (0x7).

#### 43.4.3 Concurrent BDM and Processor Operation

The debug module supports concurrent operation of the processor and most BDM commands. BDM commands may be executed while the processor is running, except these following operations that access processor/memory registers:

- Read/write address and data registers
- Read/write control registers

For BDM commands that access memory, the debug module requests the processor's local bus. The processor responds by stalling the instruction fetch pipeline and waiting for current bus activity to complete before freeing the local bus for the debug module to perform its access. After the debug module bus cycle, the processor reclaims the bus.

#### NOTE

Breakpoint registers must be carefully configured in a development system if the processor is executing. The debug module contains no hardware interlocks, so TDR and XTDR should be disabled while breakpoint registers are loaded, after which TDR and XTDR can be written to define the exact trigger. This prevents spurious breakpoint triggers.

Because there are no hardware interlocks in the debug unit, no BDM operations are allowed while the CPU is writing the debug's registers (DSCLK must be inactive).

### 43.4.4 Real-Time Trace Support

Real-time trace, which defines the dynamic execution path and is also known as instruction trace, is a fundamental debug function. The ColdFire solution is to include a parallel output port providing encoded processor status and data to an external development system. This 8-bit port is partitioned into two consecutive 4-bit nibbles. Each nibble can either transmit information concerning the processor's execution status (PST) or debug data (DDATA). A PST marker and its data display are sent contiguously — the IDLE status (0x0) can appear anytime. As stated before, PST values and operands may appear on either nibble of PSTDDATA. The processor status may not be related to the current bus transfer, due to the decoupling FIFOs.

External development systems can use PSTDDATA outputs with an external image of the program to completely track the dynamic execution path. This tracking is complicated by any change in flow, where branch target address calculation is based on the contents of a program-visible register (variant addressing). PSTDDATA outputs can display the target address of such instructions in sequential nibble increments across multiple processor clock cycles, as described in [Section 43.4.4.2, “Begin Execution of Taken Branch \(PST = 0x5\)”](#). Four 32-bit storage elements form a FIFO buffer connecting the processor's high-speed local bus to the external development system through PSTDDATA[7:0]. The buffer captures branch target addresses and certain data values for eventual display on the PSTDDATA port, two nibbles at a time starting with the least significant bit (lsb).

Execution speed is affected only when three storage elements contain valid data to be dumped to the PSTDDATA port. This occurs only when two values are captured simultaneously in a read-modify-write operation. The core stalls until two FIFO entries are available.

[Table 43-35](#) shows the encoding of these signals.

**Table 43-35. Processor Status Encoding**

PST[3:0]	Definition
0x0	Continue execution. Many instructions execute in one processor cycle. If an instruction requires more clock cycles, subsequent clock cycles are indicated by driving PSTDDATA outputs with this encoding.
0x1	Begin execution of one instruction. For most instructions, this encoding signals the first processor clock cycle of an instruction's execution. Certain change-of-flow opcodes, plus the PULSE and WDDATA instructions, generate different encodings.
0x2	Begin execution of two instructions. For superscalar instruction dispatches, this encoding signals the first clock cycle of the simultaneous instructions' execution.
0x3	Entry into user-mode. Signaled after execution of the instruction that caused the ColdFire processor to enter user mode. If the display of the ASID is enabled (CSR[OTE] = 1), the following occurs: The 8-bit ASID follows the instruction address; that is, the PSTDDATA sequence is {0x3, 0x5, marker, instruction address, 0x8, ASID}, where 0x8 is the ASID data marker. When the current ASID is loaded by the privileged MOVEC instruction, the ASID is displayed on PSTDDATA. The resulting PSTDDATA sequence for the MOVEC instruction is then {0x1, 0x8, ASID}, where the 0x8 is the data marker for the ASID.
0x4	Begin execution of PULSE and WDDATA instructions. PULSE defines logic analyzer triggers for debug and/or performance analysis. WDDATA lets the core write any operand (byte, word, or longword) directly to the PSTDDATA port, independent of debug module configuration. When WDDATA is executed, a value of 0x4 is signaled, followed by the appropriate marker, and then the data transfer on the PSTDDATA port. Transfer length depends on the WDDATA operand size.
0x5	Begin execution of taken branch or SYNC_PC command issued. For some opcodes, a branch target address may be displayed on PSTDDATA depending on the CSR settings. CSR also controls the number of address bytes displayed, indicated by the PST marker value preceding the PSTDDATA nibble that begins the data output. See <a href="#">Section 43.4.4.2, "Begin Execution of Taken Branch (PST = 0x5)"</a> . Also indicates that the SYNC_PC command has been issued.
0x6	Begin execution of instruction plus a taken branch. The processor completes execution of a taken conditional branch instruction and simultaneously starts executing the target instruction. This is achieved through branch folding.
0x7	Begin execution of return from exception (RTE) instruction.
0x8–0xB	Indicates the number of bytes to be displayed on the PSTDDATA port on subsequent clock cycles. The value is driven onto the PSTDDATA port one cycle before the data is displayed. 0x8 Begin 1-byte transfer on PSTDDATA. 0x9 Begin 2-byte transfer on PSTDDATA. 0xA Begin 3-byte transfer on PSTDDATA. 0xB Begin 4-byte transfer on PSTDDATA.
0xC	Normal exception processing. Exceptions that enter emulation mode (debug interrupt or optionally trace) generate a different encoding, as described below. Because the 0xC encoding defines a multiple-cycle mode, PSTDDATA outputs are driven with 0xC until exception processing completes.
0xD	Emulator mode exception processing. Displayed during emulation mode (debug interrupt or optionally trace). Because this encoding defines a multiple-cycle mode, PSTDDATA outputs are driven with 0xD until exception processing completes.
0xE	A breakpoint state change causes this encoding to assert for one cycle only followed by the trigger status value. If the processor stops waiting for an interrupt, the encoding is asserted for multiple cycles. See <a href="#">Section 43.4.4.3, "Processor Stopped or Breakpoint State Change (PST = 0xE)"</a> .
0xFF	Processor is halted. Because this encoding defines a multiple-cycle mode, the PSTDDATA outputs display 0xF until the processor is restarted or reset. See <a href="#">Section 43.4.1.1, "CPU Halt"</a> .

#### 43.4.4.1 PST Trace Buffer

This device also implements an on-chip PST trace buffer (PSTB) that records compressed processor execution status and debug data in a 128-packet (6-bit each) memory. This memory is mapped into the debug register programming model, so the trace information can be retrieved via the 3-pin serial BDM interface (see the RDMREG command described in [Section 43.4.1.5.15, “Read Debug Module Register \(rdmreg\)](#)). It supports programmable start/stop recording conditions, as well as special continuous and PC-profiling modes of operation along with normal recording.

The feature sets of the classic PST/DDATA and PSTB implementations are compared in [Table 43-36](#).

**Table 43-36. ColdFire Trace Support Comparison**

Feature	Classic PST/DDATA	PST Trace Buffer
Basic capability	Program trace with optional partial data trace	Program trace
Processor status	4-bit encoded status, 1 or more values per instruction	5-bit encoded compressed status, 1 or more values per instruction
Debug data	Multiple 4-bit data nibbles, instruction addresses, or system bus data	Multiple 4-bit data nibbles, instruction addresses only
WDDATA.{b,w,l} instruction	Yes, provides hardware printf capability allowing a direct write to the port	Yes, provides hardware printf capability allowing a direct write to the trace buffer
Output method	Continuously output from core in real time; can be disabled if CSR[17] = 1	Written into trace buffer, retrieved via BDM read commands; programmable start and stop recording conditions
Trace length limitations	No restrictions; can be infinitely long	Storage of up to 128 6-bit PST/DDATA packets

Since there are one (or more) PST entry per instruction, the PSTB would fill rapidly without data compression. Therefore, a compression algorithm is added with this feature.

Consider the following example to illustrate the PST compression algorithm. Most sequential instructions generate a single  $\text{PST} = 1$  value. Without compression, the execution of ten sequential instructions generates a stream of ten  $\text{PST} = 1$  values. With PST compression, the reporting of any  $\text{PST} = 1$  value is delayed so that consecutive  $\text{PST} = 1$  values can be accumulated. When a  $\text{PST} \neq 1$  value is reported, the maximum accumulation count is reached, or a debug data value is captured, a single accumulated PST value is generated. Returning to the example with compression enabled, the execution of ten sequential instructions generates a single PST value indicating ten sequential instructions have been executed.

This technique has proven to be effective at significantly reducing the average PST entries per instruction and PST entries per machine cycle. The application of this compression technique makes the application of a useful PST trace buffer for the ColdFire core realizable.

**Table 43-37. PST Buffer Processor Status Encodings**

PSTB[4:0]	Definition
0x00	Continue execution. Many instructions execute in one processor cycle. If an instruction requires more processor clock cycles, subsequent clock cycles are indicated by driving PST with this encoding. <b>Note:</b> This encoding is never stored in the PSTB.
0x01	Begin execution of one instruction. For most instructions, this encoding signals the first processor clock cycle of an instruction's execution. Certain change-of-flow opcodes, plus the PULSE and WDDATA instructions, generate different encodings.
0x02	Begin execution of two instructions. For superscalar instruction dispatches, this encoding signals the first clock cycle of the simultaneous instructions' execution.
0x03	Entry into user-mode. Signaled after execution of the instruction that caused the ColdFire processor to enter user mode.
0x04	Begin execution of PULSE and WDDATA instructions. PULSE defines triggers or markers for debug and/or performance analysis. WDDATA lets the core write any operand (byte, word, or longword) directly as DDATA packets, independent of debug module configuration. When WDDATA is executed, a value of 0x04 is stored in the PSTB, followed by the appropriate marker, and then the data transfer as DDATA packets. The number of captured data bytes depends on the WDDATA operand size.
0x05	Begin execution of taken branch or SYNC_PC BDM command. For some opcodes, a branch target address may be stored in the PSTB depending on the CSR settings. CSR also controls the number of address bytes stored, indicated by the PST marker value preceding the DDATA packets that begin the data output. This encoding also indicates that the SYNC_PC command has been processed.
0x06	Reserved
0x07	Begin execution of return from exception (RTE) instruction.
0x08–0x0B	Indicates the number of data bytes to be stored in the PSTB. This marker is the PST stored immediately before the DDATA packets. Only data associated with the WDDATA instruction is captured. 0x08 Precedes 1-byte DDATA packets 0x09 Precedes 2-byte DDATA packets 0x0A Reserved 0x0B Precedes 4-byte DDATA packets
0x0C–0x0F	Indicates the number of address bytes to be stored in the PSTB on subsequent processor clock cycles. This marker value is stored in the PSTB immediately before the address packet is stored. The capturing of branch target addresses is controlled by CSR[BTB]. 0x0C Precedes 2-byte ASID packet 0x0D Precedes 2-byte packet (Stored address is shifted right 1: ADDR[16:1]) 0x0E Precedes 3-byte packet (Stored address is shifted right 1: ADDR[23:1]) 0x0F Precedes 4-byte packet (Stored address is shifted right 1: ADDR[31:1])
0x10–0x11	Reserved
0x12	Completed execution of 2 sequential instructions
0x13	Completed execution of 3 sequential instructions
0x14	Completed execution of 4 sequential instructions
0x15	Completed execution of 5 sequential instructions
0x16	Completed execution of 6 sequential instructions
0x17	Completed execution of 7 sequential instructions
0x18	Completed execution of 8 sequential instructions
0x19	Completed execution of 9 sequential instructions

**Table 43-37. PST Buffer Processor Status Encodings (continued)**

PSTB[4:0]	Definition
0x1A	Completed execution of 10 sequential instructions
0x1B	This value signals there has been a change in the breakpoint trigger state machine. It appears as a single PST packet marker for each state change and is immediately followed by a DDATA packet signaling the new breakpoint trigger state encoding. The DDATA breakpoint trigger state packet is defined as $(0x20 + 2 \times \text{CSR[BSTAT]})$ : 0x20 No breakpoints enabled 0x22 Waiting for a level-1 breakpoint 0x24 Level-1 breakpoint triggered 0x2A Waiting for a level-2 breakpoint 0x2C Level-2 breakpoint triggered
0x1C	Exception processing. This value signals the processor has encountered an exception condition. Although this is a multi-cycle mode, there are only two PST = 0x1C packets recorded before the mode value is suppressed.
0x1D	Emulator mode exception processing. This value signals the processor has encountered a debug interrupt or a properly-configured trace exception. Although this is a multi-cycle mode, there are only two PST = 0x1D packets recorded before the mode value is suppressed.
0x1E	Processor is stopped. This value signals the processor has executed a STOP instruction. Although this is a multi-cycle mode because the ColdFire processor remains stopped until an interrupt or reset occurs, there are only two PST = 0x1E packets recorded before the mode value is suppressed.
0x1F	Processor is halted. This value signals the processor has been halted. Although this is a multi-cycle mode because the ColdFire processor remains halted until a BDM go command is received or reset occurs, there are only two PST = 0x1F packets recorded before the mode value is suppressed.

#### 43.4.4.2 Begin Execution of Taken Branch (PST = 0x5)

PST is 0x5 when a taken branch is executed. For some opcodes, a branch target address may be displayed on PSTDDATA or loaded into the PST trace buffer depending on the CSR settings. CSR also controls the number of address bytes displayed/loaded, which is indicated by the PST marker value immediately preceding the PSTDDATA nibble that begins the data output/PSTB address entries.

The address is displayed on PSTDDATA or loaded into PSTB in least-to-most significant nibble order. The address is not shifted for the PSTDDATA ports — in other words, [15:0], [23:0], or [31:0] is displayed. For PSTB storage, the address is shifted right by one bit — in other words, [16:1], [24:1], or {0,[31:1]} is stored.

Multiple byte DDATA values are displayed in least-to-most-significant order. The processor captures only those target addresses associated with taken branches that use a variant addressing mode (RTE and RTS instructions, JMP and JSR instructions using address register indirect or indexed addressing modes, and all exception vectors).

The simplest example of a branch instruction using a variant address is the compiled code for a C language case statement. Typically, the evaluation of this statement uses the variable of an expression as an index into a table of offsets, where each offset points to a unique case within the structure. For such change-of-flow operations, the ColdFire processor uses the debug pins/PST buffer to output/load the following sequence of information on two successive processor clock cycles:

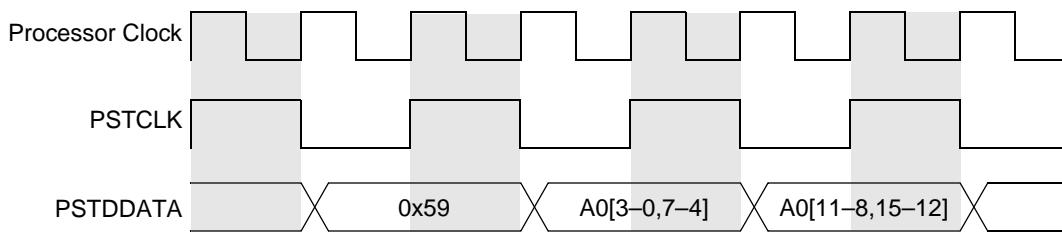
1. Use PSTDDATA (0x5) or load a PST=0x05 packet to identify that a taken branch is executed.

- Signal the target address to be displayed sequentially on the PSTDDATA pins. Encodings 0x9–0xB identify the number of bytes displayed.

Using the PSTB, optionally load the marker for the target address capture. PST packet encodings 0x0D, 0x0E, or 0x0F identify the number of bytes loaded into the PSTB. The capturing of the target instruction address is configured in CSR[BTB].

- The new target address is optionally available on subsequent cycles using the PSTDDATA port or loaded in the PSTB. The number of bytes of displayed on this port/loaded into the PSTB is configurable (2, 3, or 4 bytes, where the PSTDDATA encoding is 0x9, 0xA, and 0xB, respectively or where the PST marker packets are 0x0D, 0x0D, and 0x0F, respectively).

Another example of a variant branch instruction would be a JMP (A0) instruction. [Figure 43-54](#) shows the PSTDDATA outputs that indicate a JMP (A0) execution, assuming the CSR was programmed to display the lower 2 bytes of an address.



**Figure 43-54. Example JMP Instruction Output on PSTDDATA**

PSTDDATA is driven two nibbles at a time with a 0x59; 0x5 indicates a taken branch and the marker value 0x9 indicates a 2-byte address. Therefore, the subsequent 4 nibbles display the lower two bytes of address register A0 in least-to-most-significant nibble order. The PSTDDATA output after the JMP instruction continues with the next instruction.

Using the PST buffer, this example is slightly modified so that the CSR is programmed to display the lower three bytes of the target instruction address. [Table 43-38](#) shows the consecutive PSTB entries that indicate a JMP (A0) execution.

**Table 43-38. Example JMP Instruction Captured in PSTB**

PSTB Value	Packet Type	Description
0x05	PST	Taken branch
0x0E	PST (marker)	3-byte address marker
{10, Address[4:1]} {10, Address[8:5]} {10, Address[12:9]} {10, Address[16:13]} {10, Address[20:17]} {10, Address[24:21]}	DDATA DDATA DDATA DDATA DDATA DDATA	Target address >> 1

The PST of 0x05 indicates a taken branch and the marker value 0x0E indicates a 3-byte address. The subsequent entries display the lower three bytes of address register A0, right-shifted by 1, in least-to-most-significant nibble order. The next PSTB entry, after the JMP instruction completes, depends on the target instruction.

### 43.4.4.3 Processor Stopped or Breakpoint State Change (PST = 0xE)

The 0xE encoding is generated either as a one- or multiple-cycle issue as follows:

- When the core is stopped by a STOP instruction, this encoding appears in multiple-cycle format. The ColdFire processor remains stopped until an interrupt occurs; thus, PSTDDATA outputs display 0xE until stopped mode is exited. PSTB only stores two consecutive packets of 0x1E.
- When a breakpoint status change is to be output on PSTDDATA, 0xE is displayed for one cycle, followed immediately with the 4-bit value of the current trigger status, where the trigger status is left justified rather than in the CSR[BSTAT] description. [Section 43.3.2, “Configuration/Status Register \(CSR\),”](#) shows that status is right justified. That is, the displayed trigger status on PSTDDATA after a single 0xE is as follows:
  - 0x0 = no breakpoints enabled
  - 0x2 = waiting for level-1 breakpoint
  - 0x4 = level-1 breakpoint triggered
  - 0xA = waiting for level-2 breakpoint
  - 0xC = level-2 breakpoint triggered

Thus, 0xE can indicate multiple events, based on the next value, as [Table 43-39](#) shows.

**Table 43-39. 0xE Status Posting**

PSTDDATA Stream Includes	Result
{0xE, 0x2}	Breakpoint state changed to waiting for level-1 trigger
{0xE, 0x4}	Breakpoint state changed to level-1 breakpoint triggered
{0xE, 0xA}	Breakpoint state changed to waiting for level-2 trigger
{0xE, 0xC}	Breakpoint state changed to level-2 breakpoint triggered
{0xE, 0xE}	Stopped mode.

### 43.4.4.4 Processor Halted (PST = 0xF)

PST is 0xF when the processor is halted (see [Section 43.4.1.1, “CPU Halt”](#)). Because this encoding defines a multiple-cycle mode, the PSTDDATA outputs display 0xF until the processor is restarted or reset. Therefore, PSTDDATA[7:0] continuously are 0xFF. PSTB only stores two consecutive packets of 0x1F.

#### NOTE

HALT can be distinguished from a data output 0xFF by counting 0xFF occurrences on PSTDDATA. Because data always follows a marker (0x8, 0x9, 0xA, or 0xB), the longest occurrence in PSTDDATA of 0xFF in a data output is four.

Two scenarios exist for data 0xFFFF\_FFFF:

- The B marker occurs on the most-significant nibble of PSTDDATA with the data of 0xFF following:

PSTDDATA[7:0]

0xBF  
0xFF  
0xFF  
0xFF

0xFF (*X* indicates that the next PST value is guaranteed to not be 0xF.)

- The B marker occurs on the least-significant nibble of PSTDDATA with the data of 0xFF following:

PSTDDATA[7:0]

0xYB  
0xFF  
0xFF  
0xFF  
0xFF

0xXY (*X* indicates the PST value is guaranteed not to be 0xF, and *Y* signifies a PSTDDATA value that doesn't affect the 0xFF count.)

#### NOTE

As the result of the above, a count of nine or more sequential 0xF PSTDDATA nibbles, or five or more sequential 0xFF PSTDDATA bytes indicates the HALT condition.

#### 43.4.4.5 PST Buffer Example

In this section is an example detailing the behavior of the PSTB functionality. Consider the following interrupt service routine that counts the interrupt, negates the IRQ, performs a software IACK, and then exits. This example is presented because it exercises a considerable set of the PSTB capabilities.

```

_isr:
01074: 46fc 2700    mov.w   &0x2700,%sr      # disable interrupts
01078: 2f08          mov.l   %a0,-(%sp)    # save a0
0107a: 2f00          mov.l   %d0,-(%sp)    # save d0
0107c: 302f 0008    mov.w   (8,%sp),%d0    # load format/vector word
01080: e488          lsr.l   &2,%d0       # align vector number
01082: 0280 0000 00ff andi.l  &0xff,%d0     # isolate vector number
01088: 207c 0080 1400 mov.l   &int_count,%a0  # base of interrupt counters

_isr_entry1:
0108e: 52b0 0c00    addq.l  &1,(0,%a0,%d0.l*4) # count the interrupt
01092: 11c0 a021    mov.b   %d0,IGCR0+1.w  # negate the irq
01096: 1038 a020    mov.b   IGCR0.w,%d0    # force the write to complete
0109a: 4e71          nop                 # synchronize the pipelines
0109c: 71b8 ffe0    mvz.b   SWIACK.w,%d0  # software iack: pending irq?
010a0: 0c80 0000 0041 cmpi.l  %d0,&0x41  # level 7 or none pending?
010a6: 6f08          ble.b   _isr_exit    # yes, then exit
010a8: 52b9 0080 145c addq.l  &1,swiack_count # increment the swiack count
010ae: 60de          bra.b   _isr_entry1 # continue at entry1

_isr_exit:
010b0: 201f          mov.l   (%sp)+,%d0    # restore d0
010b2: 205f          mov.l   (%sp)+,%a0    # restore a0

```

```
010b4: 4e73           rte          # exit
```

This ISR executes mostly as straight-line code: there is a single conditional branch @ PC = 0x10A6, which is taken in this example. The following description includes the PST and DDATA packets generated as this code snippet executes. In this example, the CSR setting enables the display of 2-byte branch addresses. The sequence begins with an interrupt exception:

```
interrupt exception occurs @ pc = 5432 while in user mode
                                                # pst      = 1c, 1c, 05, 0d
                                                # ddata   = 2a, 23, 28, 20
                                                #          trg_addr = 083a << 1
                                                #          trg_addr = 1074
_isr:
01074: 46fc 2700      mov.w    &0x2700,%sr      # pst      = 01
01078: 2f08            mov.l     %a0,-(%sp)    # pst      = 01
0107a: 2f00            mov.l     %d0,-(%sp)    # pst      = 01
0107c: 302f 0008      mov.w     (8,%sp),%d0    # pst      = 01
01080: e488            lsr.l     &2,%d0        # pst      = 01
01082: 0280 0000 00ff  andi.l    &0xff,%d0      # pst      = 01
01088: 207c 0080 1400  mov.l     &int_count,%a0    # pst      = 01
0108e: 52b0 0c00      addq.l    &1,(0,%a0,%d0.1*4) # pst      = 01
01092: 11c0 a021      mov.b     %d0,IGCR0+1.w  # pst      = 01
01096: 1038 a020      mov.b     IGCR0.w,%d0    # pst      = 01
0109a: 4e71            nop         # pst      = 01
0109c: 71b8 ffe0      mvz.b    SWIACK.w,%d0    # pst      = 01
010a0: 0c80 0000 0041  cmpi.l    %d0,&0x41      # pst      = 01
010a6: 6f08            ble.b     _isr_exit    # pst      = 05 (taken branch)
010b0: 201f            mov.l     (%sp)+,%d0    # pst      = 01
010b2: 205f            mov.l     (%sp)+,%a0    # pst      = 01
010b4: 4e73            rte          # pst      = 07, 03, 05, 0d
                                         # ddata   = 29, 21, 2a, 22
                                         #          trg_addr = 2a19 << 1
                                         #          trg_addr = 5432
```

As the PSTs are compressed, the resulting stream of 6-bit hexadecimal entries is loaded into consecutive locations in the PST trace buffer:

```
PSTB[*]= 1c, 1c, 05, 0d, // interrupt exception
                2a, 23, 28, 20, // branch target addr = 1074
                1a,             // 10 sequential insts
                13,             // 3 sequential insts
                05,             // taken_branch
                12,             // 2 sequential insts
                07, 03, 05, 0d, // rte, entry into user mode
                29, 21, 2a, 22 // branch target addr = 5432 (2a19 << 1)
```

#### 43.4.4.6 Code Profiling using PC Sync Hardware Feature

To enable code profiling, the following debug registers must be set:

1. CSR[BTB] = 0x1, 0x2, or 0x3 to enable 2-, 3-, or 4-byte instruction address captures
2. XCSR[APCENB, APCSC] and CSR2[APCDIV16] define the PC sampling frequency. This can range from once every 128 processor cycles to once every 16384 cycles.
3. CSR2[PSTBRM] = 0x2 or 0x3 to enable normal or continuous PC profile recording

4. Once configured with the start/stop recording conditions defined, the PSTB only records the periodically-sampled instruction addresses with each value containing packets of the form  $\{0x0E, 6 \times (0x20 + PC_s \gg 1)\}$ , where  $PC_s$  is the sampled program counter value configured to capture three bytes (six nibbles) of address. If the ownership trace option in the CSR is enabled ( $CSR[OTE] = 1$ ), a marker and the address space identifier (ASID) are also captured. The complete entry for this configuration is  $\{0x0E, 6 \times (0x20 + PC_s \gg 1), 0x0C, 2 \times (0x20 + ASID)\}$ .
5. To retrieve the most recently sampled PC, use the RDMREG BDM command described in [Section 43.4.1.5.15, “Read Debug Module Register \(rdmreg\)”](#).

### 43.4.5 Processor Status, Debug Data Definition

This section specifies the ColdFire processor and debug module’s generation of the processor status/debug data (PSTDDATA) output on an instruction basis. In general, the PSTDDATA output for an instruction is defined as follows:

$PSTDDATA = 0x1, \{[0x89B], \text{operand}\}$

where the  $\{\dots\}$  definition is optional operand information defined by the setting of the CSR.

The CSR provides capabilities to display operands based on reference type (read, write, or both). A PST value {0x8, 0x9, or 0xB} identifies the size and presence of valid data to follow on the PSTDDATA output {1, 2, or 4 bytes}. Additionally, for certain change-of-flow branch instructions, CSR[BTB] provides the capability to display the target instruction address on the PSTDDATA output {2, 3, or 4 bytes} using a PST value of {0x9, 0xA, or 0xB}. Addresses use the markers x0D, x0E, or 0xF to store 2, 3, or 4 bytes of address packets with address shifted right by 1 bit.

#### 43.4.5.1 User Instruction Set

[Table 43-40](#) shows the PSTDDATA specification for user-mode instructions. Rn represents any {Dn, An} register. In this definition, the y suffix generally denotes the source, and x denotes the destination operand. For a given instruction, the optional operand data is displayed only for those effective addresses referencing memory.

**Table 43-40. PSTDDATA Specification for User-Mode Instructions**

Instruction	Operand Syntax	PSTDDATA Nibble
add.l	<ea>y,Dx	PSTDDATA = 0x1, {0xB, source operand}
add.l	Dy,<ea>x	PSTDDATA = 0x1, {0xB, source}, {0xB, destination}
adda.l	<ea>y,Ax	PSTDDATA = 0x1, {0xB, source operand}
addi.l	#<data>,Dx	PSTDDATA = 0x1
addq.l	#<data>,<ea>x	PSTDDATA = 0x1, {0xB, source}, {0xB, destination}
addr.l	Dy,Dx	PSTDDATA = 0x1
and.l	<ea>y,Dx	PSTDDATA = 0x1, {0xB, source operand}
and.l	Dy,<ea>x	PSTDDATA = 0x1, {0xB, source}, {0xB, destination}
andi.l	#<data>,Dx	PSTDDATA = 0x1

**Table 43-40. PSTDDATA Specification for User-Mode Instructions (continued)**

<b>Instruction</b>	<b>Operand Syntax</b>	<b>PSTDDATA Nibble</b>
asl.l	{Dy,#<data>},Dx	PSTDDATA = 0x1
asr.l	{Dy,#<data>},Dx	PSTDDATA = 0x1
bcc.{b,w}		if taken, then PSTDDATA = 0x5, else PSTDDATA = 0x1
bchg.{b,l}	#<data>,<ea>x	PSTDDATA = 0x1, {0x8, source}, {0x8, destination}
bchg.{b,l}	Dy,<ea>x	PSTDDATA = 0x1, {0x8, source}, {0x8, destination}
bclr.{b,l}	#<data>,<ea>x	PSTDDATA = 0x1, {0x8, source}, {0x8, destination}
bclr.{b,l}	Dy,<ea>x	PSTDDATA = 0x1, {0x8, source}, {0x8, destination}
bitrev.l	Dx	PSTDDATA = 0x1
bra.{b,w}		PSTDDATA = 0x5
bset.{b,l}	#<data>,<ea>x	PSTDDATA = 0x1, {0x8, source}, {0x8, destination}
bset.{b,l}	Dy,<ea>x	PSTDDATA = 0x1, {0x8, source}, {0x8, destination}
bsr.{b,w}		PSTDDATA = 0x5, {0xB, destination operand}
btst.{b,l}	#<data>,<ea>x	PSTDDATA = 0x1, {0x8, source operand}
btst.{b,l}	Dy,<ea>x	PSTDDATA = 0x1, {0x8, source operand}
byterev.l	Dx	PSTDDATA = 0x1
clr.b	<ea>x	PSTDDATA = 0x1, {0x8, destination operand}
clr.l	<ea>x	PSTDDATA = 0x1, {0xB, destination operand}
clr.w	<ea>x	PSTDDATA = 0x1, {0x9, destination operand}
cmp.b	<ea>y,Dx	PSTDDATA = 0x1, {0x8, source operand}
cmp.l	<ea>y,Dx	PSTDDATA = 0x1, {0xB, source operand}
cmp.w	<ea>y,Dx	PSTDDATA = 0x1, {0x9, source operand}
cmpa.l	<ea>y,Ax	PSTDDATA = 0x1, {0xB, source operand}
cmpa.w	<ea>y,Ax	PSTDDATA = 0x1, {0x9, source operand}
cmpl.b	#<data>,Dx	PSTDDATA = 0x1
cmpl.l	#<data>,Dx	PSTDDATA = 0x1
cmpl.w	#<data>,Dx	PSTDDATA = 0x1
divs.l	<ea>y,Dx	PSTDDATA = 0x1, {0xB, source operand}
divs.w	<ea>y,Dx	PSTDDATA = 0x1, {0x9, source operand}
divu.l	<ea>y,Dx	PSTDDATA = 0x1, {0xB, source operand}
divu.w	<ea>y,Dx	PSTDDATA = 0x1, {0x9, source operand}
eor.l	Dy,<ea>x	PSTDDATA = 0x1, {0xB, source}, {0xB, destination}
eorl.l	#<data>,Dx	PSTDDATA = 0x1
ext.l	Dx	PSTDDATA = 0x1

**Table 43-40. PSTDDATA Specification for User-Mode Instructions (continued)**

Instruction	Operand Syntax	PSTDDATA Nibble
ext.w	Dx	PSTDDATA = 0x1
extb.l	Dx	PSTDDATA = 0x1
illegal		PSTDDATA = 0x1 <sup>1</sup>
jmp	<ea>y	PSTDDATA = 0x5, {[0x9AB], target address} <sup>2</sup>
jsr	<ea>y	PSTDDATA = 0x5, {[0x9AB], target address},{0xB, destination operand} <sup>2</sup>
lea.l	<ea>y,Ax	PSTDDATA = 0x1
link.w	Ay,#<displacement>	PSTDDATA = 0x1, {0xB, destination operand}
lsl.l	{Dy,#<data>},Dx	PSTDDATA = 0x1
lsr.l	{Dy,#<data>},Dx	PSTDDATA = 0x1
mov3q.l	#<data>,<ea>x	PSTDDATA = 0x1, {0xB,destination operand}
move.b	<ea>y,<ea>x	PSTDDATA = 0x1, {0x8, source}, {0x8, destination}
move.l	<ea>y,<ea>x	PSTDDATA = 0x1, {0xB, source}, {0xB, destination}
move.w	<ea>y,<ea>x	PSTDDATA = 0x1, {0x9, source}, {0x9, destination}
move.w	CCR,Dx	PSTDDATA = 0x1
move.w	{Dy,#<data>},CCR	PSTDDATA = 0x1
movea.l	<ea>y,Ax	PSTDDATA = 0x1, {0xB, source}
movea.w	<ea>y,Ax	PSTDDATA = 0x1, {0x9, source}
movem.l	#list,<ea>x	PSTDDATA = 0x1, {0xB, destination},... <sup>3</sup>
movem.l	<ea>y,#list	PSTDDATA = 0x1, {0xB, source},... <sup>3</sup>
moveq.l	#<data>,Dx	PSTDDATA = 0x1
muls.l	<ea>y,Dx	PSTDDATA = 0x1, {0xB, source operand}
muls.w	<ea>y,Dx	PSTDDATA = 0x1, {0x9, source operand}
mulu.l	<ea>y,Dx	PSTDDATA = 0x1, {0xB, source operand}
mulu.w	<ea>y,Dx	PSTDDATA = 0x1, {0x9, source operand}
mvs.b	<ea>y,Dx	PSTDDATA = 0x1, {0x8, source operand}
mvs.w	<ea>y,Dx	PSTDDATA = 0x1, {0x9, source operand}
mvz.b	<ea>y,Dx	PSTDDATA = 0x1, {0x8, source operand}
mvz.w	<ea>y,Dx	PSTDDATA = 0x1, {0x9, source operand}
neg.l	Dx	PSTDDATA = 0x1
negx.l	Dx	PSTDDATA = 0x1
nop		PSTDDATA = 0x1
not.l	Dx	PSTDDATA = 0x1
or.l	<ea>y,Dx	PSTDDATA = 0x1, {0xB, source operand}

**Table 43-40. PSTDDATA Specification for User-Mode Instructions (continued)**

Instruction	Operand Syntax	PSTDDATA Nibble
or.l	Dy,<ea>x	PSTDDATA = 0x1, {0xB, source}, {0xB, destination}
ori.l	#<data>,Dx	PSTDDATA = 0x1
pea.l	<ea>y	PSTDDATA = 0x1, {0xB, destination operand}
pulse		PSTDDATA = 0x4
rem3.l	<ea>y,Dw:Dx	PSTDDATA = 0x1, {0xB, source operand}
remu.l	<ea>y,Dw:Dx	PSTDDATA = 0x1, {0xB, source operand}
sats.l	Dx	PSTDDATA = 0x1
scc.b	Dx	PSTDDATA = 0x1
sub.l	<ea>y,Dx	PSTDDATA = 0x1, {0xB, source operand}
sub.l	Dy,<ea>x	PSTDDATA = 0x1, {0xB, source}, {0xB, destination}
suba.l	<ea>y,Ax	PSTDDATA = 0x1, {0xB, source operand}
subi.l	#<data>,Dx	PSTDDATA = 0x1
subq.l	#<data>,<ea>x	PSTDDATA = 0x1, {0xB, source}, {0xB, destination}
subx.l	Dy,Dx	PSTDDATA = 0x1
swap.w	Dx	PSTDDATA = 0x1
tas.b	<ea>x	PSTDDATA = 0x1, {0x8, source}, {0x8, destination}
tpf		PST = 0x1
tpf.l	#<data>	PST = 0x1
tpf.w	#<data>	PST = 0x1
trap	#<data>	PSTDDATA = 0x1 <sup>1</sup>
tst.b	<ea>x	PSTDDATA = 0x1, {0x8, source operand}
tst.l	<ea>y	PSTDDATA = 0x1, {0xB, source operand}
tst.w	<ea>y	PSTDDATA = 0x1, {0x9, source operand}
unlk	Ax	PSTDDATA = 0x1, {0xB, destination operand}
wddata.b	<ea>y	PSTDDATA = 0x4, {0x8, source operand}
wddata.l	<ea>y	PSTDDATA = 0x4, {0xB, source operand}
wddata.w	<ea>y	PSTDDATA = 0x4, {0x9, source operand}

- <sup>1</sup> During normal exception processing, the PSTDDATA output is driven to a 0xCC indicating the exception processing state. The exception stack write operands, as well as the vector read and target address of the exception handler may also be displayed.

Exception Processing:

```
PSTDDATA = 0xCC,  
{ 0xB,destination}, // stack frame  
{ 0xB,destination}, // stack frame  
{ 0xB,source}, // vector read  
PSTDDATA = 0x5,{ [ 0x9AB ],target } // handler PC
```

The PSTDDATA specification for the reset exception is shown below:

Exception Processing:

```
PSTDDATA = 0xCC,  
PSTDDATA = 0x5,{ [ 0x9AB ],target } // handler PC
```

The initial references at address 0 and 4 are never captured nor displayed because these accesses are treated as instruction fetches.

For all types of exception processing, the PSTDDATA = 0xCC value is driven at all times, unless the PSTDDATA output is needed for one of the optional marker values or for the taken branch indicator (0x5).

- 2 For JMP and JSR instructions, the optional target instruction address is displayed only for those effective address fields defining variant addressing modes. This includes the following <ea>x values: (An), (d16,An), (d8,An,Xi), (d8,PC,Xi).
  - 3 For move multiple instructions (MOVEM), the processor automatically generates line-sized transfers if the operand address reaches a 0-modulo-16 boundary and there are four or more registers to be transferred. For these line-sized transfers, the operand data is never captured nor displayed, regardless of the CSR value.
- The automatic line-sized burst transfers are provided to maximize performance during these sequential memory access operations.

Table 43-41 shows the PSTDDATA specification for multiply-accumulate instructions.

**Table 43-41. PSTDDATA Values for User-Mode Multiply-Accumulate Instructions**

Instruction	Operand Syntax	PSTDDATA Nibble
mac.l	Ry,Rx,ACCx	PSTDDATA = 0x1
mac.l	Ry,Rx,<ea>y,Rw,ACCx	PSTDDATA = 0x1, {0xB, source operand}
mac.w	Ry,Rx,ACCx	PSTDDATA = 0x1
mac.w	Ry,Rx,ea,Rw,ACCx	PSTDDATA = 0x1, {0xB, source operand}
move.l	{Ry,#<data>},ACCx	PSTDDATA = 0x1
move.l	{Ry,#<data>},MACSR	PSTDDATA = 0x1
move.l	{Ry,#<data>},MASK	PSTDDATA = 0x1
move.l	{Ry,#<data>},ACCext01	PSTDDATA = 0x1
move.l	{Ry,#<data>},ACCext23	PSTDDATA = 0x1
move.l	ACCext01,Rx	PSTDDATA = 0x1
move.l	ACCext23,Rx	PSTDDATA = 0x1
move.l	ACCy,ACCx	PSTDDATA = 0x1
move.l	ACCy,Rx	PSTDDATA = 0x1
move.l	MACSR,CCR	PSTDDATA = 0x1
move.l	MACSR,Rx	PSTDDATA = 0x1

**Table 43-41. PSTDDATA Values for User-Mode Multiply-Accumulate Instructions (continued)**

<b>Instruction</b>	<b>Operand Syntax</b>	<b>PSTDDATA Nibble</b>
move.l	MASK,Rx	PSTDDATA = 0x1
msac.l	Ry,Rx,ACCx	PSTDDATA = 0x1
msac.l	Ry,Rx,<ea>y,Rw,ACCx	PSTDDATA = 0x1, {0xB, source operand}
msac.w	Ry,Rx,ACCx	PSTDDATA = 0x1
msac.w	Ry,Rx,<ea>y,Rw,ACCx	PSTDDATA = 0x1, {0xB, source operand}

### 43.4.5.2 Supervisor Instruction Set

The supervisor instruction set has complete access to the user mode instructions plus the opcodes shown below. The PSTDDATA specification for these opcodes is shown in [Table 43-42](#).

**Table 43-42. PSTDDATA Specification for Supervisor-Mode Instructions**

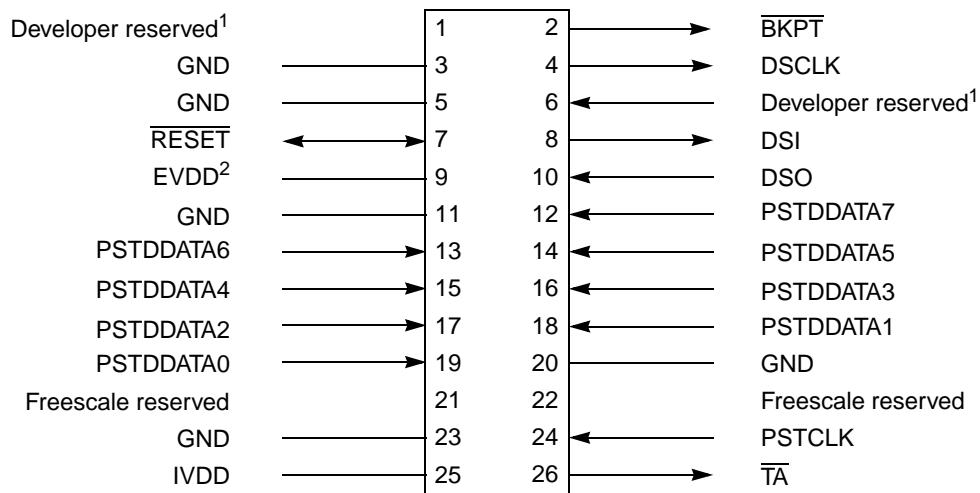
<b>Instruction</b>	<b>Operand Syntax</b>	<b>PSTDDATA Nibble</b>
cpushl	dc,(Ax) ic,(Ax) bc,(Ax)	PSTDDATA = 0x1
halt		PSTDDATA = 0x1, PSTDDATA = 0xF
intouch	(Ay)	PSTDDATA = 0x1
move.l	Ay,USP	PSTDDATA = 0x1
move.l	USP,Ax	PSTDDATA = 0x1
move.w	SR,Dx	PSTDDATA = 0x1
move.w	{Dy,#<data>},SR	PSTDDATA = 0x1, {0x3}
movec.l	Ry,Rc	PSTDDATA = 0x1, {8, ASID}
rte		PSTDDATA = 0x7, {0xB, source operand}, {0x3},{ 0xB, source operand}, {DD}, PSTDDATA = 0x5, {[0x9AB], target address}
stop	#<data>	PSTDDATA = 0x1, PSTDDATA = 0xE
wdebug.l	<ea>y	PSTDDATA = 0x1, {0xB, source, 0xB, source}

The move-to-SR and RTE instructions include an optional PSTDDATA = 0x3 value, indicating an entry into user mode. Additionally, if the execution of a RTE instruction returns the processor to emulator mode, a multiple-cycle status of 0xD is signaled.

Similar to the exception processing mode, the stopped state (PSTDDATA nibble = 0xE) and the halted state (PSTDDATA = 0xFF) display this status throughout the entire time the ColdFire processor is in the given mode.

### 43.4.6 Freescale-Recommended BDM Pinout

The ColdFire BDM connector is a 26-pin Berg connector arranged 2 x 13 as shown below.



<sup>1</sup> Pins reserved for BDM developer use.

<sup>2</sup> Supplied by target

Figure 43-55. Recommended BDM Connector



# Chapter 44

## IEEE 1149.1 Test Access Port (JTAG)

### 44.1 Introduction

The Joint Test Action Group (JTAG) is a dedicated user-accessible test logic compliant with the IEEE 1149.1 standard for boundary-scan testability, which helps with system diagnostic and manufacturing testing.

This architecture provides access to all data and chip control pins from the board-edge connector through the standard four-pin test access port (TAP) and the JTAG reset pin, TRST.

#### 44.1.1 Block Diagram

Figure 44-1 shows the block diagram of the JTAG module.

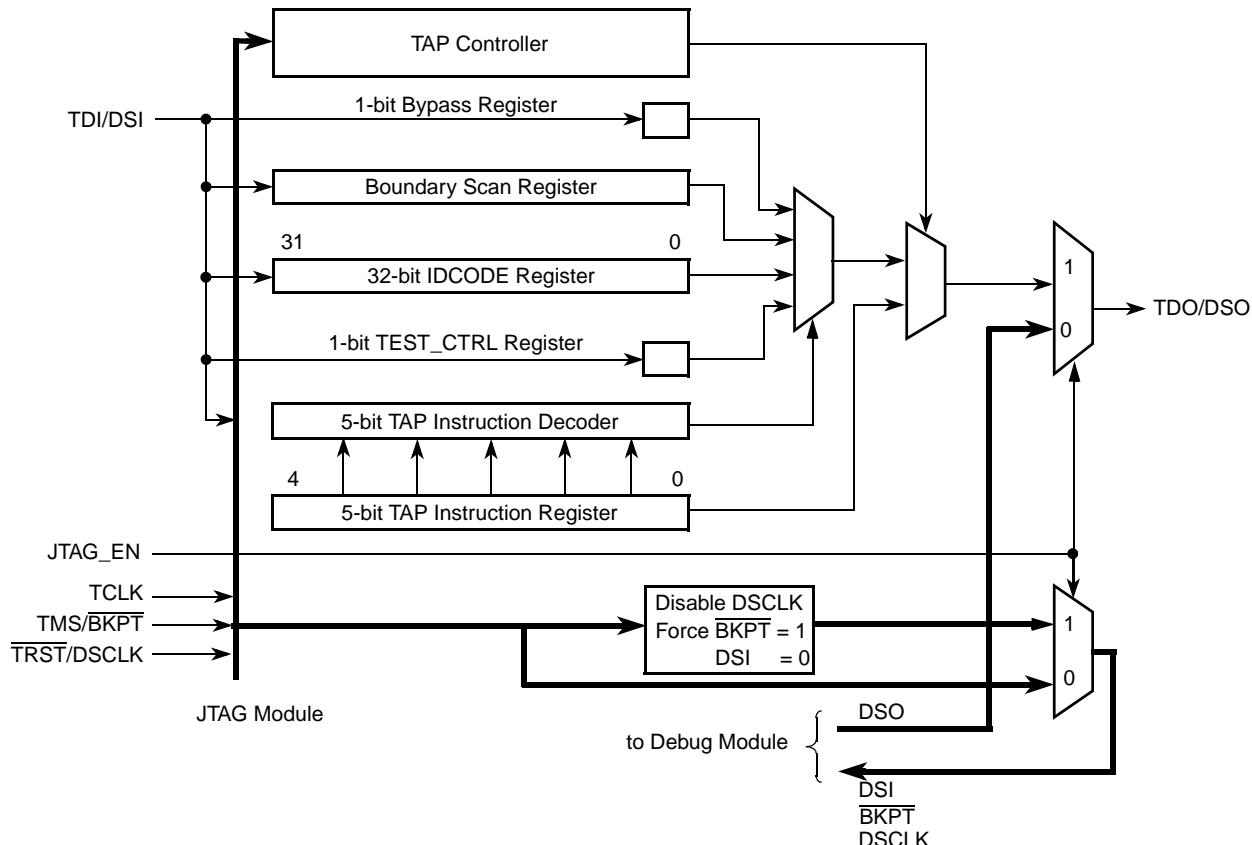


Figure 44-1. JTAG Block Diagram

## 44.1.2 Features

The basic features of the JTAG module are the following:

- Performs boundary-scan operations to test circuit board electrical continuity
- Bypasses instruction to reduce the shift register path to a single cell
- Sets chip output pins to safety states while executing the bypass instruction
- Samples the system pins during operation and transparently shifts out the result
- Selects between JTAG TAP controller and Background Debug Module (BDM) using a dedicated JTAG\_EN pin

## 44.1.3 Modes of Operation

The JTAG\_EN pin can select between the following modes of operation:

- JTAG mode (JTAG\_EN = 1)
- Background debug mode (BDM)—for more information, refer to [Section 43.4.1, “Background Debug Mode \(BDM\)”](#); (JTAG\_EN = 0).

## 44.2 External Signal Description

The JTAG module has five input and one output external signals, as described in [Table 44-1](#).

**Table 44-1. Signal Properties**

Name	Direction	Function	Reset State	Pull up
JTAG_EN	Input	JTAG/BDM selector input	—	—
TCLK	Input	JTAG Test clock input	—	Active
TMS/BKPT	Input	JTAG Test mode select / BDM Breakpoint	—	Active
TDI/DSI	Input	JTAG Test data input / BDM Development serial input	—	Active
TRST/DSCLK	Input	JTAG Test reset input / BDM Development serial clock	—	Active
TDO/DSO	Output	JTAG Test data output / BDM Development serial output	Hi-Z / 0	—

### 44.2.1 JTAG Enable (JTAG\_EN)

The JTAG\_EN pin selects between the debug module and JTAG. If JTAG\_EN is low, the debug module is selected; if it is high, the JTAG is selected. [Table 44-2](#) summarizes the pin function selected depending on JTAG\_EN logic state.

**Table 44-2. Pin Function Selected**

	<b>JTAG_EN = 0</b>	<b>JTAG_EN = 1</b>	<b>Pin Name</b>
Module selected	BDM	JTAG	—
Pin Function	— BKPT DSI DSO DSCLK	TCLK TMS TDI TDO TRST	TCLK BKPT DSI DSO DSCLK

When one module is selected, the inputs into the other module are disabled or forced to a known logic level, as shown in [Table 44-3](#), to disable the corresponding module.

**Table 44-3. Signal State to the Disable Module**

	<b>JTAG_EN = 0</b>	<b>JTAG_EN = 1</b>
Disabling JTAG	TRST = 0 TMS = 1	—
Disabling BDM	—	Disable DSCLK DSI = 0 BKPT = 1

### NOTE

The JTAG\_EN does not support dynamic switching between JTAG and BDM modes.

#### 44.2.2 Test Clock Input (TCLK)

The TCLK pin is a dedicated JTAG clock input to synchronize the test logic. Pulses on TCLK shift data and instructions into the TDI pin on the rising edge and out of the TDO pin on the falling edge. TCLK is independent of the processor clock. The TCLK pin has an internal pull-up resistor, and holding TCLK high or low for an indefinite period does not cause JTAG test logic to lose state information.

#### 44.2.3 Test Mode Select/Breakpoint (TMS/BKPT)

The TMS pin is the test mode select input that sequences the TAP state machine. TMS is sampled on the rising edge of TCLK. The TMS pin has an internal pull-up resistor.

The BKPT pin is used to request an external breakpoint. Assertion of BKPT puts the processor into a halted state after the current instruction completes.

#### 44.2.4 Test Data Input/Development Serial Input (TDI/DSI)

The TDI pin receives serial test and data, which is sampled on the rising edge of TCLK. Register values are shifted in least significant bit (lsb) first. The TDI pin has an internal pull-up resistor.

The DSI pin provides data input for the debug module serial communication port.

## 44.2.5 Test Reset/Development Serial Clock (TRST/DSCLK)

The **TRST** pin is an active low asynchronous reset input with an internal pull-up resistor that forces the TAP controller to the test-logic-reset state.

The DSCLK pin clocks the serial communication port to the debug module. Maximum frequency is 1/5 the processor clock speed. At the rising edge of DSCLK, data input on DS<sub>I</sub> is sampled and DSO changes state.

## 44.2.6 Test Data Output/Development Serial Output (TDO/DSO)

The TDO pin is the lsb-first data output. Data is clocked out of TDO on the falling edge of TCLK. TDO is tri-stateable and actively driven in the shift-IR and shift-DR controller states.

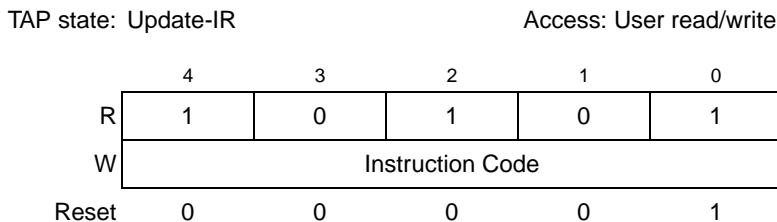
The DSO pin provides serial output data in BDM mode.

## 44.3 Memory Map/Register Definition

The JTAG module registers are not memory mapped and are only accessible through the TDO/DSO pin.

### 44.3.1 Instruction Shift Register (IR)

The JTAG module uses a 5-bit shift register with no parity. The IR transfers its value to a parallel hold register and applies an instruction on the falling edge of TCLK when the TAP state machine is in the update-IR state. To load an instruction into the shift portion of the IR, place the serial data on the TDI pin before each rising edge of TCLK. The msb of the IR is the bit closest to the TDI pin, and the lsb is the bit closest to the TDO pin. See [Section 44.4.3, “JTAG Instructions”](#) for a list of possible instruction codes.



**Figure 44-2. 5-Bit Instruction Register (IR)**

### 44.3.2 IDCODE Register

The IDCODE is a read-only register; its value is chip dependent. For more information, see Section 44.4.3.1, “[IDCODE Instruction](#)”.

<sup>1</sup> The reset values for PRN and PIN are device-dependent.

<sup>2</sup> Varies, depending on design center location.

**Figure 44-3. IDCODE Register**

**Table 44-4. IDCODE Field Descriptions**

Field	Description
31–28 PRN	Part revision number. Indicate the revision number of the device.
27–22 DC	Freescale design center number.
21–12 PIN	Part identification number. Indicate the device number.  0x09F MCF54410 0x0A0 MCF54415 0x0A1 MCF54416 0x0A2 MCF54417 0x0A3 MCF54418
11–1 JEDEC	Joint Electron Device Engineering Council ID bits. Indicate the reduced JEDEC ID for Freescale (0x0E).
0 ID	IDCODE register ID. This bit is set to 1 to identify the register as the IDCODE register and not the bypass register according to the IEEE standard 1149.1.

### 44.3.3 Bypass Register

The bypass register is a single-bit shift register path from TDI to TDO when the BYPASS, CLAMP, or HIGHZ instructions are selected. 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.

#### 44.3.4 TEST\_CTRL Register

The TEST\_CTRL register is a 1-bit shift register path from TDI to TDO when the ENABLE\_TEST\_CTRL instruction is selected. The TEST\_CTRL transfers its value to a parallel hold register on the rising edge of TCLK when the TAP state machine is in the update-DR state. The DSE bit selects the drive strength used in JTAG mode.



Figure 44-4. 1-Bit TEST\_CTRL Register

### 44.3.5 Boundary Scan Register

The boundary scan register is connected between TDI and TDO when the EXTEST or SAMPLE/PRELOAD instruction is selected. It captures input pin data, forces fixed values on output pins, and selects a logic value and direction for bidirectional pins or high impedance for tri-stated pins.

The boundary scan register contains bits for bonded-out and non bonded-out signals, excluding JTAG signals, analog signals, power supplies, compliance enable pins, device configuration pins, and clock signals.

## 44.4 Functional Description

### 44.4.1 JTAG Module

The JTAG module consists of a TAP controller state machine, which is responsible for generating all control signals that execute the JTAG instructions and read/write data registers.

### 44.4.2 TAP Controller

The TAP controller is a state machine that changes state based on the sequence of logical values on the TMS pin. [Figure 44-5](#) 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 TCLK signal.

Asserting the  $\overline{\text{TRST}}$  signal asynchronously resets the TAP controller to the test-logic-reset state. As [Figure 44-5](#) shows, holding TMS at logic 1 while clocking TCLK through at least five rising edges also causes the state machine to enter the test-logic-reset state, whatever the initial state.

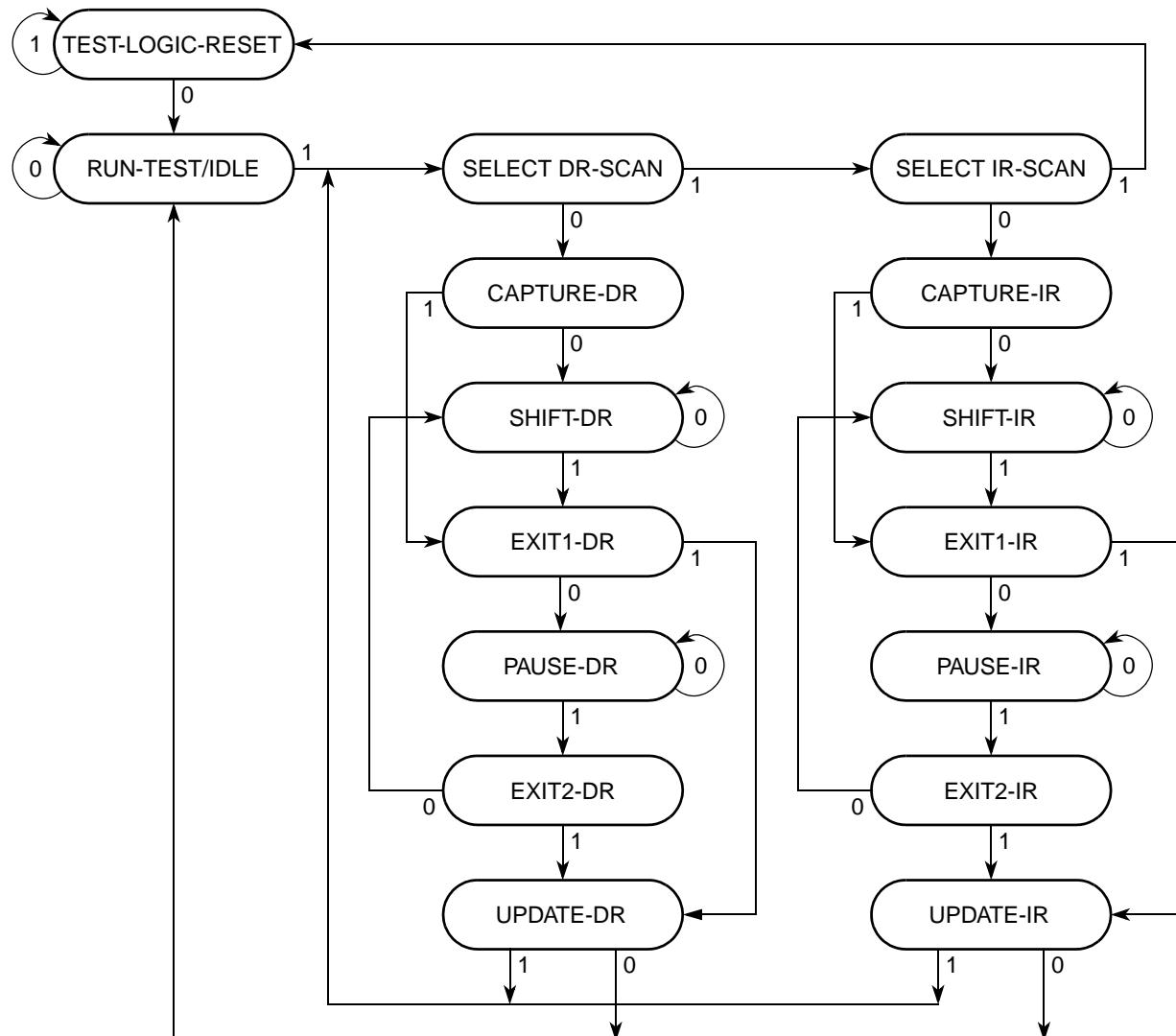


Figure 44-5. TAP Controller State Machine Flow

#### 44.4.3 JTAG Instructions

Table 44-5 describes public and private instructions.

Table 44-5. JTAG Instructions

Instruction	IR[4:0]	Instruction Summary
IDCODE	00001	Selects IDCODE 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

**Table 44-5. JTAG Instructions (continued)**

Instruction	IR[4:0]	Instruction Summary
EXTEST	00100	Selects boundary scan register while applying preloaded values to output pins and asserting functional reset
ENABLE_TEST_CTRL	00110	Selects TEST_CTRL register
HIGHZ	01001	Selects bypass register while tri-stating all output pins and asserting functional reset
CLAMP	01100	Selects bypass while applying fixed values to output pins and asserting functional reset
BYPASS	11111	Selects bypass register for data operations
Reserved	all others <sup>1</sup>	Decoded to select bypass register

<sup>1</sup> Freescale reserves the right to change the decoding of the unused opcodes in the future.

#### 44.4.3.1 IDCODE Instruction

The IDCODE instruction selects the 32-bit IDCODE register for connection as a shift path between the TDI and TDO pin. This instruction allows interrogation of the MCU to determine its version number and other part identification data. The shift register lsb is forced to logic 1 on the rising edge of TCLK following entry into the capture-DR state. Therefore, the first bit to be shifted out after selecting the IDCODE register is always a logic 1. The remaining 31 bits are also forced to fixed values on the rising edge of TCLK following entry into the capture-DR state.

IDCODE is the default instruction placed into the instruction register when the TAP resets. Thus, after a TAP reset, the IDCODE register is selected automatically.

#### 44.4.3.2 SAMPLE/PRELOAD Instruction

The SAMPLE/PRELOAD instruction has two functions:

- SAMPLE - See [Section 44.4.3.3, “SAMPLE Instruction,”](#) for description of this function.
- PRELOAD - initialize the boundary scan register update cells before selecting EXTEST or CLAMP. This is achieved by ignoring the data shifting out on the TDO pin and shifting in initialization data. The update-DR state and the falling edge of TCLK can then transfer this data to the update cells. The data is applied to the external output pins by the EXTEST or CLAMP instruction.

#### 44.4.3.3 SAMPLE Instruction

The SAMPLE instruction obtains a sample of the system data and control signals present at the MCU input pins and before the boundary scan cell at the output pins. This sampling occurs on the rising edge of TCLK in the capture-DR state when the IR contains the 0x2 opcode. The sampled data is accessible by shifting it through the boundary scan register to the TDO output by using the shift-DR state. The data capture and the shift operation are transparent to system operation.

## NOTE

External synchronization is required to achieve meaningful results because there is no internal synchronization between TCLK and the system clock.

### **44.4.3.4 EXTEST Instruction**

The external test (EXTEST) instruction selects the boundary scan register. It forces all output pins and bidirectional pins configured as outputs to the values preloaded with the SAMPLE/PRELOAD instruction and held in the boundary scan update registers. EXTEST can also configure the direction of bidirectional pins and establish high-impedance states on some pins. EXTEST asserts internal reset for the MCU system logic to force a predictable internal state while performing external boundary scan operations.

### **44.4.3.5 ENABLE\_TEST\_CTRL Instruction**

The ENABLE\_TEST\_CTRL instruction selects a 1-bit shift register (TEST\_CTRL) for connection as a shift path between the TDI and TDO pin. When the user transitions the TAP controller to the UPDATE\_DR state, the register transfers its value to a parallel hold register.

### **44.4.3.6 HIGHZ Instruction**

The HIGHZ instruction eliminates the need to backdrive the output pins during circuit-board testing. HIGHZ turns off all output drivers, including the 2-state drivers, and selects the bypass register. HIGHZ also asserts internal reset for the MCU system logic to force a predictable internal state.

### **44.4.3.7 CLAMP Instruction**

The CLAMP instruction selects the 1-bit bypass register and asserts internal reset while simultaneously forcing all output pins and bidirectional pins configured as outputs to the fixed values that are preloaded and held in the boundary scan update register. 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.

### **44.4.3.8 BYPASS Instruction**

The BYPASS instruction selects the bypass register, creating a single-bit shift register path from the TDI pin to the TDO pin. BYPASS enhances test efficiency by reducing the overall shift path when a device other than the ColdFire processor is the device under test on a board design with multiple chips on the overall boundary scan chain. The shift register lsb is forced to logic 0 on the rising edge of TCLK after entry into the capture-DR state. Therefore, the first bit shifted out after selecting the bypass register is always logic 0. This differentiates parts that support an IDCODE register from parts that support only the bypass register.

## 44.5 Initialization/Application Information

### 44.5.1 Restrictions

The test logic is a static logic design, and TCLK can be stopped in a high or low state without loss of data. However, the system clock is not synchronized to TCLK internally. Any mixed operation using the test logic and system functional logic requires external synchronization.

Using the EXTEST instruction requires a circuit-board test environment that avoids device-destructive configurations in which MCU output drivers are enabled into actively driven networks.

Low-power stop mode considerations:

- The TAP controller must be in the test-logic-reset state to enter or remain in the low-power stop mode. Leaving the test-logic-reset state negates the ability to achieve low-power, but does not otherwise affect device functionality.
- The TCLK input is not blocked in low-power stop mode. To consume minimal power, the TCLK input should be externally connected to EV<sub>DD</sub>.
- The TMS, TDI, and TRST pins include on-chip pull-up resistors. For minimal power consumption in low-power stop mode, these three pins should be connected to EV<sub>DD</sub> or left unconnected.

### 44.5.2 Nonscan Chain Operation

Keeping the TAP controller in the test-logic-reset state ensures that the scan chain test logic is transparent to the system logic. It is recommended that TMS, TDI, TCLK, and TRST be pulled up. TRST could be connected to ground. However, because there is a pull-up on TRST, some amount of current results. The internal power-on reset input initializes the TAP controller to the test-logic-reset state on power-up without asserting TRST.