

MCF5208 Reference Manual

Devices Supported:
MCF5207
MCF5208

Document Number: MCF5208RM
Rev. 0.1
08/2005



How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:

Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

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)
support@freescale.com

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

Asia/Pacific:

Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 26668334
support.asia@freescale.com

For Literature Requests Only:

Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
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. © Freescale Semiconductor, Inc. 2005. All rights reserved.

MCF5208RM
Rev. 0.1
08/2005

Contents

Paragraph Number	Title	Page Number
About This Book		
Audience	xxv	
Organization.....	xxv	
Suggested Reading.....	xxvii	
Hardware Specification.....	xxvii	
General Information.....	xxviii	
ColdFire Documentation.....	xxviii	
Conventions	xxviii	
Register Figure Conventions	xxix	
Acronyms and Abbreviations	xxx	
Terminology Conventions	xxxi	
Revision History	xxxiii	
Chapter 1 Overview		
1.1	MCF5207/8 Device Configurations	1-1
1.2	Block Diagram	1-2
1.3	Features	1-3
1.3.1	V2 Core Overview	1-6
1.3.2	Debug Module	1-6
1.3.3	JTAG	1-7
1.3.4	On-chip Memories	1-7
1.3.4.1	Cache	1-7
1.3.4.2	SRAM	1-8
1.3.5	SDR/DDR SDRAM Controller	1-8
1.3.6	Fast Ethernet Controller (FEC)	1-8
1.3.7	UARTs	1-8
1.3.8	I ² C Bus	1-8
1.3.9	QSPI	1-8
1.3.10	DMA Timers (DTIM0-DTIM3)	1-9
1.3.11	On-chip Watchdog Timer	1-9
1.3.12	Periodic Interrupt Timers (PIT0-PIT3)	1-9
1.3.13	Clock Module and Phase Locked Loop (PLL)	1-9
1.3.14	Interrupt Controller	1-9
1.3.15	DMA Controller	1-9
1.3.16	FlexBus External Interface	1-10

Contents

Paragraph Number	Title	Page Number
1.3.17	Reset Controller Module	1-10
1.3.18	GPIO	1-10
1.4	Documentation	1-11

Chapter 2 Signal Descriptions

2.1	Introduction	2-1
2.2	Signal Properties Summary	2-1
2.2.1	Internal Pull-up/Pull-downs Resistors	2-6
2.3	Signal Primary Functions	2-7
2.3.1	Reset Signals	2-7
2.3.2	PLL and Clock Signals	2-7
2.3.3	Mode Selection	2-7
2.3.4	FlexBus Signals	2-8
2.3.5	SDRAM Controller Signals	2-9
2.3.6	External Interrupt Signals	2-9
2.3.7	DMA Signals	2-10
2.3.8	Ethernet Module (FEC) Signals	2-10
2.3.9	I2C I/O Signals	2-11
2.3.10	Queued Serial Peripheral Interface (QSPI)	2-11
2.3.11	UART Module Signals	2-12
2.3.12	DMA Timer Signals	2-12
2.3.13	Debug Support Signals	2-13
2.3.14	Test Signals	2-14
2.3.15	Power and Ground Pins	2-14
2.4	External Boot Mode	2-15

Chapter 3 ColdFire Core

3.1	Processor Pipelines	3-1
3.2	Memory Map/Register Definition	3-2
3.2.1	Data Registers (D0–D7)	3-3
3.2.2	Address Registers (A0–A6)	3-4
3.2.3	Stack Pointer (A7)	3-4
3.2.4	EMAC Register Description	3-5
3.2.5	Condition Code Register (CCR)	3-5
3.2.6	Program Counter (PC)	3-6
3.2.7	Cache Control Register (CACR)	3-6

Contents

Paragraph Number	Title	Page Number
3.2.8	Access Control Registers (ACR0, ACR1)	3-6
3.2.9	Supervisor/User Stack Pointers (A7 and OTHER_A7)	3-6
3.2.10	Vector Base Register (VBR)	3-7
3.2.11	Status Register (SR)	3-7
3.2.12	RAM Base Address Register (RAMBAR1)	3-8
3.3	Additions to the Instruction Set Architecture	3-8
3.4	Exception Processing Overview	3-9
3.5	Exception Stack Frame Definition	3-11
3.6	Processor Exceptions	3-12
3.6.1	Access Error Exception	3-12
3.6.2	Address Error Exception	3-12
3.6.3	Illegal Instruction Exception	3-12
3.6.4	Divide-By-Zero	3-13
3.6.5	Privilege Violation	3-13
3.6.6	Trace Exception	3-13
3.6.7	Unimplemented Line-A Opcode	3-13
3.6.8	Unimplemented Line-F Opcode	3-14
3.6.9	Debug Interrupt	3-14
3.6.10	RTE and Format Error Exception	3-14
3.6.11	TRAP Instruction Exception	3-14
3.6.12	Interrupt Exception	3-14
3.6.13	Fault-on-Fault Halt	3-15
3.6.14	Reset Exception	3-15
3.6.15	Reset Vector	3-15
3.7	Instruction Execution Timing	3-18
3.7.1	Timing Assumptions	3-18
3.7.2	MOVE Instruction Execution Times	3-19
3.8	Standard One Operand Instruction Execution Times	3-20
3.9	Standard Two Operand Instruction Execution Times	3-21
3.10	Miscellaneous Instruction Execution Times	3-22
3.11	EMAC Instruction Execution Times	3-23
3.12	Branch Instruction Execution Times	3-25

Chapter 4 Enhanced Multiply-Accumulate Unit (EMAC)

4.1	Multiply-Accumulate Unit	4-1
4.2	Introduction to the MAC	4-2
4.3	General Operation	4-2
4.4	Memory Map/Register Definition	4-5
4.4.1	MAC Status Register (MACSR)	4-5

Contents

Paragraph Number	Title	Page Number
4.4.1.1	Fractional Operation Mode	4-7
4.4.1.1.1	Rounding	4-7
4.4.1.1.2	Saving and Restoring the EMAC Programming Model	4-8
4.4.1.1.3	MULS/MULU	4-9
4.4.1.1.4	Scale Factor in MAC or MSAC Instructions	4-9
4.4.2	Mask Register (MASK)	4-9
4.4.3	Accumulator Registers (ACC0–3)	4-10
4.4.4	Accumulator Extension Registers (ACCext01, ACCext23)	4-10
4.5	EMAC Instruction Set Summary	4-11
4.5.1	EMAC Instruction Execution Times	4-12
4.5.2	Data Representation	4-13
4.5.3	MAC Opcodes	4-13

Chapter 5 Cache

5.1	Introduction	5-1
5.1.1	Features	5-1
5.1.2	Physical Organization	5-1
5.1.3	Operation	5-3
5.1.3.1	Interaction with Other Modules	5-3
5.1.3.2	Memory Reference Attributes	5-3
5.1.3.3	Cache Coherency and Invalidation	5-3
5.1.3.4	Reset	5-4
5.1.3.5	Cache Miss Fetch Algorithm/Line Fills	5-4
5.2	Memory Map/Register Definition	5-5
5.2.1	Cache Control Register (CACR)	5-6
5.2.2	Access Control Registers (ACR0, ACR1)	5-9

Chapter 6 Static RAM (SRAM)

6.1	Introduction	6-1
6.1.1	Features	6-1
6.1.2	Operation	6-1
6.2	Memory Map/Register Description	6-1
6.2.1	SRAM Base Address Register (RAMBAR)	6-2
6.3	Initialization/Application Information	6-4
6.3.1	SRAM Initialization Code	6-4
6.3.2	Power Management	6-4

Contents

Paragraph Number	Title	Page Number
	Chapter 7 Clock Module	
7.1	Introduction	7-1
7.1.1	Block Diagram	7-2
7.1.2	Features	7-3
7.1.3	Modes of Operation	7-3
7.1.3.1	Normal PLL Mode with Crystal Reference	7-3
7.1.3.2	Normal PLL Mode with External Reference	7-3
7.1.3.3	Input Clock (Limp) Mode	7-3
7.1.3.4	Low-power Mode Operation	7-4
7.2	Memory Map/Register Definition	7-5
7.2.1	PLL Output Divider Register (PODR)	7-5
7.2.2	PLL Control Register (PCR)	7-6
7.2.3	PLL Modulation Divider Register (PMDR)	7-7
7.2.4	PLL Feedback Divider Register (PFDR)	7-8
7.3	Functional Description	7-8
7.3.1	PLL Dithered and Non-Dithered Operation	7-8
7.3.2	Dithering Waveform Definition	7-9
7.3.3	PLL Frequency Multiplication Factor Select	7-10
7.3.4	System Clock Modes	7-10
7.3.5	Clock Operation During Reset	7-11
7.3.5.1	Power-On Reset (POR)	7-11
7.3.5.2	External Reset	7-11

Chapter 8 Power Management

8.1	Introduction	8-1
8.1.1	Features	8-1
8.2	Memory Map/Register Definition	8-1
8.2.1	Wake-up Control Register	8-2
8.2.2	Peripheral Power Management Set Register (PPMSR0)	8-3
8.2.3	Peripheral Power Management Clear Register (PPMCR)	8-4
8.2.4	Peripheral Power Management Registers (PPMHR0 & PPMLR0)	8-4
8.2.5	Low-Power Control Register (LPCR)	8-6
8.2.6	Miscellaneous Control Register (MISCCR)	8-7
8.3	Functional Description	8-8
8.3.1	Peripheral Shut Down	8-8
8.3.2	Limp mode	8-8
8.3.3	Low-Power Modes	8-9

Contents

Paragraph Number	Title	Page Number
8.3.3.1	Run Mode	8-9
8.3.3.2	Wait Mode	8-9
8.3.3.3	Doze Mode	8-9
8.3.3.4	Stop Mode	8-9
8.3.4	Peripheral Behavior in Low-Power Modes	8-10
8.3.4.1	ColdFire Core	8-10
8.3.4.2	Internal SRAM	8-10
8.3.4.3	Clock Module	8-10
8.3.4.4	Chip Configuration Module	8-10
8.3.4.5	Reset Controller	8-11
8.3.4.6	System Control Module (SCM)	8-11
8.3.4.7	Cross-Bar Switch	8-11
8.3.4.8	GPIO Ports	8-11
8.3.4.9	Interrupt Controllers (INTC0)	8-11
8.3.4.10	Edge Port	8-12
8.3.4.11	eDMA Controller	8-12
8.3.4.12	FlexBus Module	8-12
8.3.4.13	SDRAM Controller (SDRAMC)	8-12
8.3.4.14	Fast Ethernet Controller (FEC)	8-12
8.3.4.15	On-chip Watchdog Timer	8-13
8.3.4.16	Programmable Interrupt Timers (PIT0–3)	8-13
8.3.4.17	DMA Timers (DTIM0–3)	8-13
8.3.4.18	Queued Serial Peripheral Interface (QSPI)	8-13
8.3.4.19	UART Modules (UART0–2)	8-13
8.3.4.20	I2C Module	8-14
8.3.4.21	BDM	8-14
8.3.4.22	JTAG	8-14
8.3.5	Summary of Peripheral State During Low-power Modes	8-14

Chapter 9 Chip Configuration Module (CCM)

9.1	Introduction	9-1
9.1.1	Block Diagram	9-1
9.1.2	Features	9-1
9.1.3	Modes of Operation	9-1
9.2	External Signal Descriptions	9-2
9.2.1	RCON	9-2
9.2.2	D[9:1] (Reset Configuration Override)	9-2
9.3	Memory Map/Register Definition	9-2
9.3.1	Chip Configuration Register (CCR)	9-2

Contents

Paragraph Number	Title	Page Number
9.3.2	Reset Configuration Register (RCON)	9-3
9.3.3	Chip Identification Register (CIR)	9-4
9.4	Functional Description	9-4
9.4.1	Reset Configuration	9-5
9.4.2	PLL Mode Selection	9-6
9.4.3	Oscillator Mode Selection	9-6
9.4.4	Boot Device Selection	9-7
9.4.5	Output Pad Strength Configuration	9-7
9.4.6	Chip Select Configuration	9-7

Chapter 10 Reset Controller Module

10.1	Introduction	10-1
10.1.1	Block Diagram	10-1
10.1.2	Features	10-1
10.2	External Signal Description	10-2
10.2.1	RESET	10-2
10.2.2	RSTOUT	10-2
10.3	Memory Map/Register Definition	10-2
10.3.1	Reset Control Register (RCR)	10-2
10.3.2	Reset Status Register (RSR)	10-3
10.4	Functional Description	10-4
10.4.1	Reset Sources	10-4
10.4.1.1	Power-On Reset	10-5
10.4.1.2	External Reset	10-5
10.4.1.3	On-chip Watchdog Timer Reset	10-5
10.4.1.4	Core Watchdog Timer Reset	10-5
10.4.1.5	Loss-of-Lock Reset	10-5
10.4.1.6	Software Reset	10-5
10.4.2	Reset Control Flow	10-5
10.4.2.1	Synchronous Reset Requests	10-6
10.4.2.2	Internal Reset Request	10-7
10.4.2.3	Power-On Reset	10-7
10.4.3	Concurrent Resets	10-7
10.4.3.1	Reset Flow	10-7
10.4.3.2	Reset Status Flags	10-7

Contents

Paragraph Number	Title	Page Number
Chapter 11 System Control Module (SCM)		
11.1	Introduction	11-1
11.1.1	Overview	11-1
11.1.2	Features	11-1
11.2	Memory Map/Register Definition	11-2
11.2.1	Master Privilege Register (MPR)	11-2
11.2.2	Peripheral Access Control Registers (PACRx)	11-3
11.2.3	Bus Monitor Timeout Register (BMT)	11-6
11.2.4	Core Watchdog Control Register (CWCR)	11-7
11.2.5	Core Watchdog Service Register (CWSR)	11-8
11.2.6	SCM Interrupt Status Register (SCMISR)	11-8
11.2.7	Core Fault Address Register (CFADR)	11-9
11.2.8	Core Fault Interrupt Enable Register (CFIER)	11-9
11.2.9	Core Fault Location Register (CFLOC)	11-10
11.2.10	Core Fault Attributes Register (CFATR)	11-10
11.2.11	Core Fault Data Register (CFDTR)	11-11
11.3	Functional Description	11-12
11.3.1	Access Control	11-12
11.3.2	Core Watchdog Timer	11-12
11.3.3	Core Data Fault Recovery Registers	11-13
Chapter 12 Cross-Bar Switch Module (XBS)		
12.1	Overview	12-1
12.2	Features	12-2
12.3	Modes of Operation	12-2
12.4	Memory Map / Register Definition	12-2
12.4.1	XBS Priority Registers (XBS_PRSn)	12-3
12.4.2	XBS Control Registers (XBS_CRSn)	12-4
12.5	Functional Description	12-5
12.5.1	Arbitration	12-5
12.5.1.1	Fixed-Priority Operation	12-5
12.5.1.2	Round-Robin Priority Operation	12-6
12.5.1.3	Priority Assignment	12-6
12.6	Initialization / Application Information	12-6

Contents

Paragraph Number	Title	Page Number
Chapter 13 General Purpose I/O Module		
13.1	Introduction	13-1
13.1.1	Overview	13-2
13.1.2	Features	13-3
13.2	External Signal Description	13-3
13.3	Memory Map/Register Definition	13-8
13.3.1	Port Output Data Registers (PODR_x)	13-11
13.3.2	Port Data Direction Registers (PDDR_x)	13-12
13.3.3	Port Pin Data/Set Data Registers (PPDSDR_x)	13-13
13.3.4	Port Clear Output Data Registers (PCLRR_x)	13-14
13.3.5	Pin Assignment Registers (PAR_x)	13-15
13.3.5.1	External Bus Control Pin Assignment Register (PAR_BUSCTL)	13-16
13.3.5.2	Byte Enable Pin Assignment Register (PAR_BE)	13-16
13.3.5.3	Chip Select Pin Assignment Register (PAR_CS)	13-17
13.3.5.4	FEC/I2C Pin Assignment Register (PAR_FECI2C)	13-17
13.3.5.5	QSPI Pin Assignment Register (PAR_QSPI)	13-18
13.3.5.6	Timer Pin Assignment Registers (PAR_TIMER)	13-19
13.3.5.7	UART Pin Assignment Register (PAR_UART)	13-19
13.3.5.8	FEC Pin Assignment Register (PAR_FEC)	13-21
13.3.5.9	IRQ Pin Assignment Register (PAR_IRQ)	13-22
13.3.6	FlexBus Mode Select Control Register (MSCR_FLEXBUS)	13-22
13.3.7	SDRAM Mode Select Control Register (MSCR_SDRAM)	13-23
13.3.8	Drive Strength Control Registers (DSCR_x)	13-24
13.3.8.1	I ² C Drive Strength Control Register (DSCR_I2C)	13-24
13.3.8.2	Miscellaneous Drive Strength Control Register (DSCR_MISC)	13-24
13.3.8.3	FEC Drive Strength Control Register (DSCR_FEC)	13-25
13.3.8.4	UART/IRQ Drive Strength Control Register (DSCR_UART)	13-25
13.3.8.5	QSPI Drive Strength Control Register (DSCR_QSPI)	13-26
13.4	Functional Description	13-27
13.4.1	Overview	13-27
13.4.2	Port Digital I/O Timing	13-27
13.5	Initialization/Application Information	13-28

Chapter 14 Interrupt Controller Module

14.1	Introduction	14-1
14.1.1	68K/ColdFire Interrupt Architecture Overview	14-1
14.2	Memory Map/Register Definition	14-2

Contents

Paragraph Number	Title	Page Number
14.2.1	Interrupt Pending Registers (IPRH, IPRL)	14-3
14.2.2	Interrupt Mask Register (IMRH, IMRL)	14-4
14.2.3	Interrupt Force Registers (INTFRCH, INTFRCL)	14-6
14.2.4	Interrupt Configuration Register (ICONFIG)	14-6
14.2.5	Set Interrupt Mask Register (SIMR)	14-7
14.2.6	Clear Interrupt Mask Register (CIMR)	14-8
14.2.7	Current Level Mask Register (CLMASK)	14-8
14.2.8	Saved Level Mask Register (SLMASK)	14-9
14.2.9	Interrupt Control Register (ICR n , ($n = 01, 02, \dots, 63$))	14-10
14.2.9.1	Interrupt Sources	14-11
14.2.10	Software and Level n IACK Registers (SWIACK n , L1IACK–L7IACK n)	14-12
14.3	Functional Description	14-13
14.3.1	Interrupt Controller Theory of Operation	14-13
14.3.1.1	Interrupt Recognition	14-14
14.3.1.2	Interrupt Prioritization	14-14
14.3.1.3	Interrupt Vector Determination	14-14
14.3.2	Low-Power Wake-up Operation	14-15
14.4	Initialization/Application Information	14-16
14.4.1	Interrupt Service Routines	14-16

Chapter 15 Edge Port Module (EPORT)

15.1	Introduction	15-1
15.2	Low-Power Mode Operation	15-2
15.3	Interrupt/GPIO Pin Descriptions	15-2
15.4	Memory Map/Register Definition	15-2
15.4.1	EPORT Pin Assignment Register (EPPAR)	15-3
15.4.2	EPORT Data Direction Register (EPDDR)	15-4
15.4.3	Edge Port Interrupt Enable Register (EPIER)	15-4
15.4.4	Edge Port Data Register (EPDR)	15-5
15.4.5	Edge Port Pin Data Register (EPPDR)	15-5
15.4.6	Edge Port Flag Register (EPFR)	15-6

Chapter 16 Enhanced Direct Memory Access (eDMA)

16.1	Overview	16-1
16.2	Block Diagram	16-1
16.3	Features	16-2

Contents

Paragraph Number	Title	Page Number
16.4	eDMA Modes of Operation	16-2
16.4.1	Normal Mode	16-2
16.4.2	Debug Mode	16-2
16.5	External Signal Description	16-3
16.6	Memory Map/Register Definition	16-3
16.6.1	eDMA Control Register (EDMA_CR)	16-4
16.6.2	eDMA Error Status Register (EDMA_ES)	16-4
16.6.3	eDMA Enable Request Register (EDMA_ERQ)	16-7
16.6.4	eDMA Enable Error Interrupt Registers (EDMA_EEI)	16-8
16.6.5	eDMA Set Enable Request Register (EDMA_SERQ)	16-8
16.6.6	eDMA Clear Enable Request Register (EDMA_CERQ)	16-9
16.6.7	eDMA Set Enable Error Interrupt Register (EDMA_SEEI)	16-10
16.6.8	eDMA Clear Enable Error Interrupt Register (EDMA_CEEI)	16-10
16.6.9	eDMA Clear Interrupt Request Register (EDMA_CINT)	16-11
16.6.10	eDMA Clear Error Register (EDMA_CERR)	16-12
16.6.11	eDMA Set START Bit Register (EDMA_SSRT)	16-12
16.6.12	eDMA Clear DONE Status Bit Register (EDMA_CDNE)	16-13
16.6.13	eDMA Interrupt Request Register (EDMA_INT)	16-14
16.6.14	eDMA Error Register (EDMA_ERR)	16-14
16.6.15	eDMA Channel n Priority Registers (DCHPRIn)	16-15
16.6.16	Transfer Control Descriptors (TCDn)	16-16
16.7	Functional Description	16-23
16.7.1	eDMA Microarchitecture	16-23
16.7.2	eDMA Basic Data Flow	16-24
16.8	Initialization/Application Information	16-27
16.8.1	eDMA Initialization	16-27
16.8.2	DMA Programming Errors	16-30
16.8.3	DMA Arbitration Mode Considerations	16-30
16.8.3.1	Fixed Channel Arbitration	16-30
16.8.3.2	Round Robin Channel Arbitration	16-31
16.8.4	DMA Transfer	16-31
16.8.4.1	Single Request	16-31
16.8.4.2	Multiple Requests	16-32
16.8.4.3	Modulo Feature	16-33
16.8.5	eDMA TCDn Status Monitoring	16-34
16.8.5.1	Minor Loop Complete	16-34
16.8.5.2	Active Channel TCDn Reads	16-35
16.8.5.3	Preemption Status	16-35
16.8.6	Channel Linking	16-35
16.8.7	Dynamic Programming	16-36
16.8.7.1	Dynamic Channel Linking and Dynamic Scatter/Gather	16-37

Contents

Paragraph Number	Title	Page Number
	Chapter 17 FlexBus	
17.1	Introduction	17-1
17.1.1	Overview	17-1
17.1.2	Features	17-1
17.2	External Signals	17-2
17.2.1	Address & Data Buses (A[23:0], D[31:0])	17-2
17.2.2	Chip-Selects ($\overline{FB_CS}[5:0]$)	17-2
17.2.3	Byte Enables/Byte Write Enables ($\overline{BE/BWE}[3:0]$)	17-2
17.2.4	Output Enable (\overline{OE})	17-3
17.2.5	Read/Write (R/W)	17-3
17.2.6	Transfer Start (TS)	17-3
17.2.7	Transfer Acknowledge (TA)	17-3
17.3	Chip-Select Operation	17-3
17.3.1	General Chip-Select Operation	17-4
17.3.1.1	8-, 16-, and 32-Bit Port Sizing	17-4
17.3.1.2	Global Chip-Select Operation	17-4
17.3.2	Memory Map/Register Definition	17-5
17.3.2.1	Chip-Select Address Registers (CSAR0–CSAR5)	17-5
17.3.2.2	Chip-Select Mask Registers (CSMR0–CSMR5)	17-6
17.3.2.3	Chip-Select Control Registers (CSCR0–CSCR5)	17-6
17.4	Functional Description	17-9
17.4.1	Data Transfer Operation	17-9
17.4.2	Data Byte Alignment and Physical Connections	17-9
17.4.3	Bus Cycle Execution	17-10
17.4.3.1	Data Transfer Cycle States	17-10
17.4.4	FlexBus Timing Examples	17-11
17.4.4.1	Basic Read Bus Cycle	17-11
17.4.4.2	Basic Write Bus Cycle	17-13
17.4.4.3	Bus Cycle Sizing	17-14
17.4.4.4	Timing Variations	17-16
17.4.4.4.1	Wait States	17-16
17.4.4.4.2	Address Setup and Hold	17-18
17.4.5	Burst Cycles	17-21
17.4.6	Misaligned Operands	17-25
17.4.7	Bus Errors	17-26

Contents

Paragraph Number	Title	Page Number
Chapter 18 SDRAM Controller (SDRAMC)		
18.1	Introduction	18-1
18.1.1	Block Diagram	18-2
18.1.2	Features	18-2
18.1.3	Terminology	18-3
18.2	External Signal Description	18-3
18.3	Interface Recommendations	18-4
18.3.1	Supported Memory Configurations	18-4
18.3.2	SDRAM SDR Connections	18-9
18.3.3	SDRAM DDR Component Connections	18-10
18.3.4	DDR SDRAM Layout Considerations	18-10
18.3.4.1	Termination Example	18-11
18.4	Memory Map/Register Definition	18-11
18.4.1	SDRAM Mode/Extended Mode Register (SDMR)	18-12
18.4.2	SDRAM Control Register (SDCR)	18-13
18.4.3	SDRAM Configuration Register 1 (SDCFG1)	18-14
18.4.4	SDRAM Configuration Register 2 (SDCFG2)	18-16
18.4.5	SDRAM Chip Select Configuration Registers (SDCS n)	18-17
18.5	Functional Overview	18-19
18.5.1	SDRAM Commands	18-19
18.5.1.1	Row and Bank Active Command (ACTV)	18-19
18.5.1.2	Read Command (READ)	18-20
18.5.1.3	Write Command (WRITE)	18-20
18.5.1.4	Burst Terminate Command (BST)	18-20
18.5.1.5	Precharge All Banks (PALL) and Selected Bank (PRE) Commands	18-20
18.5.1.6	Load Mode/Extended Mode Register Command (LMR, LEMR)	18-21
18.5.1.6.1	Mode Register Definition	18-21
18.5.1.6.2	Extended Mode Register Definition	18-22
18.5.1.7	Auto-Refresh Command (REF)	18-22
18.5.1.8	Self-Refresh (SREF) and Power Down (PDWN) Commands	18-23
18.5.2	Power-Up Initialization	18-23
18.5.3	Page Management	18-24
18.5.4	Transfer Size	18-25

Chapter 19 Fast Ethernet Controller (FEC)

19.1	Introduction	19-1
19.1.1	Overview	19-1

Contents

Paragraph Number	Title	Page Number
19.1.2	Block Diagram	19-1
19.1.3	Features	19-3
19.1.4	Modes of Operation	19-4
19.1.4.1	Full and Half Duplex Operation	19-4
19.1.5	Interface Options	19-4
19.1.5.1	10 Mbps and 100 Mbps MII Interface	19-4
19.1.5.2	10 Mpbs 7-Wire Interface Operation	19-4
19.1.6	Address Recognition Options	19-5
19.1.7	Internal Loopback	19-5
19.2	Memory Map/Register Definition	19-5
19.2.1	High-Level Module Memory Map	19-5
19.2.2	Register Memory Map	19-5
19.2.3	MIB Block Counters Memory Map	19-6
19.2.4	Register Description	19-8
19.2.4.1	Ethernet Interrupt Event Register (EIR)	19-8
19.2.4.2	Interrupt Mask Register (EIMR)	19-10
19.2.4.3	Receive Descriptor Active Register (RDAR)	19-11
19.2.4.4	Transmit Descriptor Active Register (TDAR)	19-11
19.2.4.5	Ethernet Control Register (ECR)	19-12
19.2.4.6	MII Management Frame Register (MMFR)	19-13
19.2.4.7	MII Speed Control Register (MSCR)	19-14
19.2.4.8	MIB Control Register (MIBC)	19-15
19.2.4.9	Receive Control Register (RCR)	19-15
19.2.4.10	Transmit Control Register (TCR)	19-16
19.2.4.11	Physical Address Low Register (PALR)	19-17
19.2.4.12	Physical Address High Register (PAUR)	19-18
19.2.4.13	Opcode/Pause Duration Register (OPD)	19-18
19.2.4.14	Descriptor Individual Upper Address Register (IAUR)	19-19
19.2.4.15	Descriptor Individual Lower Address Register (IALR)	19-19
19.2.4.16	Descriptor Group Upper Address Register (GAUR)	19-20
19.2.4.17	Descriptor Group Lower Address Register (GALR)	19-20
19.2.4.18	FIFO Transmit FIFO Watermark Register (TFWR)	19-21
19.2.4.19	FIFO Receive Bound Register (FRBR)	19-21
19.2.4.20	FIFO Receive Start Register (FRSR)	19-22
19.2.4.21	Receive Descriptor Ring Start Register (ERDSR)	19-22
19.2.4.22	Transmit Buffer Descriptor Ring Start Registers (ETSDR)	19-23
19.2.4.23	Receive Buffer Size Register (EMRBR)	19-23
19.2.5	Buffer Descriptors	19-24
19.2.5.1	Driver/DMA Operation with Buffer Descriptors	19-24
19.2.5.1.1	Driver/DMA Operation with Transmit BDs	19-25
19.2.5.1.2	Driver/DMA Operation with Receive BDs	19-25

Contents

Paragraph Number	Title	Page Number
19.2.5.2	Ethernet Receive Buffer Descriptor (RxBD)	19-26
19.2.5.3	Ethernet Transmit Buffer Descriptor (TxBD)	19-27
19.3	Functional Description	19-29
19.3.1	Initialization Sequence	19-29
19.3.1.1	Hardware Controlled Initialization	19-29
19.3.2	User Initialization (Prior to Setting ECR[ETHER_EN])	19-30
19.3.3	Microcontroller Initialization	19-30
19.3.4	User Initialization (After Setting ECR[ETHER_EN])	19-31
19.3.5	Network Interface Options	19-31
19.3.6	FEC Frame Transmission	19-32
19.3.7	FEC Frame Reception	19-33
19.3.8	Ethernet Address Recognition	19-34
19.3.9	Hash Algorithm	19-36
19.3.10	Full Duplex Flow Control	19-39
19.3.11	Inter-Packet Gap (IPG) Time	19-40
19.3.12	Collision Handling	19-40
19.3.13	Internal and External Loopback	19-40
19.3.14	Ethernet Error-Handling Procedure	19-41
19.3.14.1	Transmission Errors	19-41
19.3.14.1.1	Transmitter Underrun	19-41
19.3.14.1.2	Retransmission Attempts Limit Expired	19-41
19.3.14.1.3	Late Collision	19-41
19.3.14.1.4	Heartbeat	19-41
19.3.14.2	Reception Errors	19-42
19.3.14.2.1	Overrun Error	19-42
19.3.14.2.2	Non-Octet Error (Dribbling Bits)	19-42
19.3.14.2.3	CRC Error	19-42
19.3.14.2.4	Frame Length Violation	19-42
19.3.14.2.5	Truncation	19-42

Chapter 20 Watchdog Timer Module

20.1	Introduction	20-1
20.1.1	Low-Power Mode Operation	20-1
20.1.2	Block Diagram	20-2
20.2	Memory Map/Register Definition	20-2
20.2.1	Watchdog Control Register (WCR)	20-2
20.2.2	Watchdog Modulus Register (WMR)	20-3
20.2.3	Watchdog Count Register (WCNTR)	20-4
20.2.4	Watchdog Service Register (WSR)	20-4

Contents

Paragraph Number	Title	Page Number
Chapter 21 Programmable Interrupt Timer Modules (PIT0–PIT1)		
21.1	Introduction	21-1
21.1.1	Overview	21-1
21.1.2	Block Diagram	21-1
21.1.3	Low-Power Mode Operation	21-1
21.2	Memory Map/Register Definition	21-2
21.2.1	PIT Control and Status Register (PCSR n)	21-3
21.2.2	PIT Modulus Register (PMR n)	21-4
21.2.3	PIT Count Register (PCNTR n)	21-5
21.3	Functional Description	21-5
21.3.1	Set-and-Forget Timer Operation	21-5
21.3.2	Free-Running Timer Operation	21-6
21.3.3	Timeout Specifications	21-6
21.3.4	Interrupt Operation	21-6
Chapter 22 DMA Timers (DTIM0–DTIM3)		
22.1	Introduction	22-1
22.1.1	Overview	22-1
22.1.2	Features	22-2
22.2	Memory Map/Register Definition	22-2
22.2.1	DMA Timer Mode Registers (DTMR n)	22-3
22.2.2	DMA Timer Extended Mode Registers (DTXMR n)	22-4
22.2.3	DMA Timer Event Registers (DTER n)	22-5
22.2.4	DMA Timer Reference Registers (DTRR n)	22-6
22.2.5	DMA Timer Capture Registers (DTCR n)	22-7
22.2.6	DMA Timer Counters (DTCN n)	22-7
22.3	Functional Description	22-8
22.3.1	Prescaler	22-8
22.3.2	Capture Mode	22-8
22.3.3	Reference Compare	22-8
22.3.4	Output Mode	22-8
22.4	Initialization/Application Information	22-8
22.4.1	Code Example	22-9
22.4.2	Calculating Time-Out Values	22-10

Contents

Paragraph Number	Title	Page Number
Chapter 23 Queued Serial Peripheral Interface (QSPI)		
23.1	Introduction	23-1
23.1.1	Block Diagram	23-1
23.1.2	Overview	23-2
23.1.3	Features	23-2
23.1.4	Modes of Operation	23-2
23.2	External Signal Description	23-2
23.3	Memory Map/Register Definition	23-3
23.3.1	QSPI Mode Register (QMR)	23-3
23.3.2	QSPI Delay Register (QDLYR)	23-5
23.3.3	QSPI Wrap Register (QWR)	23-6
23.3.4	QSPI Interrupt Register (QIR)	23-6
23.3.5	QSPI Address Register (QAR)	23-8
23.3.6	QSPI Data Register (QDR)	23-8
23.3.7	Command RAM Registers (QCR0–QCR15)	23-8
23.4	Functional Description	23-10
23.4.1	QSPI RAM	23-11
23.4.1.1	Receive RAM	23-12
23.4.1.2	Transmit RAM	23-12
23.4.1.3	Command RAM	23-13
23.4.2	Baud Rate Selection	23-13
23.4.3	Transfer Delays	23-14
23.4.4	Transfer Length	23-15
23.4.5	Data Transfer	23-15
23.5	Initialization/Application Information	23-16

Chapter 24 UART Modules

24.1	Introduction	24-1
24.1.1	Overview	24-1
24.1.2	Features	24-2
24.2	External Signal Description	24-3
24.3	Memory Map/Register Definition	24-4
24.3.1	UART Mode Registers 1 (UMR1 n)	24-5
24.3.2	UART Mode Register 2 (UMR2 n)	24-6
24.3.3	UART Status Registers (USR n)	24-7
24.3.4	UART Clock Select Registers (UCSR n)	24-9
24.3.5	UART Command Registers (UCR n)	24-9

Contents

Paragraph Number	Title	Page Number
24.3.6	UART Receive Buffers (URBn)	24-11
24.3.7	UART Transmit Buffers (UTBn)	24-12
24.3.8	UART Input Port Change Registers (UIPCRn)	24-12
24.3.9	UART Auxiliary Control Register (UACRn)	24-13
24.3.10	UART Interrupt Status/Mask Registers (UISRn/UIMRn)	24-13
24.3.11	UART Baud Rate Generator Registers (UBG1n/UBG2n)	24-15
24.3.12	UART Input Port Register (UIPn)	24-16
24.3.13	UART Output Port Command Registers (UOP1n/UOP0n)	24-16
24.4	Functional Description	24-17
24.4.1	Transmitter/Receiver Clock Source	24-17
24.4.1.1	Programmable Divider	24-17
24.4.1.2	Calculating Baud Rates	24-18
24.4.1.2.1	Internal Bus Clock Baud Rates	24-18
24.4.1.2.2	External Clock	24-18
24.4.2	Transmitter and Receiver Operating Modes	24-18
24.4.2.1	Transmitter	24-19
24.4.2.2	Receiver	24-20
24.4.2.3	FIFO	24-22
24.4.3	Looping Modes	24-23
24.4.3.1	Automatic Echo Mode	24-23
24.4.3.2	Local Loop-back Mode	24-23
24.4.3.3	Remote Loop-back Mode	24-24
24.4.4	Multidrop Mode	24-24
24.4.5	Bus Operation	24-26
24.4.5.1	Read Cycles	24-26
24.4.5.2	Write Cycles	24-26
24.4.6	Programming	24-26
24.4.6.1	Interrupt and DMA Request Initialization	24-26
24.4.6.1.1	Setting up the UART to Generate Core Interrupts	24-26
24.4.6.1.2	Setting up the UART to Request DMA Service	24-27
24.4.6.2	UART Module Initialization Sequence	24-28

Chapter 25 **I²C Interface**

25.1	Introduction	25-1
25.2	Overview	25-1
25.3	Features	25-1
25.4	I ² C System Configuration	25-3
25.4.1	START Signal	25-3
25.4.2	Slave Address Transmission	25-3

Contents

Paragraph Number	Title	Page Number
25.4.3	Data Transfer	25-4
25.4.4	Acknowledge	25-4
25.4.5	STOP Signal	25-5
25.4.6	Repeated START	25-5
25.4.7	Clock Synchronization and Arbitration	25-6
25.4.8	Handshaking and Clock Stretching	25-7
25.5	Memory Map/Register Definition	25-8
25.5.1	I ² C Address Register (I2ADR)	25-8
25.5.2	I ² C Frequency Divider Register (I2FDR)	25-8
25.5.3	I ² C Control Register (I2CR)	25-9
25.5.4	I ² C Status Register (I2SR)	25-10
25.5.5	I ² C Data I/O Register (I2DR)	25-11
25.6	I ² C Programming Examples	25-12
25.6.1	Initialization Sequence	25-12
25.6.2	Generation of START	25-13
25.6.3	Post-Transfer Software Response	25-13
25.6.4	Generation of STOP	25-14
25.6.5	Generation of Repeated START	25-15
25.6.6	Slave Mode	25-15
25.6.7	Arbitration Lost	25-15

Chapter 26 Debug Module

26.1	Introduction	26-1
26.1.1	Overview	26-1
26.1.1.1	The New Debug Module Hardware (Rev. B+)	26-2
26.1.1.2	Enhancements over Revision A	26-2
26.2	External Signal Description	26-2
26.3	Real-Time Trace Support	26-3
26.3.1	Begin Execution of Taken Branch (PST = 0x5)	26-5
26.4	Memory Map/Register Definition	26-6
26.4.1	Shared Debug Resources	26-7
26.4.2	Configuration/Status Register (CSR)	26-7
26.4.3	BDM Address Attribute (BAAR)	26-10
26.4.4	Address Attribute Trigger Register (AATR)	26-10
26.4.5	Trigger Definition Register (TDR)	26-12
26.4.6	Program Counter Breakpoint/Mask Registers (PBR0–3, PBMR)	26-14
26.4.7	Address Breakpoint Registers (ABLR, ABHR)	26-15
26.4.8	Data Breakpoint/Mask Registers (DBR, DBMR)	26-16
26.5	Background Debug Mode (BDM)	26-17

Contents

Paragraph Number	Title	Page Number
26.5.1	CPU Halt	26-17
26.5.2	BDM Serial Interface	26-18
26.5.2.1	Receive Packet Format	26-19
26.5.2.2	Transmit Packet Format	26-19
26.5.3	BDM Command Set	26-20
26.5.3.1	ColdFire BDM Command Format	26-21
26.5.3.1.1	Extension Words as Required	26-22
26.5.3.2	Command Sequence Diagrams	26-22
26.5.3.3	Command Set Descriptions	26-24
26.5.3.3.1	Read A/D Register (RAREG/RDREG)	26-24
26.5.3.3.2	Write A/D Register (WAREG/WDREG)	26-24
26.5.3.3.3	Read Memory Location (READ)	26-25
26.5.3.3.4	Write Memory Location (WRITE)	26-26
26.5.3.3.5	Dump Memory Block (DUMP)	26-28
26.5.3.3.6	Fill Memory Block (FILL)	26-30
26.5.3.3.7	Resume Execution (GO)	26-31
26.5.3.3.8	No Operation (NOP)	26-32
26.5.3.3.9	Synchronize PC to the PST/DDATA Lines (SYNC_PC)	26-32
26.5.3.3.10	Read Control Register (RCREG)	26-33
26.5.3.3.11	Write Control Register (WCREG)	26-35
26.5.3.3.12	Read Debug Module Register (RDMREG)	26-36
26.5.3.3.13	Write Debug Module Register (WDMREG)	26-37
26.6	Real-Time Debug Support	26-38
26.6.1	Theory of Operation	26-38
26.6.1.1	Emulator Mode	26-39
26.6.2	Concurrent BDM and Processor Operation	26-40
26.7	Processor Status, DDATA Definition	26-40
26.7.1	User Instruction Set	26-41
26.7.2	Supervisor Instruction Set	26-44
26.8	Freescale-Recommended BDM Pinout	26-45

Chapter 27

IEEE 1149.1 Test Access Port (JTAG)

27.1	Introduction	27-1
27.1.1	Block Diagram	27-1
27.1.2	Features	27-2
27.1.3	Modes of Operation	27-2
27.2	External Signal Description	27-2
27.2.1	JTAG Enable (JTAG_EN)	27-2
27.2.2	Test Clock Input (TCLK)	27-3

Contents

Paragraph Number	Title	Page Number
27.2.3	Test Mode Select/Breakpoint (TMS/ <u>BKPT</u>)	27-3
27.2.4	Test Data Input/Development Serial Input (TDI/DSI)	27-3
27.2.5	Test Reset/Development Serial Clock (TRST/DSCLK)	27-4
27.2.6	Test Data Output/Development Serial Output (TDO/DSO)	27-4
27.3	Memory Map/Register Definition	27-4
27.3.1	Instruction Shift Register (IR)	27-4
27.3.2	IDCODE Register	27-4
27.3.3	Bypass Register	27-5
27.3.4	TEST_CTRL Register	27-5
27.3.5	Boundary Scan Register	27-6
27.4	Functional Description	27-6
27.4.1	JTAG Module	27-6
27.4.2	TAP Controller	27-6
27.4.3	JTAG Instructions	27-7
27.4.3.1	IDCODE Instruction	27-8
27.4.3.2	SAMPLE/PRELOAD Instruction	27-8
27.4.3.3	SAMPLE Instruction	27-8
27.4.3.4	EXTEST Instruction	27-9
27.4.3.5	ENABLE_TEST_CTRL Instruction	27-9
27.4.3.6	HIGHZ Instruction	27-9
27.4.3.7	CLAMP Instruction	27-9
27.4.3.8	BYPASS Instruction	27-9
27.5	Initialization/Application Information	27-10
27.5.1	Restrictions	27-10
27.5.2	Nonscan Chain Operation	27-10

Contents

Paragraph Number	Title	Page Number
---------------------	-------	----------------

About This Book

The primary objective of this reference manual is to define the functionality of the MCF5208 processor for use by software and hardware developers. In addition, this manual supports the MCF5207. This book is written from the perspective of the MCF5208, and unless otherwise noted, the information applies also to the MCF5207. The MCF5207 has the same functionality as the MCF5208 and any differences in data regarding bus timing, signal behavior, and AC, DC, and thermal characteristics are in the hardware specifications. Please refer to [Table 1-1](#) to see a summary of the differences.

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, it is the reader's responsibility to be sure he is using 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>.

Audience

This manual is intended for system software and hardware developers and applications programmers who want to develop products with the MCF5208. 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.

Organization

Following is a summary and brief description of the major sections of this manual:

- [Chapter 1, “Overview,”](#) includes general descriptions of the modules and features incorporated in the device, focusing in particular on new features.
- [Chapter 2, “Signal Descriptions,”](#) describes the device signals. It includes a 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.
- [Chapter 3, “ColdFire Core,”](#) provides an overview of the microprocessor core. The chapter describes the organization of the Version 2 (V2) ColdFire processor core and an overview of the programming model as they are implemented on the device.
- [Chapter 4, “Enhanced Multiply-Accumulate Unit \(EMAC\),”](#) describes the multiply/accumulate unit, which executes integer multiply, multiply-accumulate, and

miscellaneous register instructions. The EMAC is integrated into the operand execution pipeline (OEP).

- [Chapter 5, “Cache,”](#) describes the cache implementation, including organization, configuration, and coherency. It describes cache operations and how the cache interacts with other memory structures.
- [Chapter 6, “Static RAM \(SRAM\),”](#) describes the on-chip static RAM (SRAM) implementation. It covers general operations, configuration, and initialization. It also provides information and examples of how to minimize power consumption when using the SRAM.
- [Chapter 7, “Clock Module,”](#) describes the device’s different clocking methods. It also describes clock module operation in low power modes.
- [Chapter 8, “Power Management,”](#) describes the low power operation of the device and peripheral behavior in low power modes.
- [Chapter 9, “Chip Configuration Module \(CCM\),”](#) details the various operating configurations of the device. This chapter provides a description of signals used by the CCM and a programming model.
- [Chapter 10, “Reset Controller Module,”](#) describes the operation of the reset controller module, detailing the different types of reset that can occur.
- [Chapter 11, “System Control Module \(SCM\),”](#) describes the functionality of the SCM, which provides the programming model for peripheral access control, the software core watchdog timer (CWT), and the generic access error information.
- [Chapter 12, “Cross-Bar Switch Module \(XBS\),”](#) details the interaction between bus masters and bus slaves within the device, including arbitration schemes.
- [Chapter 13, “General Purpose I/O Module,”](#) describes the operation and programming model of the general purpose I/O (GPIO) ports on the device.
- [Chapter 14, “Interrupt Controller Module,”](#) describes operation of the interrupt controller portion of the SCM. Includes descriptions of the registers in the interrupt controller memory map and the interrupt priority scheme.
- [Chapter 15, “Edge Port Module \(EPORT\),”](#) describes EPORT module functionality, including operation in low power mode.
- [Chapter 16, “Enhanced Direct Memory Access \(eDMA\),”](#) describes the direct memory access (DMA) controller module. It provides an overview of the module and describes in detail its signals and registers. The latter sections of this chapter describe operations, features, and supported data transfer modes in detail.
- [Chapter 17, “FlexBus,”](#) describes data-transfer operations, chip-select operation, error conditions, bus arbitration, and reset operations.
- [Chapter 18, “SDRAM Controller \(SDRAMC\),”](#) describes the configuration and operation of the SDRAM controller. It begins with a general description and includes a description of

signals involved in DRAM operations. The remainder of the chapter describes the programming model and signal timing, as well as the command set required for synchronous operations.

- Chapter 19, “Fast Ethernet Controller (FEC),” provides a feature-set overview, a functional block diagram, and transceiver connection information for both MII (media independent interface) and 7-wire serial interfaces. It also provides describes operation and the programming model.
- Chapter 20, “Watchdog Timer Module,” describes software watchdog timer functionality, including operation in low power mode.
- Chapter 21, “Programmable Interrupt Timer Modules (PIT0–PIT1),” describes the functionality of the PIT timers, including operation in low power mode.
- Chapter 22, “DMA Timers (DTIM0–DTIM3),” describes the configuration and operation of the DMA timer modules. These 32-bit timers provide input capture and reference compare capabilities with optional signaling of events using interrupts or triggers. This chapter also provides programming examples.
- Chapter 23, “Queued Serial Peripheral Interface (QSPI),” provides a feature-set overview and a description of operation, including details of the QSPI’s internal storage organization. The chapter concludes with the programming model and a timing diagram.
- Chapter 24, “UART Modules,” describes the use of the universal asynchronous receiver/transmitters (UARTs) implemented on the device and includes programming examples.
- Chapter 25, “I²C Interface,” describes the I²C module, including I²C protocol, clock synchronization, and I²C programming model registers.
- Chapter 26, “Debug Module,” describes the hardware debug support in the device.
- Chapter 27, “IEEE 1149.1 Test Access Port (JTAG),” describes configuration and operation of the Joint Test Action Group (JTAG) implementation. It describes those items required by the IEEE 1149.1 standard and provides additional information specific to the device. For internal details and sample applications, see the IEEE 1149.1 document.

Suggested Reading

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

Hardware Specification

The MCF5208EC document contains the mechanical and electrical specifications of the MCF5208. It can be found at <http://www.freescale.com/coldfire>.

General Information

The following documentation provides useful information about the ColdFire architecture and computer architecture in general:

- *ColdFire Programmers Reference Manual, R1.0* (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.

- Reference manuals (formerly called user's manuals)—These books provide details about individual ColdFire implementations and are intended to be used in conjunction with *The ColdFire Programmers Reference Manual*.
- Addenda/errata to reference manuals—Because some processors have follow-on parts, an addendum is provided that describes the additional features and functionality changes. Also, if mistakes are found within a reference manual, an errata document will be issued before the next published release of the reference manual. These addenda/errata are intended for use with the corresponding reference manuals.
- Hardware specifications—Hardware specifications provide specific data regarding 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 implementation'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.

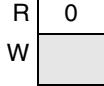
MNEMONICS	In text, instruction mnemonics are shown in uppercase.
mnemonics	In code and tables, instruction mnemonics are shown in lowercase.
<i>italics</i>	Italics indicate variable command parameters.
	Book titles in text are set in italics.
0x0	Prefix to denote hexadecimal number
0b0	Prefix to denote 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 ¹
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
<u>OVERBAR</u>	An overbar indicates that a signal is active-low.

Register Figure Conventions

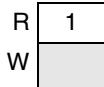
This document uses the following conventions for the register reset values:

- 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:

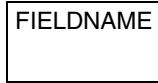
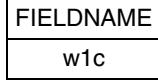
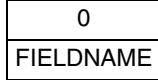


Indicates a reserved bit field in a memory-mapped register. These bits are always read as zeros.



Indicates a reserved bit field in a memory-mapped register. These bits are always read as ones.

1. 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.

R		Indicates a read/write bit.
W		Indicates a read-only bit field in a memory-mapped register.
R		Indicates a write-only bit field in a memory-mapped register.
W		Write 1 to clear: indicates that writing a 1 to this bit field clears it.
R		Indicates a self-clearing bit.
W		

Acronyms and Abbreviations

Table 1 lists acronyms and abbreviations used in this document.

Table 1. Acronyms and Abbreviated Terms

Term	Meaning
ADC	Analog-to-digital conversion
ALU	Arithmetic logic unit
BDM	Background debug mode
BIST	Built-in self test
BSDL	Boundary-scan description language
CODEC	Code/decode
DAC	Digital-to-analog conversion
DMA	Direct memory access
DSP	Digital signal processing
EA	Effective address
FIFO	First-in, first-out
GPIO	General-purpose I/O
I ² C	Inter-integrated circuit
IEEE	Institute for Electrical and Electronics Engineers
IFP	Instruction fetch pipeline
IPL	Interrupt priority level
JEDEC	Joint Electron Device Engineering Council
JTAG	Joint Test Action Group

Table 1. Acronyms and Abbreviated Terms (continued)

Term	Meaning
LIFO	Last-in, first-out
LRU	Least recently used
LSB	Least-significant byte
lsb	Least-significant bit
MAC	Multiply accumulate unit, also Media access controller
MBAR	Memory base address register
MSB	Most-significant byte
msb	Most-significant bit
Mux	Multiplex
NOP	No operation
OEP	Operand execution pipeline
PC	Program counter
PCLK	Processor clock
PLIC	Physical layer interface controller
PLL	Phase-locked loop
POR	Power-on reset
PQFP	Plastic quad flat pack
PWM	Pulse width modulation
QSPI	Queued serial peripheral interface
RISC	Reduced instruction set computing
Rx	Receive
SIM	System integration module
SOF	Start of frame
TAP	Test access port
TTL	Transistor transistor logic
Tx	Transmit
UART	Universal asynchronous/synchronous receiver transmitter
USB	Universal serial bus

Terminology Conventions

Table 2 shows terminology conventions used throughout this document.

Table 2. Notational Conventions

Instruction	Operand Syntax
Opcode Wildcard	
cc	Logical condition (example: NE for not equal)
Register Specifications	
An	Any address register n (example: A3 is address register 3)
Ay,Ax	Source and destination address registers, respectively
Dn	Any data register n (example: D5 is data register 5)
Dy,Dx	Source and destination data registers, respectively
Rc	Any control register (example VBR is the vector base register)
Rm	MAC registers (ACC, MAC, MASK)
Rn	Any address or data register
Rw	Destination register w (used for MAC instructions only)
Ry,Rx	Any source and destination registers, respectively
Xi	Index register i (can be an address or data register: Ai, Di)
Miscellaneous Operands	
#<data>	Immediate data following the 16-bit operation word of the instruction
<ea>	Effective address
<ea>y,<ea>x	Source and destination effective addresses, respectively
<label>	Assembly language program label
<list>	List of registers for MOVEM instruction (example: D3–D0)
<shift>	Shift operation: shift left (<<), shift right (>>)
<size>	Operand data size: byte (B), word (W), longword (L)
bc	Both instruction and data caches
dc	Data cache
ic	Instruction cache
# <vector>	Identifies the 4-bit vector number for trap instructions
<>	identifies an indirect data address referencing memory
<xxx>	identifies an absolute address referencing memory
dn	Signal displacement value, n bits wide (example: d16 is a 16-bit displacement)
SF	Scale factor (x1, x2, x4 for indexed addressing mode, <<1n>> for MAC operations)

Table 2. Notational Conventions (continued)

Instruction	Operand Syntax
Operations	
+	Arithmetic addition or postincrement indicator
-	Arithmetic subtraction or predecrement indicator
x	Arithmetic multiplication
/	Arithmetic division
~	Invert; operand is logically complemented
&	Logical AND
	Logical OR
^	Logical exclusive OR
<<	Shift left (example: D0 << 3 is shift D0 left 3 bits)
>>	Shift right (example: D0 >> 3 is shift D0 right 3 bits)
→	Source operand is moved to destination operand
↔	Two operands are exchanged
sign-extended	All bits of the upper portion are made equal to the high-order bit of the lower portion
If <condition> then <operations> else <operations>	Test the condition. If true, the operations after 'then' are performed. If the condition is false and the optional 'else' clause is present, the operations after 'else' are performed. If the condition is false and else is omitted, the instruction performs no operation. Refer to the Bcc instruction description as an example.
Subfields and Qualifiers	
{}	Optional operation
()	Identifies an indirect address
d _n	Displacement value, n-bits wide (example: d ₁₆ is a 16-bit displacement)
Address	Calculated effective address (pointer)
Bit	Bit selection (example: Bit 3 of D0)
lsb	Least significant bit (example: lsb of D0)
LSB	Least significant byte
LSW	Least significant word
msb	Most significant bit
MSB	Most significant byte
MSW	Most significant word

Revision History

Table 3 provides a revision history for this document.

Table 3. MCF5208RM Revision History

Rev Number	Date of Release	Substantive Changes
0	5/23/2005	Initial release.
0.1	8/19/2005	Many small functional edits throughout. Most substantive changes were to the SCM and clock module chapters.

Chapter 1

Overview

The MCF5207 and MCF5208 devices are highly-integrated 32-bit microprocessors based on the Version 2 ColdFire microarchitecture. Both devices contain a 16-Kbyte internal SRAM, 8-Kbyte configurable cache, a 2-bank SDR/DDR SDRAM controller, a 16-channel DMA controller, up to three UARTs, a queued SPI, as well as other peripherals that enable the MCF5207 and MCF5208 for use in general purpose industrial control applications. The MCF5208 device also features a 10/100 Mbps Fast Ethernet controller.

This chapter provides an overview of the MCF5207 and MCF5208 microprocessors. It was written from the perspective of the MCF5208 device. See the following section for a summary of differences between the two devices.

1.1 MCF5207/8 Device Configurations

The following table compares the two devices:

Table 1-1. MCF5207 & MCF5208 Configurations

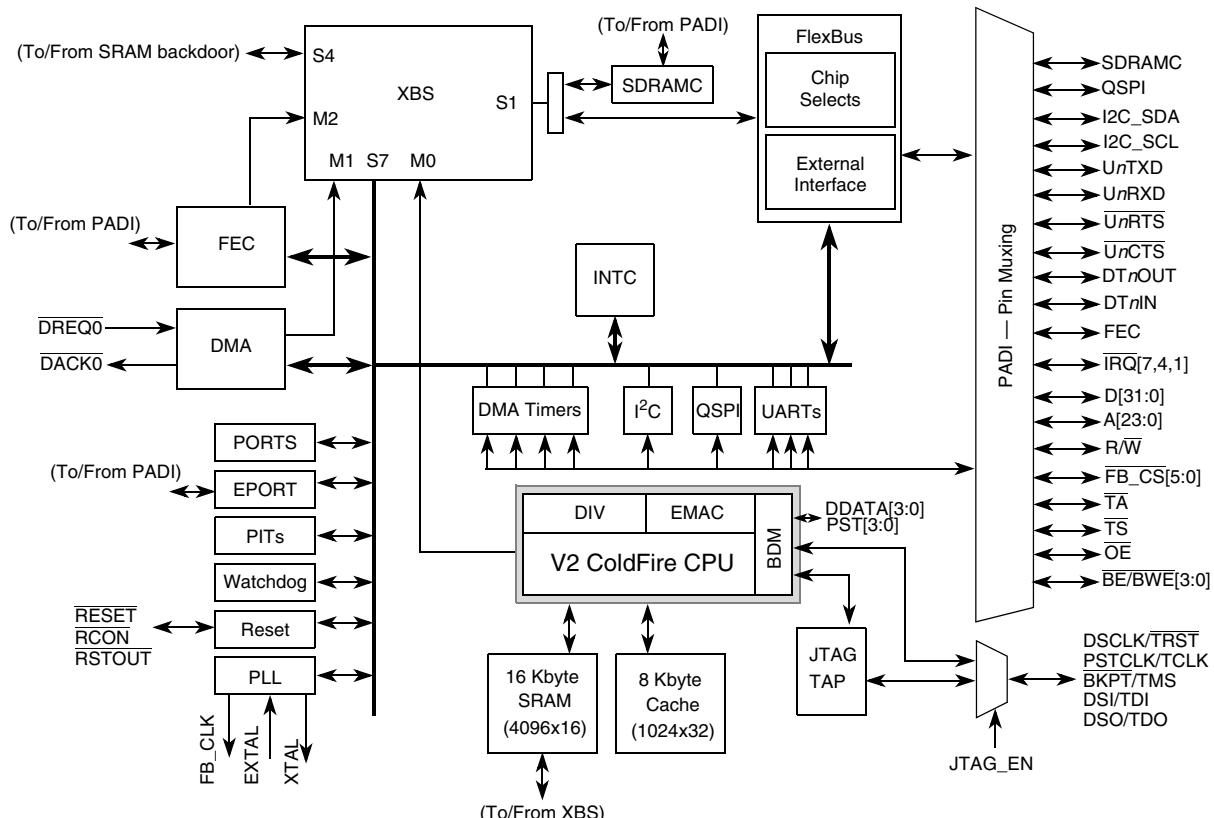
Module	MCF5207	MCF5208
Version 2 ColdFire Core with EMAC (Enhanced Multiply-Accumulate Unit)	x	x
Core (System) Clock	up to 166.67 MHz	
Peripheral and External Bus Clock (Core clock ÷ 2)	up to 83.33 MHz	
Performance (Dhrystone/2.1 MIPS)	up to 159	
Unified Cache	8 Kbytes	
Static RAM (SRAM)	16 Kbytes	
SDR/DDR SDRAM Controller	x	x
Fast Ethernet Controller (FEC)	—	x
UARTs	3	3
I ² C	x	x
QSPI	x	x
32-bit DMA Timers	4	4
Watchdog Timer (WDT)	x	x
Periodic Interrupt Timers (PIT)	2	2
Edge Port Module (EPORT)	x	x

Table 1-1. MCF5207 & MCF5208 Configurations (continued)

Module	MCF5207	MCF5208
Interrupt Controllers (INTC)	1	1
16-channel Direct Memory Access (DMA)	x	x
FlexBus External Interface	x	x
General Purpose I/O Module (GPIO)	x	x
JTAG - IEEE® 1149.1 Test Access Port	x	x
Package	144 LQFP 144 MAPBGA	160 QFP 196 MAPBGA

1.2 Block Diagram

The MCF5208 superset device is available in a 196 mold array process ball grid array (MAPBGA) or 160-pin quad flat pack (QFP) package. [Figure 1](#) shows a top-level block diagram of the MCF5208.

**Figure 1. MCF5208 Block Diagram**

1.3 Features

The following is a brief summary of the functional blocks in the MCF5208 superset device.

- Version 2 ColdFire variable-length RISC processor core
 - Static operation
 - 32-bit address and data path on-chip
 - Processor core runs at twice the bus frequency
 - Sixteen general-purpose 32-bit data and address registers
 - Implements the ColdFire instruction set architecture, ISA_A+, with extensions to support the user stack pointer register, and 4 new instructions for improved bit processing
 - Enhanced multiply-accumulate (EMAC) unit with four 48-bit accumulators to support 32-bit signal processing algorithms
 - Hardware divide execution unit supporting various 32-bit operations
 - Illegal instruction decode that allows for 68K emulation support
- System debug support
 - Background debug mode (BDM) revision B+ for in-circuit debugging
 - Real time debug support, with nine user-visible hardware breakpoint registers (PC and address with optional data) that can be configured into a 1- or 2-level trigger
- JTAG support for system level board testing
- On-chip memories
 - 8-Kbyte cache, configurable as instruction-only, data-only, or split I/D-cache
 - 16-Kbyte dual-ported SRAM on CPU internal bus, accessible by core and non-core bus masters (e.g., DMA and FEC)
- Power management
 - Fully static operation with processor wait, doze, and stop modes
 - Very rapid response to interrupts from sleep mode
 - Global clock disable register to disable clocks to each peripheral
 - Ability to bypass PLL circuitry for low power and low speed mode
- SDR/DDR SDRAM controller
 - Supports a glueless interface to SDR and DDR SDRAM devices
 - 16-bit (DDR) or 32-bit (SDR) fixed memory port width
 - 16 bytes critical word first burst transfer
 - Up to 14 lines of row address, up to 12 (in 32-bit mode) or 13 (in 16-bit mode) column address lines, 2 bits of bank address, and a maximum of two pinned-out chip selects. The maximum row bits plus column bits equals 24 in 32-bit bus mode or 25 in 16-bit mode.
 - Supports up to 256 MBytes of memory per chip select, 512 MBytes total
 - Supports page mode to maximize the data rate
 - Supports sleep and self-refresh modes

- Fast Ethernet controller (FEC)
 - 10/100 BaseT/TX capability, half duplex or full duplex
 - On-chip transmit and receive FIFOs
 - Built-in dedicated DMA controller
 - Memory-based flexible descriptor rings
 - Media independent interface (MII) to external transceiver (PHY)
- Three universal asynchronous receiver transmitters (UARTs)
 - 16-bit divider for clock generation
 - Interrupt control logic
 - DMA support with separate transmit and receive requests
 - 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
 - Flow control support includes request-to-send (\overline{UnRTS}) and clear-to-send (\overline{UnCTS}) lines
- I²C module
 - Interchip bus interface for EEPROMs, A/D converters, and keypads
 - Fully compatible with industry-standard I²C bus
 - Master or slave modes support multiple masters
 - Automatic interrupt generation with programmable level
- Queued serial peripheral interface (QSPI)
 - Full-duplex, three-wire synchronous transfers
 - Up to three chip selects available
 - Master mode operation only with programmable master bit rates
 - Up to 16 pre-programmed transfers
- Four 32-bit DMA timers
 - 12-ns resolution at 83.33 MHz
 - Programmable prescaler and sources for clock input, including an external clock option
 - Input-capture capability with programmable trigger edge on input pin
 - Output-compare with programmable mode for the output pin
 - Free run and restart modes
 - Maskable interrupts & DMA trigger capability on input capture or output-compare
- On-chip software watchdog timer
 - 16-bit counter
 - Low power mode support
- Two periodic interrupt timers (PITs)
 - 16-bit counter
 - Selectable as free running or count down

- Phase locked loop (PLL)
 - 16 MHz reference frequency
 - Programmable dithering
- Interrupt controller
 - Support for up to 63 interrupt sources
 - Unique vector number for each interrupt source
 - Ability to mask any individual interrupt source or all interrupt sources (global mask-all)
 - Support for hardware and software interrupt acknowledge (IACK) cycles
 - Combinatorial path to provide wake-up from low power modes
- DMA controller
 - 16 fully programmable channels with 32-byte transfer control
 - Data movement via dual-address transfers for 8-, 16-, and 32-bit data values
 - Programmable source and destination addresses, transfer size, and 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 pins for a single channels
- FlexBus (external interface)
 - Glueless connections to 8-, 16-, or 32-bit external memory devices (SRAM, Flash, ROM, etc.)
 - Support for independent primary and secondary wait states per chip select
 - Programmable address setup & hold time with respect to chip select negation, per transfer direction
 - Glueless interface to SRAM devices with or without byte strobe inputs
 - Programmable wait state generator
 - 32-bit bidirectional data bus and 24-bit address bus
 - 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
- Chip configuration module (CCM)
 - System configuration during reset
 - Unique part identification number and part revision number
- Reset controller
 - Separate reset in and reset out signals
 - Five reset sources: power-on reset (POR), external, software, watchdog, PLL loss of lock
 - Status flag indication of source of last reset

- General purpose I/O interface
 - Up to 30 bits of GPIO for the MCF5207
 - Up to 50 bits of GPIO for the MCF5208 (196 MAPBGA)
 - Up to 46 bits of GPIO for the MCF5208 (160 QFP)
 - Bit manipulation supported via set/clear functions
 - Unused peripheral pins may be used as extra GPIO
 - Programmable drive strength or slew rate control for related group of pins

1.3.1 V2 Core Overview

The processor core is comprised of two separate pipelines that are decoupled by an instruction buffer. The two-stage instruction fetch pipeline (IFP) is responsible for instruction-address generation and instruction fetch. The instruction buffer is a first-in-first-out (FIFO) buffer that holds prefetched instructions awaiting execution in the operand execution pipeline (OEP). The OEP includes two pipeline stages. The first stage decodes instructions and selects operands (DSOC); the second stage (AGEX) performs instruction execution and calculates operand effective addresses, if needed.

The V2 core implements the ColdFire instruction set architecture revision A+ with added support for a separate user stack pointer register and four new instructions to assist in bit processing. Additionally, the MCF5208 core includes the enhanced multiply-accumulate unit (EMAC) for improved signal processing capabilities. The EMAC implements a 4-stage execution pipeline, optimized for 32 x 32 bit operations, with support for four 48-bit accumulators. Supported operands include 16- and 32-bit signed and unsigned integers, as well as signed fractional operands, plus a complete set of instructions to process these data types. The EMAC provides superb support for execution of DSP operations within the context of a single processor at a minimal hardware cost.

The core also includes a hardware divide unit that performs a number of integer-divide operations. The supported divide functions include: 32-bit dividend and 16-bit divisor producing a 16-bit quotient and a 16-bit remainder, 32-bit dividend and 32-bit divisor producing a 32-bit quotient, and 32-bit dividend and 32-bit divisor producing a 32-bit remainder.

1.3.2 Debug Module

The ColdFire processor core debug interface is provided to support system debugging in conjunction with low-cost debug and emulator development tools. Through a standard debug interface, users can access debug information. This allows the processor and system to be debugged without the need for costly in-circuit emulators.

The on-chip breakpoint resources include a total of nine programmable registers—a pair of upper and lower address registers, a pair of data registers (a 32-bit data register and a 32-bit data mask register), and four 32-bit PC registers plus a 32-bit PC mask register. These registers can be accessed through the dedicated debug serial communication channel or from the processor’s supervisor mode programming model. The breakpoint registers can be configured to generate triggers by combining the address, data, and PC conditions in a variety of single or dual-level definitions. The trigger event can be programmed to generate a processor halt or initiate a debug interrupt exception.

To support program trace, the debug module provides processor status (PST[3:0]) and debug data (DDATA[3:0]) ports. These busses and the PSTCLK output provide execution status, captured operand data, and branch target addresses defining processor activity at one-half the CPU's clock rate.

The MCF5207L and MCF5208L contain an ALLPST signal in place of the PST and DDATA signals. It is the result of an AND operation of the four PST lines and signals when the processor has halted.

1.3.3 JTAG

The device supports circuit board test strategies based on the Test Technology Committee of IEEE and the Joint Test Action Group (JTAG). The test logic includes a test access port (TAP) consisting of a 16-state controller, an instruction register, and three test registers (a bypass register, a boundary-scan register, and an ID register). The boundary scan register links the device's pins into one shift register. Test logic, implemented using static logic design, is independent of the device system logic.

The implementation can do the following:

- Perform boundary-scan operations to test circuit board electrical continuity
- Sample device system pins during operation and transparently shift out the result in the boundary scan register
- Bypass the device for a given circuit board test by effectively reducing the boundary-scan register to a single bit
- Disable the output drive to pins during circuit-board testing
- Drive output pins to stable levels

1.3.4 On-chip Memories

1.3.4.1 Cache

The MCF5208 architecture includes a 8-Kbyte configurable cache. The 8-Kbyte cache can be configured into one of three possible organizations: an 8-Kbyte instruction cache, an 8-Kbyte data cache or a split 4-Kbyte instruction/4-Kbyte data cache. The configuration is software-programmable by control bits within the privileged Cache Configuration Register (CACR). In all configurations, the cache is a direct-mapped single-cycle memory, organized as 512 lines, each containing 16 bytes of data. The memories consist of a 512-entry tag array (containing addresses and control bits) and a 8-Kbyte data array, organized as 2048 x 32 bits.

If the desired address is mapped into the cache memory, the output of the data array is driven onto the ColdFire core's local data bus, completing the access in a single cycle. If the data is not mapped into the tag memory, a cache miss occurs and the processor core initiates a 16-byte line-sized fetch. The cache module includes a 16-byte line fill buffer used as temporary storage during miss processing. For all data cache configurations, the memory operates in write-through mode and all operand writes generate an external bus cycle.

1.3.4.2 SRAM

The SRAM module provides a general-purpose 16-Kbyte memory block that the ColdFire core can access in a single cycle. 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 quickly service core-initiated accesses or memory-referencing commands from the debug module.

The dual-port SRAM module is also accessible by the DMA and FEC non-core bus masters through the crossbar switch. The dual-ported nature of the SRAM makes it ideal for implementing applications with double-buffer schemes, where the processor and a bus-mastering device operate in alternate regions of the SRAM to maximize system performance. As an example, system performance can be increased significantly if Ethernet packets are moved from the FEC into the SRAM (rather than external memory) prior to any processing.

1.3.5 SDR/DDR SDRAM Controller

The SDRAM controller provides a glueless interface to both SDR and DDR SDRAM memory devices. The module uses a 32-bit (for SDR) or a 16-bit (for DDR) memory port and can address up to 512 MB of data (256 MB per chip select). The controller supports DDR and SDR SDRAM, but both cannot be used at the same time.

1.3.6 Fast Ethernet Controller (FEC)

The device's integrated fast Ethernet controller (FEC) performs the full set of IEEE® 802.3/Ethernet CSMA/CD media access control and channel interface functions. The FEC supports connection and functionality for the 10/100 Mbps 802.3 media independent interface (MII). It requires an external transceiver (PHY) to complete the interface to the media.

1.3.7 UARTs

The device contains three independent, full-duplex UARTs. The three UARTs can be clocked by the system bus clock, eliminating the need for an externally supplied clock. They can use DMA requests on transmit-ready and receive-ready as well as interrupt requests for servicing.

1.3.8 I²C Bus

The I²C bus 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 requiring occasional communications over a short distance between many devices.

1.3.9 QSPI

The queued serial peripheral interface module provides a high-speed synchronous serial peripheral interface with queued transfer capability. It allows up to 16 transfers to be queued at once, eliminating CPU intervention between transfers.

1.3.10 DMA Timers (DTIM0-DTIM3)

There are four independent, DMA-transfer-generating 32-bit timers (DTIM[3:0]). Each timer module incorporates a 32-bit timer with a separate register set for configuration and control. The timers can be configured to operate from the system clock or from an external clock source using one of the DT_nIN signals. If the system clock is selected, it can be divided by 16 or 1. The input clock is further divided by a user-programmable 8-bit prescaler which clocks the actual timer counter register (TCR_n). Each of these timers can be configured for input capture or output compare mode. By configuring the internal registers, each timer may be configured to assert an external pin, generate an interrupt on a particular event, or cause a DMA transfer.

1.3.11 On-chip Watchdog Timer

The on-chip watchdog timer is a 16-bit timer that facilitates recovery from runaway code. The watchdog counter is a free-running down-counter that generates a reset on underflow. To prevent a reset, software must periodically restart the countdown.

1.3.12 Periodic Interrupt Timers (PIT0-PIT3)

The four periodic interrupt timers (PIT[3:0]) are 16-bit timers that provide precise interrupts at regular intervals with minimal processor intervention. Each timer can either count down from the value written in its PIT modulus register, or it can be a free-running down-counter.

1.3.13 Clock Module and Phase Locked Loop (PLL)

The device contains a 16 MHz crystal oscillator, a phase-locked loop, as well as status and control registers. The PLL's output dividers and dithering waveform are register programmable. The system operates via two main clocks generated by the PLL, typically 166.67 MHz (core) and 83.33 MHz (peripherals). To improve noise immunity, the PLL has its own power supply inputs, PLL_VDD and PLL_VSS. All other circuits are powered by the normal internal supply pins, IVDD (core), EVDD (I/O), SD_VDD (SDRAM), and VSS.

The PLL circuitry may be bypassed in order to reduce system speed and therefore decrease power consumption. The external clock (EXTAL) is used directly, with an optional programmable divider, to produce the internal core and bus clocks.

1.3.14 Interrupt Controller

There is a single interrupt controller on the MCF5208, which can support up to 63 interrupt sources. Each interrupt source has a unique interrupt vector, and all sources of the controller provide a programmable level (1-7).

1.3.15 DMA Controller

The implementation of the DMA is targeted towards cost sensitive applications while providing a high level of functionality. The DMA executes in parallel with the core, enabling transfers of data between the

memory and peripherals with little intervention from the core, thus increasing system performance, as well as simplifying software development. The DMA is capable of performing complex data transfers via 16 programmable DMA channels. The hardware microarchitecture includes the DMA engine (which performs source/destination address calculations and data movement operations), and a dedicated memory array containing transfer control descriptors.

1.3.16 FlexBus External Interface

The FlexBus provides an external interface to 8-, 16-, or 32-bit memory devices (e.g. SRAM, Flash, ROM, etc.). The FlexBus's internal data lines are shared with the SDRAM controller. When the SDRAMC is in DDR mode (DRAMSEL = 1) the data bus signals, D[31:16], are dedicated to the SDRAM controller and the D[15:0] data bus signals are dedicated to the FlexBus. In SDR mode (DRAMSEL = 0), all 32 data lines are shared between the FlexBus and SDRAM controller.

Features are available to support external Flash modules, for secondary wait states on reads and writes and a signal to support active-low address valid (\overline{TS}). Six programmable chip-select outputs provide signals to enable external memory and peripheral circuits, providing all handshaking and timing signals for automatic wait-state insertion and data bus sizing.

Base memory address and block size are programmable, with some restrictions. For example, the starting address must be on a boundary that is a multiple of the block size. Each chip select can be configured to provide read and write enable signals suitable for use with most popular static RAMs and peripherals. Data bus width (8-bit, 16-bit, or 32-bit) is programmable on all chip selects, and further decoding is available for protection from user mode access or read-only access.

1.3.17 Reset Controller Module

The reset controller is provided to determine the cause of reset, assert the appropriate reset signals to the system, and keep track of what caused the last reset. There are five sources of reset:

- External
- Power-on reset (POR)
- Watchdog timer
- Phase locked-loop (PLL) loss of lock
- Software

External reset on the \overline{RSTOUT} pin is software-assertable independent of chip reset state. There are also software-readable status flags indicating the cause of the last reset.

1.3.18 GPIO

Unused bus interface and peripheral pins can be used as discrete general-purpose inputs and outputs. These are managed by a dedicated GPIO module that logically groups all pins into ports located within a contiguous block of memory-mapped control registers. Each port has registers that configure, monitor, and control the port pins. Slew rate control or output pad drive strength control is available on all pins.

Most of the pins associated with the FlexBus interface may be used for several different functions. Their primary function is to provide an external interface to access off-chip resources. When not used for this, the pins may be used as general-purpose digital I/O pins.

1.4 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 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 SD_SRAS and 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., A23), while designations for multiple signals within a group use brackets (i.e., A[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. Pins that are muxed with GPIO will default to their GPIO functionality.

Table 2-1. MCF5207/8 Signal Information and Muxing

Signal Name	GPIO	Alternate 1	Alternate 2	Dir. ¹	MCF5207 144 LQFP	MCF5207 144 MAPBGA	MCF5208 160 QFP	MCF5208 196 MAPBGA
Reset								
RESET ²	—	—	—	I	82		90	J14
RSTOUT	—	—	—	O	74		82	N14

Signal Descriptions

Table 2-1. MCF5207/8 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir. ¹	MCF5207 144 LQFP	MCF5207 144 MAPBGA	MCF5208 160 QFP	MCF5208 196 MAPBGA
Clock								
EXTAL	—	—	—	I	78		86	L14
XTAL	—	—	—	O	80		88	K14
FB_CLK	—	—	—	O	34		40	N1
Mode Selection								
RCON ²	—	—	—	I	144		160	C3
DRAMSEL	—	—	—	I	79		87	K11
FlexBus								
A[23:22]	—	FB_CS[5:4]	—	O	118, 117		126, 125	B11, A11
A[21:16]	—	—	—	O	116–114, 112, 108, 107		124, 123, 122, 120, 116, 115	B12, A12, A13, B13, B14, C13
A[15:14]	—	SD_BA[1:0]	—	O	106, 105		114, 113	C14, D12
A[13:11]	—	SD_A[13:11]	—	O	104–102		112, 111, 110	D13, D14, E11
A10	—	—	—	O	101		109	E12
A[9:0]	—	SD_A[9:0]	—	O	100–91		108–99	E13, E14, F11–14, G11–14
D[31:16]	—	SD_D[31:16] ³	—	O	21–28, 40–47		27–34, 46–53	J4–1, K4–1, M3, N3, M4, N4, P4, L5, M5, N5
D[15:0]	—	FB_D[31:16] ³	—	O	8–15, 51–58		16–23, 57–64	F3–1, G4–1, H1, N6, P6, L7, M7, N7, P7, N8, P8
BE/BWE[3:0]	PBE[3:0]	SD_DQM[3:0]	—	O	18, 20, 48, 50		26, 54, 24, 56	H2, P5, H4, M6
OE	PBUSCTL3	—	—	O	60		66	M8
TA ²	PBUSCTL2	—	—	I	90		98	H14
R/W	PBUSCTL1	—	—	O	59		65	L8
TS	PBUSCTL0	DACK0	—	O	4		12	E3
Chip Selects								

Table 2-1. MCF5207/8 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir. ¹	MCF5207 144 LQFP	MCF5207 144 MAPBGA	MCF5208 160 QFP	MCF5208 196 MAPBGA
FB_CS[3:2]	PCS[3:2]	—	—	O	119, 120		—	C11, A10
FB_CS1	PCS1	SD_CS1	—	O	121		127	B10
FB_CS0	—	—	—	O	122		128	C10
SDRAM Controller								
SD_A10	—	—	—	O	37		43	N2
SD_CKE	—	—	—	O	6		14	E1
SD_CLK	—	—	—	O	31		37	L1
SD_CLK	—	—	—	O	32		38	M1
SD_CS0	—	—	—	O	7		15	F4
SD_DQS[3:2]	—	—	—	O	19, 49		25, 55	H3, L6
SD_SCAS	—	—	—	O	38		44	P2
SD_SRAS	—	—	—	O	39		45	P3
SD_SDR_DQS	—	—	—	O	29		35	L3
SD_WE	—	—	—	O	5		13	E2
External Interrupts Port⁴								
IRQ7 ²	PIRQ7 ²	—	—	I	134		142	C7
IRQ4 ²	PIRQ4 ²	DREQ0 ²	—	I	133		141	D7
IRQ1 ²	PIRQ1 ²	—	—	I	132		140	D8
FEC								
FEC_MDC	PFECl2C3	I2C_SCL ²	U2TXD	O	—	—	148	D6
FEC_MDIO	PFECl2C2	I2C_SDA ²	U2RXD	I/O	—	—	147	C6
FEC_TXCLK	PFECH7	—	—	I	—	—	157	B3
FEC_TXEN	PFECH6	—	—	O	—	—	158	A2
FEC_TXD0	PFECH5	—	—	O	—	—	3	B1
FEC_COL	PFECH4	—	—	I	—	—	7	D3
FEC_RXCLK	PFECH3	—	—	I	—	—	154	B4
FEC_RXDV	PFECH2	—	—	I	—	—	153	A4
FEC_RXD0	PFECH1	—	—	I	—	—	152	D5
FEC_CRS	PFECH0	—	—	I	—	—	8	D2
FEC_TXD[3:1]	PFECL[7:5]	—	—	O	—	—	6–4	C1, C2, B2
FEC_TXER	PFECL4	—	—	O	—	—	156	A3

Table 2-1. MCF5207/8 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir. ¹	MCF5207 144 LQFP	MCF5207 144 MAPBGA	MCF5208 160 QFP	MCF5208 196 MAPBGA
FEC_RXD[3:1]	PFECL[3:1]	—	—	I	—	—	149–151	A5, B5, C5
FEC_RXER	PFECL0	—	—	I	—	—	155	C4
I²C								
I2C_SDA ²	PFECI2C0 ²	U2RXD ²	—	I/O	—	—	—	D1
I2C_SCL ²	PFECI2C1 ²	U2TXD ²	—	I/O	—	—	—	E4
DMA								
<p>DACK0 and DREQ0 do not have a dedicated bond pads. Please refer to the following pins for muxing: TS and QSPI_CS2 for DACK0, IRQ4 and QSPI_DIN for DREQ0.</p>								
QSPI								
QSPI_CS2	PQSPI3	DACK0	U2RTS	O	126		132	D10
QSPI_CLK	PQSPI0	I2C_SCL ²	—	O	127		133	A9
QSPI_DOUT	PQSPI1	I2C_SDA ²	—	O	128		134	B9
QSPI_DIN	PQSPI2	DREQ0 ²	U2CTS	I	129		135	C9
<p>Note: The QSPI_CS1 and QSPI_CS0 signals are available on the U1CTS, U1RTS, U0CTS, or U0RTS pins for the 196 and 160-pin packages.</p>								
UARTs								
U1CTS	PUARTL7	—	—	I	139		—	—
U1RTS	PUARTL6	—	—	O	142		—	—
U1CTS	PUARTL7	DT1IN	QSPI_CS1	I	—	—	136	D9
U1RTS	PUARTL6	DT1OUT	QSPI_CS1	O	—	—	137	C8
U1TXD	PUARTL5	—	—	O	131		139	A8
U1RXD	PUARTL4	—	—	I	130		138	B8
U0CTS	PUARTL3	—	—	I	140		—	—
U0RTS	PUARTL2	—	—	O	141		—	—
U0CTS	PUARTL3	DT0IN	QSPI_CS0	I	—	—	76	N12
U0RTS	PUARTL2	DT0OUT	QSPI_CS0	O	—	—	77	P12
U0TXD	PUARTL1	—	—	O	71		79	P13
U0RXD	PUARTL0	—	—	I	70		78	N13
<p>Note: The UART2 signals are multiplexed on the DMA Timers, QSPI, FEC, and I2C pins.</p>								
DMA Timers								
DT3IN	PTIMER3	DT3OUT	U2CTS	I	135		143	B7

Table 2-1. MCF5207/8 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir. ¹	MCF5207 144 LQFP	MCF5207 144 MAPBGA	MCF5208 160 QFP	MCF5208 196 MAPBGA
DT2IN	PTIMER2	DT2OUT	U2RTS	I	136		144	A7
DT1IN	PTIMER1	DT1OUT	U2RXD	I	137		145	A6
DT0IN	PTIMER0	DT0OUT	U2TXD	I	138		146	B6
BDM/JTAG⁵								
JTAG_EN ⁶	—	—	—	I	83		91	J13
DSCLK	—	TRST ²	—	I	76		84	L12
PSTCLK	—	TCLK ²	—	O	64		70	P9
BKPT	—	TMS ²	—	I	75		83	M14
DSI	—	TDI ²	—	I	77		85	K12
DSO	—	TDO	—	O	69		75	M12
DDATA[3:0]	—	—	—	O	—		—	P11, N11, M11, P10
PST[3:0]	—	—	—	O	—		—	N10, M10, L10, L9
ALLPST	—	—	—	O	67	—	73	—
Test								
TEST ⁶	—	—	—	I	109		117	C12
PLL_TEST	—	—	—	I	—		—	M13
Power Supplies								
EVDD	—	—	—		1, 63, 66, 72, 81, 87, 125		2, 9, 69, 72, 80, 89, 95, 131	E5–7, F5, F6, G5, H10, J9, J10, K8–10, K13, M9
IVDD	—	—	—		30, 68, 84, 113, 143		36, 74, 92, 121, 159	J12, D4, D11, H11, L4, L11,
PLL_VDD	—	—	—		86		94	H13

Table 2-1. MCF5207/8 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir. ¹	MCF5207 144 LQFP	MCF5207 144 MAPBGA	MCF5208 160 QFP	MCF5208 196 MAPBGA
SD_VDD	—	—	—		3, 17, 33, 35, 61, 89, 110, 123		11, 39, 41, 67, 97, 118, 129	E8–10, F9, F10, G10, H5, J5, J6, K5–7, L2
VSS	—	—	—		2, 16, 36, 62, 65, 73, 88, 111, 124		1, 10, 42, 68, 71, 81, 96, 119, 130	A1, A14, F7–8, G6–9, H6–9, J7–8, L13, M2, N9, P1, P14
PLL_VSS	—	—	—		85		93	H12

¹ Refers to pin's primary function.

2 Pull-up enabled internally on this signal for this mode.

3 Alternate functionality selected by asserting the DRAMSEL signal (DDR mode). The GPIO module is not responsible for assigning these pins.

4 GPIO functionality is determined by the edge port module. The GPIO module is only responsible for assigning the alternate functions.

5 If JTAG_EN is asserted, these pins default to Alternate 1 (JTAG) functionality. The GPIO module is not responsible for assigning these pins.

6 Pull-down enabled internally on this signal for this mode.

2.2.1 Internal Pull-up/Pull-down Resistors

The following table summarizes which external signals contain internal pull-up or pull-down resistors.

Table 2-2. Internal Pull-up/down Resistors

Pin Name	Pull-Up	Pull-Down	Comment
RESET	x		
TEST		x	
RCON	x		
IRQ[7,4,1]	x		
TA	x		
QSPI_DOUT	x		I2C mode only (I2C_SDA)
QSPI_CLK	x		I2C mode only (I2C_SCL)
FEC_MDIO	x		I2C mode only (I2C_SDA)
FEC_MDC	x		I2C mode only (I2C_SCL)
QSPI_DIN	x		DMA mode only ($\overline{DREQ_0}$)

Table 2-2. Internal Pull-up/down Resistors (continued)

Pin Name	Pull-Up	Pull-Down	Comment
I2C_SDA	x		
I2C_SCL	x		
JTAG_EN		x	
TDI	x		JTAG mode only
TMS	x		JTAG mode only
TRST	x		JTAG mode only
TCLK	x		JTAG mode only

2.3 Signal Primary Functions

2.3.1 Reset Signals

Table 2-3 describes signals that are used to either reset the chip or as a reset indication.

Table 2-3. Reset Signals

Signal Name	Abbreviation	Function	I/O
Reset In	RESET	Primary reset input to the device. Asserting <u>RESET</u> immediately resets the core and peripherals, which stay in reset until <u>RESET</u> is negated.	I
Reset Out	RSTOUT	Reset output (<u>RSTOUT</u>) is an indicator that the chip is in reset. <u>RSTOUT</u> is driven low for 512 FB_CLK clock cycles in response to any internal or external reset.	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 when the internal oscillator circuit is used. The clock source may be configured during reset by asserting <u>RCON</u> . See Chapter 9, “Chip Configuration Module (CCM)” 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.	O
FlexBus Clock Out	FB_CLK	Reflects the internal bus clock (or one-half the core/system clock). ($f_{sys}/2$)	O

2.3.3 Mode Selection

Table 2-5 describes signals used in mode selection.

Table 2-5. Mode Selection Signals

Signal Name	Abbreviation	Function	I/O
Reset Configuration	RCON	Indicates whether the external D[15:0] pin states affect chip configuration at reset.	I
SDR/DDR SDRAM Select	DRAMSEL	Controls whether certain pins act as FlexBus or SDRAMC signals. When asserted, D[31:0] dynamically switches between SDR data and FlexBus data. When negated, D[31:16] are dedicated for DDR data while D[15:0] are dedicated for FlexBus data.	I

2.3.4 FlexBus Signals

Table 2-6 describes signals that are used for doing transactions on the external bus.

Table 2-6. FlexBus Signals

Signal Name	Abbreviation	Function	I/O
Address Bus	A[23:0]	The 24 dedicated address signals define the address of external byte, word, and longword accesses. These three-state outputs are the 24 lsbs of the internal 32-bit address bus and multiplexed with the SDRAM controller row and column addresses.	O
Data Bus	D[31:0]	These three-state bidirectional signals provide the general purpose data path between the processor and all other devices.	I/O
Byte Enables	BW/BWE[3:0]	<p>Define the flow of data on the data bus. During SRAM and peripheral accesses, these output signals indicate that data is to be latched or driven onto a byte of the data when driven low. The BE/BWE[3:0] signals are asserted only to the memory bytes used during a read or write access. $\overline{BE/BWE}0$ controls access to the most significant byte lane of data, and $\overline{BE/BWE}3$ controls access to the least significant byte lane of data.</p> <p>The $\overline{BE/BWE}n$ signals are asserted during accesses to on-chip peripherals but not to on-chip SRAM or cache. During SDRAM accesses, these signals act as the DQM[3:0] signals, which indicate a byte transfer between SDRAM and the chip when driven high.</p> <p>For SRAM or Flash devices, the $\overline{BE/BWE}n$ outputs should be connected to individual byte strobe signals.</p> <p>For SDRAM devices, the $\overline{BE/BWE}n$ should be connected to individual SDRAM DQM signals. Note that most SDRAMs associate DQM3 with the MSB, in which case $\overline{BE/BWE}0$ should be connected to the SDRAM's DQM3 input.</p>	O
Output Enable	OE	Indicates when an external device can drive data during external read cycles.	O
Transfer Acknowledge	TA	Indicates that the external data transfer is complete. During a read cycle, when the processor recognizes TA, it latches the data and then terminates the bus cycle. During a write cycle, when the processor recognizes TA, the bus cycle is terminated.	I

Table 2-6. FlexBus Signals (continued)

Signal Name	Abbreviation	Function	I/O
Read/Write	R/W	Indicates the 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	TS	Bus control output signal indicating the start of a transfer.	O
Chip Selects	FB_CS[5:0]	These output signals select external devices for external bus transactions.	O

2.3.5 SDRAM Controller Signals

Table 2-7 describes signals that are used for SDRAM accesses.

Table 2-7. SDRAM Controller Signals

Signal Name	Abbreviation	Function	I/O
SDRAM A10	SD_A10	Bit 10 of the SDRAM Address bus	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_CLK	Inverted output clock for DDR SDRAM.	O
SDRAM Chip Selects	SD_CS[1:0]	SDRAM chip select signals.	O
DDR SDRAM Data Strobes	SD_DQS[3:2]	Indicates when valid data is on the data bus. SD_DQS1 is tied to SD_DQS3 and SD_DQS0 is tied to SD_DQS2 internally.	I/O
SDR SDRAM Write Data Byte Mask	SD_DQM[3:0]	Used to determine which byte lanes of the data bus should be latched during a write cycle.	O
SDRAM Synchronous Column Address Strobe	SD_SCAS	SDRAM synchronous column address strobe.	O
SDRAM Synchronous Row Address Strobe	SD_SRAS	SDRAM synchronous row address strobe.	O
SDR SDRAM Data Strobe	SD_SDRDQS	Generated by the memory controller in SDR mode, to mimic the DQS generated by DDR memories during reads. It is routed out and connected back to SD_DQS inputs.	O
SDRAM Write Enable	SD_WE	Indicates the direction of the data transfer on the 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

2.3.6 External Interrupt Signals

Table 2-8 describes the external interrupt signals.

Table 2-8. External Interrupt Signals

Signal Name	Abbreviation	Function	I/O
External Interrupts	IRQ[7,4,1]	External interrupt sources.	I

2.3.7 DMA Signals

Table 2-9 describes the external DMA signals.

Table 2-9. DMA Signals

Signal Name	Abbreviation	Function	I/O
DMA Request	DREQ0	Active low external DMA request lines.	I
DMA Acknowledge	DACK0	Active low external DMA acknowledge lines.	O

2.3.8 Ethernet Module (FEC) Signals

The following signals are used by the Ethernet module for data and clock signals.

Table 2-10. Ethernet Module (FEC) Signals

Signal Name	Abbreviation	Function	I/O
Management Data	FEC_MDIO	Transfers control information between the external PHY and the media-access controller. Data is synchronous to FEC_MDC. Applies to MII mode operation. This signal is an input after reset. When the FEC is operated in 10Mbps 7-wire interface mode, this signal should be connected to VSS.	I/O
Management Data Clock	FEC_MDC	In Ethernet mode, FEC_MDC is an output clock which provides a timing reference to the PHY for data transfers on the FEC_MDIO signal. Applies to MII mode operation.	O
Collision	FEC_COL	Asserted upon detection of a collision and remains asserted while the collision persists. This signal is not defined for full-duplex mode.	I
Carrier Receive Sense	FEC_CRS	When asserted, indicates that transmit or receive medium is not idle. Applies to MII mode operation.	I
Transmit Clock	FEC_TXCLK	Input clock which provides a timing reference for FEC_TXEN, FEC_TXD[3:0] and FEC_TXER	I
Transmit Enable	FEC_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 FEC_TXCLK following the final nibble of the frame.	O
Transmit Data 0	FEC_TXD0	FEC_TXD0 is the serial output Ethernet data and is only valid during the assertion of FEC_TXEN. This signal is used for 10-Mbps Ethernet data. It is also used for MII mode data in conjunction with FEC_TXD[3:1].	O
Transmit Data 1–3	FEC_TXD[3:1]	In Ethernet mode, these pins contain the serial output Ethernet data and are valid only during assertion of FEC_TXEN in MII mode.	O
Transmit Error	FEC_TXER	In Ethernet mode, when FEC_TXER is asserted for one or more clock cycles while FEC_TXEN is also asserted, the PHY sends one or more illegal symbols. FEC_TXER has no effect at 10 Mbps or when FEC_TXEN is negated. Applies to MII mode operation.	O
Receive Clock	FEC_RXCLK	Provides a timing reference for FEC_RXDV, FEC_RXD[3:0], and FEC_RXER.	I

Table 2-10. Ethernet Module (FEC) Signals (continued)

Signal Name	Abbreviation	Function	I/O
Receive Data Valid	FEC_RXDV	Asserting the FEC_RXDV input indicates that the PHY has valid nibbles present on the MII. FEC_RXDV should remain asserted from the first recovered nibble of the frame through to the last nibble. Assertion of FEC_RXDV must start no later than the SFD and exclude any EOF.	I
Receive Data 0	FEC_RXD0	FEC_RXD0 is the Ethernet input data transferred from the PHY to the media-access controller when FEC_RXDV is asserted. This signal is used for 10-Mbps Ethernet data. This signal is also used for MII mode Ethernet data in conjunction with FEC_RXD[3:1].	I
Receive Data 1–3	FEC_RXD[3:1]	In Ethernet mode, these pins contain the Ethernet input data transferred from the PHY to the media access controller when FEC_RXDV is asserted in MII mode operation.	I
Receive Error	FEC_RXER	In Ethernet mode, FEC_RXER—when asserted with FEC_RXDV—indicates that the PHY has detected an error in the current frame. When FEC_RXDV is not asserted FEC_RXER has no effect. Applies to MII mode operation.	I

2.3.9 I²C I/O Signals

Table 2-11 describes the I²C serial interface module signals.

Table 2-11. I²C I/O Signals

Signal Name	Abbreviation	Function	I/O
Serial Clock	I2C_SCL	Open-drain clock signal for the I ² C interface. Either it is driven by the I ² C module when the bus is in the master mode, or it becomes the clock input when the I ² C is in the slave mode.	I/O
Serial Data	I2C_SDA	Open-drain signal that serves as the data input/output for the I ² C interface.	I/O

2.3.10 Queued Serial Peripheral Interface (QSPI)

Table 2-12 describes QSPI signals.

Table 2-12. Queued Serial Peripheral Interface (QSPI) Signals

Signal Name	Abbreviation	Function	I/O
QSPI Syncrhonous Serial Output	QSPI_DOUT	Provides the serial data from the QSPI and can be programmed to be driven on the rising or falling edge of QSPI_CLK. Each byte is sent msb first.	O
QSPI Synchronous Serial Data Input	QSPI_DIN	Provides the serial data to the QSPI and can be programmed to be sampled on the rising or falling edge of QSPI_CLK. Each byte is written to RAM lsb first.	I
QSPI Serial Clock	QSPI_CLK	Provides the serial clock from the QSPI. The polarity and phase of QSPI_CLK are programmable. The output frequency is programmed according to the following formula, in which n can be any value between 1 and 255: $\text{SPI_CLK} = f_{\text{sys}/2} \div (2 \times n)$	O
Synchronous Peripheral Chip Selects	QSPI_CS[2:0]	Provide QSPI peripheral chip selects that can be programmed to be active high or low.	O

2.3.11 UART Module Signals

Table 2-13 describes the signals of the three UART modules, where $n=0\text{--}2$. Baud rate clock inputs are not supported.

Table 2-13. UART Module Signals

Signal Name	Abbreviation	Function	I/O
Transmit Serial Data Output	UnTXD	Transmitter serial data outputs. Data is shifted out lsb first on this pin at the falling edge of the serial clock source. The output is held high when the transmitter is disabled, idle, or in local loopback mode.	O
Receive Serial Data Input	UnRXD	Receiver serial data inputs. Data is sampled on the rising edge of the serial clock source lsb first. When the UART clock is stopped for power-down mode, any transition on this pin restarts it.	I
Clear-to-Send	UnCTS	Indicates that the UART modules can begin data transmission	I
Request-to-Send	UnRTS	Automatic request-to-send outputs from the UART modules. They may also be asserted and negated as a function of the receive FIFO level.	O

2.3.12 DMA Timer Signals

Table 2-14 describes the signals of the four DMA timer modules, where $n=0\text{--}3$.

Table 2-14. DMA Timer Signals

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

2.3.13 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 2-15. Debug Support Signals

Signal Name	Abbreviation	Function	I/O
Test Reset	TRST	This active-low signal is 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
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 is actively driven in the shift-IR and shift-DR controller states. TDO changes on the falling edge of TCLK.	O
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	This internally-synchronized signal provides data input for the serial communication port to the BDM module.	I
Development Serial Output	DSO	This internally-registered signal provides serial output communication for BDM module responses.	O
Processor Status Clock	PSTCLK	Used by the development system to known when to sample the 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]. Only present on the BGA devices (MCF5208CVM and MCF5207CVM).	O
Processor Status Outputs	PST[3:0]	Indicate core status, as shown in Table 2-16 . 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]. Only present on the BGA devices (MCF5208CVM and MCF5207CVM).	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 (MCF5207CABxx and MCF5208CABxx). When asserted, reflects that the core is halted.	O

Table 2-16. Processor Status

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

2.3.14 Test Signals

Table 2-17 describes test signals which are reserved for factory testing.

Table 2-17. 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
PLL Test	PLL_TEST	Reserved for factory testing only and should be treated as a no-connect (NC).	O

2.3.15 Power and Ground Pins

The pins described in Table 2-18 provide system power and ground to the chip. Multiple pins are provided for adequate current capability. All power supply pins must have adequate bypass capacitance for high-frequency noise suppression.

Table 2-18. Power and Ground Pins

Signal Name	Abbreviation	Function	I/O
PLL Analog Supply	PLL_VDD PLL_VSS	Dedicated power supply signals to isolate the sensitive 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.	
SDRAMC Supply	SD_VDD	These pins supply positive power to the SDRAM controller.	
Ground	VSS	These pins are the negative supply (ground) for the device.	

2.4 External Boot Mode

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

Chapter 3

ColdFire Core

This section describes the organization of the Version 2 (V2) ColdFire® processor core and an overview of the program-visible registers. For detailed information on instructions, use the ColdFire ISA opcodes classified as ISA_A+ in the *ColdFire Family Programmer's Reference Manual*.

3.1 Processor Pipelines

Figure 3-1 is a block diagram showing the processor pipelines of a V2 ColdFire core.

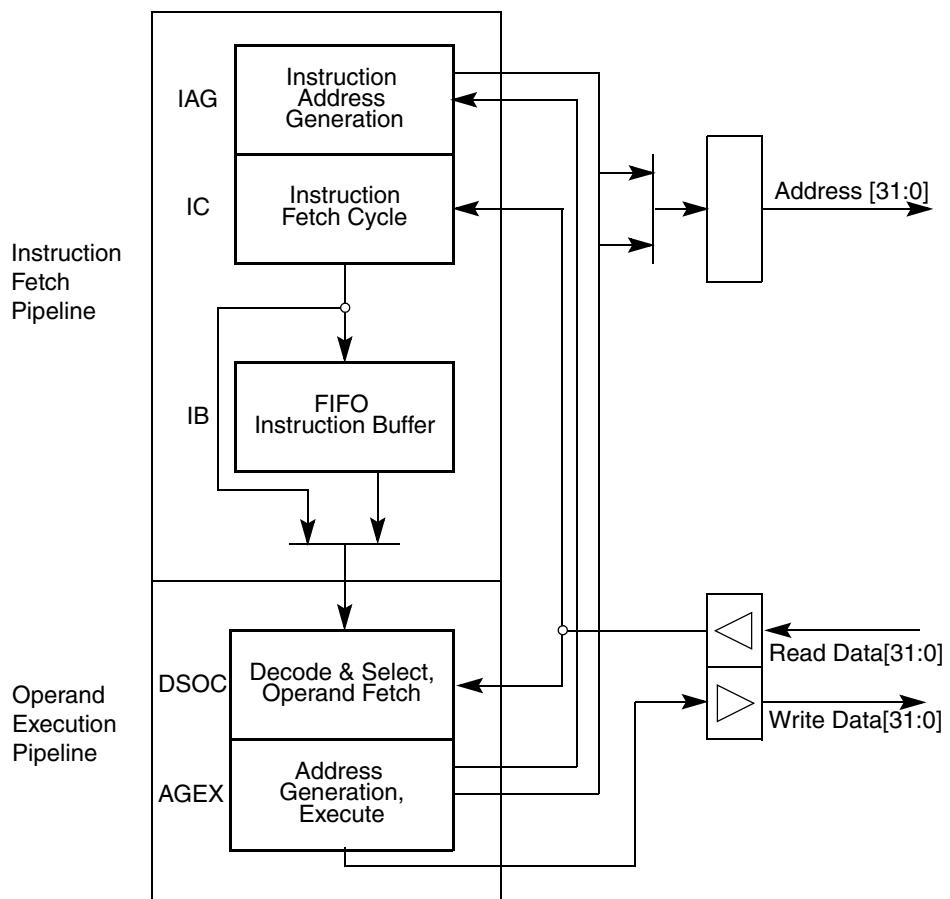


Figure 3-1. ColdFire Processor Core Pipelines

The processor core is comprised of two separate pipelines that are decoupled by an instruction buffer.

The instruction fetch pipeline (IFP) is a two-stage pipeline for prefetching instructions. The prefetched instruction stream is then gated into the two-stage operand execution pipeline (OEP), which decodes the

instruction, fetches the required operands, and then executes the required function. Since the IFP and OEP pipelines are decoupled by an instruction buffer that serves 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 instruction fetch pipeline consists of two stages with an instruction buffer stage:

- Instruction address generation (IAG cycle)
- Instruction fetch cycle (IC cycle)
- Instruction buffer (IB cycle)

When the instruction buffer is empty, opcodes are loaded directly from the IC cycle into the operand execution pipeline. If the buffer is not empty, the IFP stores the contents of the fetch cycle in the FIFO queue until it is required by the OEP. In the Version 2 implementation, the instruction buffer contains three 32-bit longwords of storage.

The operand execution pipeline is implemented in a two-stage pipeline featuring a traditional RISC datapath with a dual-read-ported register file (RGF) feeding an arithmetic/logic unit. In this design, the pipeline stages have multiple functions:

- Decode and select/operand cycle (DSOC cycle)
- Address generation/execute cycle (AGEX cycle)

3.2 Memory Map/Register Definition

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

The user access model is the same as the M68000 family microprocessors, consisting 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)

Only system control software is intended to use the supervisor programming model 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, which 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)
- Two 32-bit access control registers (ACR0, ACR1)
- 32-bit base address registers (RAMBAR0, RAMBAR1)

Table 3-1. ColdFire Core Programming Model

CPU Space (Rc)	Register	Width (bits)	Access	Reset Value	Written w/ MOVEC	Section/Page
User Access Registers						
Load: 0x080 Store: 0x180	Data Register 0 (D0)	32	R/W	0xCF20_6089	No	3.2.1/3-3
Load: 0x081 Store: 0x181	Data Register 1 (D1)	32	R/W	0x1500_1060	No	3.2.1/3-3
Load: 0x082–7 Store: 0x182–7	Data Register 2–7 (D2–D7)	32	R/W	Undefined	No	3.2.1/3-3
Load: 0x088–8E Store: 0x188–8E	Address Register 0–6 (A0–A6)	32	R/W	Undefined	No	3.2.2/3-4
Load: 0x08F Store: 0x18F	Address Register 7/User Stack Pointer (A7)	32	R/W	Undefined	No	3.2.3/3-4
0x804	MAC Status Register (MACSR)	32	R/W	0x0000_0000	No	3.2.4/3-5
0x805	MAC Address Mask Register (MASK)	32	R/W	0xFFFF_FFFF	No	3.2.4/3-5
0x806, 0x809, 0x80A, 0x80B	MAC Accumulators 0–3 (ACC0–3)	32	R/W	Undefined	No	3.2.4/3-5
0x807	MAC Accumulator 0,1 Extension Bytes (ACCext01)	32	R/W	Undefined	No	3.2.4/3-5
0x808	MAC Accumulator 2,3 Extension Bytes (ACCext23)	32	R/W	Undefined	No	3.2.4/3-5
0x80E	Condition Code Register (CCR)	8	R	0x00	No	3.2.5/3-5
0x80F	Program Counter (PC)	32	R/W	Undefined	Yes	3.2.6/3-6
Supervisor Access Only Registers						
0x002	Cache Control Register (CACR)	32	R/W	0x0000_0000	Yes	3.2.7/3-6
0x004	Access Control Register 0 (ACR0)	32	R/W	Undefined	Yes	3.2.8/3-6
0x005	Access Control Register 1 (ACR1)	32	R/W	Undefined	Yes	3.2.8/3-6
Load: 0x08F Store: 0x18F	Supervisor/User A7 Stack Pointer (A7)	32	R/W	Undefined	No	3.2.9/3-6
0x800	User/Supervisor A7 Stack Pointer (OTHER_A7)	32	R/W	Undefined	No	3.2.9/3-6
0x801	Vector Base Register (VBR)	32	R/W	0x0000_0000	Yes	3.2.10/3-7
0x80E	Status Register (SR)	16	R/W	0x2700	No	3.2.11/3-7
0xC05	RAM Base Address Register 1 (RAMBAR1)	32	R/W	0x0000_0000	Yes	3.2.12/3-8

3.2.1 Data Registers (D0–D7)

Registers D0–D7 are used as data registers 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 3.6.14, “Reset Exception”](#) for more details.

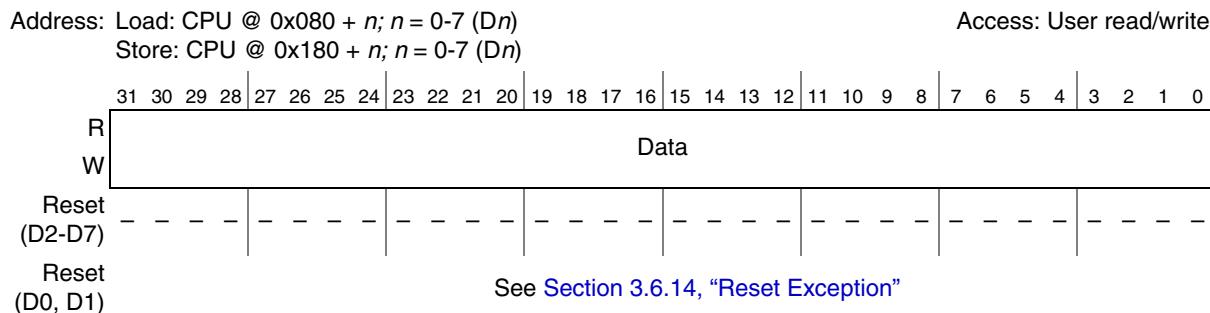


Figure 3-2. Data Registers (D0–D7)

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.

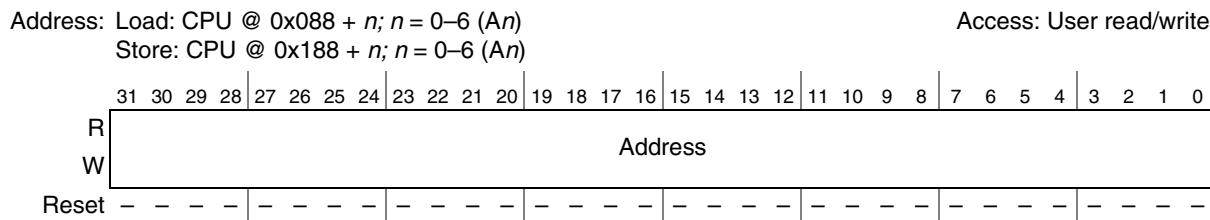


Figure 3-3. Address Registers (A0–A6)

3.2.3 Stack Pointer (A7)

This ColdFire implementation supports two unique stack pointer (A7) registers—the supervisor stack pointer (SSP) and the user stack pointer (USP). This support provides the required isolation between operating modes of the processor. The SSP is described in [Section 3.2.9, “Supervisor/User Stack Pointers \(A7 and OTHER A7\).”](#)

A subroutine call saves the PC on the stack and the return restores it from the stack. Both the PC and the SR are saved on the supervisor stack during the processing of exceptions and interrupts. The return from exception (RTE) instruction restores the SR and PC values from the supervisor stack.

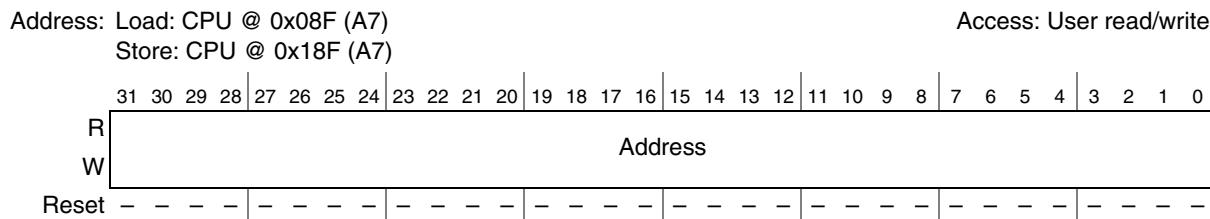


Figure 3-4. Stack Pointer Register (A7)

3.2.4 EMAC Register Description

The registers in the EMAC portion of the user programming model are described in [Section Chapter 4, “Enhanced Multiply-Accumulate Unit \(EMAC\)”](#) and include the following registers:

- 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, and 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)

3.2.5 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. Bit 4, the extend bit (X bit), is also used as an input operand during multiprecision arithmetic computations.

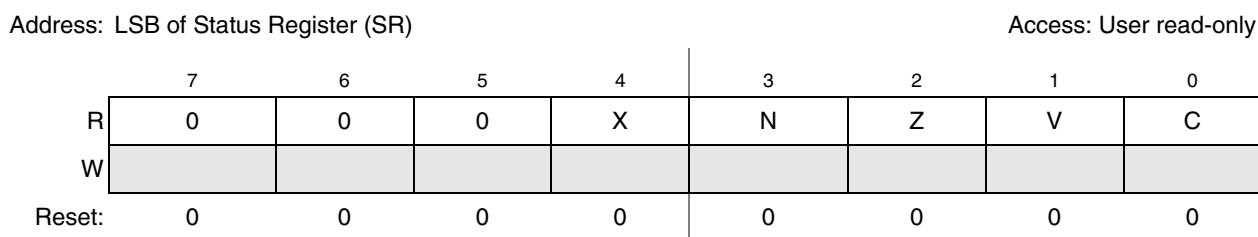


Figure 3-5. Condition Code Register (CCR)

Table 3-2. CCR Field Descriptions

Field	Description
7–5	Reserved, should be cleared.
4 X	Extend condition code bit. Set to the value of the C-bit for arithmetic operations; otherwise not affected or set to a specified result.
3 N	Negative condition code bit. Set if the most significant bit of the result is set; otherwise cleared.
2 Z	Zero condition code bit. Set if the result equals zero; otherwise cleared.
1 V	Overflow condition code bit. Set if an arithmetic overflow occurs implying that the result cannot be represented in the operand size; otherwise cleared.
0 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. Set to the value of the C bit for arithmetic operations; otherwise not affected.

3.2.6 Program Counter (PC)

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

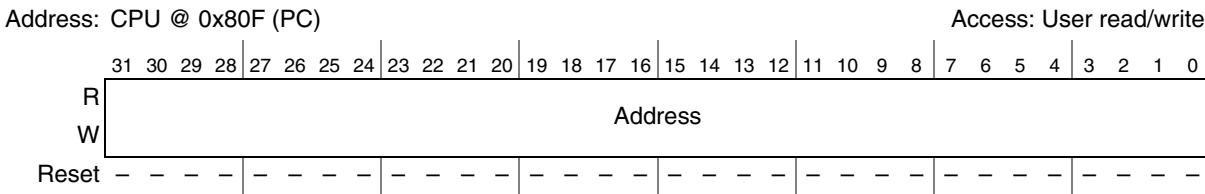


Figure 3-6. Program Counter Register (PC)

3.2.7 Cache Control Register (CACR)

The CACR controls operation of the instruction/data cache memories. It includes bits for enabling, freezing, and invalidating cache contents. It also includes bits for defining the default cache mode and write-protect fields. The supervisor access-only CACR is described in [Section 5.2.1, “Cache Control Register \(CACR\).”](#)

3.2.8 Access Control Registers (ACR0, ACR1)

The access control registers, ACR0 and ACR1, define attributes for two user-defined memory regions and is only accessible in supervisor mode. These attributes include the definition of cache mode, write protect, and buffer write enables. The ACRs are described in [Section 5.2.2, “Access Control Registers \(ACR0, ACR1\).”](#)

3.2.9 Supervisor/User Stack Pointers (A7 and OTHER_A7)

This 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 programmable-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 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 the stack pointer (A7), defined for previous ColdFire versions, is available. EUSP is zero at reset.

If EUSP is set, the appropriate stack pointer register (SSP or USP) is accessed as a function of the processor's operating mode. To support dual stack pointers, the following two privileged M68000 instructions are added to 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*.

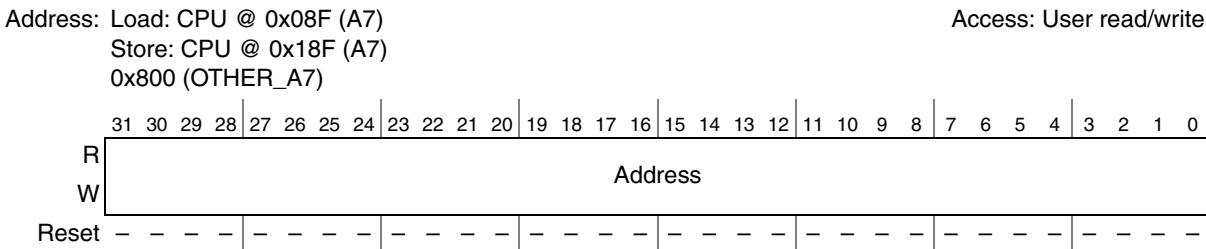


Figure 3-7. Stack Pointer Registers (A7 and OTHER_A7)

3.2.10 Vector Base Register (VBR)

The VBR contains the base address of the exception vector table in 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 MByte boundary.

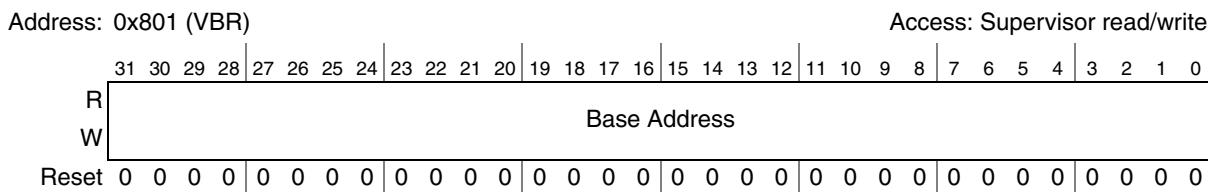


Figure 3-8. Vector Base Register (VBR)

3.2.11 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 are accessible (CCR). 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.

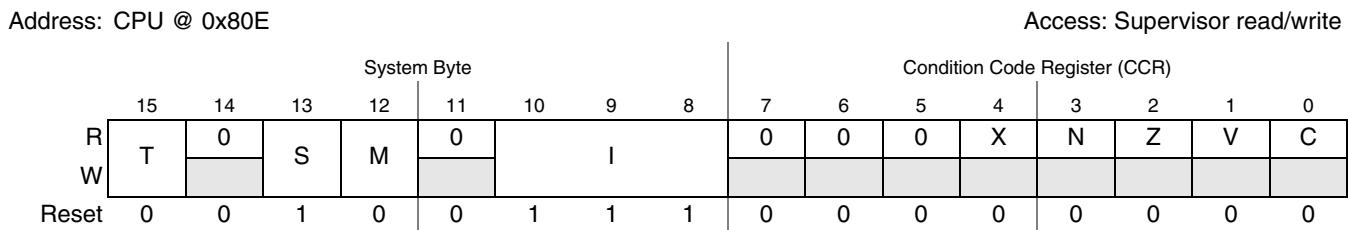


Figure 3-9. Status Register (SR)

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, should be cleared.
13 S	Supervisor/user state. Denotes whether the processor is in supervisor mode (S = 1) or user mode (S = 0).
12 M	Master/interrupt state. This bit is cleared by an interrupt exception, and can be set by software during execution of the RTE or move to SR instructions.
11	Reserved, should be cleared.
10–8 I	Interrupt level mask. Defines the current interrupt level. Interrupt requests are inhibited for all priority levels less than or equal to the current level, except the edge-sensitive level 7 request, which cannot be masked.
7–5	Reserved, should be cleared.
4–0 CCR	Refer to Table 3-2 .

3.2.12 RAM Base Address Register (RAMBAR1)

The RAMBAR register is used to specify the base address of the internal SRAM and indicate 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. For more information, refer to [Section 6.2.1, “SRAM Base Address Register \(RAMBAR\).”](#)

3.3 Additions to the Instruction Set Architecture

The original ColdFire instruction set architecture (ISA) was derived from the M68000-family opcodes based on extensive analysis of embedded application code. After the initial ColdFire compilers were created, developers identified ISA additions that would enhance both code density and overall performance. Additionally, as users implemented ColdFire-based designs into a wide range of embedded systems, they identified frequently used instruction sequences that could be improved by the creation of new instructions. This observation was especially prevalent in development environments that made use of substantial amounts of assembly language code.

[Table 3-4](#) summarizes the new instructions added to the Revision A+ ISA. For more details see the *ColdFire Family Programmer’s Reference Manual*.

Table 3-4. ISA Revision A+ New Instructions

Instruction	Description
BITREV	The contents of the destination data register are bit-reversed; that is, new Dn[31] = old Dn[0], new Dn[30] = old Dn[1], ..., new Dn[0] = old Dn[31].
BYTEREV	The contents of the destination data register are byte-reversed; that is, new Dn[31:24] = old Dn[7:0], ..., new Dn[7:0] = old Dn[31:24].

Table 3-4. ISA Revision A+ New Instructions (continued)

Instruction	Description
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.
STLDSR	Pushes the contents of the status register onto the stack and then reloads the status register with the immediate data value.

3.4 Exception Processing Overview

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

- A simplified exception vector table
- Reduced relocation capabilities using the vector base register
- A single exception stack frame format
- Use of a single self-aligning system stack

All ColdFire processors use an instruction restart exception model, but certain microarchitectures (V2 and V3) require more software support to recover from certain access errors. See [Section 3.6.1, “Access Error Exception”](#) for details.

Exception processing includes all actions from the detection of the fault condition to the initiation of fetch for the first handler instruction. Exception processing is comprised of these four major steps:

1. First, the processor makes an internal copy of the SR and enters supervisor mode by setting the S bit and disabling trace mode by negating the T bit. The occurrence of an interrupt exception also forces the M bit to be cleared and the interrupt priority mask to be set to the level of the current interrupt request.
2. Second, the processor determines the exception vector number. For all faults except interrupts, the processor performs this calculation based on the 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 a special acknowledge address space with the interrupt level encoded in the address.
3. Third, the processor saves the current context by creating an exception stack frame on the supervisor system stack. As a result, the exception stack frame is created at a 0-modulo-4 address on the top of the current system stack. Additionally, the processor uses a simplified fixed-length stack frame for all exceptions. 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. Fourth, the processor calculates the address of the first instruction of the exception handler. By definition, the exception vector table is aligned on a 1-Mbyte 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 x vector number). Once the exception vector has been fetched, the contents of the vector determine the address of the first

instruction of the desired handler. After the instruction fetch for the first opcode of the handler has been 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 by Freescale and the remaining 192 are user-defined interrupt vectors.

Table 3-5. Exception Vector Assignments

Vector Number(s)	Vector Offset (Hex)	Stacked Program Counter	Assignment
0	0x000	—	Initial 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	Debug interrupt
13	0x034	—	Reserved
14	0x038	Fault	Format error
15–23	0x03C–0x05C	—	Reserved
24	0x060	Next	Spurious interrupt
25–31	0x064–0x07C	—	Reserved
32–47	0x080–0x0BC	Next	Trap # 0–15 instructions
48–63	0x0C0–0x0FC	—	Reserved
64–255	0x100–0x3FC	Next	User-defined interrupts

'Fault' refers to the PC of the instruction that caused the exception; "Next" refers to the PC of the next 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 effectively disable interrupts, if necessary, by raising the interrupt mask level contained in the status register. In addition, the V2 core includes a new instruction (STLDSR) that stores the current interrupt mask level and loads a value into the SR. This instruction is specifically intended for use as the first instruction of an interrupt service routine that services multiple interrupt requests with different interrupt levels. For more details, see the *ColdFire Family Programmer's Reference Manual*.

3.5 Exception Stack Frame Definition

The exception stack frame is shown in [Figure 3-10](#). The first longword of the exception stack frame 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:

1. 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	4
01	Original SSP - 9	5
10	Original SSP - 10	6
11	Original SSP - 11	7

2. 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 types of exceptions. See [Table 3-7](#).

Table 3-7. Fault Status Encodings

FS[3:0]	Definition
00xx	Reserved
0100	Error on instruction fetch
0101	Reserved
011x	Reserved
1000	Error on operand write
1001	Attempted write to write-protected space
101x	Reserved
1100	Error on operand read
1101	Reserved
111x	Reserved

3. 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 the case of an interrupt. Refer to [Table 3-5](#).

3.6 Processor Exceptions

3.6.1 Access Error Exception

The exact processor response to an access error depends on the type of 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 that occur during instruction prefetches that are then 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 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. In this situation, any address register updates attributable to the auto-addressing modes, (for example, (An)+, -(An)), have already been performed, so the programming model contains the updated An value. In addition, if an access error occurs during the execution of a MOVEM instruction loading from memory, any registers already updated *before* the fault occurs contain the operands from memory.

The V2 ColdFire processor uses an imprecise reporting mechanism for access errors on operand writes. Because the actual write cycle may be decoupled from the processor's issuing of the operation, the signaling of an access error appears to be decoupled from the instruction that generated the write. Accordingly, the PC contained in the exception stack frame merely represents the location in the program when the access error was signaled. All programming model updates associated with the write instruction are completed. The NOP instruction can collect access errors for writes. This instruction delays its execution until all previous operations, including all pending write operations, are complete. If any previous write terminates with an access error, it is guaranteed to be reported on the NOP instruction.

3.6.2 Address Error Exception

Any attempted execution transferring control to an odd instruction address (that is, 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 8 on an indexed effective addressing mode generates an address error, as does an attempted execution of a full-format indexed addressing mode.

3.6.3 Illegal Instruction Exception

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 opcode generates their unique exception types, vector numbers 10 and 11, respectively. The

V2 core does not provide illegal instruction detection on the extension words on any instruction, including MOVEC.

3.6.4 Divide-By-Zero

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

3.6.5 Privilege Violation

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

3.6.6 Trace Exception

To aid in program development, all ColdFire processors provide an instruction-by-instruction tracing capability. While in trace mode, indicated by the assertion of the T-bit in the status register ($SR[15] = 1$), 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 is executed, 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 the execution of a TRAP instruction while in trace mode. The processor will initiate the TRAP exception and then pass 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 ($SR[15]$ in the exception stack frame asserted) and pass control to the trace handler before returning from the original exception.

3.6.7 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.

3.6.8 Unimplemented Line-F Opcode

A line-F opcode is defined when bits 15-12 of the opword are 0b1111. This exception is generated by attempted execution of an undefined line-F opcode.

3.6.9 Debug Interrupt

This special type of program interrupt is discussed in detail in [Chapter 26, “Debug Module.”](#) This exception is generated in response to a hardware breakpoint register trigger. The processor does not generate an IACK cycle, but rather calculates the vector number internally (vector number 12).

3.6.10 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
4. Transfers control to the instruction address defined by the second longword operand within the stack frame

3.6.11 TRAP Instruction Exception

The TRAP #n instruction always forces an exception as part of its execution and is useful for implementing system calls.

3.6.12 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 14, “Interrupt Controller Module”](#) for details on the interrupt controller.

3.6.13 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 force the processor to exit this halted state.

3.6.14 Reset Exception

Asserting the reset input signal 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 S-bit and disables tracing by clearing the T bit in the SR. This exception also clears the M-bit and sets the processor's interrupt priority mask in the SR to the highest level (level 7). Next, the VBR is initialized to zero (0x00000000). 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 supervisor registers are also affected. Refer to each of the modules in this reference manual for details on these registers.

Once the processor is granted the bus, it then performs two longword read bus cycles. The first longword at address 0 is loaded into the stack pointer, and the second longword at address 4 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 halted state.

3.6.15 Reset Vector

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 BDM to determine the hardware configuration.

Information loaded into D0 defines the processor hardware configuration as shown in [Figure 3-11](#).

Address: D0

Access: User read-only

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	PF								VER				REV			
W																
Reset	1	1	0	0	1	1	1	1	0	0	1	0	0	0	0	0
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MAC	DIV	EMAC	FPU	MMU	0	0	0	ISA				DEBUG			
W																
Reset	0	1	1	0	0	0	0	0	1	0	0	0	1	0	0	1

Figure 3-11. D0 Hardware Configuration Info

Table 3-8. 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. This field is fixed to a hex value of 0x2 indicating a Version 2 ColdFire core.
19–16 REV	Processor revision number.
15 MAC	MAC execute engine status. 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 execute engine status. 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 execute engine status. 0 EMAC execute engine not present in core. 1 EMAC execute engine is present in core. (This is the value used for this device.)
12 FPU	FPU execute engine status. 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	Virtual memory management unit status. 0 MMU execute engine not present in core. (This is the value used for this device.) 1 MMU execute engine is present in core.
10–8	Reserved.

Table 3-8. D0 Hardware Configuration Info Field Description (continued)

Field	Description
7–4 ISA	Instruction set architecture (ISA) revision number. 0000 ISA_A 0001 ISA_B 0010 ISA_C 1000 ISA_A+ (ISA_A with the addition of the BYTEREV, BITREV, FF1, and STLDSR instructions. This is the value used for this device.) Else Reserved.
3–0 DEBUG	Debug module revision number. 0000 DEBUG_A 0001 DEBUG_B 0010 DEBUG_C 0011 DEBUG_D 0100 DEBUG_E 1001 DEBUG_B+ (This is the value used for this device) Else Reserved.

Information loaded into D1 defines the local memory hardware configuration as shown in [Figure 3-12](#).

Address: D1												Access: User read-only							
R												ICSIZ							
W																			
Reset	0	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
R												DCSIZ							
W																			
Reset	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0
R												RAM1SIZ							
W																			
Reset	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0

Figure 3-12. D1 Hardware Configuration Info**Table 3-9. D1 Local Memory Hardware Configuration Information Field Description**

Field	Description
31–30 CL	Cache line size. This field is fixed to a hex value of 0x0 indicating a 16-byte cache line size.
29–28 ICA	Instruction cache associativity. 00 Four-way. 01 Direct mapped. (This is the value used for this device)
27–24 ICSIZ	Instruction cache size. 0101 8KB instruction cache. (This is the value used for this device) Else Reserved.
23–16	Reserved, should be cleared.
15–14 BUSW	Encoded bus data width. 00 32-bit data bus (only configuration currently in use).

Table 3-9. D1 Local Memory Hardware Configuration Information Field Description (continued)

Field	Description
13–12 DCA	Data cache associativity. 00 Four-way. 01 Direct mapped. (This is the value used for this device)
11–8 DCSIZ	Data cache size. 0000 No data cache. (This is the value used for this device) Else Reserved.
7–4 RAM1SIZ	RAM bank 1 size. 0110 16KB RAM. (This is the value used for this device) Else Reserved.
3–0	Reserved, should be cleared.

3.7 Instruction Execution Timing

This section presents V2 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.

3.7.1 Timing Assumptions

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

1. The operand execution pipeline (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 instruction fetch pipeline (IFP) to supply opwords and/or extension words.
2. The OEP does not experience any sequence-related pipeline stalls. For V2 ColdFire processors, the most common example of this type of stall involves consecutive store operations, excluding the MOVEM instruction. For all STORE operations (except MOVEM), certain hardware resources within the processor are marked as busy for two processor clock cycles after the final DSOC cycle of the store instruction. If a subsequent STORE instruction is encountered within this 2-cycle window, it will be stalled until the resource again becomes available. Thus, the maximum pipeline stall involving consecutive STORE operations is 2 cycles. The MOVEM instruction uses a different set of resources and this stall does not apply.
3. 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.
4. All operand data accesses are aligned on the same byte boundary as the operand size: that is, 16 bit operands aligned on 0-modulo-2 addresses and 32-bit operands aligned on 0-modulo-4 addresses.

If the operand alignment fails these guidelines, it is misaligned. The processor core decomposes the misaligned operand reference into a series of aligned accesses, as shown in [Table 3-10](#).

Table 3-10. Misaligned Operand References

Address[1:0]	Size	Kbus Operations	Additional C(R/W)
X1	Word	Byte, Byte	2(1/0) if read 1(0/1) if write
X1	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

3.7.2 MOVE Instruction Execution Times

The execution times for the MOVE.{B,W} instructions are shown in [Table 3-11](#), while [Table 3-12](#) provides the timing for MOVE.L.

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.

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

Table 3-11. Move Byte and Word Execution Times

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d ₁₆ ,Ax)	(d ₈ ,Ax,Xi)	(xxx).wl
Dn	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
An	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
(An)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)
(An)+	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)
-(An)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)
(d ₁₆ ,An)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	—	—
(d ₈ ,An,Xi)	4(1/0)	4(1/1)	4(1/1)	4(1/1)	—	—	—
(xxx).w	3(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
(xxx).l	3(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
(d ₁₆ ,PC)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	—	—
(d ₈ ,PC,Xi)	4(1/0)	4(1/1)	4(1/1)	4(1/1)	—	—	—
#<xxx>	1(0/0)	3(0/1)	3(0/1)	3(0/1)	—	—	—

Table 3-12. Move Long Execution Times

Source	Destination						
	Rx	(Ax)	(Ax)+	-(Ax)	(d ₁₆ ,Ax)	(d ₈ ,Ax,Xi)	(xxx).wl
Dn	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
An	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)
(An)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
(An)+	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
- (An)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
(d ₁₆ ,An)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—
(d ₈ ,An,Xi)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
(xxx).w	2(1/0)	2(1/1)	2(1/1)	2(1/1)	—	—	—
(xxx).l	2(1/0)	2(1/1)	2(1/1)	2(1/1)	—	—	—
(d ₁₆ ,PC)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—
(d ₈ ,PC,Xi)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
#<xxx>	1(0/0)	2(0/1)	2(0/1)	2(0/1)	—	—	—

3.8 Standard One Operand Instruction Execution Times

Table 3-13. One Operand Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d ₁₆ ,An)	(d ₈ ,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)	—	—	—	—	—	—	—
scc	Dx	1(0/0)	—	—	—	—	—	—	—
stldsr	#imm	—	—	—	—	—	—	—	5(0/1)

Table 3-13. One Operand Instruction Execution Times (continued)

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
swap	Dx	1(0/0)	—	—	—	—	—	—	—
tst.b	<ea>	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
tst.w	<ea>	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
tst.l	<ea>	1(0/0)	2(1/0)	2(1/0)	2(1/0)	2(1/0)	3(1/0)	2(1/0)	1(0/0)

3.9 Standard Two Operand Instruction Execution Times

Table 3-14. 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)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
add.l	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
addi.l	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
addq.l	#imm,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
addir.l	Dy,Dx	1(0/0)	—	—	—	—	—	—	—
and.l	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
and.l	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
andi.l	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
asl.l	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
asrl.l	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
bchg	Dy,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1)	4(1/1)	—
bchg	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	—	—	—
bclr	Dy,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1)	4(1/1)	—
bclr	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	—	—	—
bset	Dy,<ea>	2(0/0)	4(1/1)	41/1	4(1/1)	4(1/1)	5(1/1)	4(1/1)	—
bset	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	—	—	—
btst	Dy,<ea>	2(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
btst	#imm,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	—	—	1(0/0)
cmp.l	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
cmpl.l	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
divs.w ¹	<ea>,Dx	20(0/0)	23(1/0)	23(1/0)	23(1/0)	23(1/0)	24(1/0)	23(1/0)	20(0/0)
divu.w ¹	<ea>,Dx	20(0/0)	23(1/0)	23(1/0)	23(1/0)	23(1/0)	24(1/0)	23(1/0)	20(0/0)
divs.l ¹	<ea>,Dx	≤35(0/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	—	—	—
divu.l ¹	<ea>,Dx	≤35(0/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	—	—	—

Table 3-14. 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
eor.l	Dy,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(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	#imm,Dx	—	—	—	—	—	—	—	1(0/0)
muls.w	<ea>y, Dx	4(0/0)	6(1/0)	6(1/0)	6(1/0)	6(1/0)	7(1/0)	6(1/0)	4(1/0)
mulu.w	<ea>y, Dx	4(0/0)	6(1/0)	6(1/0)	6(1/0)	6(1/0)	7(1/0)	6(1/0)	4(1/0)
muls.l	<ea>y, Dx	4(0/0)	6(1/0)	6(1/0)	6(1/0)	6(1/0)	—	—	—
mulu.l	<ea>y, Dx	4(0/0)	6(1/0)	6(1/0)	6(1/0)	6(1/0)	—	—	—
or.l	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
or.l	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
ori.l	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
rems.l ¹	<ea>,Dx	≤35(0/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	—	—	—
remu.l ¹	<ea>,Dx	≤35(0/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	≤38(1/0)	—	—	—
sub.l	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
sub.l	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
subi.l	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
subq.l	#imm,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
subx.l	Dy,Dx	1(0/0)	—	—	—	—	—	—	—

¹ For divide and remainder instructions the times listed represent the worst-case timing. Depending on the operand values, the actual execution time may be less.

3.10 Miscellaneous Instruction Execution Times

Table 3-15. Miscellaneous Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
link.w	Ay,#imm	2(0/1)	—	—	—	—	—	—	—
move.w	CCR,Dx	1(0/0)	—	—	—	—	—	—	—
move.w	<ea>,CC R	1(0/0)	—	—	—	—	—	—	1(0/0)
move.w	SR,Dx	1(0/0)	—	—	—	—	—	—	—
move.w	<ea>,SR	7(0/0)	—	—	—	—	—	—	7(0/0) ²
movec	Ry,Rc	9(0/1)	—	—	—	—	—	—	—

Table 3-15. Miscellaneous Instruction Execution Times (continued)

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
movem.l	<ea>,&list	—	1+n(n/0)	—	—	1+n(n/0)	—	—	—
movem.l	&list,<ea>	—	1+n(0/n)	—	—	1+n(0/n)	—	—	—
nop		3(0/0)	—	—	—	—	—	—	—
pea	<ea>	—	2(0/1)	—	—	2(0/1) ⁴	3(0/1) ⁵	2(0/1)	—
pulse		1(0/0)	—	—	—	—	—	—	—
stop	#imm	—	—	—	—	—	—	—	3(0/0) ³
trap	#imm	—	—	—	—	—	—	—	15(1/2)
trapf		1(0/0)	—	—	—	—	—	—	—
trapf.w		1(0/0)	—	—	—	—	—	—	—
trapf.l		1(0/0)	—	—	—	—	—	—	—
unlk	Ax	2(1/0)	—	—	—	—	—	—	—
wddata	<ea>	—	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	3(1/0)
wdebug	<ea>	—	5(2/0)	—	—	5(2/0)	—	—	—

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

3.11 EMAC Instruction Execution Times

Table 3-16. EMAC Instruction Execution Times

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,X n*SF)	xxx.wl	#xxx
muls.w	<ea>y, Dx	4(0/0)	6(1/0)	6(1/0)	6(1/0)	6(1/0)	7(1/0)	6(1/0)	4(1/0)
mulu.w	<ea>y, Dx	4(0/0)	6(1/0)	6(1/0)	6(1/0)	6(1/0)	7(1/0)	6(1/0)	4(1/0)
muls.l	<ea>y, Dx	4(0/0)	6(1/0)	6(1/0)	6(1/0)	6(1/0)	—	—	—
mulu.l	<ea>y, Dx	4(0/0)	6(1/0)	6(1/0)	6(1/0)	6(1/0)	—	—	—
mac.w	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—
mac.l	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—
msac.w	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—
msac.l	Ry, Rx, Raccx	1(0/0)	—	—	—	—	—	—	—
mac.w	Ry, Rx, <ea>, Rw, Raccx	—	2(1/0)	2(1/0)	2(1/0)	2(1/0) ¹	—	—	—

Table 3-16. EMAC Instruction Execution Times (continued)

Opcode	<EA>	Effective Address							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,X n*SF)	xxx.wl	#xxx
mac.l	Ry, Rx, <ea>, Rw, Raccx	—	2(1/0)	2(1/0)	2(1/0)	2(1/0) ¹	—	—	—
msac.w	Ry, Rx, <ea>, Rw	—	2(1/0)	2(1/0)	2(1/0)	2(1/0) ¹	—	—	—
msac.l	Ry, Rx, <ea>, Rw, Raccx	—	2(1/0)	2(1/0)	2(1/0)	2(1/0) ¹	—	—	—
mov.l	<ea>y, Raccx	1(0/0)	—	—	—	—	—	—	1(0/0)
mov.l	Raccy, Raccx	1(0/0)	—	—	—	—	—	—	—
mov.l	<ea>y, MACSR	5(0/0)	—	—	—	—	—	—	5(0/0)
mov.l	<ea>y, Rmask	4(0/0)	—	—	—	—	—	—	4(0/0)
mov.l	<ea>y, Racceext01	1(0/0)	—	—	—	—	—	—	1(0/0)
mov.l	<ea>y, Racceext23	1(0/0)	—	—	—	—	—	—	1(0/0)
mov.l	Raccx, <ea>x	1(0/0) ²	—	—	—	—	—	—	—
mov.l	MACSR, <ea>x	1(0/0)	—	—	—	—	—	—	—
mov.l	Rmask, <ea>x	1(0/0)	—	—	—	—	—	—	—
mov.l	Racceext01, <ea>x	1(0/0)	—	—	—	—	—	—	—
mov.l	Racceext23, <ea>x	1(0/0)	—	—	—	—	—	—	—

¹ Effective address of (d16,PC) not supported

² Storing an accumulator requires one additional processor clock cycle when saturation is enabled, or fractional rounding is performed (MACSR[7:4] = 1--, -11-, --11)

NOTE

The execution times for moving the contents of the Racc, Racceext[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.

3.12 Branch Instruction Execution Times

Table 3-17. 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
bsr		—	—	—	—	3(0/1)	—	—	—
jmp	<ea>	—	3(0/0)	—	—	3(0/0)	4(0/0)	3(0/0)	—
jsr	<ea>	—	3(0/1)	—	—	3(0/1)	4(0/1)	3(0/1)	—
rte		—	—	10(2/0)	—	—	—	—	—
rts		—	—	5(1/0)	—	—	—	—	—

Table 3-18. BRA, Bcc Instruction Execution Times

Opcode	Forward Taken	Forward Not Taken	Backward Taken	Backward Not Taken
bra	2(0/0)	—	2(0/0)	—
bcc	3(0/0)	1(0/0)	2(0/0)	3(0/0)

Chapter 4

Enhanced Multiply-Accumulate Unit (EMAC)

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

4.1 Multiply-Accumulate Unit

The MAC design provides a set of DSP operations that can be used to 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 multiplies
2. Multiply-accumulate operations supporting signed and unsigned integer operands as well as signed, fixed-point, 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 16×16 multiply array and a single 32-bit accumulator. The EMAC features a four-stage pipeline optimized for 32-bit operands, with a fully pipelined 32×32 multiply array and four 48-bit accumulators.

The first ColdFire MAC supported signed and unsigned integer operands and was optimized for 16×16 operations, such as those found in a variety of 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 32×32 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 the use of a 40-bit product, plus the addition of 8 extension bits to 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, as shown in [Figure 4-1](#).

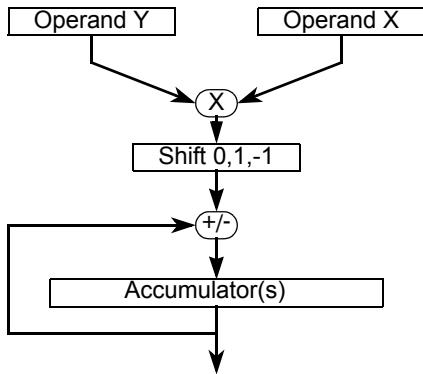


Figure 4-1. Multiply-Accumulate Functionality Diagram

4.2 Introduction to the MAC

The MAC is an extension of the basic multiplier found 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 the scope of any processor architecture and may require full DSP implementation.

To strike a balance between 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 also has been 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 4-1](#).

$$y(i) = \sum_{k=1}^{N-1} a(k)y(i-k) + \sum_{k=0}^{N-1} b(k)x(i-k) \quad \text{Eqn. 4-1}$$

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 the above equation to a simple, four-tap FIR filter, shown in [Equation 4-2](#), in which the accumulated sum is a sum of past data values and coefficients.

$$y(i) = \sum_{k=0}^3 b(k)x(i-k) = b(0)x(i) + b(1)x(i-1) + b(2)x(i-2) + b(3)x(i-3) \quad \text{Eqn. 4-2}$$

4.3 General Operation

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 of the following formats:

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

The EMAC is optimized for single-cycle, pipelined 32×32 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 either 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 4-2](#) and [Figure 4-3](#) show relative alignment of input operands, the full 64-bit product, the resulting 40-bit product used for accumulation, and 48-bit accumulator formats.

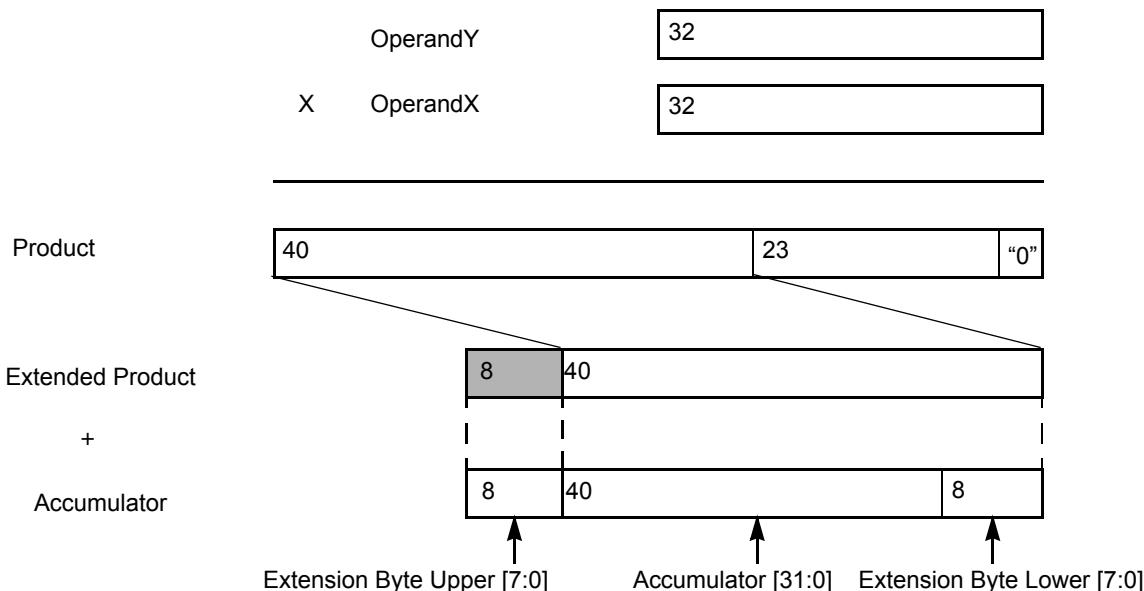


Figure 4-2. Fractional Alignment

Enhanced Multiply-Accumulate Unit (EMAC)

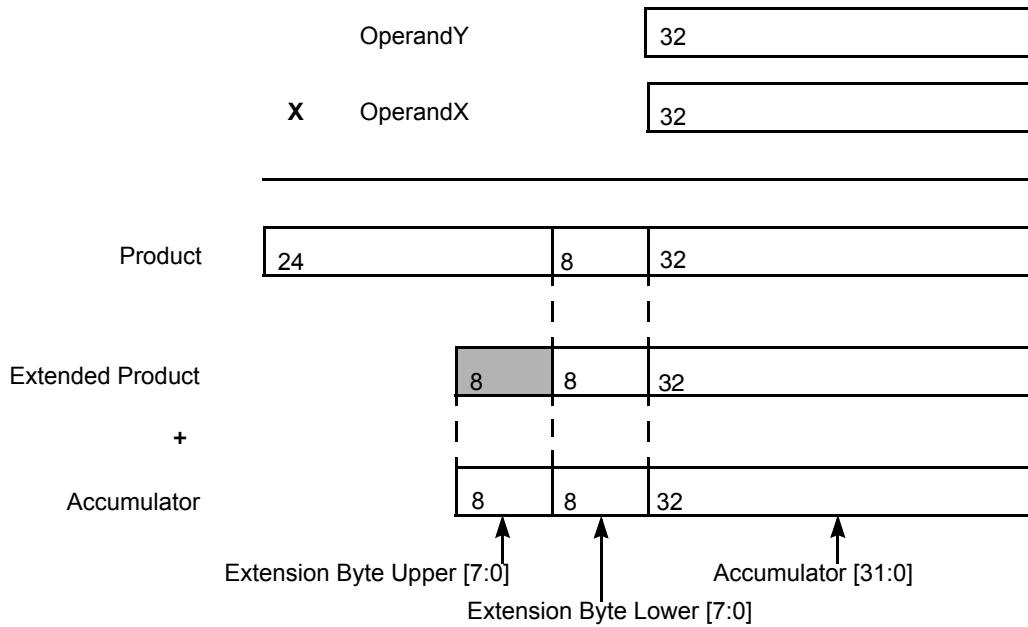


Figure 4-3. Signed and Unsigned Integer Alignment

Thus, 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 as follows:

```

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, ACC n , 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 internally in an accumulator. Thus, an additional move instruction is needed to store data in a general-purpose register. One new feature found 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 the word choice during the 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. New and existing ColdFire instructions can accommodate these requirements. A MOVEM instruction can move large blocks of data efficiently by generating line-sized burst references. The ability to simultaneously load an operand 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 application of this register with auto-increment addressing mode supports efficient implementation of circular data queues for memory operands.

4.4 Memory Map/Register Definition

The following table and sections explain the MAC registers:

Table 4-1. EMAC Memory Map

CPU Space (Rc)	Register	Width (bits)	Access	Reset Value	Section/Page
0x804	MAC Status Register (MACSR)	32	R/W	0x0000_0000	4.4.1/4-5
0x805	MAC Address Mask Register (MASK)	32	R/W	0xFFFF_FFFF	4.4.2/4-9
0x806, 0x809, 0x80A, 0x80B	MAC Accumulators 0–3 (ACC0–3)	32	R/W	Undefined	4.4.3/4-10
0x807	MAC Accumulator 0,1 Extension Bytes (ACCext01)	32	R/W	Undefined	4.4.4/4-10
0x808	MAC Accumulator 2,3 Extension Bytes (ACCext23)	32	R/W	Undefined	4.4.4/4-10

4.4.1 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.

Address: CPU @ 0x804

Access: User read/write

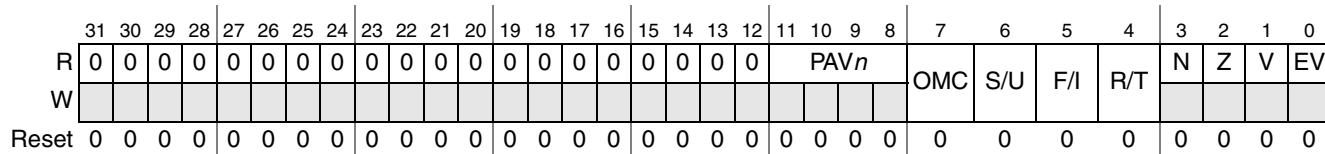


Figure 4-4. MAC Status Register (MACSR)

Table 4-2. MACSR Field Descriptions

Field	Description
31–12	Reserved, should 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 is used to form the general overflow flag, MACSR[V]. Once set, each flag remains set until V is cleared by a MOV.L, MACSR instruction or the accumulator is loaded directly.
7 OMC	Overflow/saturation mode. Used to enable or disable saturation mode on overflow. If set, the accumulator is set to the appropriate constant on any operation which overflows the accumulator. Once saturated, the accumulator remains unaffected by any other MAC or MSAC instructions until either the overflow bit is cleared or the accumulator is directly loaded.
6 S/U	<p>Signed/unsigned operations.</p> <p>In integer mode: S/U determines whether operations performed are signed or unsigned. It also determines the accumulator value during saturation, if enabled.</p> <ul style="list-style-type: none"> 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 both the instruction and the value of the product 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. <p>In fractional mode: S/U controls rounding while storing an accumulator to a general-purpose register.</p> <ul style="list-style-type: none"> 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 it is moved to a general-purpose register. See Section 4.4.1.1.1, “Rounding.” The resulting 16-bit value is stored in the lower word of the destination register. The upper word is zero-filled. The accumulator value is not affected by this rounding procedure.
5 F/I	Fractional/integer mode. Determines whether input operands are treated as fractions or integers.
4 R/T	<p>Round/truncate mode. Controls the rounding procedure for MOV.L ACCx,Rx, or MSAC.L instructions when operating in fractional mode.</p> <ul style="list-style-type: none"> 0 Truncate. The product's lsbs are dropped before it is combined with the accumulator. Additionally, when a store accumulator instruction is executed (MOV.L ACCx,Rx), the 8 lsbs of the 48-bit accumulator logic are simply 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 Section 4.4.1.1.1, “Rounding.” Additionally, when a store accumulator instruction is executed (MOV.L 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] = 0 and MACSR[R/T] = 1, the low-order 8 bits are used to round the resulting 32-bit fraction. If MACSR[S/U] = 1, the low-order 24 bits are used to round the resulting 16-bit fraction.
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.

Table 4-2. MACSR Field Descriptions (continued)

Field	Description
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 is still 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 4-3 summarizes the interaction of the MACSR[S/U,F/I,R/T] control bits.

Table 4-3. 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

4.4.1.1 Fractional Operation Mode

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

4.4.1.1.1 Rounding

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

- Execution of a store accumulator instruction (MOVL 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 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.
- 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 that is 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 = 1 and R0.L = 0x8000, the number is rounded up.
 - If the lsb of R0.U = 0 and R0.L = 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.

4.4.1.1.2 Saving and Restoring the EMAC Programming Model

The presence of rounding logic in the output datapath of the EMAC requires that special care be taken 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 following 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

```

The following code performs the EMAC state restore:

```

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

```

By executing this type of sequence, the exact state of the EMAC programming model can be correctly saved and restored.

4.4.1.1.3 MULS/MULU

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

4.4.1.1.4 Scale Factor in MAC or MSAC Instructions

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

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

This feature 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 use of MASK and causes bit 5 of the extension word to be set. The exact algorithm for the use of MASK is as follows:

Enhanced Multiply-Accumulate Unit (EMAC)

```

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 calculation of the updated An value is also shown.

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

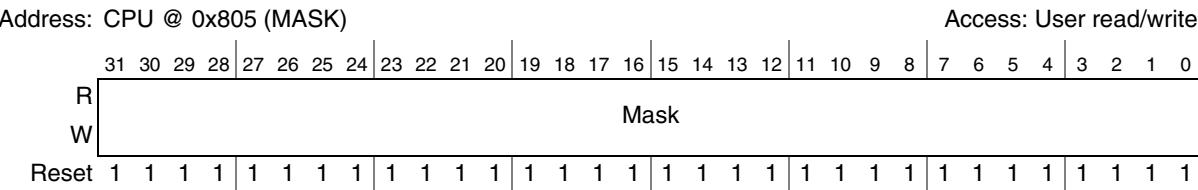


Figure 4-5. Mask Register (MASK)

4.4.3 Accumulator Registers (ACC0–3)

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

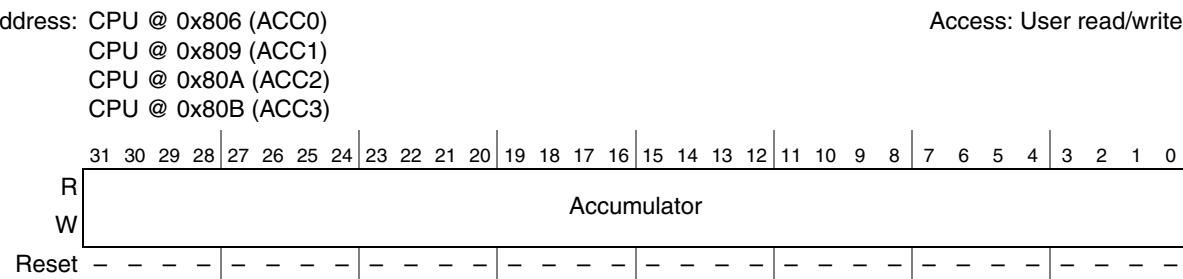


Figure 4-6. Accumulator Registers (ACC0–3)

4.4.4 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 on their use, see [Section 4.3, “General Operation.”](#)

Address: CPU @ 0x807 (ACCext01)																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	Accumulator 0 Upper Extension Byte				Accumulator 0 Lower Extension Byte				Accumulator 1 Upper Extension Byte				Accumulator 1 Lower Extension Byte																			
W																																
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-			

Figure 4-7. Accumulator Extension Register (ACCext01)

Address: CPU @ 0x808 (ACCext23)																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	Accumulator 2 Upper Extension Byte				Accumulator 2 Lower Extension Byte				Accumulator 3 Upper Extension Byte				Accumulator 3 Lower Extension Byte																			
W																																
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-				

Figure 4-8. Accumulator Extension Register (ACCext23)

4.5 EMAC Instruction Set Summary

Table 4-4 summarizes EMAC unit instructions.

Table 4-4. 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	MOV.L {Ry,#imm},ACCx	Loads an accumulator with a 32-bit operand
Store Accumulator	MOV.L ACCx,Rx	Writes the contents of an accumulator to a CPU register
Copy Accumulator	MOV.L ACCy,ACCx	Copies a 48-bit accumulator
Load MACSR	MOV.L {Ry,#imm},MACSR	Writes a value to MACSR
Store MACSR	MOV.L MACSR,Rx	Write the contents of MACSR to a CPU register
Store MACSR to CCR	MOV.L MACSR,CCR	Write the contents of MACSR to the CCR
Load MAC Mask Reg	MOV.L {Ry,#imm},MASK	Writes a value to the MASK register
Store MAC Mask Reg	MOV.L MASK,Rx	Writes the contents of the MASK to a CPU register
Load AccExtensions01	MOV.L {Ry,#imm},ACCext01	Loads the accumulator 0,1 extension bytes with a 32-bit operand
Load AccExtensions23	MOV.L {Ry,#imm},ACCext23	Loads the accumulator 2,3 extension bytes with a 32-bit operand

Table 4-4. EMAC Instruction Summary (continued)

Command	Mnemonic	Description
Store AccExtensions01	MOV.L ACCext01,Rx	Writes the contents of accumulator 0,1 extension bytes into a CPU register
Store AccExtensions23	MOV.L ACCext23,Rx	Writes the contents of accumulator 2,3 extension bytes into a CPU register

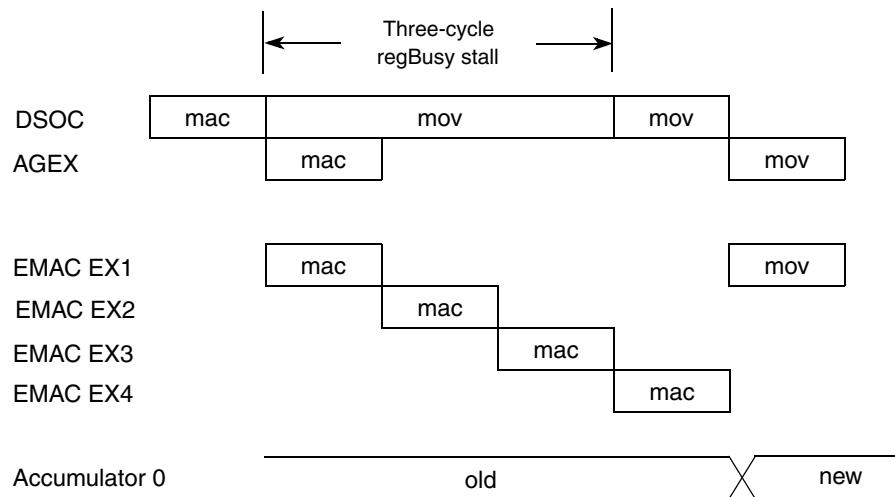
4.5.1 EMAC Instruction Execution Times

The instruction execution times for the EMAC can be found in [Section 3.11, “EMAC Instruction Execution Times.”](#)

The EMAC execution pipeline overlaps the AGEX stage of the OEP; that is, 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, as in the following:

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

The mov.l instruction that stores the accumulator to an integer register (Rz) stalls until the program-visible copy of the accumulator is available. [Figure 4-9](#) shows EMAC timing.

**Figure 4-9. EMAC-Specific OEP Sequence Stall**

In [Figure 4-9](#), the OEP stalls the store-accumulator instruction for 3 cycles: the depth of the EMAC pipeline minus 1. The minus 1 factor is needed because the OEP and EMAC pipelines overlap by a cycle, the AGEX stage. As the store-accumulator instruction reaches the AGEX stage where the operation is performed, the just-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. In fact, a major benefit of the EMAC is the addition of three accumulators to minimize stalls caused by exchanges between the accumulator(s) and the general-purpose registers.

4.5.2 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 4-3](#).

$$\text{value} = -(1 \cdot a_{N-1}) + \sum_{i=0}^{N-2} 2^{(i+1-N)} \cdot a_i \quad \text{Eqn. 4-3}$$

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 $(1 - 2^{-31})$.

4.5.3 MAC Opcodes

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

Note 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 handled differently. It is set if the complete product cannot be represented as a 40-bit value (this applies to 32×32 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 4.4.1, “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, it is expected that assemblers support this syntax and that no explicit reference to an accumulator is interpreted as a reference to ACC0. These assemblers would 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
                            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)

```

Enhanced Multiply-Accumulate Unit (EMAC)

```
        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 {
        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 */

```

Enhanced Multiply-Accumulate Unit (EMAC)

```
        result[47:0] = 0xfffff_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 5

Cache

5.1 Introduction

This chapter describes cache operation on the ColdFire processor.

5.1.1 Features

Features include the following:

- Configurable as instruction, data, or split instruction/data cache
- 8-Kbyte direct-mapped cache
- Single-cycle access on cache hits
- Physically located on the ColdFire core's high-speed local bus
- Nonblocking design to maximize performance
- Separate instruction and data 16-Byte line-fill buffers
- Configurable instruction cache miss-fetch algorithm

5.1.2 Physical Organization

The cache is a direct-mapped, single-cycle memory. It may be configured as an instruction cache, a write-through data cache, or a split instruction/data cache. The cache storage is organized as 512 lines, each containing 16 bytes. The memory storage consists of a 512-entry tag array (containing addresses and a valid bit), and a data array containing 8 Kbytes, organized as 2048×32 bits.

Cache configuration is controlled by bits in the cache control register (CACR), detailed later in this chapter. For the instruction or data-only configurations, only the associated instruction or data line-fill buffer is used. For the split cache configuration, one-half of the tag and storage arrays is used for an instruction cache and one-half is used for a data cache. The split cache configuration uses both the instruction and the data line-fill buffers. The core's local bus is a unified bus used for both instruction and data fetches. Therefore, the cache can have only one fetch, either instruction or data, active at one time.

For the instruction- or data-only configurations, the cache tag and storage arrays are accessed in parallel: fetch address bits [12:4] addressing the tag array, and fetch address bits [12:2] addressing the storage array. For the split cache configuration, the cache tag and storage arrays are accessed in parallel. The msb of the tag array address is set for instruction fetches and cleared for operand fetches; fetch address bits [11:4] provide the rest of the tag array address. The tag array outputs the address mapped to the given cache location along with the valid bit for the line. This address field is compared to bits [31:13] for instruction- or data-only configurations and to bits [31:12] for a split configuration of the fetch address from the local bus to determine if a cache hit has occurred. If the desired address is mapped into the cache memory, the

Cache

output of the storage array is driven onto the ColdFire core's local data bus, thereby completing the access in a single cycle.

The tag array maintains a single valid bit per line entry. Accordingly, only entire 16-byte lines are loaded into the cache.

The cache also contains separate 16-byte instruction and data line-fill buffers that provide temporary storage for the last line fetched in response to a cache miss. With each fetch, the contents of the associated line fill buffer are examined. Thus, each fetch address examines both the tag memory array and the associated line fill buffer to see if the desired address is mapped into either hardware resource. A cache hit in either the memory array or the associated line-fill buffer is serviced in a single cycle. Because the line fill buffer maintains valid bits on a longword basis, hits in the buffer can be serviced immediately without waiting for the entire line to be fetched.

If the referenced address is not contained in the memory array or the associated line-fill buffer, the cache initiates the required external fetch operation. In most situations, this is a 16-byte line-sized burst reference.

The hardware implementation is a nonblocking design, meaning the ColdFire core's local bus is released after the initial access of a miss. Thus, the cache or the SRAM module can service subsequent requests while the remainder of the line is being fetched and loaded into the fill buffer.

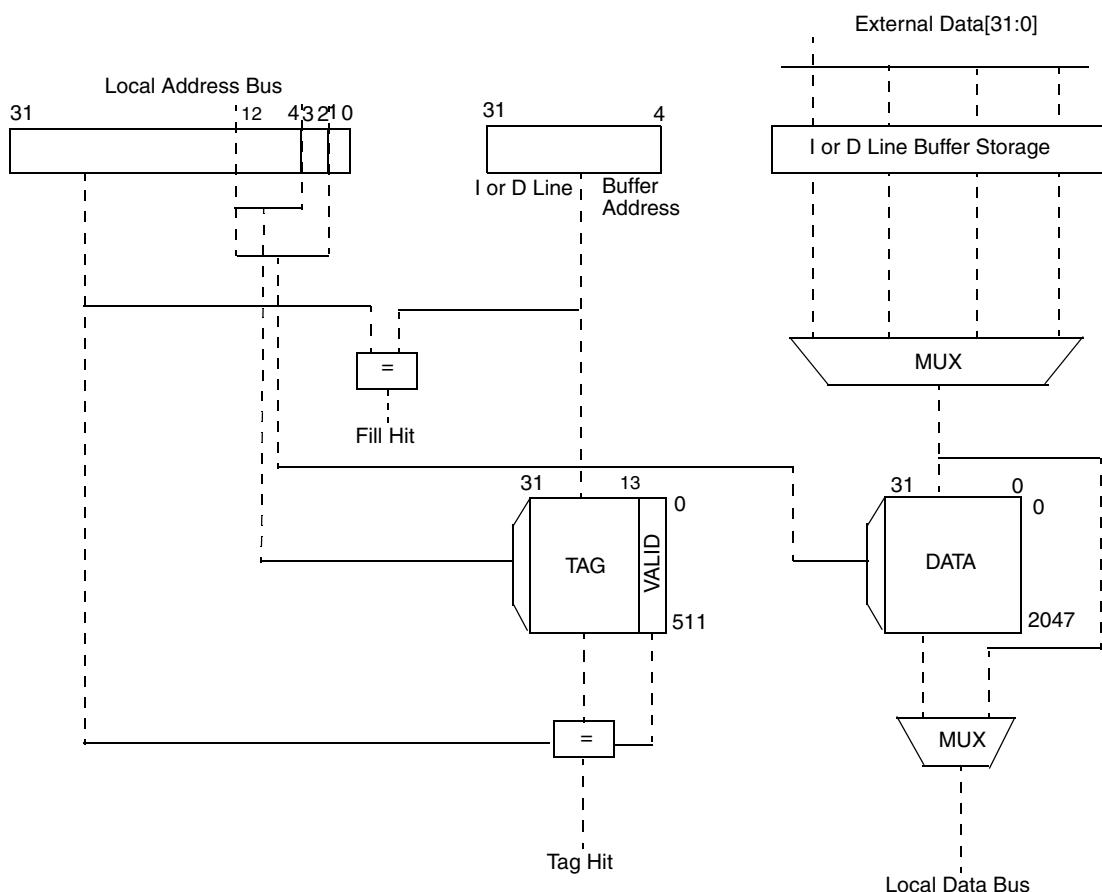


Figure 5-1. Cache Block Diagram

5.1.3 Operation

The cache is physically connected to the ColdFire core's local bus, allowing it to service all fetches from the ColdFire core and certain memory fetches initiated by the debug module. Typically, the debug module's memory references appear as supervisor data accesses but the unit can be programmed to generate user-mode accesses and/or instruction fetches. The cache processes any fetch access in the normal manner.

5.1.3.1 Interaction with Other Modules

Because both the cache and high-speed SRAM module are connected to the ColdFire core's local data bus, certain user-defined configurations can result in simultaneous fetch processing.

If the referenced address is mapped into the SRAM module, that module will service the request in a single cycle. In this case, data accessed from the cache is simply discarded and no external memory references are generated. If the address is not mapped into the SRAM space, the cache handles the request in the normal fashion.

5.1.3.2 Memory Reference Attributes

For every memory reference the ColdFire core or the debug module generates, a set of effective attributes is determined based on the address and the access control registers (ACRs). This set of attributes includes the cacheable/noncacheable definition, the precise/imprecise handling of operand write, and the write-protect capability.

In particular, each address is compared to the values programmed in the ACRs. If the address matches one of the ACR values, the access attributes from that ACR are applied to the reference. If the address does not match either ACR, then the default value defined in the cache control register (CACR) is used. 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 Effective Attributes = CACR default attributes
```

5.1.3.3 Cache Coherency and Invalidation

The cache does not monitor ColdFire core data references for accesses to cached instructions. Therefore, software must maintain instruction cache coherency by invalidating the appropriate cache entries after modifying code segments if instructions are cached.

The cache invalidation can be performed in several ways. For the instruction- or data-only configurations, setting CACR[CINV] forces the entire cache to be marked as invalid. The invalidation operation requires 512 cycles because the cache sequences through the entire tag array, clearing a single location each cycle. For the split configuration, CACR[INVI] and CACR[INVD] can be used in addition to CACR[CINV] to clear the entire cache, only the instruction half, or only the data half. Any subsequent fetch accesses are postponed until the invalidation sequence is complete.

The privileged CPUSHL instruction can invalidate a single cache line. When this instruction is executed, the cache entry defined by bits [12:4] of the source address register is invalidated, provided CACR[CPDI] is cleared. For the split data/instruction cache configuration, software directly controls bit 12 which selects whether an instruction cache or data cache line is being accessed.

These invalidation operations can be initiated from the ColdFire core or the debug module.

5.1.3.4 Reset

A hardware reset clears the CACR and disables the cache. The contents of the tag array are not affected by the reset. Accordingly, the system startup code must explicitly perform a cache invalidation by setting CACR[CINV] before the cache can be enabled.

5.1.3.5 Cache Miss Fetch Algorithm/Line Fills

As discussed in [Section 5.1.2, “Physical Organization,”](#) the cache hardware includes a 16-byte, line-fill buffer for providing temporary storage for the last fetched line.

With the cache enabled as defined by CACR[CENB], a cacheable fetch that misses in both the tag memory and the line-fill buffer generates an external fetch. For data misses, the size of the external fetch is always 16 bytes. For instruction misses, the size of the external fetch is determined by the value contained in the 2-bit CLNF field of the CACR and the miss address. [Table 5-1](#) shows the relationship between the CLNF bits, the miss address, and the size of the external fetch.

Table 5-1. Initial Fetch Offset vs. CLNF Bits

CLNF[1:0]	Longword Address Bits			
	00	01	10	11
00	Line	Line	Line	Longword
01	Line	Line	Longword	Longword
1X	Line	Line	Line	Line

Depending on the runtime characteristics of the application and the memory response speed, overall performance may be increased by programming the CLNF bits to values {00, 01}.

For all cases of a line-sized fetch, the critical longword defined by bits [3:2] of the miss address is accessed first followed by the remaining three longwords that are accessed by incrementing the longword address in a modulo-16 fashion as shown below:

```

if miss address[3:2] = 00
    fetch sequence = {0x0, 0x4, 0x8, 0xC}
if miss address[3:2] = 01
    fetch sequence = {0x4, 0x8, 0xC, 0x0}
if miss address[3:2] = 10
    fetch sequence = {0x8, 0xC, 0x0, 0x4}
if miss address[3:2] = 11
    fetch sequence = {0xC, 0x0, 0x4, 0x8}

```

Once an external fetch has been initiated and the data is loaded into the line-fill buffer, the cache maintains a special ‘most-recently-used’ indicator that tracks the contents of the associated line-fill buffer versus its corresponding cache location. At the time of the miss, the hardware indicator is set, marking the line-fill buffer as ‘most recently used.’ If a subsequent access occurs to the cache location defined by bits [12:4] (or bits [11:4] for split configurations of the fill buffer address), the data in the cache memory array is now most recently used, so the hardware indicator is cleared. In all cases, the indicator defines whether the contents of the line-fill buffer or the memory data array are most recently used. At the time of the next cache miss, the contents of the line-fill buffer are written into the memory array if the entire line is present, and the line-fill buffer data is still most recently used compared to the memory array.

Generally, longword references are used for sequential instruction fetches. If the processor branches to an odd word address, a word-sized instruction fetch is generated.

For instruction fetches, the fill buffer can also be used as temporary storage for line-sized bursts of non-cacheable references under control of CACR[CEIB]. With this bit set, a noncacheable instruction fetch is processed, as defined by [Table 5-2](#). For this condition, the line-fill buffer is loaded and subsequent references can hit in the buffer, but the data is never loaded into the memory array.

[Table 5-2](#) shows the relationship between CACR bits CENB and CEIB and the type of instruction fetch.

Table 5-2. Instruction Cache Operation as Defined by CACR

CACR [CENB]	CACR [CEIB]	Type of Instruction Fetch	Description
0	0	N/A	Cache is completely disabled; all instruction fetches are word or longword in size.
0	1	N/A	All instruction fetches are word or longword in size
1	X	Cacheable	Fetch size is defined by Table 5-1 and contents of the line-fill buffer can be written into the memory array
1	0	Noncacheable	All instruction fetches are word or longword in size, and not loaded into the line-fill buffer
1	1	Noncacheable	Instruction fetch size is defined by Table 5-1 and loaded into the line-fill buffer, but are never written into the memory array.

5.2 Memory Map/Register Definition

Three supervisor registers define the operation of the cache and local bus controller: the cache control register (CACR) and two access control registers (ACR0, ACR1). [Table 5-3](#) below shows the memory map of the cache and access control registers.

The following lists several keynotes regarding the programming model table:

- The CACR and ACRs can only be accessed in supervisor mode using the MOVEC instruction with an Rc value of 0x002, 0x004 and 0x005, respectively.
- Addresses not assigned to the registers and undefined register bits are reserved for future expansion. The user should write zeros to these reserved address spaces and read accesses will return zeros.

- The reset value column indicates the register initial value at reset. Certain registers may be uninitialized upon reset; that is, they may contain random values after reset.
- The access column indicates if the corresponding register allows both read/write functionality (R/W), read-only functionality (R), or write-only functionality (W). If a read access to a write-only register is attempted, zeros will be returned. If a write access to a read-only register is attempted, the access will be ignored and no write will occur.

Table 5-3. Cache Memory Map

CPU Space (Rc)	Register	Width (bits)	Access ¹	Reset Value	Section/Page
0x002	Cache Control Register (CACR)	32	W	0x0000_0000	5.2.1/5-6
0x004	Access Control Register 0 (ACR0)	32	W	Undefined	5.2.2/5-9
0x005	Access Control Register 1 (ACR1)	32	W	Undefined	5.2.2/5-9

¹ Readable through debug.

5.2.1 Cache Control Register (CACR)

The CACR controls the operation of the cache. The CACR provides a set of default memory access attributes used when a reference address does not map into the spaces defined by the ACRs.

The CACR is a 32-bit, write-only supervisor control register. It is accessed in the CPU address space via the MOVEC instruction with an Rc encoding of 0x002. The CACR can be read when in background debug mode (BDM). At system reset, the entire register is cleared.

Address: CPU @ 0x002 (CACR)

Access: Supervisor write-only
Debug read/write

31 30 29 28				27 26 25 24				23 22 21 20				19 18 17 16				
R																
W	CENB	0	0	CPD	CFRZ	0	0	CINV	DISI	DISD	INVI	INVD	0	0	0	0
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																
W	0	0	0	0	0	CEIB	DCM	DBWE	0	0	DWP	EUSP	0	0	CLNF	
Reset 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0																

Figure 5-2. Cache Control Register (CACR)**Table 5-4. CACR Field Descriptions**

Field	Description
31 CENB	Cache enable. The memory array of the cache is enabled only if CENB is asserted. This bit, along with the DISI (disable instruction caching) and DISD (disable data caching) bits, control the cache configuration. 0 Cache disabled 1 Cache enabled Table 5-5 describes cache configuration.
30–29	Reserved, should be cleared.

Table 5-4. CACR Field Descriptions (continued)

Field	Description
28 CPDI	Disable CPUSHL invalidation. When the privileged CPUSHL instruction is executed, the cache entry defined by bits [12:4] of the address is invalidated if CPDI = 0. If CPDI = 1, no operation is performed. 0 Enable invalidation 1 Disable invalidation
27 CFRZ	Cache freeze. This field allows the user to freeze the contents of the cache. When CFRZ is asserted line fetches can be initiated and loaded into the line-fill buffer, but a valid cache entry can not be overwritten. If a given cache location is invalid, the contents of the line-fill buffer can be written into the memory array while CFRZ is asserted. 0 Normal Operation 1 Freeze valid cache lines
26–25	Reserved, should be cleared.
24 CINV	Cache invalidate. The cache invalidate operation is not a function of the CENB state (that is, this operation is independent of the cache being enabled or disabled). Setting this bit forces the cache to invalidate all, half, or none of the tag array entries depending on the state of the DISI, DISD, INVI, and INVD bits. The invalidation process requires several cycles of overhead plus 512 machine cycles to clear all tag array entries and 64256 cycles to clear half of the tag array entries, with a single cache entry cleared per machine cycle. The state of this bit is always read as a zero. After a hardware reset, the cache must be invalidated before it is enabled. 0 No operation 1 Invalidate all cache locations Table 5-6 describes how to set the cache invalidate all bit.
23 DISI	Disable instruction caching. When set, this bit disables instruction caching. This bit, along with the CENB (cache enable) and DISD (disable data caching) bits, control the cache configuration. See the CENB definition for a detailed description. 0 Enable instruction caching 1 Disable instruction caching Table 5-5 describes cache configuration and Table 5-6 describes how to set the cache invalidate all bit.
22 DISD	Disable data caching. When set, this bit disables data caching. This bit, along with the CENB (cache enable) and DISI (disable instruction caching) bits, control the cache configuration. See the CENB definition for a detailed description. 0 Enable data caching 1 Disable data caching Table 5-5 describes cache configuration and Table 5-6 describes how to set the cache invalidate all bit.
21 INVI	CINV instruction cache only. This bit can not be set unless the cache configuration is split (both DISI and DISD cleared). For instruction or data cache configurations this bit is a don't-care. For the split cache configuration, this bit is part of the control for the invalidate all operation. See the CINV definition for a detailed description Table 5-6 describes how to set the cache invalidate all bit.
20 INVD	CINV data cache only. This bit can not be set unless the cache configuration is split (both DISI and DISD cleared). For instruction or data cache configurations this bit is a don't-care. For the split cache configuration, this bit is part of the control for the invalidate all operation. See the CINV definition for a detailed description Table 5-6 describes how to set the cache invalidate all bit.
19–11	Reserved, should be cleared.
10 CEIB	Cache enable noncacheable instruction bursting. Setting this bit enables the line-fill buffer to be loaded with burst transfers under control of CLNF[1:0] for noncacheable accesses. Noncacheable accesses are never written into the memory array. See Table 5-2 . 0 Disable burst fetches on noncacheable accesses 1 Enable burst fetches on noncacheable accesses

Table 5-4. CACR Field Descriptions (continued)

Field	Description
9 DCM	Default cache mode. This bit defines the default cache mode: 0 is cacheable, 1 is noncacheable. For more information on the selection of the effective memory attributes, see Section 5.1.3.2, "Memory Reference Attributes . 0 Caching enabled 1 Caching disabled
8 DBWE	Default buffered write enable. This bit defines the default value for enabling buffered writes. If DBWE = 0, the termination of an operand write cycle on the processor's local bus is delayed until the external bus cycle is completed. If DBWE = 1, the write cycle on the local bus is terminated immediately and the operation buffered in the bus controller. In this mode, operand write cycles are effectively decoupled between the processor's local bus and the external bus. Generally, enabled buffered writes provide higher system performance but recovery from access errors can be more difficult. For the ColdFire core, reporting access errors on operand writes is always imprecise and enabling buffered writes further decouples the write instruction and the signaling of the fault 0 Disable buffered writes 1 Enable buffered writes
7–6	Reserved, should be cleared.
5 DWP	Default write protection 0 Read and write accesses permitted 1 Only read accesses permitted
4 EUSP	Enable user stack pointer. See Section 3.2.9, "Supervisor/User Stack Pointers (A7 and OTHER_A7)" for more information on the dual stack pointer implementation. 0 Disable the processor's use of the User Stack Pointer 1 Enable the processor's use of the User Stack Pointer
3–2	Reserved, should be cleared.
1–0 CLNF	Cache line fill. These bits control the size of the memory request the cache issues to the bus controller for different initial instruction line access offsets. See Table 5-1 for external fetch size based on miss address and CLNF.

[Table 5-5](#) shows the relationship between CACR[CENB, DISI, & DISD] bits and the cache configuration.

Table 5-5. Cache Configuration as Defined by CACR

CACR [CENB]	CACR [DISI]	CACR [DISD]	Configuration	Description
0	x	x	N/A	Cache is completely disabled
1	0	0	Split Instruction/ Data Cache	4 KByte direct-mapped instruction cache (uses lower half of tag and storage arrays) and 4 KByte direct-mapped write-through data cache (uses upper half of tag and storage arrays)
1	0	1	Instruction Cache	8 KByte direct-mapped instruction cache (uses all of tag and storage arrays)
1	1	0	Data Cache	8 KByte direct-mapped write-through data cache (uses all of tag and storage arrays)

[Table 5-6](#) shows the relationship between CACR[DISI, DISD, INVI, & INVD] and setting the cache invalidate all bit.

Table 5-6. Cache Invalidate All as Defined by CACR

CACR [DISI]	CACR [DISD]	CACR [INVI]	CACR [INVD]	Configuration	Operation
0	0	0	0	Split Instruction/ Data Cache	Invalidate all entries in both 4 KByte instruction cache and 4 KByte data cache
0	0	0	1	Split Instruction/ Data Cache	Invalidate only 4 KByte data cache
0	0	1	0	Split Instruction Data Cache	Invalidate only 4 KByte instruction cache
0	0	1	1	Split Instruction/ Data Cache	No invalidate
1	0	x	x	Instruction Cache	Invalidate 8 KByte instruction cache
0	1	x	x	Data Cache	Invalidate 8 KByte data cache

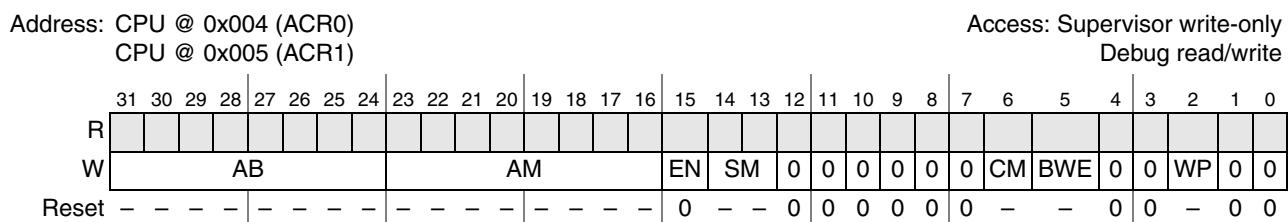
5.2.2 Access Control Registers (ACR0, ACR1)

The ACRs provide a definition of memory reference attributes for two memory regions (one per ACR). This set of effective attributes is defined for every memory reference using the ACRs or the set of default attributes contained in the CACR. The ACRs are examined for every processor memory reference that is not mapped to the SRAM memories.

The ACRs are 32-bit, write-only supervisor control register. They are accessed in the CPU address space via the MOVEC instruction with an Rc encoding of 0x004 and 0x005. The ACRs can be read when in background debug mode (BDM). At system reset, both registers are cleared.

NOTE

Peripheral space (0xE000_0000-0xFFFF_FFFF) cannot be cached. Ensure that ACR[AB] does not fall within this space.

**Figure 5-3. Access Control Registers (ACRn)****Table 5-7. ACRn Field Descriptions**

Field	Description
31–24 AB	Address base. This 8-bit field is compared to address bits [31:24] from the processor's local bus under control of the ACR address mask. If the address matches, the attributes for the memory reference are sourced from the given ACR.
23–16 AM	Address mask. This 8-bit field can mask any bit of the AB field comparison. If a bit in the AM field is set, then the corresponding bit of the address field comparison is ignored.

Table 5-7. ACRn Field Descriptions (continued)

Field	Description
15 EN	ACR Enable. The EN bit defines the ACR enable. Hardware reset clears this bit, disabling the ACR. 0 ACR disabled 1 ACR enabled
14–13 SM	Supervisor mode. This two-bit field allows the given ACR to be applied to references based on operating privilege mode of the ColdFire processor. The field uses the ACR for user references only, supervisor references only, or all accesses. 00 Match if user mode 01 Match if supervisor mode 1x Match always—ignore user/supervisor mode
12–7	Reserved, should be cleared.
6 CM	Cache mode. This bit defines the cache mode: 0 is cacheable, 1 is noncacheable. 0 Caching enabled 1 Caching disabled
5 BWE	Buffered write enable. This bit defines the value for enabling buffered writes. If BWE = 0, the termination of an operand write cycle on the processor's local bus is delayed until the external bus cycle is completed. If BWE = 1, the write cycle on the local bus is terminated immediately and the operation is then buffered in the bus controller. In this mode, operand write cycles are effectively decoupled between the processor's local bus and the external bus. Generally, the enabling of buffered writes provides higher system performance but recovery from access errors may be more difficult. For the V2 ColdFire core, the reporting of access errors on operand writes is always imprecise, and enabling buffered writes simply decouples the write instruction from the signaling of the fault even more. 0 Writes are not buffered. 1 Writes are buffered.
4–3	Reserved, should be cleared.
2 WP	Write protect. The WP bit defines the write-protection attribute. If the effective memory attributes for a given access select the WP bit, an access error terminates any attempted write with this bit set. 0 Read and write accesses permitted 1 Only read accesses permitted
1–0	Reserved, should be cleared.

Chapter 6

Static RAM (SRAM)

6.1 Introduction

This chapter is a description of the on-chip static RAM (SRAM) implementation that covers general operations, configuration, and initialization. It also provides information and examples showing how to minimize power consumption when using the SRAM.

6.1.1 Features

Features include the following:

- One 16-Kbyte SRAM
- Single-cycle access
- Physically located on processor's high-speed local bus
- Memory location programmable on any 0-modulo-16 Kbyte address
- Byte, word, and longword address capabilities

6.1.2 Operation

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-16K 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, instruction fetches may be sent to both the cache and the SRAM block simultaneously. If the reference is mapped 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 cross-bar switch. The SRAM is partitioned into two physical memory arrays to allow simultaneous access to both arrays by the processor core and another bus master. See [Chapter 12, “Cross-Bar Switch Module \(XBS\),”](#) for more information on arbitration between multiple masters accessing the SRAM.

6.2 Memory Map/Register Description

The SRAM programming model includes a description of the SRAM base address register (RAMBAR),

Static RAM (SRAM)

SRAM initialization, and power management.

Table 6-1. SRAM Programming Model

CPU Space (Rc)	Register	Width (bits)	Access	Reset Value	Written w/ MOVEC	Section/Page
Supervisor Access Only Registers						
0xC05	RAM Base Address Register 1 (RAMBAR1)	32	R/W	0x0000_0000	Yes	6.2.1/6-2

6.2.1 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 base address of the SRAM. 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.
- The RAMBAR valid bit is cleared by reset, disabling the SRAM module. All other bits are unaffected.

NOTE

The only applicable address ranges for the SRAM module's base address are 0x8000_0000–0x8FFF_C000. Be sure to set the RAMBAR register appropriately.

The RAMBAR contains several control fields. These fields are shown in [Figure 6-1](#).

Address: CPU @ 0x0C05

Access: Core 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	0	0	0	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																				0	0	PRIU	PRII	BDE	WP	0	BWP	C/I	SC	SD	UC	UD	V
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 6-1. SRAM Base Address Register (RAMBAR)

Table 6-2. RAMBAR Field Descriptions

Field	Description
31–14 BA	Base address. Defines the 0-modulo-16K base address of the SRAM module. By programming this field, the SRAM may be located on any 16-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_C000.
13–12	Reserved, should be cleared.

Table 6-2. RAMBAR Field Descriptions (continued)

Field	Description																	
11–10 PRIU PRIL	<p>Priority bit. PRIU determines if the SRAM backdoor or CPU has priority in the upper 8K bank of memory. PRIL determines if the SRAM backdoor or CPU has priority in the lower 8K 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:</p> <table border="1"> <thead> <tr> <th>PRIU,PRIL</th> <th>Upper Bank Priority</th> <th>Lower Bank Priority</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>SRAM Backdoor</td> <td>SRAM Backdoor</td> </tr> <tr> <td>01</td> <td>SRAM Backdoor</td> <td>CPU</td> </tr> <tr> <td>10</td> <td>CPU</td> <td>SRAM Backdoor</td> </tr> <tr> <td>11</td> <td>CPU</td> <td>CPU</td> </tr> </tbody> </table> <p>Note: The recommended setting (maximum performance) for the priority bits is 00.</p>			PRIU,PRIL	Upper Bank Priority	Lower Bank Priority	00	SRAM Backdoor	SRAM Backdoor	01	SRAM Backdoor	CPU	10	CPU	SRAM Backdoor	11	CPU	CPU
PRIU,PRIL	Upper Bank Priority	Lower Bank Priority																
00	SRAM Backdoor	SRAM Backdoor																
01	SRAM Backdoor	CPU																
10	CPU	SRAM Backdoor																
11	CPU	CPU																
9 BDE	<p>Backdoor enable. Allows access by non-core bus masters via the SRAM backdoor on the cross-bar switch</p> <ul style="list-style-type: none"> 0 Non-core cross-bar switch master access to memory is disabled. 1 Non-core cross-bar switch master access to memory is enabled. 																	
8 WP	<p>Write protect. Allows only read accesses to the SRAM. When this bit is set, any attempted write access from the core will generate an access error exception to the ColdFire processor core.</p> <ul style="list-style-type: none"> 0 Allows core read and write accesses to the SRAM module 1 Allows only core read accesses to the SRAM module <p>Note: Non-core write accesses are not affected by this bit.</p>																	
7	Reserved, should be cleared.																	
6 BWP	<p>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.</p> <ul style="list-style-type: none"> 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. 																	
5–1 C/I, SC, SD, UC, UD	<p>Address space masks (ASn)</p> <p>These five bit fields allow certain 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"> C/I = CPU space/interrupt acknowledge cycle mask SC = Supervisor code address space mask SD = Supervisor data address space mask UC = User code address space mask UD = User data address space mask <p>For each address space bit:</p> <ul style="list-style-type: none"> 0 An access to the SRAM module can occur for this address space 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. <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 Section 6.3.2, “Power Management.” 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"> 0 Processor accesses of the SRAM are masked. 1 Processor accesses of the SRAM are enabled. 																	

6.3 Initialization/Application Information

After a hardware reset, the contents of the SRAM module are undefined. The valid bit of the RAMBAR is cleared, disabling the module. If the SRAM requires initialization with instructions or data, the following steps should be performed:

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. There are various instructions to 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 has been loaded 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.

6.3.1 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 0x00000000          ;set this variable to 0x00000000
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       #4096,D0             ;load loop counter into D0 (SRAM size/4)

SRAM_INIT_LOOP:
clr.l        (A0)+                ;clear 4 bytes of SRAM
subq.l       #1,D0                ;decrement loop counter
bne.b        SRAM_INIT_LOOP      ;if done, then exit; else continue looping
```

6.3.2 Power Management

As noted previously, depending on the configuration defined by the RAMBAR, instruction fetch and operand read accesses may be sent to the SRAM and cache simultaneously. If the access is mapped to the SRAM module, it sources the read data and the unified cache access is discarded. If the SRAM is used only for data operands, setting the `ASn` bits associated with instruction fetches can decrease power dissipation. Additionally, if the SRAM contains only instructions, masking operand accesses can reduce power dissipation. [Table 6-3](#) shows some examples of typical RAMBAR settings.

Table 6-3. Typical RAMBAR Setting Examples

Data Contained in SRAM	RAMBAR[7:0]
Instruction Only	0x2B
Data Only	0x35
Both Instructions And Data	0x21

Chapter 7

Clock Module

7.1 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 either 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 be used to clock the device directly. The clock module contains:

- Crystal amplifier and oscillator (OSC)
- Dithering 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 7-1](#) is a high level representation of the clock connections. The exact functionality of the blocks is not illustrated (e.g. clocks to the SDRAMC are disabled when the device is in limp mode, and the clocks to individual modules may be disabled via the peripheral power management registers).

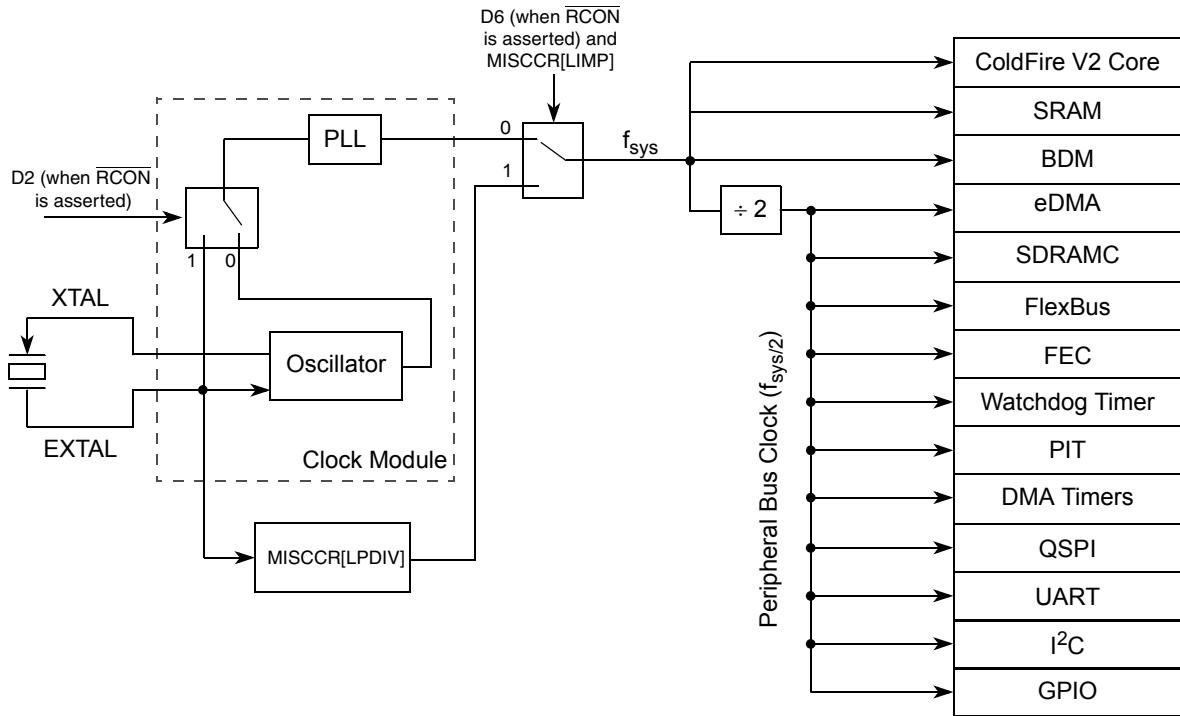


Figure 7-1. Device Clock Connections

7.1.1 Block Diagram

Figure 7-2 shows a block diagram of the clock module.

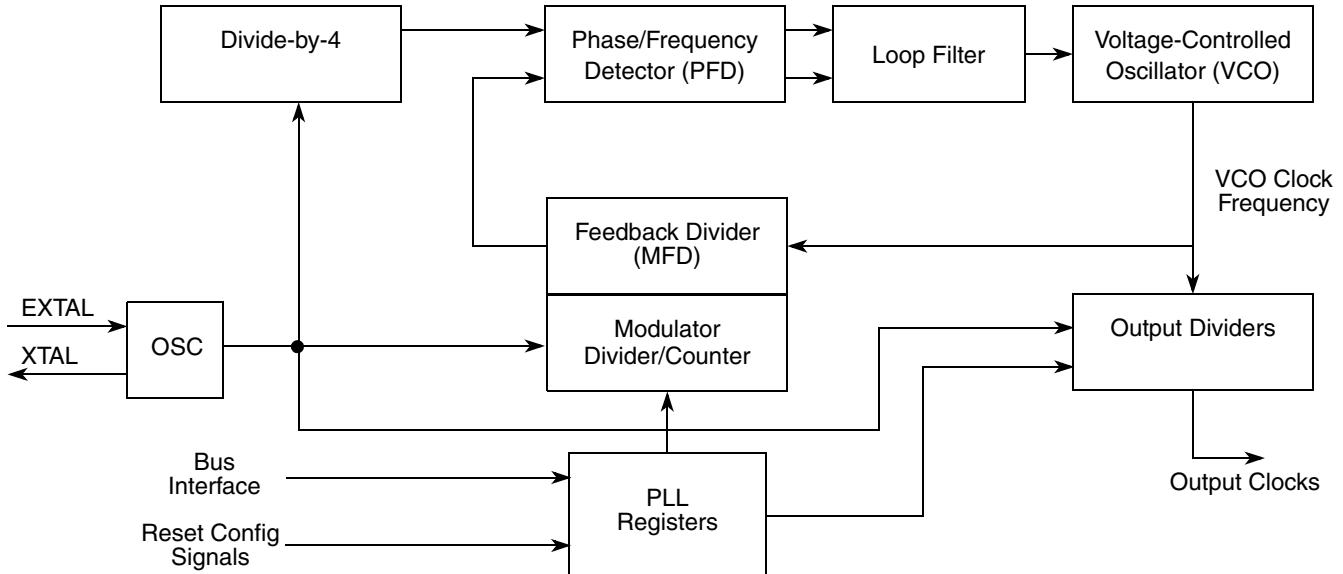


Figure 7-2. Clock Module Diagram

7.1.2 Features

Features of the clock module include the following:

- 16-MHz reference crystal oscillator
- Voltage controlled oscillator range from 350 MHz to 540 MHz, resulting in a core frequency ($f_{vco} \div 3$ (or $f_{vco} \div 4$)) of 87.5 MHz to 166.67 MHz (maximum rated for device)
- Programmable dithering
- Support for low-power modes
- Direct clocking of system by input clock, bypassing the PLL
- Loss-of-lock reset

7.1.3 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 9, “Chip Configuration Module \(CCM\).”](#)

The clock module can operate in normal PLL mode with crystal reference, normal PLL mode with external reference, and input clock limp mode.

7.1.3.1 Normal PLL Mode with Crystal Reference

In normal mode with a crystal reference, the PLL receives an input clock frequency from the crystal oscillator circuit and multiplies the frequency to create the PLL output clock. It can synthesize frequencies ranging from 22x to 33.75x the input frequency. The user must supply a crystal oscillator that is within the appropriate input frequency range, the crystal manufacturer’s recommended external support circuitry, and short signal route from the device to the crystal. In normal mode, the PLL can generate a dithered clock or a non-dithered clock (locked on a single frequency). The dithering deviation, dither modulation frequency, and whether the PLL is modulating or not can be programmed by writing to the PLL registers through the bus interface.

7.1.3.2 Normal PLL Mode with External Reference

Same as [Section 7.1.3.1, “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. 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 9, “Chip Configuration Module \(CCM\),”](#) for details on setting the device for external reference.

7.1.3.3 Input Clock (Limp) Mode

Through the use of RCON, the device may be booted 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 5-bit programmable counter that divides the input clock by 2^n , where n is the value of the programmable counter field, MISCCR[LPDIV]. For more information on programming the divider, see [Chapter 8,](#)

“Power Management.” 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. A 2:1 ratio is still maintained between the core and the primary bus clock. Because they will not function at speeds as low as the minimum input clock frequency, the SDRAM controller and FEC are not functional in limp mode.

When switching from LIMP mode to normal functional mode, the user must wait for the PLL to lock before continuing with code execution. The user must also ensure than any peripheral transactions in progress (e.g. Ethernet frame reception/transmission) are allowed to complete in order to avoid data loss or corruption.

Limp mode may also be entered and exited from by writing to the MISCCR[LIMP] bit. This is useful since it places the PLL in a state where the multiplication factor (PFMDR) can be altered. Entering limp mode also requires a special procedure with the SDRAM module. As noted above the SDRAM controller is disabled in limp mode, so two critical steps must be followed before setting the MISCCR[LIMP] bit.

1. Code execution must be transferred to another memory resource. Primary options are whatever 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 limp mode entrance or exit).
2. The SDRAM controller must be placed in self-refresh mode to avoid data loss while the SDRAMC is shut down.

7.1.3.4 Low-power Mode Operation

This subsection describes the operation of the clock module in low-power and halted modes of operation. Low-power modes are described in [Chapter 8, “Power Management.”](#) [Table 7-1](#) shows the clock module operation in low-power modes.

Table 7-1. Clock Module Operation in Low-power Modes

Low-power Mode	Clock Operation	Mode Exit
Wait	Clocks sent to peripheral modules only	Exit not caused by clock module, but normal clocking resumes upon mode exit
Doze	Clocks sent to peripheral modules only	Exit not caused by clock module, but normal clocking resumes upon mode exit
Stop	All system clocks disabled	Exit not caused by clock module, but clock sources are re-enabled and normal clocking resumes upon mode exit
Halted	Normal	Exit not caused by clock module

In wait and doze modes, the system clocks to the peripherals are enabled, and the clocks to the core, and SRAM are stopped. Each module can disable its clock locally at the module level.

In stop mode, all system clocks are disabled. There are several options for enabling or disabling the PLL or crystal oscillator in stop mode, compromising between stop mode current and wakeup recovery time. The PLL can be disabled in stop mode, but requires a wakeup period before it can relock. The oscillator can also be disabled during stop mode, but requires a wakeup 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 8.2.5, “Low-Power Control Register \(LPCR\),”](#) for more information about operating the PLL in stop mode.

There is also a fast wakeup option for quickly enabling the system clocks during stop recovery (LPCR[FWKUP]). This eliminates the wakeup recovery time but at the risk of sending a potentially unstable clock to the system.

7.2 Memory Map/Register Definition

The PLL module programming model consists of the following registers:

Table 7-2. PLL Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC09_0000	PLL Output Divider Register (PODR)	8	R/W	0x36 ¹	7.2.1/7-5
0xFC09_0002	PLL Control Register (PCR)	8	R/W	0x00	7.2.2/7-6
0xFC09_0004	PLL Modulation Divider Register (PMDR)	8	R/W	0x00	7.2.3/7-7
0xFC09_0006	PLL Feedback Divider Register (PFDR)	8	R/W	0x42 ¹	7.2.4/7-8

¹ With default reset configuration ($\overline{\text{RCON}}$ is negated).

7.2.1 PLL Output Divider Register (PODR)

The PODR register controls the output divider for generating the core and bus clocks. The value of this register may be either 0x48 or 0x36 and writing any other value to this register will result in unpredictable behavior.

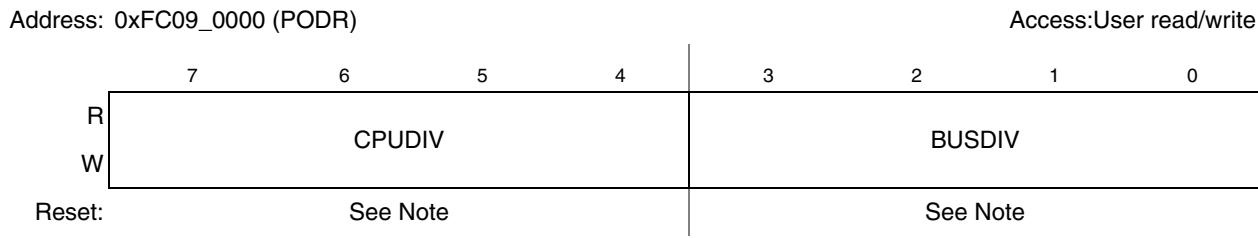


Figure 7-3. PLL Output Divider Register (PODR)

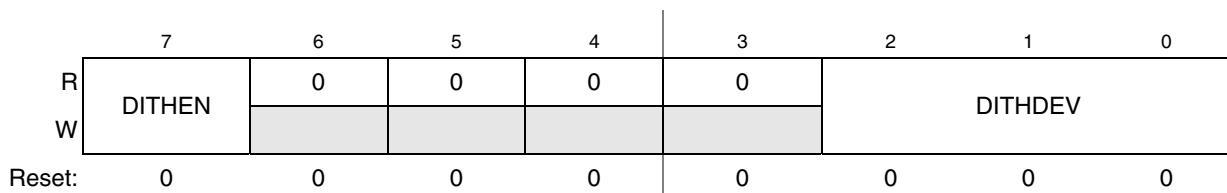
Table 7-3. PODR Field Descriptions

Field	Description																		
7–4 CPUDIV	Divider for generating the core frequency. See BUSDIV for a table of possible values.																		
3–0 BUSDIV	Divider for generating the internal bus frequency. <table border="1" data-bbox="616 418 1122 713"> <thead> <tr> <th>Value</th> <th>Core Clock</th> <th>Bus Clock</th> </tr> </thead> <tbody> <tr> <td>0011</td> <td>VCO/3</td> <td>Reserved</td> </tr> <tr> <td>0100</td> <td>VCO/4</td> <td>Reserved</td> </tr> <tr> <td>0110</td> <td>Reserved</td> <td>VCO/6</td> </tr> <tr> <td>1000</td> <td>Reserved</td> <td>VCO/8</td> </tr> <tr> <td>Else</td> <td>Reserved</td> <td>Reserved</td> </tr> </tbody> </table>	Value	Core Clock	Bus Clock	0011	VCO/3	Reserved	0100	VCO/4	Reserved	0110	Reserved	VCO/6	1000	Reserved	VCO/8	Else	Reserved	Reserved
Value	Core Clock	Bus Clock																	
0011	VCO/3	Reserved																	
0100	VCO/4	Reserved																	
0110	Reserved	VCO/6																	
1000	Reserved	VCO/8																	
Else	Reserved	Reserved																	

7.2.2 PLL Control Register (PCR)

Address: 0xFC09_0002 (PCR)

Access:User read/write

**Figure 7-4. PLL Control Register (PCR)****Table 7-4. PCR Field Descriptions**

Field	Description
7 DITHEN	Dithering enable bit. 0 Dithering disabled. 1 Dithering enabled.

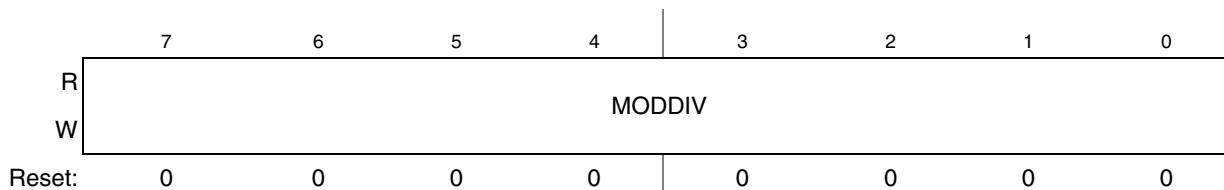
Table 7-4. PCR Field Descriptions (continued)

Field	Description																		
6–3	Reserved, should be cleared.																		
2–0 DITHDEV	<p>Dither Deviation. The dither deviation settings are target percentages based on simulation. The actual percentages observed will be slightly larger than these targets. See Section 7.3.2, “Dithering Waveform Definition” for more information.</p> <p>Deviation = -0.75% - (DITHDEV × 0.75%)</p> <table border="1"> <thead> <tr> <th>DITHDEV</th> <th>Deviation</th> </tr> </thead> <tbody> <tr> <td>000</td> <td>- 0.75%</td> </tr> <tr> <td>001</td> <td>- 1.00%</td> </tr> <tr> <td>010</td> <td>- 1.25%</td> </tr> <tr> <td>011</td> <td>- 1.50%</td> </tr> <tr> <td>100</td> <td>- 1.75%</td> </tr> <tr> <td>101</td> <td>- 2.00%</td> </tr> <tr> <td>110</td> <td>- 2.25%</td> </tr> <tr> <td>111</td> <td>- 2.50%</td> </tr> </tbody> </table> <p>Note: This field should only be written when dithering mode is disabled (PCR[DITHEN] = 0). Else, unpredictable PLL operation will result.</p>	DITHDEV	Deviation	000	- 0.75%	001	- 1.00%	010	- 1.25%	011	- 1.50%	100	- 1.75%	101	- 2.00%	110	- 2.25%	111	- 2.50%
DITHDEV	Deviation																		
000	- 0.75%																		
001	- 1.00%																		
010	- 1.25%																		
011	- 1.50%																		
100	- 1.75%																		
101	- 2.00%																		
110	- 2.25%																		
111	- 2.50%																		

7.2.3 PLL Modulation Divider Register (PMDR)

Address: 0xFC09_0004 (PMDR)

Access: User read/write

**Figure 7-5. PLL Modulation Divider Register (PMDR)****Table 7-5. PMDR Field Descriptions**

Field	Description
7–0 MODDIV	<p>Dither modulation divider.</p> <p>Dither Modulation Frequency = Input Frequency / (MODDIV × 32)</p> <p>A dither modulation frequency greater than 105 kHz or less than 9.95 kHz is invalid. For example, for a 16 MHz input frequency, MODDIV may be programmed between 5 (100 kHz) and 50 (10 kHz). Programming MODDIV outside the specified range will result in unpredictable PLL operation.</p> <p>Note: This field should only be written when dithering mode is disabled (PCR[DITHEN] = 0). Else, unpredictable PLL operation will result.</p>

7.2.4 PLL Feedback Divider Register (PFDR)

Address: 0xFC09_0006 (PFDR)

Access: User read/write



Note: Reset value determined by reset configuration. See [Chapter 9, “Chip Configuration Module \(CCM\),”](#) for more information. For the default reset configuration (RCON negated), the reset value is 0x58. If RCON and D1 is asserted at reset, the reset value of PFDR is 0x7D.

Figure 7-6. PLL Feedback Divider Register (PFDR)

Table 7-6. PFDR Field Descriptions

Field	Description
7–0 MFD	<p>The MFD bits control the value of the divider in the PLL feedback loop. The value specified by the MFD bits establish the multiplication factor applied to the reference frequency. See Section 7.3.3, “PLL Frequency Multiplication Factor Select” for more details.</p> <p>0x58 88 0x59 89 0x5A 90 ... 0x86 134 0x87 135 Else Reserved</p> <p>Note: The MFD bits may only be written when the device is in limp mode (MISCCR[LIMP] = 1).</p>

7.3 Functional Description

This subsection provides a functional description of the clock module.

7.3.1 PLL Dithered and Non-Dithered Operation

The PLL is capable of generating output clocks with a frequency that modulates in a triangular waveform with a specified percentage frequency deviation and a specified dither modulation frequency. This modulation of the output clock is called dithered operation. When the PLL operates at a fixed frequency, this operation is known as non-dithered operation. The selection of dithered or non-dithered operation is controlled by the PCR[DITHEN] bit. The percent frequency deviation and dither modulation frequency are also controlled by the PCR and PMDR registers, whose operation is described in [Section 7.3.2, “Dithering Waveform Definition.”](#)

After reset, dithering is disabled. After the PLL has locked (indicated by the MISCCR[PLLLOCK] bit described in [Section 8.2.6, “Miscellaneous Control Register \(MISCCR\)”](#)), the PLL may be changed from non-dithered operation to dithered operation by writing to the PCR. Once the PCR[DITHEN] bit has been set, the PLL will synchronize the new value with the VCO clock domain. Then the transition from non-dithered operation to dithered operation will take place such that the PLL output clocks remain glitch-free. However, the dithering waveform and deviation percentages are not guaranteed to meet

specifications until two modulation periods have passed after the time of the write to the PCR register. During the transition the frequency of the PLL output clocks will not exceed 10% of the respective non-dithered frequency.

The PLL may also be changed from dithered operation to non-dithered operation by clearing the PCR[DITHEN] bit. Once this occurs, the PLL will synchronize the new value with the VCO clock domain. Then, the transition from dithered operation to non-dithered operation will take place such that PLL output clocks remain glitch-free. However, the frequency of the PLL output clocks are not guaranteed to be completely stable until one modulation period has passed after the time of the write to the PCR. During the transition the frequency of the PLL output clocks will not exceed 10% of the respective non-dithered frequency.

Since the transition between dithered and non-dithered operation (and vice-versa) takes a period of time to change, the PCR may not be written back-to-back without waiting two modulation periods between writes.

NOTE

Failure to wait two modulation periods between writes to the PCR will result in unpredictable PLL operation.

7.3.2 Dithering Waveform Definition

The dithering waveform created by the changes in the frequency of the PLL output clocks is defined by the percent frequency deviation and dither modulation frequency. The definitions of the percent frequency deviation (or dithering deviation) and the modulation period (which is the inverse of the dither modulation frequency) are shown in [Figure 7-7](#). The dithering deviation is controlled by PCR[DITHDEV] field, while the dither modulation frequency is controlled by the PMDR[MODDIV] field.

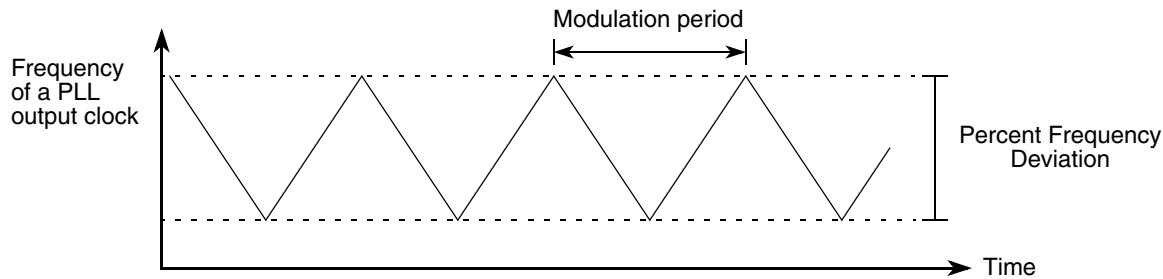


Figure 7-7. Ideal Triangular Dithering Waveform

After reset, the PCR and PMDR registers will be cleared, disabling dithering. After reset has been de-asserted and the PLL has locked (indicated by the MISCCR[PLLLOCK] bit), the dithering waveform definition may be changed by writing to the PCR[DITHDEV] field and the PMDR register. However, the PCR[DITHDEV] field and PMDR register may only be written to when the PLL is in non-dithered operation (i.e. the PCR[DITHEN] bit must be cleared).

NOTE

Writing to the PCR[DITHDEV] field or the PMDR during dithering operation will result in unpredictable PLL operation.

The dither deviation can be programmed to be between -0.75% of the non-dithered frequency up to -2.50% of the non-dithered frequency in steps of 0.25%.

NOTE

The dither deviation settings in the PCR[DITHDEV] field are target percentages based on simulation. The actual percentages achieved may be numerically different than the percentages listed in the specification; however, the actual percentages achieved will be in proportion to each other and will be stable within $\pm 10\%$ across process, voltage, and temperature conditions.

The dither modulation frequency can be programmed to be between approximately 10kHz and 100kHz. Since the dither modulation frequency is determined as a division of the input frequency, the dither modulation frequency is given by the following equation:

$$\text{Dither Modulation Frequency} = \frac{\text{Input Frequency}}{\text{PMDR[MODDIV]} \times 32}$$

Eqn. 7-1

NOTE

PMDR[MODDIV] field values which result in a dither modulation frequency greater than 105kHz or less than 9.95kHz are invalid and will result in unpredictable PLL operation.

7.3.3 PLL Frequency Multiplication Factor Select

The frequency multiplication factor of the PLL is defined by the feedback divider in the following equation:

$$f_{\text{sys}} = f_{\text{ref}} \times \left(\frac{\text{PFDR}}{4 \times \text{PODR[CPUDIV]}} \right) \quad \text{Eqn. 7-2}$$

where f_{sys} is the core frequency. The allowable range of values for the PFDR is 88 to 135, resulting in a frequency multiplication factor range of 7.33 to 11.25 times the input reference frequency (typically 16 MHz).

The PFDR can only be modified while the device is in limp mode, which is entered by setting the MISCCR[LIMP] bit. Once the PFDR register has been changed, re-enter normal mode by clearing the MISCCR[LIMP] bit. The PLL will then begin to acquire lock accordingly on the new frequency.

7.3.4 System Clock Modes

The system clock source is determined during reset. By default the PLL is placed in crystal reference mode and generates a core/bus frequency of 88/44 MHz. This default mode can be overridden by asserting the RCON pin. See [Chapter 9, “Chip Configuration Module \(CCM\),”](#) for more information on overriding the default configuration during reset.

[Table 7-7](#) shows the clock-out frequency to clock-in frequency relationships for the possible system clock modes. Refer to [Section 7.1.3, “Modes of Operation”](#) for details on each mode.

Table 7-7. Clock Out and Clock In Relationships

System Clock Mode	PLL Options ¹	Cross-Reference
Normal PLL clock mode	<p>$\overline{\text{RCON}}$ negated (default): $f_{\text{sys}} = f_{\text{ref}} \times \left(\frac{\text{PFDR}}{16} \right)$</p> <p>$\overline{\text{RCON}}$ and D1 asserted: $f_{\text{sys}} = f_{\text{ref}} \times \left(\frac{\text{PFDR}}{12} \right)$</p>	Section 7.1.3.1, "Normal PLL Mode with Crystal Reference" and Section 7.1.3.2, "Normal PLL Mode with External Reference"
Limp mode	$f_{\text{sys}} = \frac{f_{\text{ref}}}{2^{\text{MISCCR[LPDIV]}}}$	Section 7.1.3.3, "Input Clock (Limp) Mode"

¹ f_{ref} = input reference frequency = 16 MHz

PFDR ranges from 88 to 135

MISCCR[LPDIV] ranges from 0 to 15

7.3.5 Clock Operation During Reset

This section describes the reset operation of the PLL. Both power-on reset and normal reset are described.

7.3.5.1 Power-On Reset (POR)

Once V_{DDPLL} and the input clock are within specification, the PLL is held in reset for at least 10 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 will be output to the device. Once $\overline{\text{RESET}}$ is de-asserted, PLL output clocks will be generated; however, until the MISCCR[PLLLOCK] bit is set the PLL output clock frequencies will not be stable and will not be within specification. The MISCCR[PLLLOCK] bit will be set after $\overline{\text{RESET}}$ has negated for a minimum of 1 ms. When this bit is set, the PLL will be in frequency lock.

7.3.5.2 External Reset

When $\overline{\text{RESET}}$ is asserted, the PLL input clock will be output to the device and the PLL will not begin acquiring lock until $\overline{\text{RESET}}$ is negated. The MISCCR[PLLLOCK] bit will be cleared and will remain cleared while the PLL is acquiring lock. This bit will set after the PLL lock period of 1 ms has passed.

CAUTION

When running in an unlocked state, the clocks generated by the PLL are not guaranteed to be stable and may exceed the maximum specified frequency of the device.

Chapter 8

Power Management

8.1 Introduction

This chapter explains the low-power operation of the device.

8.1.1 Features

The following features support low-power operation:

- Four modes of operation: 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

8.2 Memory Map/Register Definition

The power management programming model consists of registers from both the SCM and CCM memory space, as shown below:

Table 8-1. Power Management Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
Supervisor Access Only Registers¹					
0xFC04_0013	Wakeup Control Register (WCR)	8	R/W	0x00	8.2.1/8-2
0xFC04_002C	Peripheral Power Management Set Register 0 (PPMSR0)	8	W	0x00	8.2.2/8-3
0xFC04_002D	Peripheral Power Management Clear Register 0 (PPMCR0)	8	W	0x00	8.2.3/8-4
0xFC04_0030	Peripheral Power Management High Register 0 (PPMHR0)	32	R/W	0x0000_0000	8.2.4/8-4
0xFC04_0034	Peripheral Power Management Low Register 0 (PPMLR0)	32	R/W	0x0000_0000	8.2.4/8-4
0xFC0A_0007	Low-Power Control Register (LPCR)	8	R/W	0x00	8.2.5/8-6
0xFC0A_0010	Miscellaneous Control Register (MISCCR)	16	R/W	See Section	8.2.6/8-7

¹ User access to supervisor only address locations have no effect and result in a bus error

8.2.1 Wake-up Control Register

Implementation of low-power stop mode and exit from a low-power mode via an interrupt require communication between the core and logic associated with the interrupt controller. The WCR enables entry into low-power modes, and includes the setting of the interrupt level needed to exit a low-power mode.

NOTE

The setting of the low-power mode select field, LPCR[LPMD], determines which low-power mode the device enters when a STOP instruction is issued.

The following sequence of operations is 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. Once 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 an interrupt request with a priority level greater than the value programmed in WCR[PRILVL].
5. Once 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.

Address: 0xFC04_0013 (WCR)

Access: Supervisor read/write

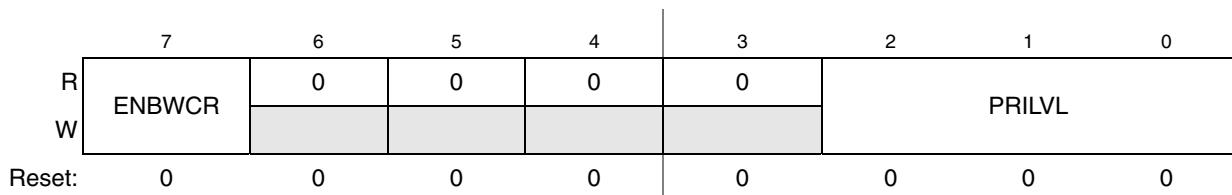


Figure 8-1. Wake-up Control Register (WCR)

Table 8-2. WCR Field Descriptions

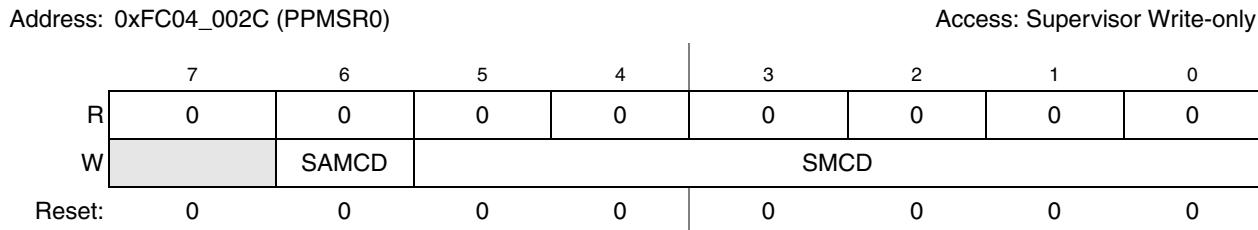
Field	Description
7 ENBWCR	Enable low-power mode entry. The mode entered is specified in LPCR[LPMD]. 0 Low-power mode entry is disabled 1 Low-power mode entry is enabled.

Table 8-2. WCR Field Descriptions (continued)

Field	Description																	
6–3	Reserved, should be cleared.																	
2–0 PRILVL	Exit low-power mode interrupt priority level. This field defines the interrupt priority level needed to exit the low-power mode.																	
	<table border="1"> <thead> <tr> <th>PRILVL</th> <th>Interrupt Level Needed to Exit Low-Power Mode</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>		PRILVL	Interrupt Level Needed to Exit Low-Power Mode	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
PRILVL	Interrupt Level Needed to Exit Low-Power Mode																	
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																	

8.2.2 Peripheral Power Management Set Register (PPMSR0)

The PPMSR register provides a simple mechanism to set a given bit in the PPM{H,L}R registers to disable the clock for a given peripheral module without the need 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 to be set. The SAMCD bit provides a global set function forcing the entire contents of the PPMR to be set, disabling all peripheral module clocks. Reads of these registers return all zeroes.

**Figure 8-2. Peripheral Power Management Set Register (PPMSR0)****Table 8-3. PPMSR0 Field Descriptions**

Field	Description
7	Reserved, should 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.

8.2.3 Peripheral Power Management Clear Register (PPMCR)

The PPMCR register provides a simple mechanism to clear a given bit in the PPMHR & PPMLR registers, enabling the clock for a given peripheral module without the need 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 to be clear. A value of 64 to 127 (setting the CAMCD bit) provides a global clear function forcing the entire contents of the PPMR to be clear, enabling all peripheral module clocks. Reads of these registers return all zeroes.

Address: 0xFC04_002D (PPMCR0)								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 8-3. Peripheral Power Management Clear Register (PPMCR0)

Table 8-4. PPMCR0 Field Descriptions

Field	Description
7	Reserved, should be cleared.
6 CAMCD	Clear all module clock disables. 0 Clear only those bits specified in the CMCD field 1 Clear all bits in PPMRH and PPMLR, enabling all peripheral clocks
5–0 CMCD	Clear module clock disable. Clear the corresponding bit in PPMR{H,L}, enabling the peripheral clock.

8.2.4 Peripheral Power Management Registers (PPMHR0 & PPMLR0)

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 of these address spaces that defines whether the module clock for the given space is enabled or disabled.

Since 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.

The individual bits of the PPMR can be modified using a read-modify-write to this register directly or indirectly through writes to the PPMSR and PPMCR registers to set/clear individual bits.

Address: 0xFC04_0030 (PPMHR0) Access: Supervisor 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	CD42	CD41	CD40	0	0	0	CD36	CD35	CD34	CD33	CD32
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-4. Peripheral Power Management High Register (PPMHR0)

Table 8-5. PPMHR0[CDn] Assignments

Slot Number	CDn	Peripheral
32	CD32	PIT 0
33	CD33	PIT 1
34	CD34	Edge Port
35	CD35	On-chip Watchdog Timer
36	CD36	PLL
40	CD40	CCM, Reset Controller, Power Management
41	CD41	GPIO Module
42	CD42	SDRAM 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	0	CD26	CD25	CD24	CD23	CD22	CD21	0	0	CD18	CD17	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	CD12	0	0	0	0	0	0	0	0	0	CD2	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 8-5. Peripheral Power Management Low Registers (PPMLR0)

Table 8-6. PPMLR0[CDn] Assignments

Slot Number	CDn	Peripheral
2	CD2	FlexBus
12	CD12	FEC
17	CD17	eDMA Controller

Table 8-6. PPMLR0[CD*n*] Assignments (continued)

Slot Number	CD <i>n</i>	Peripheral
18	CD18	Interrupt Controller
21	CD21	IACK
22	CD22	I ² C
23	CD23	QSPI
24	CD24	UART0
25	CD25	UART1
26	CD26	UART2
28	CD28	DMA Timer 0
29	CD29	DMA Timer 1
30	CD30	DMA Timer 2
31	CD31	DMA Timer 3

Table 8-7. PPMHR & PPMLR Field Descriptions

Field	Description
CD <i>n</i>	Module slot <i>n</i> clock disable. 0 The clock for this module is enabled. 1 The clock for this module is disabled.

CAUTION

Extreme caution should be taken by the customer when setting PPMR[CD40] to disable clocking of the CCM, reset controller, and power management modules. This may disable logic to reset the chip, disable the external bus monitor, and other logic contained within these blocks.

8.2.5 Low-Power Control Register (LPCR)

The LPCR register controls chip operation and module operation during low-power modes.

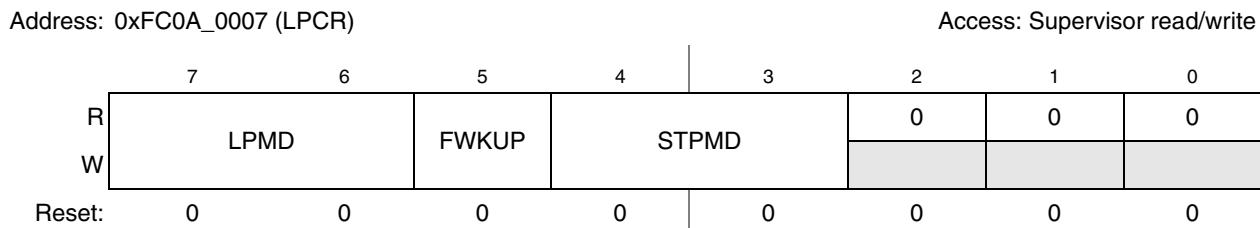
**Figure 8-6. Low-Power Control Register (LPCR)**

Table 8-8. LPCR Field Descriptions

Field	Description																									
7–6 LPMD	Low-power mode select. Used to select the low-power mode the chip enters once the ColdFire core executes the STOP instruction. These bits must be written prior to instruction execution for them to take effect. The LPMD bits are readable and writable in all modes. 00 Run 01 Doze 10 Wait 11 Stop Note: If LPCR[LPMD] is cleared, then the device will stop executing code upon issue of a STOP instruction. However, no clocks will be disabled.																									
5 FWKUP	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. 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. Note: 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 the MISCCR[PLLLOCK] bit. Also note that FWKUP is not effective in limp mode, since the PLL never locks in this mode. The system clocks are always enabled upon wake-up from stop mode, regardless of the value of FWKUP.																									
4–3 STPMD	FB_CLK stop mode bits. Controls the operation of the clocks, PLL, and oscillator in stop mode as shown below.																									
	<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, should be cleared.																									

8.2.6 Miscellaneous Control Register (MISCCR)

The MISCCR provides clock source selection and configuration for internal clocks.

Address: 0xFC0A_0010 (MISCCR)

Access: Supervisor read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	PLL LOCK	LIMP	0	0	0	0	0	0	0	0	LPDIV			
W																
Reset	0	0	1	See Note	0	0	0	0	0	0	0	0	0	0	0	0

Note: Reset value depends upon chosen reset configuration (RCON[RЛИMP]).

Figure 8-7. Miscellaneous Control Register (MISCCR)

Table 8-9. MISCCR Field Descriptions

Field	Description
15–14	Reserved, should be cleared.
13 PLLLOCK	PLL lock status. 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 all internal clocks. 1 Limp mode; low-power clock divider drives internal clocks.
11–4	Reserved, should be cleared.
3–0 LPDIV	Low power divider. Specifies the limp mode divide value used to produce the clock for the ColdFire core, bus, and other system clocks. A 2:1 ratio is maintained between the core and the internal bus. This field is used only when LIMP is set, and ignored otherwise. Limp mode clock = Oscillator clock / 2^{LPDIV} Note: When LPDIV is 0 (divide-by-1), the internal bus clock and FB_CLK will not have a 50/50 duty cycle.

8.3 Functional Description

The functions and characteristics of the low-power modes, and how each module is affected by, or affects these modes are discussed in this section.

8.3.1 Peripheral Shut Down

All peripherals, except for the SCM and crossbar switch, may have their input clocks individually removed by software in order to reduce power consumption. See [Section 8.2.4, “Peripheral Power Management Registers \(PPMHR0 & PPMLR0\)”](#) for more information. A peripheral may be disabled at any time and will remain disabled during any low-power mode of operation.

8.3.2 Limp mode

The device may also be booted 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 5-bit programmable counter that divides the input clock by 2^n , where n is the value of the programmable counter field, MISCCR[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. When switching from limp mode to normal functional mode, the user must wait for the PLL to lock before continuing with code execution.

Limp mode may also be entered and exited from by writing to the MISCCR[LIMP] bit.

While in this mode a 2:1 ratio is maintained between the core and the primary bus clock. Because they will not function at speeds as low as the minimum input clock frequency, the SDRAM controller, and FEC are not functional in limp mode.

8.3.3 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 LPCR[LPMD]. 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 which is 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.

8.3.3.1 Run Mode

Run mode is the normal system operating mode. Current consumption in this mode is related directly to the system clock frequency.

8.3.3.2 Wait Mode

Wait mode is intended to be used 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, which cause the CPU to exit from wait mode.

8.3.3.3 Doze Mode

Doze mode affects the CPU in the same manner as wait mode, except that some peripherals define individual operational characteristics in doze mode. Peripherals which continue to run and have the capability of producing interrupts may cause the CPU to exit the doze mode and return to run mode. Peripherals which are stopped will restart operation on exit from doze mode as defined for each peripheral.

8.3.3.4 Stop Mode

Stop mode affects the CPU in the same manner 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.

The following subsections specify the operation of each module while in and when exiting low-power modes.

NOTE

Entering stop mode will disable the SDRAMC including the refresh counter. If SDRAM is used, then code is required to insure proper entry and exit from stop mode. See [Chapter 18, “SDRAM Controller \(SDRAMC\),”](#) for more information.

8.3.4 Peripheral Behavior in Low-Power Modes

8.3.4.1 ColdFire Core

The ColdFire core is disabled during any low-power mode. No recovery time is required when exiting any low-power mode.

8.3.4.2 Internal SRAM

The SRAM is disabled during any low-power mode. No recovery time is required when exiting any low-power mode.

8.3.4.3 Clock Module

In wait and doze modes, the clocks to the CPU and SRAM will be stopped and the system clocks to the peripherals are enabled. 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 8.2.4, “Peripheral Power Management Registers \(PPMHR0 & PPMLR0\)”](#)). In stop mode, all clocks to the system will be stopped.

There are several options for enabling or disabling the PLL or crystal oscillator in stop mode, compromising between stop mode current and wakeup recovery time. The PLL can be disabled in stop mode, but requires a wakeup period before it can relock. The oscillator can also be disabled during stop mode, but requires a wakeup 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 8.2.5, “Low-Power Control Register \(LPCR\),”](#) for more information about operating the PLL in stop mode.

There is also a fast wakeup option for quickly enabling the system clocks during stop recovery (LPCR[FWKUP]). This eliminates the wakeup recovery time but at the risk of sending a potentially unstable clock to the system. This is also explained in [Section 8.2.5, “Low-Power Control Register \(LPCR\).”](#)

8.3.4.4 Chip Configuration Module

The chip configuration module is unaffected by entry into a low-power mode. If low-power mode is exited by a reset, chip configuration may be executed if configured to do so.

8.3.4.5 Reset Controller

A power-on reset (POR) will always cause a chip reset and exit from any low-power mode.

In wait and doze modes, asserting the external RESET pin for at least four clocks will cause an external reset that will reset the chip and exit any low-power modes.

In stop mode, the RESET pin synchronization is disabled and asserting the external RESET pin will asynchronously generate an internal reset and exit any low-power modes. Registers will lose current values and must be reconfigured from reset state if needed.

If the core or on-chip watchdog timer is still enabled during wait or doze modes, then a watchdog timer timeout may generate a reset to exit these low-power modes.

When the CPU is inactive, a software reset cannot be generated to exit any low-power mode.

8.3.4.6 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 cause a reset of 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 8.3.3, “Low-Power Modes”](#) for the core watchdog interrupt to bring the part out of low-power mode.

8.3.4.7 Cross-Bar Switch

8.3.4.8 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 will revert to their default direction settings.

8.3.4.9 Interrupt Controllers (INTC0)

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 will be combinational to allow the ability to wake up the core during low-power stop mode when all system clocks are stopped.

An interrupt request will cause the CPU 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.

8.3.4.10 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, there is no system clock available to perform the edge detect function. Thus, 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.

8.3.4.11 eDMA Controller

In wait and doze modes, the eDMA controller is capable of bringing the device out of a low-power mode by generating an interrupt either upon completion of a transfer or upon an error condition. The completion of transfer interrupt is generated when DMA interrupts are enabled by the setting of a EDMA_INTR[INTn] bit, and an interrupt is generated when the TCDn[DONE] bit is set. The interrupt upon error condition is generated when the EDMA_EEIR[EEIn] bit is set, and an interrupt is generated when any of the EDMA_ESR bits becomes set.

The eDMA controller is stopped in stop mode and thus cannot cause an exit from this low-power mode.

8.3.4.12 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.

8.3.4.13 SDRAM Controller (SDRAMC)

SDRAM controller operation is unaffected by either the wait or doze modes; however, the SDRAMC is disabled by stop mode. Since all clocks to the SDRAMC are disabled by stop mode, the SDRAMC will not generate refresh cycles.

To prevent loss of data the SDRAMC should be placed in self-refresh mode by clearing SDCR[CKE] and setting SDCR[REF_EN]. The SDRAM self-refresh mode allows the SDRAM to enter a low-power state where internal refresh operations are used to maintain the integrity of the data stored in the SDRAM.

When stop mode is exited, setting the SDCR[CKE] bit will cause the SDRAM controller to exit the self-refresh mode and allow bus cycles to the SDRAM to resume.

NOTE

The SDRAM is inaccessible while in the self-refresh mode. Therefore, if stop mode is used the vector table and any interrupt handlers that could wake the processor should not be stored in or attempt to access SDRAM.

8.3.4.14 Fast Ethernet Controller (FEC)

In wait and doze modes, the FEC is unaffected and may generate an interrupt to exit these low-power modes. In stop mode, the FEC stops immediately and freezes operation, register values, state machines, and external pins. During this mode, the FEC clocks are shut down. Coming out of stop mode returns the FEC to operation from the state prior to stop mode entry.

8.3.4.15 On-chip Watchdog Timer

In stop mode (or in wait/doze mode, if so programmed in the WCR register), the watchdog ceases operation and freezes at the current value. When exiting these modes, the watchdog resumes operation from the stopped value. It is the responsibility of software to avoid erroneous operation.

When not stopped, the watchdog may generate a reset to exit the low-power modes.

8.3.4.16 Programmable Interrupt Timers (PIT0–3)

In stop mode (or in 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.

8.3.4.17 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 be generated when the DMA timer is in either 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 is issued upon a captured input. In reference compare mode, where the output reference request interrupt enable (ORRI) bit of DTMR is set and the DTXMR[DMAEN] bit is cleared, an interrupt is issued when the timer counter reaches the reference value.

DMA timer operation is disabled in stop mode. Upon exiting stop mode, the timer will resume operation unless stop mode was exited by reset.

8.3.4.18 Queued Serial Peripheral Interface (QSPI)

In wait and doze modes, the QSPI module is unaffected and may generate an interrupt to exit these low-power modes.

In stop mode, the QSPI stops immediately and freezes operation, register values, state machines, and external pins. During this mode, the QSPI clocks are shut down. Coming out of stop mode returns the QSPI to operation from the state prior to stop mode entry.

8.3.4.19 UART Modules (UART0–2)

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 are shut down. Coming out of stop mode returns the UARTs to operation from the state prior to stop mode entry.

8.3.4.20 I²C Module

When the I²C Module is enabled by the setting of the I2CR[IEN] bit and when the device is not in stop mode, the I²C 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 either 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 I²C module stops immediately and freezes operation, register values, and external pins. Upon exiting stop mode, the I²C resumes operation unless stop mode was exited by reset.

8.3.4.21 BDM

Entering halt (debug) mode via the BDM port (by asserting the external BKPT pin) will cause the CPU to exit any low-power mode.

8.3.4.22 JTAG

The JTAG (Joint Test Action Group) controller logic is clocked using the TCLK input and is not affected by the system clock. The JTAG cannot generate an event to cause the CPU to exit any low-power mode. Toggling TCLK during any low-power mode will increase the system current consumption.

8.3.5 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 8-10](#). 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 LPCR[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 8-10. CPU and Peripherals in Low-Power Modes

Module	Peripheral Status ¹ / Wake-up Procedure					
	Wait Mode		Doze Mode		Stop Mode	
ColdFire Core	Stopped	N/A	Stopped	N/A	Stopped	N/A
SRAM	Stopped	N/A	Stopped	N/A	Stopped	N/A
Clock Module	Enabled	Interrupt	Enabled	Interrupt	Program	Interrupt
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	Reset	Enabled	Reset	Stopped	N/A
GPIO Module	Enabled	N/A	Enabled	N/A	Enabled	N/A

Table 8-10. CPU and Peripherals in Low-Power Modes (continued)

Module	Peripheral Status ¹ / Wake-up Procedure					
	Wait Mode		Doze Mode		Stop Mode	
Interrupt controller	Enabled	Interrupt	Enabled	Interrupt	Enabled	Interrupt
Edge port	Enabled	Interrupt	Enabled	Interrupt	Stopped	Interrupt
eDMA Controller	Enabled	Yes	Enabled	Yes	Stopped	N/A
FlexBus Module	Enabled	N/A	Enabled	N/A	Stopped	N/A
SDRAM Controller	Enabled	N/A	Enabled	N/A	Stopped	N/A
Fast Ethernet Controller	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
On-chip Watchdog Timer	Program	Reset	Program	Reset	Stopped	N/A
Programmable Interrupt Timers	Enabled	Interrupt	Program	Interrupt	Stopped	N/A
DMA Timers	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
QSPI	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
UARTs	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
I ² C Module	Enabled	Interrupt	Enabled	Interrupt	Stopped	N/A
JTAG	Enabled	N/A	Enabled	N/A	Enabled	N/A
BDM ²	Enabled	Yes	Enabled	Yes	Enabled	Yes

¹ “Program” indicates that the peripheral function during the low-power mode is dependent on programmable bits in the peripheral register map.

² The BDM logic is clocked by a separate TCLK clock. Entering halt mode via the BDM port exits any lower-power mode. Upon exit from halt mode, the previous low-power mode will be re-entered and changes made in halt mode will remain in effect.

Chapter 9

Chip Configuration Module (CCM)

9.1 Introduction

The chip configuration module (CCM) controls the chip configuration for the device.

9.1.1 Block Diagram

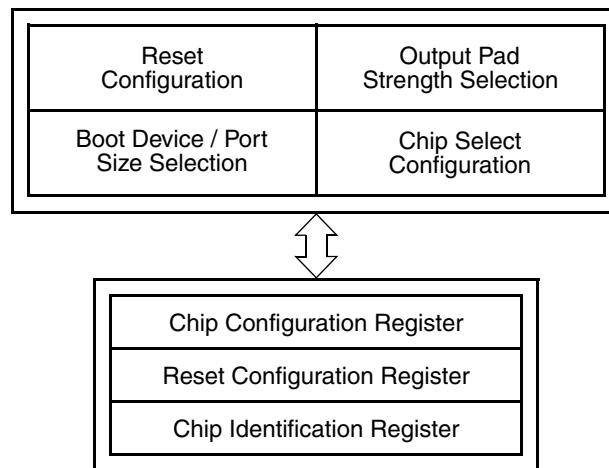


Figure 9-1. Chip Configuration Module Block Diagram

9.1.2 Features

The CCM performs the following operations:

- Selects the chip operating mode
- Selects external clock or phase-lock loop (PLL) mode with internal or external reference
- Selects output pad drive strength
- Selects boot device and data port size
- Selects bus monitor configuration
- Selects low-power configuration

9.1.3 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 bus and 24 address lines. The available bus control signals include R/W, TS, TA, OE, and BE/BWE[3:0]. Up to six chip selects

can be programmed to select and control external devices and to provide bus cycle termination. When interfacing to 16-bit ports, the BE/BWE[1:0] pins can be configured as general-purpose input/output (I/O); when interfacing to 8-bit ports, the BE/BWE[2:0] pins can be configured as general purpose input/output (I/O).

9.2 External Signal Descriptions

[Table 9-1](#) provides an overview of the CCM signals.

Table 9-1. Signal Properties

Name	Function	Reset State
RCON	Reset configuration select	Internal weak pull-up device
D[9:1]	Reset configuration override pins	—

9.2.1 RCON

If the external RCON pin is asserted during reset, then various chip functions, including the reset configuration pin functions after reset, are configured according to the levels driven onto the external data pins (see [Section 9.4, “Functional Description”](#)). The internal configuration signals are driven to reflect the levels on the external configuration pins to allow for module configuration.

9.2.2 D[9:1] (Reset Configuration Override)

If the external RCON pin is asserted during reset, then the states of these data pins during reset determine the chip mode of operation, boot device, clock mode, and certain module configurations after reset.

9.3 Memory Map/Register Definition

The CCM programming model consists of the registers listed in the below table.

Table 9-2. CCM Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
Supervisor Access Only Registers¹					
0xFC0A_0004	Chip Configuration Register (CCR)	16	R	See Section	9.3.1/9-2
0xFC0A_0008	Reset Configuration Register (RCON)	16	R	0x0001	9.3.2/9-3
0xFC0A_000A	Chip Identification Register (CIR)	16	R	See Section	9.3.3/9-4

¹ User access to supervisor only address locations have no effect and result in a bus error.

9.3.1 Chip Configuration Register (CCR)

The CCR is a read-only register; writing to the CCR has no effect. At reset, the CCR reflects the chosen operation of certain chip functions. These functions may either be set to the defaults defined by the RCON

register values or may be overridden during reset configuration using the external RCON and D[15:0] pins. (See [Figure 9-3](#) for the RCON register definition.)

Address: 0xFC0A_0004 (CCR)												Access: Supervisor read-only							
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
R	0	0	0	0	0	0	CSC	0	OSC FREQ	LIMP	LOAD	BOOTPS	OSC MODE	PLL MODE	1				
W																			
Reset	0	0	0	0	0	0										1			
																See Note			

Note: Reset value depends upon chosen reset configuration. Default reset value ([RCON](#) is not asserted) is 0x0001.

Figure 9-2. Chip Configuration Register (CCR)

Table 9-3. CCR Field Descriptions

Field	Description
15–10	Reserved, should be cleared.
9 CSC	Chip select configuration field. Reflects the chosen chip select configuration. 0 A[23:22] = A[23:22] 1 A[23] = FB_CS5 and A[22] = FB_CS4
8	Reserved, should be cleared.
7 OSCFREQ	Oscillator frequency select bit. Reflects the chosen oscillator input frequency for the device. 0 16 MHz 1 16.67 MHz
6 LIMP	Limp mode bit. 0 Normal operation; PLL drives internal clocks. 1 Limp mode; low-power clock divider drives internal clocks.
5 LOAD	Pad driver load bit. Reflects the chosen pad driver strength for those pins with drive strength control and the chosen pad slew rate for those pins with slew rate control. 0 Low drive strength, low slew rate 1 High drive strength, high slew rate
4–3 BOOTPS	Boot port size field. Indicates the selection for the boot port size. 00 32 bits 01 16 bits 10 8 bits 11 32 bits
2 OSCMODE	Oscillator clock mode bit. 0 Crystal oscillator mode 1 Oscillator bypass mode
1 PLLMODE	PLL clock mode 0 88/44 MHz operation 1 166.67/83.33 MHz operation
0	Reserved.

9.3.2 Reset Configuration Register (RCON)

At reset, RCON determines the default operation of certain chip functions. All default functions defined by the RCON values can only be overridden during reset configuration (see [Section 9.4.1, “Reset](#)

Chip Configuration Module (CCM)

Configuration") if the external RCON pin is asserted. RCON is a read-only register and contains the same fields as the CCR register. See [Table 9-3](#) for field descriptions.

Address: 0xFC0A_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	0	0	0	CSC	0	OSC FREQ	LIMP	LOAD	BOOTPS	OSC MODE	PLL MODE	1	
W																
Reset	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	

Figure 9-3. Reset Configuration Register (RCON)

9.3.3 Chip Identification Register (CIR)

Address: 0xFC0A_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 9-4. Chip Identification Register (CIR)

Table 9-4. CIR Field Description

Field	Description
15–6 PIN	Part identification number. Contains a unique identification number for the device. 0x044 MCF5208 0x045 MCF5207
5–0 PRN	Part revision number. This number is increased by one for each new full-layer mask set of this part. The revision numbers are assigned in chronological order.

9.4 Functional Description

Six functions are defined within the chip configuration module:

1. Reset configuration
2. PLL mode
3. Oscillator mode
4. Boot device selection
5. Output pad strength configuration
6. Chip select configuration

These functions are described below.

9.4.1 Reset Configuration

During reset, the pins for the reset override functions are immediately configured to known states. [Table 9-5](#) shows the states of the external pins while in reset.

Table 9-5. Reset Configuration Pin States During Reset

Pin	Pin Function ¹	I/O	Output State	Input State
D[15:0]	Primary function	Input	—	Must be driven by external logic
RCON	RCON function for all modes ²	Input	—	Internal weak pull-up device

¹ If the external RCON pin is not asserted during reset, pin functions are determined by the default operation mode defined in the RCON register. If the external RCON pin is asserted, pin functions are determined by the override values driven on the external data bus pins.

² During reset, the external RCON pin assumes its RCON pin function, but this pin changes to the function defined by the chip operation mode immediately after reset. See [Table 9-6](#).

If the RCON pin is not asserted during reset, the chip configuration and the reset configuration pin functions after reset are determined by the RCON register or fixed defaults, regardless of the states of the external data pins. The internal configuration signals are driven to levels specified by the RCON register's reset state for default module configuration.

If the RCON pin is asserted during reset, then various chip functions, including the reset configuration pin functions after reset, are configured according to the levels driven onto the external data pins. (See [Table 9-6](#).) The internal configuration signals are driven to reflect the levels on the external configuration pins to allow for module configuration.

Table 9-6. Configuration During Reset¹

Pin(s) Affected	Default Configuration	Override Pins in Reset ^{2,3}	Function
None	RCON[1] = 0	D1	PLL Mode
		0	88/44 MHz operation (default)
		1	166.67/83.33 MHz operation
None	RCON[2] = 0	D2	Oscillator Mode
		0	Crystal oscillator mode (default)
		1	Oscillator bypass mode
None	RCON[4:3] = 00	D[4:3]	Boot Device
		00	External with 32-bit port ⁴ (default)
		01	External with 16-bit port
		10	External with 8-bit port
		11	External with 32-bit port ⁴

Table 9-6. Configuration During Reset¹ (continued)

Pin(s) Affected	Default Configuration	Override Pins in Reset^{2,3}	Function
All output pins	RCON[5] = 0	D5	Output pad drive strength
		0	Low Drive Strength (default)
		1	High Drive Strength
None	RCON[6] = 0	D6	Limp Mode
		0	PLL mode (default)
		1	Limp mode
None	RCON[7] = 0	D7	Oscillator Frequency Select
		0	16 MHz (default)
		1	16.67 MHz
A[23:22]/FB_CS[5:4]	RCON[9] = 0	D9	Chip Select Configuration
		0	A[23:22] = A[23:22]
		1	A[23:22] = FB_CS[5:4] (default)

¹ Modifying the default configurations is possible only if the external RCON pin is asserted.

² The D[31:10,8,0] pins do not affect reset configuration.

³ The external reset override circuitry drives the data bus pins with the override values while RSTOUT is asserted. It must stop driving the data bus pins within one FB_CLK cycle after 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 RSTOUT is negated.

⁴ 32-bit port size is not available when DRAMSEL = 0. Defaults to 16-bit mode instead.

9.4.2 PLL Mode Selection

The initial device operating frequency is determined during reset configuration by the D1 pin. The default configuration with a 16 MHz input clock is 88 MHz for the core and 44 MHz for the internal bus. The user may choose to almost double these frequencies (166.67 MHz and 83.33 MHz) by either placing the device in limp mode and reconfiguring the appropriate PLL registers, or asserting D1 during reset configuration.

9.4.3 Oscillator Mode Selection

Use of the internal oscillator can be selected during reset configuration via the D2 pin. By default, the oscillator is enabled and the PLL is placed in normal mode with a crystal reference. If default configuration is over-ridden and D2 is asserted, the PLL is placed in normal mode with an external reference and the internal oscillator is bypassed. Once reset is exited, the oscillator mode cannot be changed. See Chapter 7, “Clock Module” for more details on the available modes.

9.4.4 Boot Device Selection

During reset configuration, the $\overline{\text{FB_CS0}}$ chip select pin is configured to select an external boot device. In this case, the V (valid) bit in the CSMR0 register is ignored, and $\overline{\text{FB_CS0}}$ is enabled after reset. $\overline{\text{FB_CS0}}$ 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 that the reset vector loaded from address $0x0000_0004$ causes the core to start executing from external memory space decoded by $\overline{\text{FB_CS0}}$.

9.4.5 Output Pad Strength Configuration

Output pad strength is determined during reset configuration as shown in [Table 9-7](#). Once reset is exited, the output pad strength configuration can only be changed using the GPIO module. For more information see [Chapter 13, “General Purpose I/O Module.”](#)

Table 9-7. Output Pad Driver Strength Selection¹

Optional Pin Function Selection	D5
Output pads configured for low drive strength	D5 driven low
Output pads configured for full drive strength	D5 driven high

¹ Modifying the default configurations is possible only if the external $\overline{\text{RCON}}$ pin is asserted low.

9.4.6 Chip Select Configuration

The chip select configuration ($\overline{\text{FB_CS}}[5:4]$) is selected during reset and reflected in the CCR[CSC] field. Once reset is exited, the chip select configuration cannot be changed. [Table 9-6](#) shows the different chip select configurations that can be implemented during reset configuration.

Chapter 10

Reset Controller Module

10.1 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.

10.1.1 Block Diagram

Figure 10-1 illustrates the reset controller and is explained in the following sections.

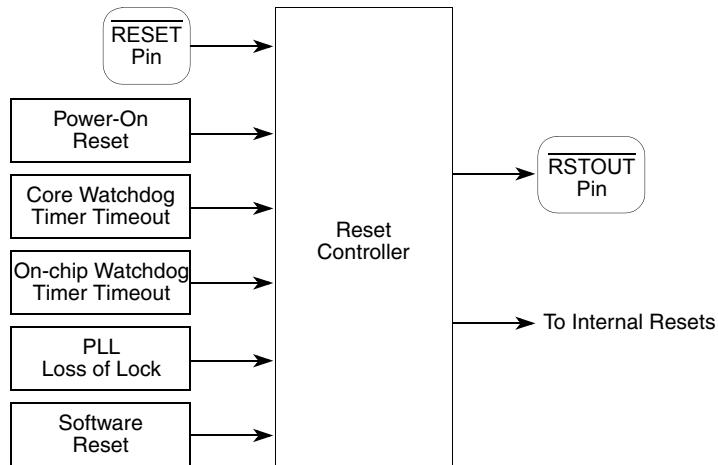


Figure 10-1. Reset Controller Block Diagram

10.1.2 Features

Module features include the following:

- Six sources of reset:
 - External
 - Power-on reset (POR)
 - Core watchdog timer
 - On-chip watchdog timer
 - Phase locked-loop (PLL) loss of lock
 - Software
- Software-assertable $\overline{\text{RSTOUT}}$ pin independent of chip reset state
- Software-readable status flags indicating the cause of the last reset

10.2 External Signal Description

[Table 10-1](#) provides a summary of the reset controller signal properties. The signals are described in the following paragraphs.

Table 10-1. Reset Controller Signal Properties

Name	Direction	Input Hysteresis	Input Synchronization
RESET	I	Y	Y ¹
RSTOUT	O	—	—

¹ RESET is always synchronized except when in low-power stop mode.

10.2.1 **RESET**

Asserting the external RESET for at least four rising FB_CLK edges causes the external reset request to be recognized and latched.

10.2.2 **RSTOUT**

This active-low output signal is driven low when the internal reset controller module resets the chip. 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 9, “Chip Configuration Module \(CCM\)](#),” for more details on these override options.

10.3 Memory Map/Register Definition

The reset controller programming model consists of these registers:

- Reset control register (RCR), which selects reset controller functions
- Reset status register (RSR), which reflects the state of the last reset source

See [Table 10-2](#) for the memory map and the following paragraphs for a description of the registers.

Table 10-2. Reset Controller Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_0000	Reset Control Register (RCR)	8	R/W	0x00	10.3.1/10-2
0xFC0A_0001	Reset Status Register (RSR)	8	R	See Section	10.3.2/10-3

10.3.1 **Reset Control Register (RCR)**

The RCR allows software control for requesting a reset, and for independently asserting the external RSTOUT pin.

Address: 0xFC0A_0000 (RCR)

Access: User read/write

	7	6	5	4	3	2	1	0
R	SOFTRST	FRCRSTOUT	0	0	0	0	0	0
W								
Reset:	0	0	0	0	0	0	0	0

Figure 10-2. Reset Control Register (RCR)

Table 10-3. RCR Field Descriptions

Field	Description
7 SOFTRST	Allows software to request a reset. The reset caused by setting this bit clears this bit. 1 Software reset request 0 No software reset request
6 FRCRSTOUT	Allows software to assert or negate the external RSTOUT pin. 1 Assert RSTOUT pin 0 Negate RSTOUT pin CAUTION: External logic driving reset configuration data during reset needs to be considered when asserting the RSTOUT pin when setting FRCRSTOUT.

10.3.2 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: 0xFC0A_0001 (RSR)

Access: User read-only

	7	6	5	4	3	2	1	0
R	0	0	SOFT	WDR CHIP	POR	EXT	WDR CORE	LOL
W								
Reset:	0	0	Reset Dependent	Reset Dependent	Reset Dependent	Reset Dependent	Reset Dependent	Reset Dependent

Figure 10-3. Reset Status Register (RSR)

Table 10-4. RSR Field Descriptions

Field	Description
7–6	Reserved, should be cleared.
5 SOFT	Software reset flag. Indicates that the last reset was caused by software. 0 Last reset not caused by software 1 Last reset caused by software

Table 10-4. RSR Field Descriptions (continued)

Field	Description
4 WDRCHIP	On-chip watchdog timer reset flag. Indicates that the last reset was caused by the on-chip watchdog timer timeout. 0 Last reset not caused by watchdog timer timeout 1 Last reset caused by watchdog timer timeout
3 POR	Power-on reset flag. Indicates that the last reset was caused by a power-on 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 <u>RESET</u> pin. 0 Last reset not caused by external reset 1 Last reset caused by external reset
1 WDRCORE	Core watchdog timer reset flag. Indicates that the last reset was caused by the core watchdog timer timeout. 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 that 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

10.4 Functional Description

10.4.1 Reset Sources

Table 10-5 defines the sources of reset and the signals driven by the reset controller.

Table 10-5. Reset Source Summary

Source	Type
Power on	Asynchronous
External <u>RESET</u> pin (not stop mode)	Synchronous
External <u>RESET</u> pin (during stop mode)	Asynchronous
Core Watchdog timer	Synchronous
On-chip watchdog timer	Synchronous
Loss of lock	Asynchronous
Software	Synchronous

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 asserted immediately to the system.

10.4.1.1 Power-On Reset

At power up, the reset controller asserts $\overline{\text{RSTOUT}}$. $\overline{\text{RSTOUT}}$ continues to be asserted until V_{DD} has reached a minimum acceptable level and, if PLL clock mode is selected, until the PLL achieves phase lock. Then after approximately another 512 cycles, $\overline{\text{RSTOUT}}$ is negated and the device begins operation.

10.4.1.2 External Reset

Asserting the external $\overline{\text{RESET}}$ for at least four rising $\overline{\text{FB_CLK}}$ edges causes the external reset request to be recognized and latched. The reset controller asserts $\overline{\text{RSTOUT}}$ for approximately 512 cycles after $\overline{\text{RESET}}$ is negated and the PLL has acquired lock. The device then exits reset and begins operation.

In low-power stop mode, the system clocks are stopped. Asserting the external $\overline{\text{RESET}}$ in stop mode causes an external reset to be recognized asynchronously.

10.4.1.3 On-chip Watchdog Timer Reset

An on-chip watchdog timer timeout causes timer reset request to be recognized and latched. If the $\overline{\text{RESET}}$ pin is negated and the PLL has acquired lock, the reset controller asserts $\overline{\text{RSTOUT}}$ for approximately 512 cycles. Then the device exits reset and begins operation.

10.4.1.4 Core Watchdog Timer Reset

A core watchdog timer timeout causes timer reset request to be recognized and latched. If the $\overline{\text{RESET}}$ pin is negated and the PLL has acquired lock, the reset controller asserts $\overline{\text{RSTOUT}}$ for approximately 512 cycles. Then the device exits reset and begins operation.

10.4.1.5 Loss-of-Lock Reset

This reset condition occurs when the PLL loses lock. The reset controller asserts $\overline{\text{RSTOUT}}$ for approximately 512 cycles after the PLL has acquired lock. The device then exits reset and resumes operation.

10.4.1.6 Software Reset

A software reset occurs when the RCR[SOFTRST] bit is set. If the $\overline{\text{RESET}}$ is negated and the PLL has acquired lock, the reset controller asserts $\overline{\text{RSTOUT}}$ for approximately 512 cycles. Then the device exits reset and resumes operation.

10.4.2 Reset Control Flow

The reset logic control flow is shown in [Figure 10-4](#). In this figure, the control state boxes have been numbered, and these numbers are referred to (within parentheses) in the flow description that follows. All cycle counts given are approximate.

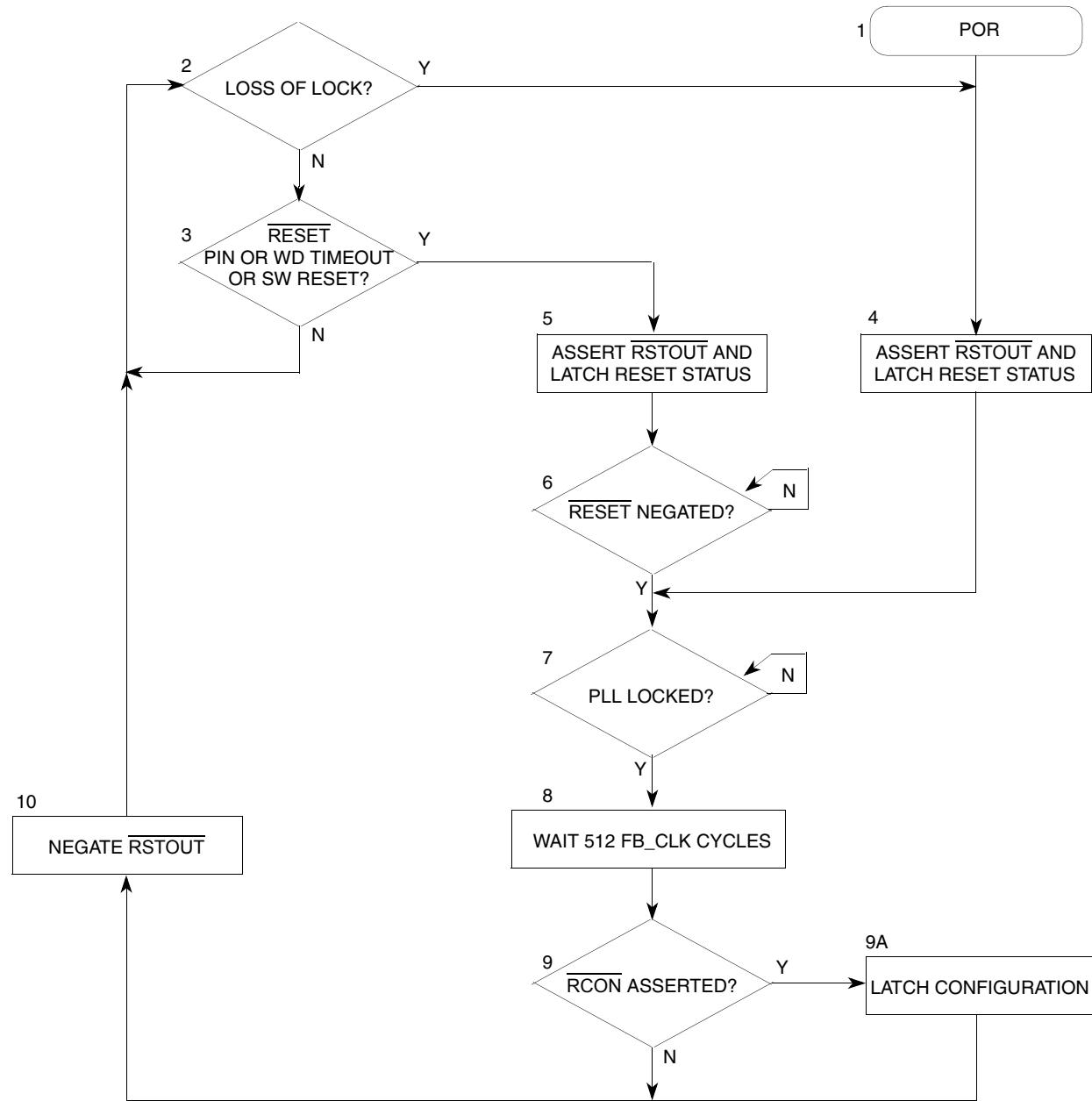


Figure 10-4. Reset Control Flow

10.4.2.1 Synchronous Reset Requests

In this discussion, the reference in parentheses refer to the state numbers in [Figure 10-4](#). All cycle counts given are approximate.

If the external RESET signal is asserted by an external device for at least four rising FB_CLK edges (3), if the watchdog timer times out, or if software requests a reset, the reset control logic latches the reset request internally. At this point the RSTOUT pin is asserted (5). The reset control logic waits until the RESET signal is negated (6) and for the PLL to attain lock (7) before waiting 512 FB_CLK cycles (8). The

reset control logic may latch the configuration according to the $\overline{\text{RCON}}$ signal level (9, 9A) before negating $\overline{\text{RSTOUT}}$ (10).

If the external $\overline{\text{RESET}}$ signal is asserted by an external device for at least four rising FB_CLK edges during the 512 count (8) or during the wait for PLL lock (7), the reset flow switches to (6) and waits for the $\overline{\text{RESET}}$ signal to be negated before continuing.

10.4.2.2 Internal Reset Request

If reset is asserted by an asynchronous internal reset source, such as loss of clock (1), the reset control logic asserts $\overline{\text{RSTOUT}}$ (4). The reset control logic waits for the PLL to attain lock (7) before waiting 512 FB_CLK cycles (8). Then the reset control logic may latch the configuration according to the $\overline{\text{RCON}}$ pin level (9, 9A) before negating $\overline{\text{RSTOUT}}$ (10).

If loss of lock occurs during the 512 count (8), the reset flow switches to (7) and waits for the PLL to lock before continuing.

10.4.2.3 Power-On Reset

When the reset sequence is initiated by power-on reset (1), the same reset sequence is followed as for the other asynchronous reset sources.

10.4.3 Concurrent Resets

This section describes the concurrent resets. As in the previous discussion references in parentheses refer to the state numbers in [Figure 10-4](#).

10.4.3.1 Reset Flow

If a power-on reset is detected during any reset sequence, the reset sequence starts immediately (1).

If the external $\overline{\text{RESET}}$ pin is asserted for at least four rising FB_CLK edges while waiting for PLL lock or the 512 cycles, the external reset is recognized. Reset processing switches to wait for the external $\overline{\text{RESET}}$ pin to negate (6).

If a loss-of-lock condition is detected during the 512 cycle wait, the reset sequence continues after a PLL lock (7).

10.4.3.2 Reset Status Flags

For a POR reset, the RSR[POR] bit is set, and all other RSR flags are cleared even if another type of reset condition is pending or concurrently asserted.

If other sources of reset are asserted after the RSR status bits have been latched (4 or 5), then the device will be held in reset (8) until all sources have negated and the subsequent sources will not be reflected in the RSR contents.

Chapter 11

System Control Module (SCM)

11.1 Introduction

This system control module (SCM) provides several control functions, including peripheral access control, a software core watchdog timer, and generic access error information for the processor core.

11.1.1 Overview

The SCM provides programmable access protections for both masters and peripherals. It allows the privilege level of a master to be overridden, forcing it to user-mode privilege, and allows masters to be designated as trusted or untrusted. Peripherals may be programmed to require supervisor privilege level for access, may restrict access to a trusted master only, and may be write-protected.

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.

NOTE

The core watchdog timer is available to provide compatibility with the watchdog timer implemented on previous ColdFire devices. However, there is a second watchdog timer available that has new features. See [Chapter 20, “Watchdog Timer Module,”](#) for more information.

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.

11.1.2 Features

The SCM includes these distinctive features:

- Access control registers
 - Master privilege register (MPR)
 - Peripheral access control registers (PACRs)
- System control registers
 - Core watchdog control register (CWCR) for watchdog timer control
 - Core watchdog service register (CWSR) to service watchdog timer
 - SCM interrupt status register (SCMISR) to service a bus fault or watchdog interrupt
 - Bus monitor timeout register (BMT)

- Core fault reporting registers

11.2 Memory Map/Register Definition

The memory map for the SCM registers is shown in [Table 11-1](#).

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 only supports 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 11-1. SCM Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC00_0000	Master Privilege Register (MPR1)	32	R/W	0x7000_0007	11.2.1/11-2
0xFC00_0020	Peripheral Access Control Register A (PACRA)	32	R/W	0x5440_0000	11.2.2/11-3
0xFC00_0024	Peripheral Access Control Register B (PACRB)	32	R/W	0x0000_4000	11.2.2/11-3
0xFC00_0028	Peripheral Access Control Register C (PACRC)	32	R/W	0x4440_0044	11.2.2/11-3
0xFC00_002C	Peripheral Access Control Register D (PACRD)	32	R/W	0x4440_4444	11.2.2/11-3
0xFC00_0040	Peripheral Access Control Register E (PACRE)	32	R/W	0x4444_4000	11.2.2/11-3
0xFC00_0044	Peripheral Access Control Register F (PACRF)	32	R/W	0x4440_0000	11.2.2/11-3
0xFC00_0054	Bus Monitor Timeout (BMT)	32	R/W	0x0000_0008	11.2.3/11-6
0xFC04_0013	Wakeup Control Register (WCR) ¹	8	R/W	0x00	8.2.1/8-2
0xFC04_0016	Core Watchdog Control Register (CWCR)	16	R/W	0x0000	11.2.4/11-7
0xFC04_001B	Core Watchdog Service Register (CWSR)	8	R/W	Undefined	11.2.5/11-8
0xFC04_001F	SCM Interrupt Status Register (SCMISR)	8	R/W	0x00	11.2.6/11-8
0xFC04_0070	Core Fault Address Register (CFADR)	32	R	0x0000_0000	11.2.7/11-9
0xFC04_0075	Core Fault Interrupt Enable Register (CFIER)	8	R/W	0x00	11.2.8/11-9
0xFC04_0076	Core Fault Location Register (CFLOC)	8	R	Undefined	11.2.9/11-10
0xFC04_0077	Core Fault Attributes Register (CFATR)	8	R	Undefined	11.2.10/11-10
0xFC04_0078	Core Fault Data Register (CFDTR)	32	R	Undefined	11.2.11/11-11

¹ The WCR register is described in [Chapter 8, “Power Management.”](#)

11.2.1 Master Privilege Register (MPR)

The MPR specifies three 4-bit fields defining the access privilege level associated with a bus master in the device to the various peripherals listed in [Table 11-4](#). The register provides one field per bus master.

Address: 0xFC00_0000 (MPR)																Access: User read/write															
R MPROT0 MPROT1 MPROT2																0 0															
W																0 0															
Reset 0 1 1 1																0 0															

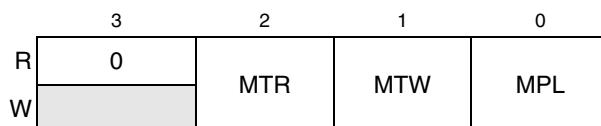
Figure 11-1. Master Privilege Register (MPR)

Each master is assigned depending on its connection to the various cross-bar switch master ports.

Table 11-2. MPROT n Assignments

Cross-bar Switch Port Number	MPROT n	Master
M0	MPROT0	ColdFire Core
M1	MPROT1	eDMA Controller
M2	MPROT2	FEC

The MPROT n field is defined as shown below.

**Figure 11-2. MPROT n Fields****Table 11-3. MPROT n Field Descriptions**

Field	Description
3	Reserved, should be cleared.
2 MTR	Master trusted for read. Determines whether the master is trusted for read accesses. 0 This master is not trusted for read accesses. 1 This master is trusted for read accesses.
1 MTW	Master trusted for writes. Determines whether the master is trusted for write accesses. 0 This master is not trusted for write accesses. 1 This master is trusted for write accesses.
0 MPL	Master privilege level. Determines how the privilege level of the master is determined. 0 Accesses from this master are forced to user-mode. 1 Accesses from this master are not forced to user-mode.

11.2.2 Peripheral Access Control Registers (PACRx)

Each of the peripherals has a four-bit PACR n field which defines the access levels supported by the given module. Eight PACRs are grouped together to form a 32-bit PACRx register (PACRA-PACRF).

Figure 11-3. Peripheral Access Control Register A (PACRA)

Figure 11-4. Peripheral Access Control Register B (PACRB)

Address: 0xFC00_0028 (PACRC)																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	PACR16				PACR17				PACR18				0 0 0 0				0 0 0 0				PACR21				PACR22				PACR23			
	W	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0
Reset	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0

Figure 11-5. Peripheral Access Control Register C (PACRC)

Address: 0xFC00_002C (PACRD)																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	PACR24				PACR25				PACR26				0 0 0 0				PACR28				PACR29				PACR30				PACR31			
	W	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0
Reset	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0

Figure 11-6. Peripheral Access Control Register D (PACRD)

Figure 11-7. Peripheral Access Control Register E (PACRE)

Figure 11-8. Peripheral Access Control Register F (PACRF)

Each peripheral is assigned to its PACR n field as shown below:

Table 11-4. PACR n Assignments

Slot Number	PACR n	Peripheral
0	PACR0	SCM (MPR & PACRs)
1	PACR1	Cross-bar switch
2	PACR2	FlexBus
12	PACR12	FEC
16	PACR16	SCM (CWT & Core Fault Registers)
17	PACR17	eDMA Controller
18	PACR18	Interrupt Controller 0
21	PACR21	Interrupt Controller IACK
22	PACR22	I ² C
23	PACR23	QSPI
24	PACR24	UART0
25	PACR25	UART1
26	PACR26	UART2
28	PACR28	DMA Timer 0
29	PACR29	DMA Timer 1
30	PACR30	DMA Timer 2
31	PACR31	DMA Timer 3
32	PACR32	PIT 0
33	PACR33	PIT 1
34	PACR34	Edge Port
35	PACR35	On-chip Watchdog Timer
36	PACR36	PLL
40	PACR40	CCM, Reset Controller, Power Management
41	PACR41	GPIO Module
42	PACR42	SDRAM Controller

The PACR n field is defined as shown below.

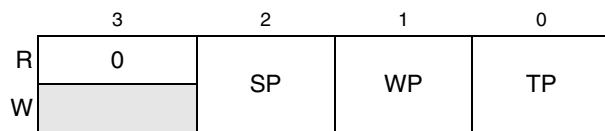
**Figure 11-9. PACR n Fields**

Table 11-5. PACR_n Field Descriptions

Field	Description
3	Reserved, should be cleared.
2 SP	Supervisor protect. Determines whether the peripheral requires supervisor privilege level for access. 0 This peripheral does not require supervisor privilege level for accesses. 1 This peripheral requires supervisor privilege level for accesses. The master privilege level must indicate supervisor access attribute, and the MPROT _n [MPL] control bit for the master must be set. If not, the access is terminated with an error response and no peripheral access is initiated.
1 WP	Write protect. Determines whether the peripheral allows write accesses 0 This peripheral allows write accesses. 1 This peripheral is write protected. If a write access is attempted, the access is terminated with an error response and no peripheral access is initiated.
0 TP	Trusted Protect. Determines whether the peripheral allows accesses from an untrusted master. 0 Accesses from an untrusted master are allowed. 1 Accesses from an untrusted master are not allowed. If an access is attempted by an untrusted master, the access is terminated with an error response and no peripheral access is initiated.

11.2.3 Bus Monitor Timeout Register (BMT)

Each AIPS controller implements a bus timeout monitor to insure that every bus cycle is properly terminated within a programmed period of time. The monitor continually checks for termination of each bus cycle and completes the cycle if there is no response when the programmed monitor cycle count is reached.

The monitor can be programmed from 8–1024 internal bus cycles under control of the BMT register. If the programmed timeout value is reached before a termination, the bus monitor completes the cycle with an error termination. Thus, the SCMSR[ECFEI] bit is set, and an interrupt to the interrupt controller is generated if the CFLOC[ECFEI] bit is set. At reset, the BMT is enabled with a maximum timeout value.

Address: 0xFC00_0054 (BMT)																																			Access: User read/write							
				BMT																																						
				BMT																																						
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	BME	BMT								
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	0	0	0	0	0	0	0	0	0	
	1	0	0	0																																						

Figure 11-10. Bus Monitor Timeout Register (BMT)**Table 11-6. BMT Field Descriptions**

Field	Description
31–4	Reserved, should be cleared.

Table 11-6. BMT Field Descriptions (continued)

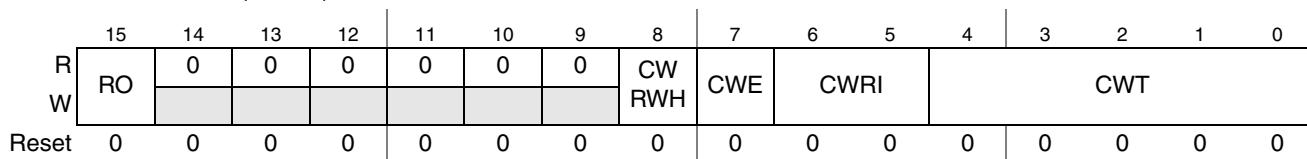
Field	Description
3 BME	Bus monitor timeout enable. 0 BMT disabled 1 BMT enabled
2–0 BMT	Bus monitor timeout period. Indicates the timeout period in internal bus cycles for the bus monitor. 000 1024 cycles 001 512 cycles 010 256 cycles 011 128 cycles 100 64 cycles 101 32 cycles 110 16 cycles 111 8 cycles

11.2.4 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.

Address: 0xFC04_0016 (CWCR)

Access: User read/write

**Figure 11-11. Core Watchdog Control Register (CWCR)****Table 11-7. 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, should be cleared.
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.

Table 11-7. CWCR Field Descriptions (continued)

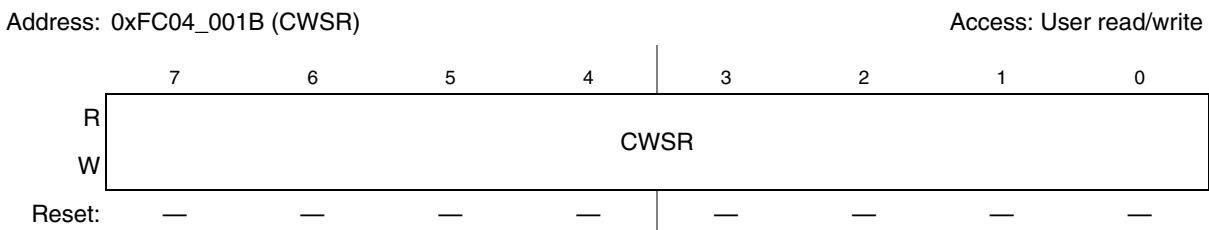
Field	Description
6–5 CWRI	Core watchdog reset/interrupt. 00 If a time-out occurs, the CWT generates an interrupt to the core. Refer to Chapter 14, “Interrupt Controller Module,” for details on setting its priority level. 01 The first time-out generates an interrupt to the processor, and if not serviced, 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 since CWCR[CWE] = 0 at reset. For CWCR[CWT] = n , time-out period = 2^n system clock cycles, where $n = 8–31$. If $n < 8$, then time-out period is forced to 2^8 .

11.2.5 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: first 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 11-12](#) illustrates the CWSR. At system reset, the contents of CWSR are uninitialized.

NOTE

If the CWT is enabled, then any write of a data value other than 0x55 or 0xAA will cause an immediate system reset, regardless of the value in the CWCR[CWRI] field.

**Figure 11-12. Core Watchdog Service Register (CWSR)**

11.2.6 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 will only be sent to the interrupt controller when the CFIER[ECFEI] bit is set.

Address: 0xFC04_001F (SCMISR)								Access: User read/write	
	7	6	5	4	3	2	1	0	
R	0	0	0	0	0	0	CFEI	CWIC	
W							w1c	w1c	
Reset:	0	0	0	0	0	0	0	0	0

Figure 11-13. SCM Interrupt Status Register (SCMISR)

Table 11-8. SCMISR Field Descriptions

Field	Description
7–2	Reserved, should be cleared.
1 CFEI	Core fault error interrupt flag. Indicates if a bus fault has occurred. 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 only enabled if CFLOC[ECFEI] is set. The interrupt request is negated by writing a 1 to this bit. Writing a 0 has no effect.
0 CWIC	Core watchdog interrupt flag. Indicates whether an CWT interrupt has occurred. 0 No CWT interrupt has occurred. 1 CWT interrupt has occurred. Writing a 1 clears this bit and negates the interrupt request. Writing a 0 has no effect.

11.2.7 Core Fault Address Register (CFADR)

The CFADR is a read-only register for indicating the address of the last core access which was terminated with an error response.

Address: 0xFC04_0070 (CFADR)																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																																
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 11-14. Core Fault Address Register (CFADR)

Table 11-9. CFADR Field Descriptions

Field	Description
31–0 ADDR	Indicates the faulting address of the last core access terminated with an error response.

11.2.8 Core Fault Interrupt Enable Register (CFIER)

The CFIER register is used to enable the system bus error interrupt. See [Chapter 14, “Interrupt Controller Module,”](#) for more information of the interrupt controller.

Address: 0xFC04_0075 (CFIER)								Access: User read/write	
	7	6	5	4	3	2	1	0	
R	0	0	0	0	0	0	0	0	ECFEI
W									
Reset:	0	0	0	0	0	0	0	0	

Figure 11-15. Core Fault Interrupt Enable Register (CFIER)

Table 11-10. CFIER Field Descriptions

Field	Description
7–1	Reserved, should 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.

11.2.9 Core Fault Location Register (CFLOC)

The read-only CFLOC register indicates the exact location within the device of the captured fault information.

Address: 0xFC04_0076 (CFLOC)								Access: User read-only	
	7	6	5	4	3	2	1	0	
R	LOC	0	0	0	0	0	0	0	
W									
Reset:	–	0	0	0	0	0	0	0	

Figure 11-16. Core Fault Location Register (CFLOC)

Table 11-11. CFLOC Field Descriptions

Field	Description
7 LOC	The location of the last captured fault. 0 Error occurred on the internal bus. 1 Error occurred within the core.
6–0	Reserved, should be cleared.

11.2.10 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.

Address: 0xFC04_0077 (CFATR)

Access: User read-only

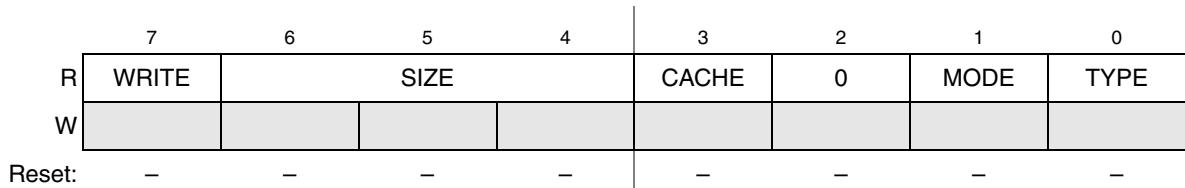


Figure 11-17. Core Fault Attributes Register (CFATR)

Table 11-12. 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, should 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

11.2.11 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 only valid for faulted internal bus write accesses, CFLOC[LOC] = 0.

Address: 0xFC04_0078 (CFDTR)

Access: User read-only

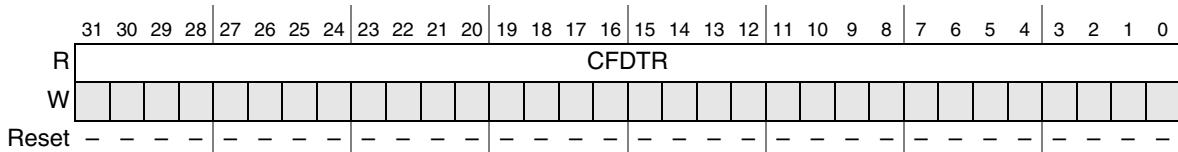


Figure 11-18. Core Fault Data Register (CFDTR)

Table 11-13. CFDTR Field Descriptions

Field	Description
31–0 CFDTR	Contains the data associated with the faulting access of the last internal bus write access. The register contains the data value taken directly from the write data bus.

11.3 Functional Description

11.3.1 Access Control

The SCM supports the traditional model of two privilege levels: supervisor and user. Typically, memory references with the supervisor attribute have total accessibility to all the resources in the system, while user mode references cannot access system control and configuration registers. In many systems, the operating system executes in supervisor mode, while application software executes in user mode.

The SCM further partitions the access control functions into two parts: one control register defines the privilege level associated with each bus master (MPR), and another set of control registers define the access levels associated with the peripheral modules (PACRx).

Each bus transaction targeted for the peripheral space is first checked to see if its privilege rights allow access to the given memory space. If the privilege rights are correct, the access proceeds on the internal bus. If the privilege rights are insufficient for the targeted memory space, the transfer is immediately aborted and terminated with an exception, and the targeted module not accessed.

11.3.2 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:

1. An interrupt may be generated to the core.
2. An immediate system reset.
3. 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.
4. 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 be executed between the two writes. This allows interrupts and exceptions to occur, if necessary, between the two writes.

NOTE

If the CWT is enabled, then any write of a data value other than 0x55 or 0xAA will cause 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, there is a write-once control bit in the CWCR that sets the CWCR to read-only to prevent accidental updates to this control register from changing the desired system configuration. Once 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.

11.3.3 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 that is terminated with an error.

Chapter 12

Cross-Bar Switch Module (XBS)

12.1 Overview

This section provides information on the layout, configuration, and programming of the cross-bar switch. The cross-bar switch connects the bus masters and bus slaves using a cross-bar switch structure. This structure allows bus masters to simultaneously access different bus slaves 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 MCF5208 devices have up to three masters and three slaves (3Mx3S) that utilize on cross-bar switch. The three masters are the ColdFire core, eDMA controller, and FEC. The slaves are SDRAM controller, FlexBus, SRAM controller, and the peripheral bus.

Figure 12-1 is a block diagram of the MCF5208 family bus architecture showing the cross-bar switch configuration.

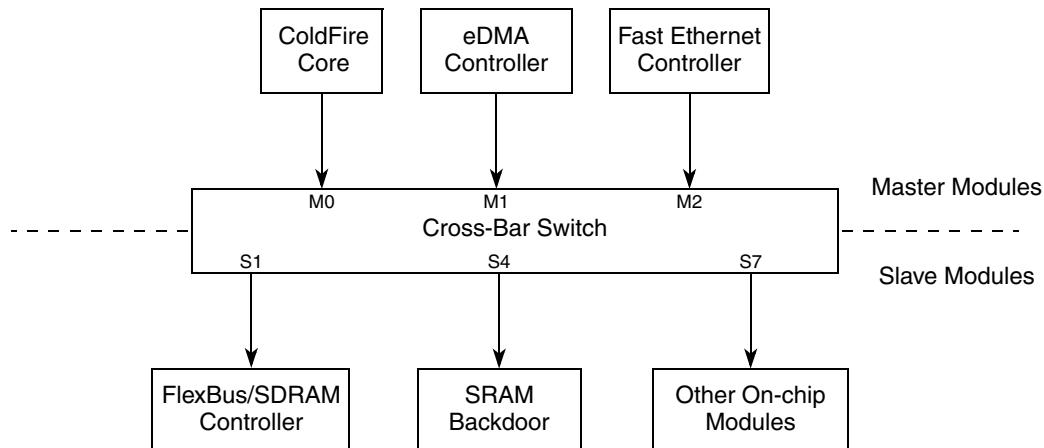


Figure 12-1. Bus Architecture Block Diagram

The modules are assigned to the numbered ports on the cross-bar switch as follows in the below table.

Table 12-1. Cross-bar Switch Master/Slave Assignments

Cross-bar Port	Module
Master Modules	
Master 0 (M0)	ColdFire Core
Master 1 (M1)	eDMA Controller
Master 2 (M2)	Fast Ethernet Controller

Table 12-1. Cross-bar Switch Master/Slave Assignments (continued)

Cross-bar Port	Module
Slave Modules	
Slave 1 (S1)	Flexbus/SDRAM Controller
Slave 4 (S4)	Internal SRAM Backdoor
Slave 7 (S7)	Other On-chip Peripherals

12.2 Features

The cross-bar switch includes these distinctive features:

- Symmetric cross-bar 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

12.3 Modes of Operation

The cross-bar switch provides two arbitration modes: fixed or round-robin. The arbitration mode 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 switch-path to that slave. A higher priority master will block access to a given slave from a lower priority master as long as the higher priority master continuously requests that slave. See [Section 12.5.1.1, “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 12.5.1.2, “Round-Robin Priority Operation.”](#)

12.4 Memory Map / Register Definition

There are two registers that reside in each slave port of the cross-bar switch. Read and write transfers both 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 cross-bar switch. See [Section 11.2.6, “SCM Interrupt Status Register \(SCMISR\).”](#)

The slave registers also feature a bit that, when set, will prevent the registers from being written. The registers will still be readable, but future write attempts will have no effect on the registers and will be terminated with a bus error response to the master initiating the write. The core, for example, will take a bus error interrupt.

Table 12-2 shows the memory map for the cross-bar switch program-visible registers.

Table 12-2. XBS Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC00_4100	Priority Register Slave 1 (XBS_PRS1)	32	R/W	0x7000_0210	12.4.1/12-3
0xFC00_4110	Control Register Slave 1 (XBS_CRS1)	32	R/W	0x0000_0000	12.4.2/12-4
0xFC00_4400	Priority Register Slave 4 (XBS_PRS4)	32	R/W	0x7000_0210	12.4.1/12-3
0xFC00_4410	Control Register Slave 4 (XBS_CRS4)	32	R/W	0x0000_0000	12.4.2/12-4
0xFC00_4700	Priority Register Slave 7 (XBS_PRS7)	32	R/W	0x7000_0210	12.4.1/12-3
0xFC00_4710	Control Register Slave 7 (XBS_CRS7)	32	R/W	0x0000_0000	12.4.2/12-4

12.4.1 XBS Priority Registers (XBS_PRS n)

The priority registers (XBS_PRS n) set the priority of each master port on a per slave port basis and resides in each slave port. The priority register can only be accessed with 32-bit accesses. Once the XBS_CRS n [RO] bit has been set, the XBS_PRS n register can only be read; attempts to write to it will have no effect on XBS_PRS n and result in a bus error response to the master initiating the write. For proper operation, XBS_CRS n [31:28] should be set to 0x7.

Additionally, no two available master ports may be programmed with the same priority level. Attempts to program two or more available masters with the same priority level will result in a bus error response (see Section 11.2.6, “SCM Interrupt Status Register (SCMISR)”) and the XBS_PRS n will not be updated.

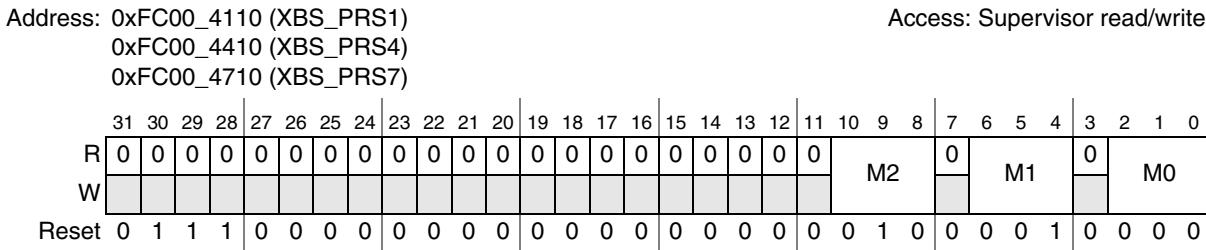


Figure 12-2. XBS Priority Registers Slave n (XBS_PRS n)

Table 12-3. XBS_PRS n Field Descriptions

Field	Description
31–28	Reserved, should be set to 0x7.
27–11	Reserved, should be cleared.

Table 12-3. XBS_PRS n Field Descriptions (continued)

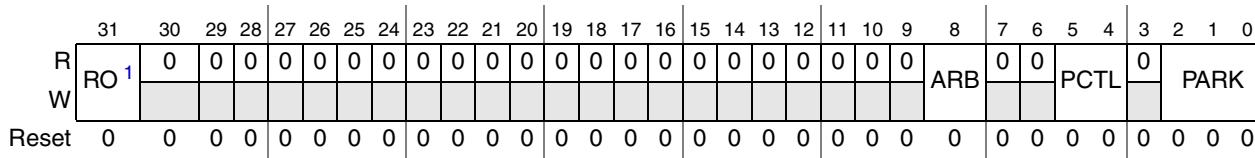
Field	Description
10–8 M2	Master 2 (FEC) priority. This bit sets the arbitration priority for this port on the associated slave port.
000	This master has level 1 (highest) priority when accessing the slave port.
001	This master has level 2 priority when accessing the slave port.
010	This master has level 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.
7	Reserved, should be cleared.
6–4 M1	Master 1 (eDMA) priority. See M2 description.
3	Reserved, should be cleared.
2–0 M0	Master 0 (ColdFire core) priority. See M2 description.

12.4.2 XBS Control Registers (XBS_CRSn)

The XBS control registers (XBS_CRS n) control several features of each slave port. The XBS_CRS n can only be accessed with 32-bit accesses. Once the XBS_CRS n [RO] bit has been set, the XBS_CRS n can only be read; attempts to write to it will have no effect on the XBS_CRS n and result in an error response.

Address: 0xFC00_4110 (XBS_CRS1)
0xFC00_4410 (XBS_CRS4)
0xFC00_4710 (XBS_CRS7)

Access: Supervisor read/write



¹ Once this bit is set, only a hardware reset will clear it.

Figure 12-3. XBS Control Registers Slave n (XBS_CRS n)

Table 12-4. XBS_CRS*n* Field Descriptions

Field	Description
31 RO	Read Only. This bit is used to force both of the slave port's registers (XBS_CRS n & XBS_PRS n) to be read only. Once set it can only be cleared by hardware reset. 0 Both of the slave port's registers can be written. 1 Both of the slave port's registers are read only and cannot be written (attempted writes have no effect and result in a bus error response).
30–9	Reserved, should be cleared.
8 ARB	Arbitration Mode. These bits are used to select the arbitration policy for the slave port. 0 Fixed Priority. 1 Round Robin (rotating) Priority.

Table 12-4. XBS_CRSn Field Descriptions (continued)

Field	Description
7–6	Reserved, should be cleared.
5–4 PCTL	Parking Control. These bits determine the parking control used by this slave port. The low-power park feature can result in an overall power savings if the slave port is not saturated; however, it will force an extra clock of latency whenever any master tries to access it when it is not in use because it will not be parked on any master. 00 When no master is making a request, the arbiter will park the slave port on the master port defined by the PARK bit field. 01 When no master is making a request, the arbiter will park the slave port on the last master to be in control of the slave port. 10 When no master is making a request, the arbiter will park the slave port on no master and will drive all outputs to a constant safe state. 11 Reserved
3	Reserved, should be cleared.
2–0 PARK	This bit is used to determine which master port this slave port parks on when no masters are actively making requests and the PCTL bits are set to 00. 000 Park on master port M0 (ColdFire Core) 001 Park on master port M1 (eDMA Controller) 010 Park on master port M2 (FEC) Else Reserved

12.5 Functional Description

12.5.1 Arbitration

The cross-bar 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.

12.5.1.1 Fixed-Priority Operation

When operating in fixed-priority mode, each master is assigned a unique priority level in the XBS_PRSn (priority registers). If two masters both request access to a slave port, the master with the highest priority in the selected priority register will gain control over the slave port.

Any time a master makes a request to a slave port, the slave port checks to see if the new requesting master's priority level is higher than that of the master that currently has control over the slave port (unless the slave port is in a parked state). The slave port does an arbitration check at every bus transfer boundary to ensure that the proper master (if any) has control of the slave port.

If the new requesting master's priority level is higher than that of the master that currently has control of the slave port the new requesting master will be 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 will have to wait until the end of the burst transfer or locked transfer before it will be granted control of the slave port.

If the new requesting master's priority level is lower than that of the master that currently has control of the slave port, the new requesting master will be forced to wait until the current master either:

- a) Runs an IDLE cycle, or
- b) Runs a non-IDLE cycle to a location other than the current slave port.

12.5.1.2 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 whose priority is the highest based on this comparison will be granted control over the slave port at the next bus transfer boundary.

Once 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 will be granted access to the slave port at the next transfer boundary.

Parking may still be used in a round-robin mode, but will not affect the round-robin pointer unless the parked master actually performs a transfer. Handoff will occur 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 will be reset to point at master port 0, giving it the highest priority.

12.5.1.3 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 cross-bar switch will respond with a bus error (refer to [Section 11.2.6, “SCM Interrupt Status Register \(SCMISR\)”](#)) and the registers will not be updated.

12.6 Initialization / Application Information

No initialization is required by or for the cross-bar switch. Hardware reset ensures all the register bits used by the cross-bar switch are properly initialized to a valid state. Settings and priorities can be programmed if needed.

Chapter 13

General Purpose I/O Module

13.1 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 13-1](#) is a block diagram of the device ports. Note that the GPIO functionality of the port IRQ pins is selected by the edge port module. They are shown in the below figure only for completeness.

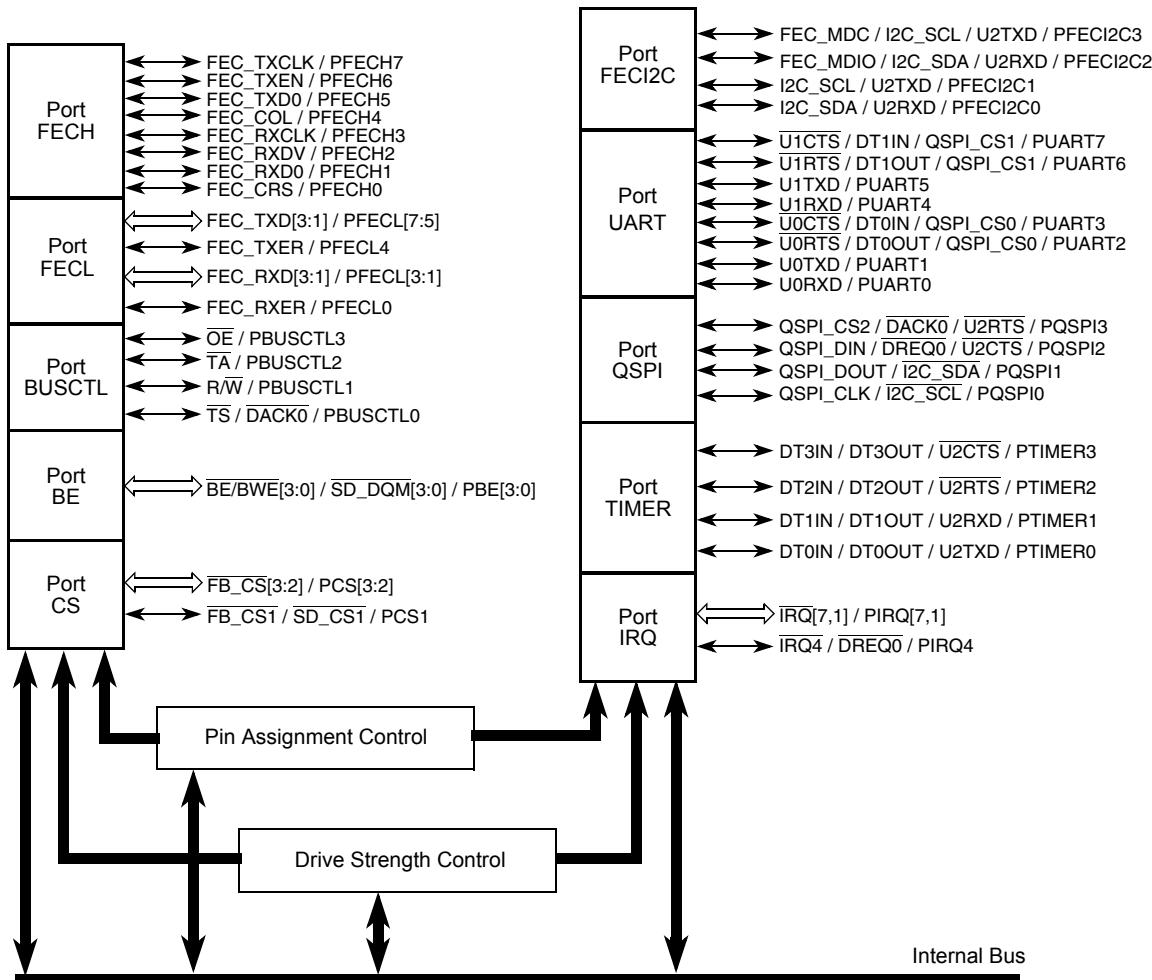


Figure 13-1. Ports Module Block Diagram

13.1.1 Overview

The GPIO module controls the configuration for various external pins, including those used for:

- External bus accesses
- External chip selection
- Ethernet data and control
- I²C serial control
- QSPI
- 32-bit DMA timers
- FEC
- UART

13.1.2 Features

The ports module includes these distinctive features:

- Control of primary function use
 - On all supported GPIO ports
 - On pins whose GPIO is not supported by ports module: $\overline{\text{IRQ4}}$
- 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

13.2 External Signal Description

The GPIO module controls the functionality of several external pins. These pins are listed in [Table 13-2](#) under the GPIO column.

After reset ports BUSCTL, BE and CS are configured for external memory. They are available for the user as GPIO if the corresponding registers are set appropriately. All other ports default to GPIO after reset.

NOTE

In this table and throughout this document a single signal within a group is designated without square brackets (i.e., A23), while designations for multiple signals within a group use brackets (i.e., A[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. Pins that are muxed with GPIO will default to their GPIO functionality.

Table 13-1. MCF5207/8 Signal Information and Muxing

Signal Name	GPIO	Alternate 1	Alternate 2	Dir. ¹	MCF5207 144 LQFP	MCF5207 144 MAPBGA	MCF5208 160 QFP	MCF5208 196 MAPBGA
Reset								
$\overline{\text{RESET}}^2$	—	—	—	I	82		90	J14
$\overline{\text{RSTOUT}}$	—	—	—	O	74		82	N14
Clock								
EXTAL	—	—	—	I	78		86	L14
XTAL	—	—	—	O	80		88	K14
FB_CLK	—	—	—	O	34		40	N1

Table 13-1. MCF5207/8 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir. ¹	MCF5207 144 LQFP	MCF5207 144 MAPBGA	MCF5208 160 QFP	MCF5208 196 MAPBGA
Mode Selection								
$\overline{\text{RCON}}^2$	—	—	—	I	144		160	C3
DRAMSEL	—	—	—	I	79		87	K11
FlexBus								
A[23:22]	—	$\overline{\text{FB_CS}}[5:4]$	—	O	118, 117		126, 125	B11, A11
A[21:16]	—	—	—	O	116–114, 112, 108, 107		124, 123, 122, 120, 116, 115	B12, A12, A13, B13, B14, C13
A[15:14]	—	SD_BA[1:0]	—	O	106, 105		114, 113	C14, D12
A[13:11]	—	SD_A[13:11]	—	O	104–102		112, 111, 110	D13, D14, E11
A10	—	—	—	O	101		109	E12
A[9:0]	—	SD_A[9:0]	—	O	100–91		108–99	E13, E14, F11–14, G11–14
D[31:16]	—	SD_D[31:16] ³	—	O	21–28, 40–47		27–34, 46–53	J4–1, K4–1, M3, N3, M4, N4, P4, L5, M5, N5
D[15:0]	—	FB_D[31:16] ³	—	O	8–15, 51–58		16–23, 57–64	F3–1, G4–1, H1, N6, P6, L7, M7, N7, P7, N8, P8
BE/BWE[3:0]	PBE[3:0]	$\overline{\text{SD_DQM}}[3:0]$	—	O	18, 20, 48, 50		26, 54, 24, 56	H2, P5, H4, M6
$\overline{\text{OE}}$	PBUSCTL3	—	—	O	60		66	M8
$\overline{\text{TA}}^2$	PBUSCTL2	—	—	I	90		98	H14
R/W	PBUSCTL1	—	—	O	59		65	L8
$\overline{\text{TS}}$	PBUSCTL0	DACK0	—	O	4		12	E3
Chip Selects								
$\overline{\text{FB_CS}}[3:2]$	PCS[3:2]	—	—	O	119, 120		—	C11, A10
$\overline{\text{FB_CS1}}$	PCS1	$\overline{\text{SD_CS1}}$	—	O	121		127	B10
$\overline{\text{FB_CS0}}$	—	—	—	O	122		128	C10
SDRAM Controller								

Table 13-1. MCF5207/8 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir. ¹	MCF5207 144 LQFP	MCF5207 144 MAPBGA	MCF5208 160 QFP	MCF5208 196 MAPBGA
SD_A10	—	—	—	O	37		43	N2
SD_CKE	—	—	—	O	6		14	E1
SD_CLK	—	—	—	O	31		37	L1
<u>SD_CLK</u>	—	—	—	O	32		38	M1
<u>SD_CS0</u>	—	—	—	O	7		15	F4
SD_DQS[3:2]	—	—	—	O	19, 49		25, 55	H3, L6
SD_SCAS	—	—	—	O	38		44	P2
<u>SD_SRAS</u>	—	—	—	O	39		45	P3
SD_SDR_DQS	—	—	—	O	29		35	L3
<u>SD_WE</u>	—	—	—	O	5		13	E2
External Interrupts Port⁴								
<u>IRQ7²</u>	PIRQ7 ²	—	—	I	134		142	C7
<u>IRQ4²</u>	PIRQ4 ²	<u>DREQ0²</u>	—	I	133		141	D7
<u>IRQ1²</u>	PIRQ1 ²	—	—	I	132		140	D8
FEC								
FEC_MDC	PFECI2C3	I2C_SCL ²	U2TXD	O	—	—	148	D6
FEC_MDIO	PFECI2C2	I2C_SDA ²	U2RXD	I/O	—	—	147	C6
FEC_TXCLK	PFECH7	—	—	I	—	—	157	B3
FEC_TXEN	PFECH6	—	—	O	—	—	158	A2
FEC_RXD0	PFECH5	—	—	O	—	—	3	B1
FEC_COL	PFECH4	—	—	I	—	—	7	D3
FEC_RXCLK	PFECH3	—	—	I	—	—	154	B4
FEC_RXDV	PFECH2	—	—	I	—	—	153	A4
FEC_RXD0	PFECH1	—	—	I	—	—	152	D5
FEC_CRS	PFECH0	—	—	I	—	—	8	D2
FEC_RXD[3:1]	PFECL[7:5]	—	—	O	—	—	6–4	C1, C2, B2
FEC_TXER	PFECL4	—	—	O	—	—	156	A3
FEC_RXD[3:1]	PFECL[3:1]	—	—	I	—	—	149–151	A5, B5, C5
FEC_RXER	PFECL0	—	—	I	—	—	155	C4
I²C								
I2C_SDA ²	PFECI2C0 ²	U2RXD ²	—	I/O	—	—	—	D1

Table 13-1. MCF5207/8 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir. ¹	MCF5207 144 LQFP	MCF5207 144 MAPBGA	MCF5208 160 QFP	MCF5208 196 MAPBGA
I2C_SCL ²	PFEI2C1 ²	U2TXD ²	—	I/O	—	—	—	E4
DMA								
DACK0 and DREQ0 do not have a dedicated bond pads. Please refer to the following pins for muxing: TS and QSPI_CS2 for DACK0, IRQ4 and QSPI_DIN for DREQ0.								
QSPI								
QSPI_CS2	PQSPI3	DACK0	U2RTS	O	126		132	D10
QSPI_CLK	PQSPI0	I2C_SCL ²	—	O	127		133	A9
QSPI_DOUT	PQSPI1	I2C_SDA ²	—	O	128		134	B9
QSPI_DIN	PQSPI2	DREQ0 ²	U2CTS	I	129		135	C9
Note: The QSPI_CS1 and QSPI_CS0 signals are available on the U1CTS, U1RTS, U0CTS, or U0RTS pins for the 196 and 160-pin packages.								
UARTs								
U1CTS	PUARTL7	—	—	I	139		—	—
U1RTS	PUARTL6	—	—	O	142		—	—
U1CTS	PUARTL7	DT1IN	QSPI_CS1	I	—	—	136	D9
U1RTS	PUARTL6	DT1OUT	QSPI_CS1	O	—	—	137	C8
U1TXD	PUARTL5	—	—	O	131		139	A8
U1RXD	PUARTL4	—	—	I	130		138	B8
U0CTS	PUARTL3	—	—	I	140		—	—
U0RTS	PUARTL2	—	—	O	141		—	—
U0CTS	PUARTL3	DT0IN	QSPI_CS0	I	—	—	76	N12
U0RTS	PUARTL2	DT0OUT	QSPI_CS0	O	—	—	77	P12
U0TXD	PUARTL1	—	—	O	71		79	P13
U0RXD	PUARTL0	—	—	I	70		78	N13
Note: The UART2 signals are multiplexed on the DMA Timers, QSPI, FEC, and I2C pins.								
DMA Timers								
DT3IN	PTIMER3	DT3OUT	U2CTS	I	135		143	B7
DT2IN	PTIMER2	DT2OUT	U2RTS	I	136		144	A7
DT1IN	PTIMER1	DT1OUT	U2RXD	I	137		145	A6
DT0IN	PTIMER0	DT0OUT	U2TXD	I	138		146	B6
BDM/JTAG⁵								

Table 13-1. MCF5207/8 Signal Information and Muxing (continued)

Signal Name	GPIO	Alternate 1	Alternate 2	Dir. ¹	MCF5207 144 LQFP	MCF5207 144 MAPBGA	MCF5208 160 QFP	MCF5208 196 MAPBGA
JTAG_EN ⁶	—	—	—	I	83		91	J13
DSCLK	—	TRST ²	—	I	76		84	L12
PSTCLK	—	TCLK ²	—	O	64		70	P9
<u>BKPT</u>	—	TMS ²	—	I	75		83	M14
DSI	—	TDI ²	—	I	77		85	K12
DSO	—	TDO	—	O	69		75	M12
DDATA[3:0]	—	—	—	O	—		—	P11, N11, M11, P10
PST[3:0]	—	—	—	O	—		—	N10, M10, L10, L9
ALLPST	—	—	—	O	67	—	73	—
Test								
TEST ⁶	—	—	—	I	109		117	C12
PLL_TEST	—	—	—	I	—		—	M13
Power Supplies								
EVDD	—	—	—		1, 63, 66, 72, 81, 87, 125		2, 9, 69, 72, 80, 89, 95, 131	E5–7, F5, F6, G5, H10, J9, J10, K8–10, K13, M9
IVDD	—	—	—		30, 68, 84, 113, 143		36, 74, 92, 121, 159	J12, D4, D11, H11, L4, L11,
PLL_VDD	—	—	—		86		94	H13
SD_VDD	—	—	—		3, 17, 33, 35, 61, 89, 110, 123		11, 39, 41, 67, 97, 118, 129	E8–10, F9, F10, G10, H5, J5, J6, K5–7, L2
VSS	—	—	—		2, 16, 36, 62, 65, 73, 88, 111, 124		1, 10, 42, 68, 71, 81, 96, 119, 130	A1, A14, F7–8, G6–9, H6–9, J7–8, L13, M2, N9, P1, P14
PLL_VSS	—	—	—		85		93	H12

¹ Refers to pin's primary function.

- ² Pull-up enabled internally on this signal for this mode.
- ³ Alternate functionality selected by asserting the DRAMSEL signal (DDR mode). The GPIO module is not responsible for assigning these pins.
- ⁴ GPIO functionality is determined by the edge port module. The GPIO module is only responsible for assigning the alternate functions.
- ⁵ If JTAG_EN is asserted, these pins default to Alternate 1 (JTAG) functionality. The GPIO module is not responsible for assigning these pins.
- ⁶ Pull-down enabled internally on this signal for this mode.

Refer to the [Chapter 2, “Signal Descriptions,”](#) for more detailed descriptions of these pins and other pins not controlled by the ports module. The function of most of the pins (primary function, GPIO, etc.) is determined by the ports module pin assignment registers.

It should be noted from [Table 13-2](#) that 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, this type of programming should be avoided for input functions to prevent unexpected behavior. All multiple-pin functions are listed in [Table 13-2](#).

Table 13-2. Multiple-Pin Functions

Function	Direction	Associated Pins
$\overline{\text{DACK}_0}$	I	QSPI_CS2, $\overline{\text{TS}}$
$\overline{\text{DREQ}_0}$	I	$\overline{\text{IRQ}_4}$, QSPI_DIN
I2C_SCL	I/O	I2C_SCL, QSPI_CLK, FEC_MDC
I2C_SDA	I/O	I2C_SDA, QSPI_DOUT, FEC_MDIO
$\overline{\text{U2CTS}}$	I	DT3IN, QSPI_DIN
$\overline{\text{U2RTS}}$	O	DT2IN, QSPI_CS2
U2TXD	O	DT0IN, FEC_MDC, I2C_SCL
U2RXD	I	DT1IN, FEC_MDIO, I2C_SDA
DT0IN	I	DT0IN, $\overline{\text{U0CTS}}$
DT0OUT	O	DT0IN, $\overline{\text{U0RTS}}$
DT1IN	I	DT1IN, $\overline{\text{U1CTS}}$
DT1OUT	O	DT1IN, $\overline{\text{U1RTS}}$
QSPI_CS0	O	$\overline{\text{U0CTS}}$, $\overline{\text{U0RTS}}$
QSPI_CS1	O	$\overline{\text{U1CTS}}$, $\overline{\text{U1RTS}}$

13.3 Memory Map/Register Definition

[Table 13-3](#) summarizes all the registers in the ports address space.

Table 13-3. GPIO Module Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
Port Output Data Registers					
0xFC0A_4000	PODR_BUSCTL	8	R/W	0x0F	13.3.1/13-11
0xFC0A_4001	PODR_BE	8	R/W	0x0F	13.3.1/13-11
0xFC0A_4002	PODR_CS	8	R/W	0x0E	13.3.1/13-11
0xFC0A_4003	PODR_FECI2C	8	R/W	0x0F	13.3.1/13-11
0xFC0A_4004	PODR_QSPI	8	R/W	0x0F	13.3.1/13-11
0xFC0A_4005	PODR_TIMER	8	R/W	0x0F	13.3.1/13-11
0xFC0A_4006	PODR_UART	8	R/W	0xFF	13.3.1/13-11
0xFC0A_4007	PODR_FECH	8	R/W	0xFF	13.3.1/13-11
0xFC0A_4008	PODR_FECL	8	R/W	0xFF	13.3.1/13-11
Port Data Direction Registers					
0xFC0A_400C	PDDR_BUSCTL	8	R/W	0x00	13.3.2/13-12
0xFC0A_400D	PDDR_BE	8	R/W	0x00	13.3.2/13-12
0xFC0A_400E	PDDR_CS	8	R/W	0x00	13.3.2/13-12
0xFC0A_400F	PDDR_FECI2C	8	R/W	0x00	13.3.2/13-12
0xFC0A_4010	PDDR_QSPI	8	R/W	0x00	13.3.2/13-12
0xFC0A_4011	PDDR_TIMER	8	R/W	0x00	13.3.2/13-12
0xFC0A_4012	PDDR_UART	8	R/W	0x00	13.3.2/13-12
0xFC0A_4013	PDDR_FECH	8	R/W	0x00	13.3.2/13-12
0xFC0A_4014	PDDR_FECL	8	R/W	0x00	13.3.2/13-12
Port Pin Data/Set Data Registers					
0xFC0A_4018	PPDSR_BUSCTL	8	R/W	See Section	13.3.3/13-13
0xFC0A_4019	PPDSR_BE	8	R/W	See Section	13.3.3/13-13
0xFC0A_401A	PPDSR_CS	8	R/W	See Section	13.3.3/13-13
0xFC0A_401B	PPDSR_FECI2C	8	R/W	See Section	13.3.3/13-13
0xFC0A_401C	PPDSR_QSPI	8	R/W	See Section	13.3.3/13-13
0xFC0A_401D	PPDSR_TIMER	8	R/W	See Section	13.3.3/13-13
0xFC0A_401E	PPDSR_UART	8	R/W	See Section	13.3.3/13-13
0xFC0A_401F	PPDSR_FECH	8	R/W	See Section	13.3.3/13-13
0xFC0A_4020	PPDSR_FECL	8	R/W	See Section	13.3.3/13-13
Port Clear Output Data Registers					

Table 13-3. GPIO Module Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_4024	PCLRR_BUSCTL	8	W	0x00	13.3.4/13-14
0xFC0A_4025	PCLRR_BE	8	W	0x00	13.3.4/13-14
0xFC0A_4026	PCLRR_CS	8	W	0x00	13.3.4/13-14
0xFC0A_4027	PCLRR_FECI2C	8	W	0x00	13.3.4/13-14
0xFC0A_4028	PCLRR_QSPI	8	W	0x00	13.3.4/13-14
0xFC0A_4029	PCLRR_TIMER	8	W	0x00	13.3.4/13-14
0xFC0A_402A	PCLRR_UART	8	W	0x00	13.3.4/13-14
0xFC0A_402B	PCLRR_FECH	8	W	0x00	13.3.4/13-14
0xFC0A_402C	PCLRR_FECL	8	W	0x00	13.3.4/13-14
Pin Assignment Registers					
0xFC0A_4030	PAR_BUSCTL	8	R/W	0x1F	13.3.5.1/13-16
0xFC0A_4031	PAR_BE	8	R/W	0x0F	13.3.5.2/13-16
0xFC0A_4032	PAR_CS	8	R/W	0x0F	13.3.5.3/13-17
0xFC0A_4033	PAR_FECI2C	8	R/W	0x00	13.3.5.4/13-17
0xFC0A_4034	PAR_QSPI	8	R/W	0x00	13.3.5.5/13-18
0xFC0A_4035	PAR_TIMER	8	R/W	0x00	13.3.5.6/13-19
0xFC0A_4036	PAR_UART	8	R/W	0x0000	13.3.5.7/13-19
0xFC0A_4038	PAR_FEC	8	R/W	0x00	13.3.5.8/13-21
0xFC0A_4039	PAR_IRQ	8	R/W	0x00	13.3.5.9/13-22
Mode Select Control Registers					
0xFC0A_403A	MSCR_FLEXBUS	8	R/W	0xFF	13.3.6/13-22
0xFC0A_403B	MSCR_SDRAM	8	R/W	0x3F	13.3.7/13-23
Drive Strength Control Registers					
0xFC0A_403C	DSCR_I2C	8	R/W	See Section	13.3.8.1/13-24
0xFC0A_403D	DSCR_MISC	8	R/W	See Section	13.3.8.2/13-241 3.3.8/13-24
0xFC0A_403E	DSCR_FEC	8	R/W	See Section	13.3.8.3/13-251 3.3.8/13-24
0xFC0A_403F	DSCR_UART	8	R/W	See Section	13.3.8.4/13-251 3.3.8/13-25
0xFC0A_4040	DSCR_QSPI	8	R/W	See Section	13.3.8.5/13-261 3.3.8/13-24

13.3.1 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 each eight bits wide, but not all ports use all eight bits. The register definitions for all ports are shown in the below figures. The PODR_x registers are read/write. At reset, all implemented bits in the PODR_x registers are set. Reserved bits always remain cleared.

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 the PPDSDR_x register. To clear bits in a PODR_x register, clear the PODR_x bits, or clear the corresponding bits in the PCLRR_x register.

Address:	0xFC0A_4000 (PODR_BUSCTL) 0xFC0A_4001 (PODR_BE) 0xFC0A_4003 (PODR_FECl2C) 0xFC0A_4004 (PODR_QSPI) 0xFC0A_4005 (PODR_TIMER)								Access:	User read/write			
										PODR _x			
Reset:	0 0 0 0									1 1 1 1	0	1	1

Figure 13-2. Port x Output Data Registers (PODR_x)

Address:	0xFC0A_4002 (PODR_CS)								Access:	User read/write			
										PODR_CS			
Reset:	0 0 0 0									1 1 1 1	0	1	0

Figure 13-3. Port CS Output Data Registers (PODR_CS)

Address:	0xFC0A_4006 (PODR_UART) 0xFC0A_4007 (PODR_FECH) 0xFC0A_4008 (PODR_FECL)								Access:	User read/write			
										PODR _x			
Reset:	1 1 1 1									1 1 1 1	0	1	1

Figure 13-4. Port x Output Data Registers (PODR_x)

Table 13-4. PODR_x Field Descriptions

Field	Description
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

Note: See above figures for bit field positions.

13.3.2 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_x registers are each eight bits wide, but not all ports use all eight bits. The register definitions for all ports are shown in the figures below.

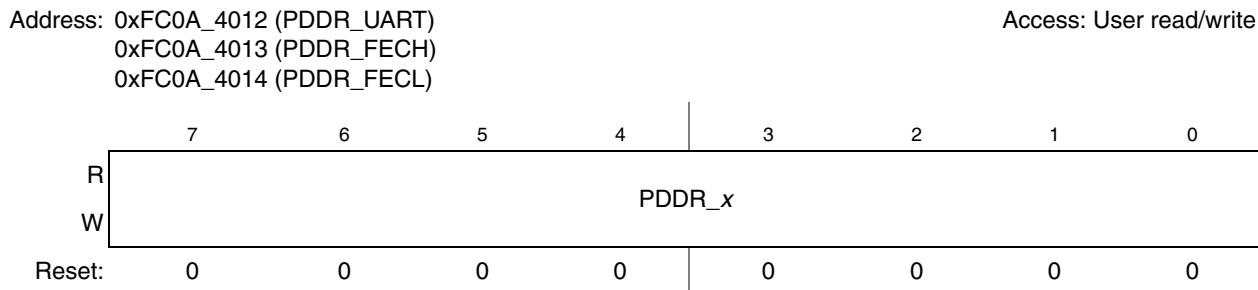
The PDDRs are read/write. At reset, all bits in the PDDRs are cleared. Setting any bit in a PDDR_x register configures the corresponding port pin as an output. Clearing any bit in a PDDR_x register configures the corresponding pin as an input.

Address: 0xFC0A_400C (PDDR_BUSCTL) 0xFC0A_400D (PDDR_BE) 0xFC0A_400F (PDDR_FECI2C) 0xFC0A_4010 (PDDR_QSPI) 0xFC0A_4011 (PDDR_TIMER)	Access: User read/write																														
<table> <tr> <td>R</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>4</td> <td>3</td> <td>2</td> <td>1</td> <td>0</td> </tr> <tr> <td>W</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>PDDR_x</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></td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> </table>	R	0	0	0	0	4	3	2	1	0	W						PDDR_x				Reset:	0	0	0	0		0	0	0	0	
R	0	0	0	0	4	3	2	1	0																						
W						PDDR_x																									
Reset:	0	0	0	0		0	0	0	0																						

Figure 13-5. Port Data Direction Registers (PDDR_x)

Address: 0xFC0A_400E (PDDR_CS)	Access: User read/write																														
<table> <tr> <td>R</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>4</td> <td>3</td> <td>2</td> <td>1</td> <td>0</td> </tr> <tr> <td>W</td> <td></td> <td></td> <td></td> <td></td> <td></td> <td>PDDR_CS</td> <td></td> <td></td> <td>0</td> </tr> <tr> <td>Reset:</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td></td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> </table>	R	0	0	0	0	4	3	2	1	0	W						PDDR_CS			0	Reset:	0	0	0	0		0	0	0	0	
R	0	0	0	0	4	3	2	1	0																						
W						PDDR_CS			0																						
Reset:	0	0	0	0		0	0	0	0																						

Figure 13-6. Port CS Data Direction Registers (PDDR_CS)

**Figure 13-7. Port Data Direction Registers (PDDR_x)****Table 13-5. PDDR_x Field Descriptions**

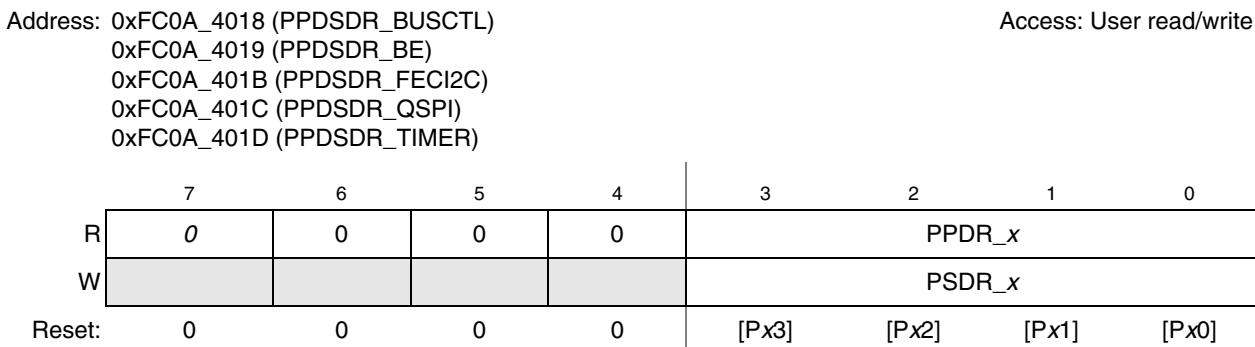
Field	Description
PDDR_x	Port x output data direction bits. 1 Port x pin configured as output 0 Port x pin configured as input

Note: See above figures for bit field positions.

13.3.3 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_x registers are each eight bits wide, but not all ports use all eight bits. The register definitions for all ports are shown in the below figures.

The PPDSDR_x registers are read/write. At reset, the bits in the PPDSDR_x registers are set to the current pin states. Reading a PPDSDR_x register returns the current state of the port x pins. Setting a PPDSDR_x register sets the corresponding bits in the PODR_x register. Writing 0s has no effect.

**Figure 13-8. Port Pin Data/Set Data Registers (PPDSDR_x)**

Address: 0xFC0A_401A (PPDSDR_CS)								Access: User read/write			
R								7	6	5	4
0 0 0 0								3	2	1	0
W								PPDR_CS	PSDR_CS		
Reset: 0 0 0 0								[PCS3]	[PCS2]	[PCS1]	0

Figure 13-9. Port CS Pin Data/Set Data Registers (PPDSDR_CS)

Address: 0xFC0A_401E (PPDSDR_UART) 0xFC0A_401F (PPDSDR_FECH) 0xFC0A_4020 (PPDSDR_FECL)								Access: User read/write			
R								7	6	5	4
PPDR_x								3	2	1	0
W								PSDR_x			
Reset: [Px7] [Px6] [Px5] [Px4]								[Px3]	[Px2]	[Px1]	[Px0]

Figure 13-10. Port Pin Data/Set Data Registers (PPDSDR_x)

Table 13-6. PPDSDR_x Field Descriptions

Field	Description
PPDR_x (read)	Port x pin data bits. 0 Port x pin state is 0 1 Port x pin state is 1
PSDR_x (write)	Port x set data bits. 0 No effect. 1 Set corresponding PODR_x bit.

Note: See above figures for bit field positions.

13.3.4 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, but not all ports use all eight bits. The register definitions for all ports are shown in the figures below.

Address: 0xFC0A_4024 (PCLRR_BUSCTL)
 0xFC0A_4025 (PCLRR_BE)
 0xFC0A_4027 (PCLRR_FECl2C)
 0xFC0A_4028 (PCLRR_QSPI)
 0xFC0A_4029 (PCLRR_TIMER)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								PCLRR_x
Reset:	0	0	0	0	0	0	0	0

Figure 13-11. Port Clear Output Data Registers (PCLRR_x)

Address: 0xFC0A_4026 (PCLRR_CS)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								PCLRR_CS
Reset:	0	0	0	0	0	0	0	0

Figure 13-12. Port CS Clear Output Data Registers (PCLRR_CS)

Address: 0xFC0A_402A (PCLRR_UART)
 0xFC0A_402B (PCLRR_FECH)
 0xFC0A_402C (PCLRR_FECL)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W								PCLRR_x
Reset:	0	0	0	0	0	0	0	0

Figure 13-13. Port Clear Output Data Registers (PCLRR_x)**Table 13-7. PCLRR_x Field Descriptions**

Field	Description
PCLRR_x	Port x clear data bits. 0 Clears corresponding PODR_x bit 1 No effect

Note: See above figures for bit field positions.

13.3.5 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.

13.3.5.1 External Bus Control Pin Assignment Register (PAR_BUSCTL)

The PAR_BUSCTL register controls the functions of the external bus control signal pins.

Address: 0xFC0A_4030 (PAR_BUSCTL)								Access: User read/write			
	7	6	5	4		3	2	1	0		
R	0	0	0		PAR_OE	PAR_TA	PAR_RWB	PAR_TS			
W											
Reset:	0	0	0	1		1	1	1	1		

Figure 13-14. External Bus Control Pin Assignment Register (PAR_BUSCTL)

Table 13-8. PAR_BUSCTL Field Descriptions

Field	Description
7–5	Reserved, should be cleared.
4 PAR_OE	0 \overline{OE} pin configured for GPIO 1 \overline{OE} pin configured for external bus \overline{OE} function
3 PAR_TA	0 \overline{TA} pin configured for GPIO 1 \overline{TA} pin configured for external bus \overline{TA} function
2 PAR_RWB	0 \overline{RW} pin configured for GPIO 1 \overline{RW} pin configured for external bus read/write function
1–0 PAR_TS	0x \overline{TS} pin configured for GPIO 10 \overline{TS} pin configured for DMA acknowledge 0 function 11 \overline{TS} pin configured for external bus \overline{TS} function

13.3.5.2 Byte Enable Pin Assignment Register (PAR_BE)

The PAR_BE register controls the functions of the byte enable pins.

Address: 0xFC0A_4031 (PAR_BE)								Access: User read/write			
	7	6	5	4		3	2	1	0		
R	0	0	0	0		PAR_BE					
W											
Reset:	0	0	0	0		1	1	1	1		

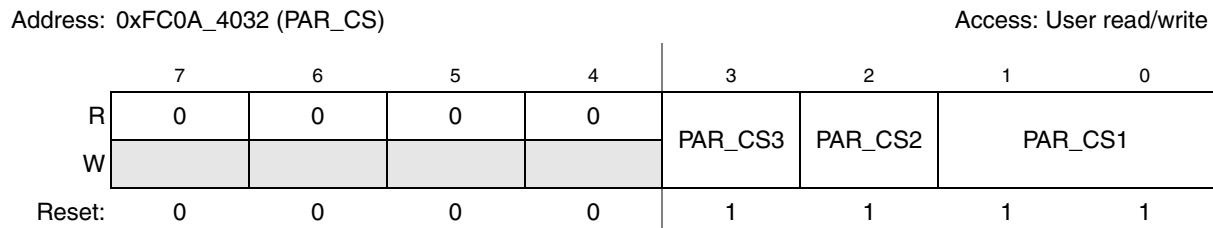
Figure 13-15. Byte Enable Pin Assignment Register (PAR_BE)

Table 13-9. PAR_BE Field Descriptions

Field	Description
7–4	Reserved, should be cleared.
3–0 PAR_BE	BE/BWE[3:0] pin assignment. The PAR_BE[3:0] bits configure the BE/BWE[3:0] pins for their primary function or GPIO. 0 BE/BWE[3:0] pin configured for GPIO 1 BE/BWE[3:0] pin configured for BE/BWE[3:0] function Refer to Chapter 9, “Chip Configuration Module (CCM) for more information on reset configuration.

13.3.5.3 Chip Select Pin Assignment Register (PAR_CS)

The PAR_CS register controls the functions of the FlexBus chip select pins.

**Figure 13-16. Byte Strobe Pin Assignment Register (PAR_BE)****Table 13-10. PAR_BE Field Descriptions**

Field	Description
7–4	Reserved, should be cleared.
3 PAR_CS3	FB_CS3 pin assignment. 0 FB_CS3 pin configured for GPIO 1 FB_CS3 pin configured for FlexBus FB_CS3 function
2 PAR_CS2	FB_CS2 pin assignment. 0 FB_CS2 pin configured for GPIO 1 FB_CS2 pin configured for FlexBus FB_CS2 function
1–0 PAR_CS1	FB_CS1 pin assignment. 0x FB_CS1 pin configured for GPIO 10 FB_CS1 pin configured for SD_CS1 function 11FB_CS1 pin configured for FlexBus FB_CS1 function

13.3.5.4 FEC/I²C Pin Assignment Register (PAR_FECI2C)

The PAR_FECI2C register controls the functions of the I²C and some of the FEC pins.

Address: 0xFC0A_4033 (PAR_FECI2C) Access: User read/write

	7	6	5	4		3	2	1	0
R	PAR_MDC		PAR_MDIO		PAR_SCL		PAR_SDA		
W	0	0	0	0	0	0	0	0	0

Reset: 0 0 0 0 0 0 0 0 0 0

Figure 13-17. FEC/I2C Pin Assignment (PAR_FECI2C)

Table 13-11. PAR_FECI2C Field Descriptions

Field	Description																									
7–6 PAR_MDC 5–4 PAR_MDIO 3–2 PAR_SCL 1–0 PAR_SDA	FEC & I ² C pin assignment. These bit fields configure the FEC_MDC, FEC_MDIO, I2C_SCL, and I2C_SDA pins for one of their primary functions or GPIO. <table border="1"> <tr> <th></th> <th>PAR_MDC</th> <th>PAR_MDIO</th> <th>PAR_SCL</th> <th>PAR_SDA</th> </tr> <tr> <td>00</td> <td>GPIO</td> <td>GPIO</td> <td>GPIO</td> <td>GPIO</td> </tr> <tr> <td>01</td> <td>U2TXD</td> <td>U2RXD</td> <td>U2TXD</td> <td>U2RXD</td> </tr> <tr> <td>10</td> <td>I2C_SCL</td> <td>I2C_SDA</td> <td>GPIO</td> <td>GPIO</td> </tr> <tr> <td>11</td> <td>FEC_MDC</td> <td>FEC_MDIO</td> <td>I2C_SCL</td> <td>I2C_SDA</td> </tr> </table>		PAR_MDC	PAR_MDIO	PAR_SCL	PAR_SDA	00	GPIO	GPIO	GPIO	GPIO	01	U2TXD	U2RXD	U2TXD	U2RXD	10	I2C_SCL	I2C_SDA	GPIO	GPIO	11	FEC_MDC	FEC_MDIO	I2C_SCL	I2C_SDA
	PAR_MDC	PAR_MDIO	PAR_SCL	PAR_SDA																						
00	GPIO	GPIO	GPIO	GPIO																						
01	U2TXD	U2RXD	U2TXD	U2RXD																						
10	I2C_SCL	I2C_SDA	GPIO	GPIO																						
11	FEC_MDC	FEC_MDIO	I2C_SCL	I2C_SDA																						

13.3.5.5 QSPI Pin Assignment Register (PAR_QSPI)

The PAR_QSPI register controls the functions of the QSPI pins.

Address: 0xFC0A_4034 (PAR_QSPI) Access: User read/write

	7	6	5	4		3	2	1	0
R	PAR_PCS2		PAR_DIN		PAR_DOUT		PAR_SCK		
W	0	0	0	0	0	0	0	0	0

Reset: 0 0 0 0 0 0 0 0 0 0

Figure 13-18. QSPI Pin Assignment (PAR_QSPI)

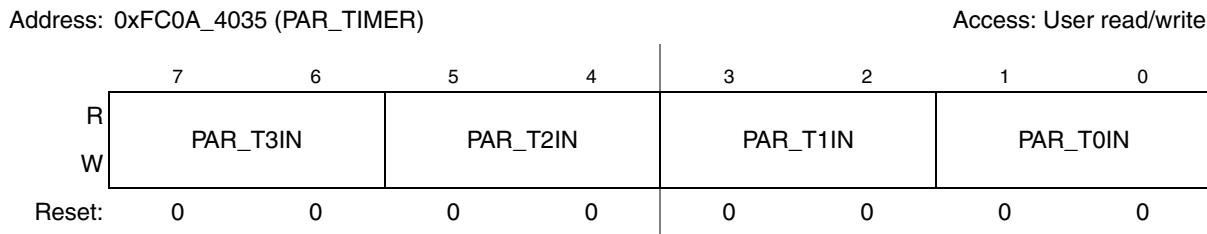
Table 13-12. PAR_QSPI Field Descriptions

Field	Description				
7–6 PAR_PCS2 5–4 PAR_DIN 3–2 PAR_DOUT 1–0 PAR_SCK	QSPI pin assignment. These bit fields configure the QSPI pins for one of their primary functions or GPIO.				

	PAR_PCS2	PAR_DIN	PAR_DOUT	PAR_SCK
00	GPIO	GPIO	GPIO	GPIO
01	$\overline{U2RTS}$	$\overline{U2CTS}$	GPIO	GPIO
10	$\overline{DACK0}$	$\overline{DREQ0}$	I2C_SDA	I2C_SCL
11	QSPI_PCS2	QSPI_DIN	QSPI_DOUT	QSPI_SCK

13.3.5.6 Timer Pin Assignment Registers (PAR_TIMER)

The PAR_TIMER register controls the functions of the DMA timer pins.

**Figure 13-19. Timer Pin Assignment (PAR_TIMER)****Table 13-13. PAR_TIMER Field Descriptions**

Field	Description				
7–6 PAR_T3IN 5–4 PAR_T2IN 3–2 PAR_T1IN 1–0 PAR_T0IN	DMA Timer pin assignment. These bit fields configure the DMA Timer pins for one of their primary functions or GPIO.				

	PAR_T3IN	PAR_T2IN	PAR_T1IN	PAR_T0IN
00	GPIO	GPIO	GPIO	GPIO
01	$\overline{U2CTS}$	$\overline{U2RTS}$	U2RXD	U2TXD
10	T3OUT	T2OUT	T1OUT	T0OUT
11	T3IN	T2IN	T1IN	T0IN

13.3.5.7 UART Pin Assignment Register (PAR_UART)

The PAR_TIMER register controls the functions of the DMA timer pins.

Address: 0xFC0A_4036 (PAR_UART)																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	PAR_U1CTS	PAR_U1RTS	PAR_U1TXD	PAR_U1RXD	PAR_U0CTS	PAR_U0RTS	PAR_U0TXD	PAR_U0RXD	0	0	0	0			
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 13-20. UART Pin Assignment (PAR_UART)

Table 13-14. PAR_UART Field Descriptions

Field	Description																	
15–12	Reserved, should be cleared.																	
11–10 PAR_U1CTS 9–8 PAR_U1RTS 7–6 PAR_U1RXD 5–4 PAR_U1TXD	UART1 control pin assignment. These bit fields configure the UART1 control pins for one of their primary functions or GPIO. <table border="1"> <tr> <th></th> <th>PAR_U1CTS</th> <th>PAR_U1RTS</th> </tr> <tr> <td>00</td> <td>GPIO</td> <td>GPIO</td> </tr> <tr> <td>01</td> <td>QSPI_PCS1</td> <td>QSPI_PCS1</td> </tr> <tr> <td>10</td> <td>T1IN</td> <td>T1OUT</td> </tr> <tr> <td>11</td> <td>$\overline{U1CTS}$</td> <td>$\overline{U1RTS}$</td> </tr> </table>				PAR_U1CTS	PAR_U1RTS	00	GPIO	GPIO	01	QSPI_PCS1	QSPI_PCS1	10	T1IN	T1OUT	11	$\overline{U1CTS}$	$\overline{U1RTS}$
	PAR_U1CTS	PAR_U1RTS																
00	GPIO	GPIO																
01	QSPI_PCS1	QSPI_PCS1																
10	T1IN	T1OUT																
11	$\overline{U1CTS}$	$\overline{U1RTS}$																
7 PAR_U1TXD	U1TXD pin assignment. 0 U1TXD pin configured for GPIO 1 U1TXD pin configured for UART1 TXD function																	
6 PAR_U1RXD	U1RXD pin assignment. 0 U1RXD pin configured for GPIO 1 U1RXD pin configured for UART1 RXD function																	
5–4 PAR_U0CTS 3–2 PAR_U0RTS	UART0 control pin assignment. These bit fields configure the UART0 control pins for one of their primary functions or GPIO. <table border="1"> <tr> <th></th> <th>PAR_U0CTS</th> <th>PAR_U0RTS</th> </tr> <tr> <td>00</td> <td>GPIO</td> <td>GPIO</td> </tr> <tr> <td>01</td> <td>QSPI_PCS0</td> <td>QSPI_PCS0</td> </tr> <tr> <td>10</td> <td>T0IN</td> <td>T0OUT</td> </tr> <tr> <td>11</td> <td>$\overline{U0CTS}$</td> <td>$\overline{U0RTS}$</td> </tr> </table>				PAR_U0CTS	PAR_U0RTS	00	GPIO	GPIO	01	QSPI_PCS0	QSPI_PCS0	10	T0IN	T0OUT	11	$\overline{U0CTS}$	$\overline{U0RTS}$
	PAR_U0CTS	PAR_U0RTS																
00	GPIO	GPIO																
01	QSPI_PCS0	QSPI_PCS0																
10	T0IN	T0OUT																
11	$\overline{U0CTS}$	$\overline{U0RTS}$																
1 PAR_U0TXD	U0TXD pin assignment. 0 U0TXD pin configured for GPIO 1 U0TXD pin configured for UART0 TXD function																	
0 PAR_U0RXD	U0RXD pin assignment. 0 U0RXD pin configured for GPIO 1 U0RXD pin configured for UART0 RXD function																	

13.3.5.8 FEC Pin Assignment Register (PAR_FEC)

The PAR_FEC register controls the functions of the FEC pins.

Address: 0xFC0A_4038 (PAR_FEC)								Access: User read/write			
	7	6	5	4		3	2	1	0		
R	0	0	0	0		PAR_FEC_7W		PAR_FEC_MII			
W						0	0	0	0		
Reset:	0	0	0	0		0	0	0	0		

Figure 13-21. FEC Pin Assignment (PAR_FEC)

Table 13-15. PAR_FEC Field Descriptions

Field	Description							
7–4	Reserved, should be cleared.							
3–2 PAR_FEC_7W	FEC 7-wire pin assignment. These bit fields configure the FEC_COL, FEC_RXCLK, FEC_RXDV, FEC_RXD0, FEC_TXCLK, FEC_TXD0, and FEC_TXEN pins for one of their primary functions or GPIO.							
	FEC_COL	FEC_RXCLK	FEC_RXDV	FEC_RXD0	FEC_TXCLK	FEC_TXD0	FEC_TXEN	
00	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	
01	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	<u>U1RTS</u>	
10	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO	
11	FEC_COL	FEC_RXCLK	FEC_RXDV	FEC_RXD0	FEC_TXCLK	FEC_TXD0	FEC_TXEN	
1–0 PAR_FEC_MII	FEC MII pin assignment. These bit fields configure the FEC_CRS, FEC_RXD3, FEC_RXD2, FEC_RXD1, FEC_RXER, FEC_TXD3, FEC_TXD2, FEC_TXD1 and FEC_TXER pins for one of their primary functions or GPIO.							
	FEC_RXD3	FEC_RXD2	FEC_RXD1	FEC_RXER	FEC_CRS			
00	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO		
01	GPIO	GPIO	<u>U1CTS</u>	<u>U0CTS</u>	GPIO			
10	GPIO	GPIO	GPIO	GPIO	GPIO	GPIO		
11	FEC_RXD3	FEC_RXD2	FEC_RXD1	FEC_RXER	FEC_CRS			
	FEC_TXD3	FEC_TXD2	FEC_TXD1	FEC_TXER				
00	GPIO	GPIO	GPIO	GPIO	GPIO			
01	GPIO	GPIO	GPIO	<u>U0RTS</u>				
10	GPIO	GPIO	GPIO	GPIO	GPIO			
11	FEC_TXD3	FEC_TXD2	FEC_TXD1	FEC_TXER				

13.3.5.9 IRQ Pin Assignment Register (PAR_IRQ)

The PAR_IRQ register controls the functions of the $\overline{\text{IRQ4}}$ pin.

Address: 0xFC0A_4039 (PAR_IRQ)								Access: User read/write	
	7	6	5	4	3	2	1	0	
R	0	0	0	0	0	0	0	0	PAR_IRQ4
W									
Reset:	0	0	0	0	0	0	0	0	

Figure 13-22. IRQ Pin Assignment (PAR_IRQ)

Table 13-16. PAR_IRQ Field Descriptions

Field	Description
7–1	Reserved, should be cleared.
0 PAR_IRQ4	$\overline{\text{IRQ4}}$ pin assignment. 0 $\overline{\text{IRQ4}}$ pin configured for $\overline{\text{IRQ4}}$ function (if IRQ4 is enabled in the Edge Port module) or GPIO (if IRQ4 is disabled in the Edge Port module). 1 $\overline{\text{IRQ4}}$ pin configured for $\overline{\text{DREQ0}}$ function.

13.3.6 FlexBus Mode Select Control Register (MSCR_FLEXBUS)

The MSCR_FLEXBUS register controls the output mode selects of the following FlexBus pins: FB_A[23:0], D[31:0], $\overline{\text{BE/BWE}}[3:0]$, $\overline{\text{OE}}$, R/W, FB_CS[5:0], TA and TS.

Address: 0xFC0A_403A (MSCR_FLEXBUS)								Access: User read/write	
	7	6	5	4	3	2	1	0	
R	MSCR_FBCLK		MSCR_DUPPER		MSCR_DLOWER		MSCR_ADDRCTL		
W	1	1	1	1	1	1	1	1	
Reset:	1	1	1	1	1	1	1	1	

Figure 13-23. FlexBus Mode Select Control Register (MSCR_FLEXBUS)

Table 13-17. MSCR_FLEXBUS Field Descriptions

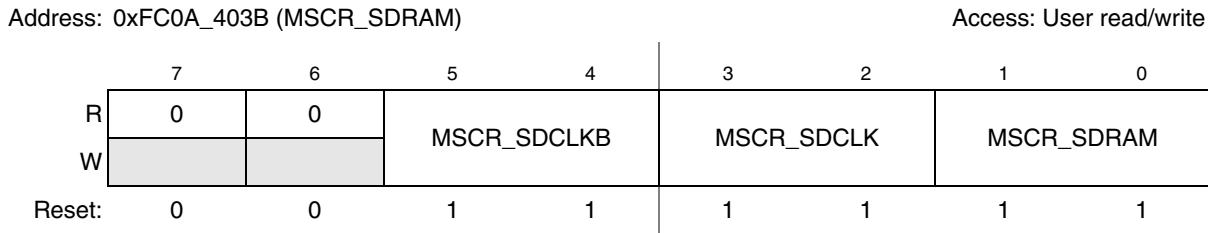
Field	Description
7–6 MSCR_FBCLK	FB_CLK mode select control. These bit fields control the strength of the FlexBus clock pin. 00 Half strength 1.8V low power/mobile DDR. 01 Open drain. 10 Full strength 1.8V low power/mobile DDR. 11 2.5V DDR1 or 3.3V CMOS with roughly equal rise and fall delays.
5–4 MSCR_DUPPER	FB_D[31:16] mode select control. These bit fields control the strength of the FlexBus upper data pins. 00 Half strength 1.8V low power/mobile DDR. 01 Open drain. 10 Full strength 1.8V low power/mobile DDR. 11 2.5V DDR1 or 3.3V CMOS with roughly equal rise and fall delays.

Table 13-17. MSCR_FLEXBUS Field Descriptions (continued)

Field	Description
3–2 MSCR_DLOWER	FB_D[15:0] mode select control. These bit fields control the strength of the FlexBus lower data pins. 00 Half strength 1.8V low power/mobile DDR. 01 Open drain. 10 Full strength 1.8V low power/mobile DDR. 11 2.5V DDR1 or 3.3V CMOS with roughly equal rise and fall delays.
1–0 MSCR_ADDRCTL	FB_A[23:0], $\overline{BE/BWE}$ [3:0], \overline{OE} , \overline{RW} , FB_CS[5:0], \overline{TA} , and \overline{TS} mode select control. These bit fields control the strength of the FlexBus address and control pins. 00 Half strength 1.8V low power/mobile DDR. 01 Open drain. 10 Full strength 1.8V low power/mobile DDR. 11 2.5V DDR1 or 3.3V CMOS with roughly equal rise and fall delays.

13.3.7 SDRAM Mode Select Control Register (MSCR_SDRAM)

The MSCR_SDRAM register controls the output mode selects of the following dedicated SDRAM pins: SD_A10, $\overline{SD_{CAS}}$, SD_CKE, SD_CLK, $\overline{SD_{CLK}}$, $\overline{SD_{CS0}}$, SD_DQS[3:2], $\overline{SD_{RAS}}$, SD_SDRDQS, and $\overline{SD_{WE}}$.

**Figure 13-24. SDRAM Mode Select Control Register (MSCR_SDRAM)****Table 13-18. MSCR_SDRAM Field Descriptions**

Field	Description
7–6	Reserved, should be cleared
5–4 MSCR_SDCLKB	SD_CLK mode select control. These bit fields control the strength of the FlexBus upper data pins. 00 Half strength 1.8V Mobile DDR. 01 Open drain. 10 Full strength 1.8V Mobile DDR. 11 2.5V DDR1 or 3.3V CMOS with roughly equal rise and fall delays.
3–2 MSCR_SDCLK	SD_CLK mode select control. These bit fields control the strength of the FlexBus lower data pins. 00 Half strength 1.8V Mobile DDR. 01 Open drain. 10 Full strength 1.8V Mobile DDR. 11 2.5V DDR1 or 3.3V CMOS with roughly equal rise and fall delays.
1–0 MSCR_SDRAM	SD_A10, $\overline{SD_{CAS}}$, SD_CKE, $\overline{SD_{CS0}}$, SD_DQS[3:2], $\overline{SD_{RAS}}$, SD_SDRDQS, SD_WE mode select control. These bit fields control the strength of the FlexBus address and control pins. 00 Half strength 1.8V Mobile DDR. 01 Open drain. 10 Full strength 1.8V Mobile DDR. 11 2.5V DDR1 or 3.3V CMOS with roughly equal rise and fall delays.

13.3.8 Drive Strength Control Registers (DSCR_x)

The drive strength control registers set the output pin drive strengths. All drive strength control registers are read/write.

13.3.8.1 I²C Drive Strength Control Register (DSCR_I2C)

The DSCR_I2C register controls the output drive strengths of the I2C_SDA and I2C_SCL pins.

Address: 0xFC0A_403C (DSCR_I2C)								Access: User read/write		
	7	6	5	4	3	2	1	0		
R	0	0	0	0	0	0			I2C_DSE	
W										
Reset:	0	0	0	0	0	0	See Note	1		

Note: Reset state is 0 when $\overline{\text{RCON}} = 1$, and is value of D[21] when $\overline{\text{RCON}} = 0$.

Figure 13-25. I²C Drive Strength Control Register (DSCR_I2C)

Table 13-19. DSCR_I2C Field Descriptions

Field	Description
7–2	Reserved, should be cleared
1–0 I2C_DSE	I ² C drive strength control. This bit field controls the drive strength of the I2C_SDA and I2C_SCL pins. 00 10pF 01 20pF 10 30pF 11 50pF

13.3.8.2 Miscellaneous Drive Strength Control Register (DSCR_MISC)

The DSCR_MISC register controls the output drive strengths of the following pins: PSTCLK, PST[3:0], DDATA[3:0], ALLPST, DSO, DT3IN, DT2IN, DT1IN, DT0IN, and $\overline{\text{RSTOUT}}$.

Address: 0xFC0A_403D (DSCR_MISC)								Access: User read/write		
	7	6	5	4	3	2	1	0		
R	0	0			DEBUG_DSE		RSTOUT_DSE		TIMER_DSE	
W										
Reset:	0	0	1	1	See Note	1	See Note	1		

Note: Reset state is 0 when $\overline{\text{RCON}} = 1$, and is value of D[21] when $\overline{\text{RCON}} = 0$.

Figure 13-26. Miscellaneous Drive Strength Control Register (DSCR_MISC)

Table 13-20. DSCR_I2C Field Descriptions

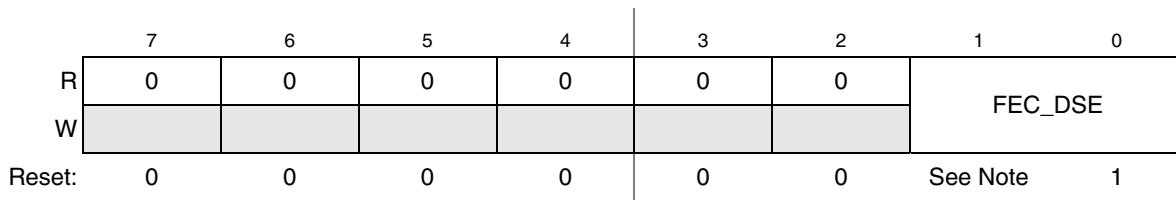
Field	Description
7–6	Reserved, should be cleared
5–4 DEBUG_ DSE	Debug drive strength control. This bit field controls the drive strength of the PST[3:0], DDATA[3:0], ALLPST, and DSO pins. 00 10pF 01 20pF 10 30pF 11 50pF
3–2 RSTOUT_ DSE	RSTOUT drive strength control. This bit field controls the drive strength of the RSTOUT pin. 00 10pF 01 20pF 10 30pF 11 50pF
1–0 TIMER_ DSE	Timer drive strength control. This bit field controls the drive strength of the DT3IN, DT2IN, DT1IN, and DT0IN pins. 00 10pF 01 20pF 10 30pF 11 50pF

13.3.8.3 FEC Drive Strength Control Register (DSCR_FEC)

The DSCR_FEC register controls the output drive strengths of the FEC_MDC and FEC_MDIO pins.

Address: 0xFC0A_403E (DSCR_FEC)

Access: User read/write



Note: Reset state is 0 when RCON = 1, and is value of D[21] when RCON = 0.

Figure 13-27. FEC Drive Strength Control Register (DSCR_FEC)**Table 13-21. DSCR_FEC Field Descriptions**

Field	Description
7–2	Reserved, should be cleared
1–0 FEC_ DSE	FEC drive strength control. This bit field controls the drive strength of the FEC_MDC and FEC_MDIO pins. 00 10pF 01 20pF 10 30pF 11 50pF

13.3.8.4 UART/IRQ Drive Strength Control Register (DSCR_UART)

The DSCR_UART register controls the output drive strengths of the UART1, UART0, and IRQ pins.

Address: 0xFC0A_403F (DSCR_UART)								Access: User read/write																	
<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="width: 10px;">R</td> <td style="width: 10px;">0</td> <td style="width: 10px;">0</td> <td colspan="2" style="width: 40px; text-align: center;">UART1_DSE</td> <td colspan="2" style="width: 40px; text-align: center;">UART0_DSE</td> <td colspan="2" style="width: 40px; text-align: center;">IRQ_DSE</td> </tr> <tr> <td style="width: 10px;">W</td> <td colspan="2"></td> <td colspan="2"></td> <td colspan="2"></td> <td colspan="2"></td> </tr> </table>								R	0	0	UART1_DSE		UART0_DSE		IRQ_DSE		W								
R	0	0	UART1_DSE		UART0_DSE		IRQ_DSE																		
W																									
Reset:								See Note																	

Note: Reset state is 0 when $\overline{\text{RCON}} = 1$, and is value of D[21] when $\overline{\text{RCON}} = 0$.

Figure 13-28. UART/IRQ Drive Strength Control Register (DSCR_UART)

Table 13-22. DSCR_UART Field Descriptions

Field	Description
7–6	Reserved, should be cleared
5–4 UART1_DSE	UART1 drive strength control. This bit field controls the drive strength of the U1RXD, U1TXD, $\overline{\text{U1CTS}}$, and $\overline{\text{U1RTS}}$ pins. 00 10pF 01 20pF 10 30pF 11 50pF
3–2 UART0_DSE	UART0 drive strength control. This bit field controls the drive strength of the U0RXD, U0TXD, $\overline{\text{U0CTS}}$, and $\overline{\text{U0RTS}}$ pins. 00 10pF 01 20pF 10 30pF 11 50pF
1–0 IRQ_DSE	IRQ drive strength control. This bit field controls the drive strength of the IRQ[7:1] pins. 00 10pF 01 20pF 10 30pF 11 50pF

13.3.8.5 QSPI Drive Strength Control Register (DSCR_QSPI)

The DSCR_QSPI register controls the output drive strengths of the QSPI_PCS2, QSPI_SCK, QSPI_DIN, and QSPI_DOUT pins.

Address: 0xFC0A_4040 (DSCR_QSPI)								Access: User read/write																	
<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="width: 10px;">R</td> <td style="width: 10px;">0</td> <td colspan="2" style="width: 40px; text-align: center;">QSPI_DSE</td> </tr> <tr> <td style="width: 10px;">W</td> <td colspan="2"></td> <td colspan="2"></td> <td colspan="2"></td> <td colspan="2"></td> </tr> </table>								R	0	0	0	0	0	0	QSPI_DSE		W								
R	0	0	0	0	0	0	QSPI_DSE																		
W																									
Reset:								See Note																	

Note: Reset state is 0 when $\overline{\text{RCON}} = 1$, and is value of D[21] when $\overline{\text{RCON}} = 0$.

Figure 13-29. QSPI Drive Strength Control Register (DSCR_QSPI)

Table 13-23. DSCR_QSPI Field Descriptions

Field	Description
7–2	Reserved, should be cleared
1–0 QSPI_DSE	QSPI drive strength control. This bit field controls the drive strength of the QSPI_PCS2, QSPI_SCK, QSPI_DIN, QSPI_DOUT pins. 00 10pF 01 20pF 10 30pF 11 50pF

13.4 Functional Description

13.4.1 Overview

Initial pin function is determined during reset configuration. The pin assignment registers allow the user to select among various primary functions and general purpose I/O after reset. Most pins are configured as GPIO by default. The notable exceptions to this are external bus control pins, address/data pins, and chip select pins. These pins are configured for their primary functions after reset.

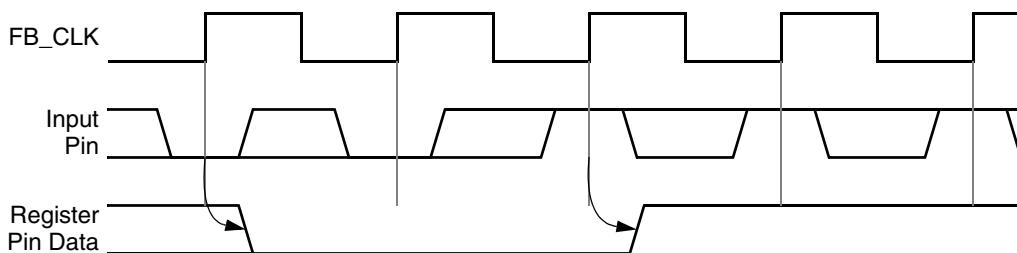
Every GPIO pin is individually configurable as an input or an output via a data direction register (PDDR_x). Every GPIO port has an output data register (PODR_x) and a pin data register (PPDSDR_x) to monitor and control the state of its pins. Data written to a PODR_x register is stored and then driven to the corresponding port x pins configured as outputs.

Reading a PODR_x register returns the current state of the register regardless of the state of the corresponding pins. Reading a PPDSDR_x register returns the current state of the corresponding pins when configured as general purpose I/O, regardless of whether the pins are inputs or outputs.

Every GPIO port has a PPDSDR_x register and a clear register (PCLR_x) for setting or clearing individual bits in the PODR_x register. Initial pin output drive strength is determined during reset configuration. The DSCR_x registers allow the pin drive strengths to be configured on a per-function basis after reset.

13.4.2 Port Digital I/O Timing

Input data on all pins configured as general purpose input is synchronized to the rising edge of the bus clock, FB_CLK, as shown in [Figure 13-30](#).

**Figure 13-30. General Purpose Input Timing**

Data written to the PODR_{_x} register of any pin configured as a general purpose output is immediately driven to its respective pin, as shown in [Figure 13-31](#).

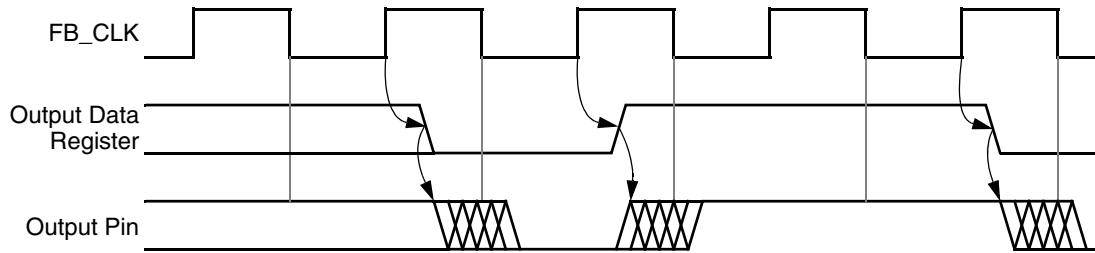


Figure 13-31. General Purpose Output Timing

13.5 Initialization/Application Information

The initialization for the ports module is done during reset configuration. All registers are reset to a predetermined state. Refer to [Section 13.3, “Memory Map/Register Definition,”](#) for more details on reset and initialization.

Chapter 14

Interrupt Controller Module

14.1 Introduction

This section details the functionality of the interrupt controller (INTC). The general features of the interrupt controller block include:

- 64 fully-programmable interrupt sources. Note, not all possible interrupt source locations are used on this device.
- Each of the sources has a unique interrupt control register (ICR_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 both hardware and software interrupt acknowledge cycles.
- Wake-up signal from low-power stop modes.

The 64, fully-programmable interrupt sources for the interrupt controller handle 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.

14.1.1 68K/ColdFire Interrupt Architecture Overview

Before continuing with the specifics of the interrupt controller, 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, once 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 encoding of the transfer type

and transfer modifier attributes to distinguish this data fetch from a normal memory access. The fetched data provides an index into the exception vector table which 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 handle reset, error conditions (access, address), arithmetic faults, system calls, etc. Once 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 3.5, “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. For many peripheral devices, the processing of the IACK cycle directly negates the interrupt request, while other devices require that request to be explicitly negated during the processing of the service routine.

The processing of the interrupt acknowledge cycle is fundamentally different than previous 68K/ColdFire cores. In the new approach, all IACK cycles are directly handled 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 14.3.1.3, “Interrupt Vector Determination.”](#)

Unlike the M68000 family, all 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.

During the execution of the service routine, the appropriate actions must be performed on the peripheral to negate the interrupt request.

For more information on exception processing, see the *ColdFire Programmer’s Reference Manual* at <http://www.freescale.com/coldfire>.

14.2 Memory Map/Register Definition

The register programming model for the interrupt controller 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 14-2](#). The base addresses for the interrupt controller is listed below.

Table 14-1. Interrupt Controller Base Addresses

Interrupt Controller Number	Base Address
INTC	0xFC04_8000

Table 14-2. Interrupt Controller Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/ Page
Interrupt Controller 0					
0xFC04_8000	Interrupt Pending Register High (IPRH)	32	R	0x0000_0000	14.2.1/14-3
0xFC04_8004	Interrupt Pending Register Low (IPRL)	32	R	0x0000_0000	14.2.1/14-3
0xFC04_8008	Interrupt Mask Register High (IMRH)	32	R/W	0xFFFF_FFFF	14.2.2/14-4
0xFC04_800C	Interrupt Mask Register Low (IMRL)	32	R/W	0xFFFF_FFFF	14.2.2/14-4
0xFC04_8010	Interrupt Force Register High (INTFRCH)	32	R/W	0x0000_0000	14.2.3/14-6
0xFC04_8014	Interrupt Force Register Low (INTFRCL)	32	R/W	0x0000_0000	14.2.3/14-6
0xFC04_801A	Interrupt Configuration Register (ICONFIG)	16	R/W	0x0000	14.2.4/14-6
0xFC04_801C	Set Interrupt Mask (SIMR)	8	W	0x00	14.2.5/14-7
0xFC04_801D	Clear Interrupt Mask (CIMR)	8	W	0x00	14.2.6/14-8
0xFC04_801E	Current Level Mask (CLMASK)	8	R/W	0x0F	14.2.7/14-8
0xFC04_801F	Saved Level Mask (SLMASK)	8	R/W	0x0F	14.2.8/14-9
0xFC04_8040 + <i>n</i> (<i>n</i> =0:63)	Interrupt Control Registers (ICR <i>n</i>)	8	R/W	0x00	14.2.9/14-10
0xFC04_80E0	Software Interrupt Acknowledge (SWIACK)	8	R	0x00	14.2.10/14-12
0xFC04_80E0 + 4 <i>n</i> (<i>n</i> =1:7)	Level <i>n</i> Interrupt Acknowledge Registers (LnIACK)	8	R	0x18	14.2.10/14-12

14.2.1 Interrupt Pending Registers (IPRH, IPRL)

The IPRH and IPRL registers, [Figure 14-1](#) and [Figure 14-2](#), are each 32 bits in size, and provide a bit map for each interrupt request to indicate if there is an active request (1 = active request, 0 = no request) for the given source. The state of the interrupt mask register does not affect the IPR. The IPR is cleared by reset and is a read-only register, so any attempted write to this register is ignored.

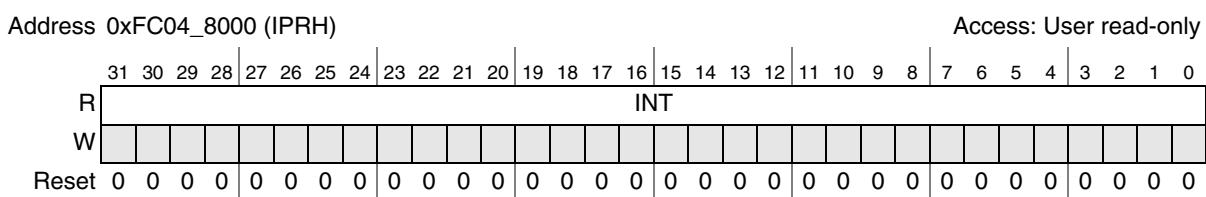
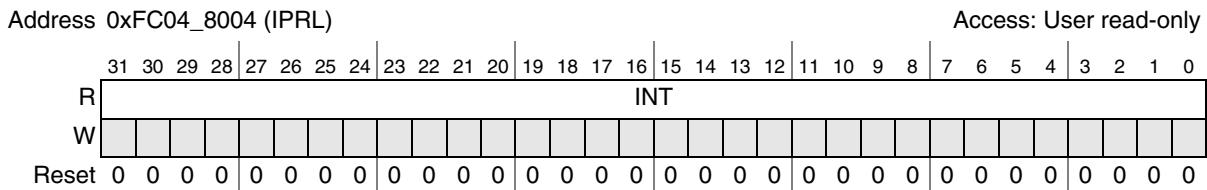


Figure 14-1. Interrupt Pending Register High (IPRH)

Table 14-3. IPRH 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 samples the signal generated by the interrupting source. The corresponding IPRH 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 14-2. Interrupt Pending Register Low (IPRL)****Table 14-4. IPRL 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 samples the signal generated by the interrupting source. The corresponding IPRL bit reflects the state of the interrupt signal even if the corresponding IMRL bit is set. 0 The corresponding interrupt source does not have an interrupt pending 1 The corresponding interrupt source has an interrupt pending

14.2.2 Interrupt Mask Register (IMRH, IMRL)

The IMRH and IMRL registers are each 32 bits in size and provide a bit map for each interrupt to allow the request to be disabled (1 = disable the request, 0 = 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 is set to all ones by reset, disabling all interrupt requests. The IMR 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. Since 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.

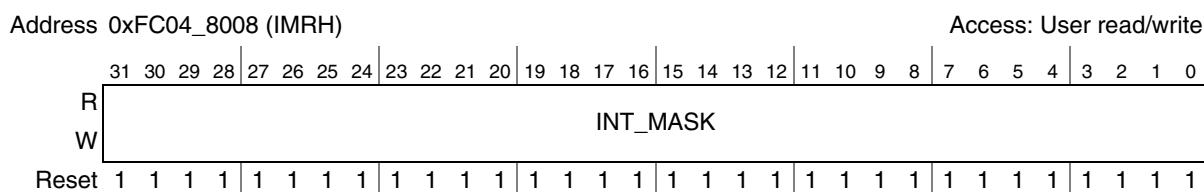


Figure 14-3. Interrupt Mask Register High (IMRH)

Table 14-5. IMRH Field Descriptions

Field	Description
31–0 INT_MASK	<p>Interrupt mask. Each bit corresponds to an interrupt source. The corresponding IMRH bit determines whether an interrupt condition can generate an interrupt. The corresponding IPRH bit reflects the state of the interrupt signal even if the corresponding IMRH bit is set.</p> <ul style="list-style-type: none"> 0 The corresponding interrupt source is not masked 1 The corresponding interrupt source is masked

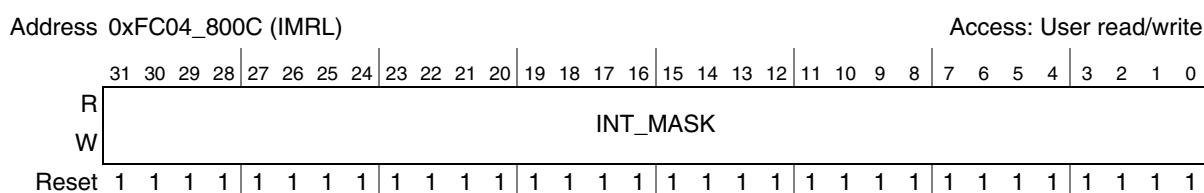


Figure 14-4. Interrupt Mask Register Low (IMRL)

Table 14-6. IMRL Field Descriptions

Field	Description
31-0 INT_MASK	<p>Interrupt mask. Each bit corresponds to an interrupt source. The corresponding IMRL bit determines whether an interrupt condition can generate an interrupt. The corresponding IPRL bit reflects the state of the interrupt signal even if the corresponding IMRL bit is set.</p> <ul style="list-style-type: none"> 0 The corresponding interrupt source is not masked 1 The corresponding interrupt source is masked

14.2.3 Interrupt Force Registers (INTFRCH, INTFRCL)

The INTFRCH and INTFRCL 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 (1 = force request, 0 = negate request) in the appropriate INTFRC register. The INTFRCH register is used for masking interrupt sources 0 to 31, while the INTFRCL register is used for masking interrupts 32 to 63. The assertion of an interrupt request via the interrupt force register is not affected by the interrupt mask register. The INTFRC registers are cleared by reset.

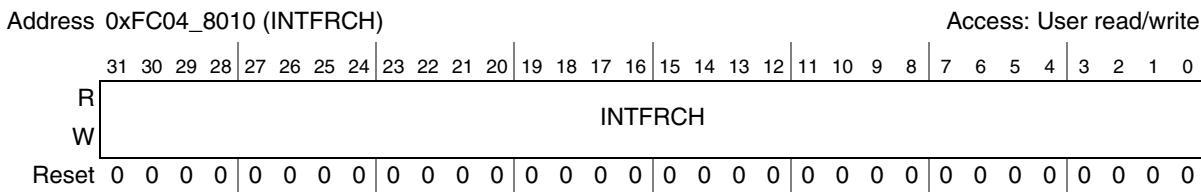


Figure 14-5. Interrupt Force Register High (INTFRCH)

Table 14-7. INTFRCH Field Descriptions

Field	Description
31-0 INTFRCH	Interrupt force. Allows software generation of interrupts for each possible source for functional or debug purposes. 0 No interrupt forced on the corresponding interrupt source 1 Force an interrupt on the corresponding source

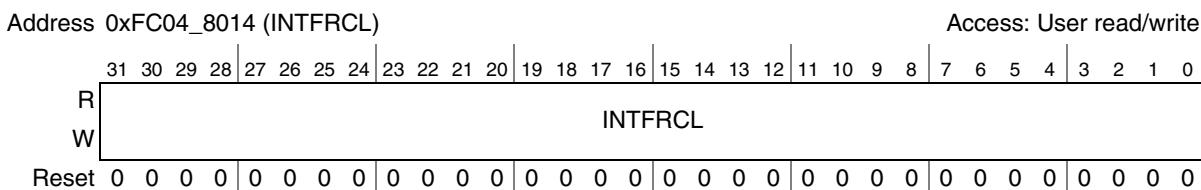


Figure 14-6. Interrupt Force Register Low (INTFRCL)

Table 14-8. INTFRCL Field Descriptions

Field	Description
31-0 INTFRCL	Interrupt force. Allows software generation of interrupts for each possible source for functional or debug purposes.

14.2.4 Interrupt Configuration Register (ICONFIG)

This 16-bit register defines the operating configuration for the interrupt controller module.

Address: 0xFC04_801A (ICONFIG)

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	ELVLPRI							0	0	0	EMASK	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 14-7. Interrupt Configuration Register (ICONFIG)

Table 14-9. 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- <i>n</i> 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- <i>n</i> request is negated. If round-robin arbitration is enabled, this bit has no effect. If cleared, the assertion of a level- <i>n</i> request does not affect the processor's bus master priority.
8–6	Reserved, should 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, should be cleared.

14.2.5 Set Interrupt Mask Register (SIMR)

The SIMR register provides a simple mechanism to set a given bit in the IMR registers to mask the corresponding interrupt request. The value written to the SIMR field causes the corresponding bit in the IMR register to be set. The SIMR[SALL] bit provides a global set function, forcing the entire contents of IMR 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 register.

Address: 0xFC04_801C (SIMR)

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W	SALL	SIMR						
Reset:	0	0	0	0	0	0	0	0

Figure 14-8. Set Interrupt Mask Register (SIMR)

Table 14-10. SIMR Field Descriptions

Field	Description
7	Reserved, should be cleared.
6 SALL	Set all bits in the IMR register, masking all interrupt requests. 0 Only set those bits specified in the SIMR field. 1 Set all bits in IMR register. The SIMR field is ignored.
5-0 SIMR	Set the corresponding bit in the IMR register, masking the interrupt request.

14.2.6 Clear Interrupt Mask Register (CIMR)

The CIMR register provides a simple mechanism to clear a given bit in the IMR registers to enable the corresponding interrupt request. The value written to the CIMR field causes the corresponding bit in the IMR register to be cleared. The CIMR[CALL] bit provides a global clear function, forcing the entire contents of IMR 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 register.

In the event of a simultaneous write to both the CIMR and SIMR, the SIMR has priority and the resulting function would be a set of the interrupt mask register.

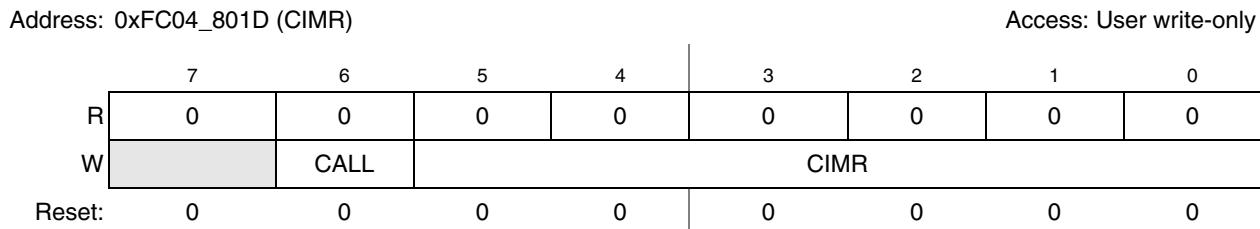


Figure 14-9. Clear Interrupt Mask Register (CIMR)

Table 14-11. CIMR Field Descriptions

Field	Description
7	Reserved, should be cleared.
6 CALL	Clear all bits in the IMR register, enabling all interrupt requests. 0 Only set those bits specified in the CIMR field. 1 Clear all bits in IMR register. The CIMR field is ignored.
5-0 CIMR	Clear the corresponding bit in the IMR register, enabling the interrupt request.

14.2.7 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. Once 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, once a level-*n* interrupt request is handled, 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.

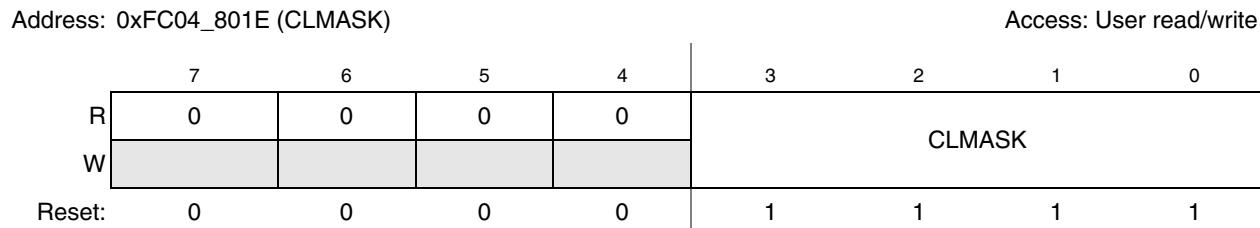


Figure 14-10. Current Level Mask Register (CLMASK)

Table 14-12. CLMASK Field Descriptions

Field	Description
7–4	Reserved, should be cleared.
3–0 CLMASK	Current level mask. Defines the level mask, where only interrupt levels greater than the current value are processed by the controller 0000 Level 1–7 requests are processed. 0001 Level 2–7 requests are processed. 0010 Level 3–7 requests are processed. 0011 Level 4–7 requests are processed. 0100 Level 5–7 requests are processed. 0101 Level 6–7 requests are processed. 0110 Level 7 requests are processed. 0111 All requests are masked. 1111 Level 1–7 requests are processed.

14.2.8 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. Once 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, once a level-*n* interrupt request is handled, the service routine restores the saved level mask value into the current level mask register to re-enable the lower priority requests.

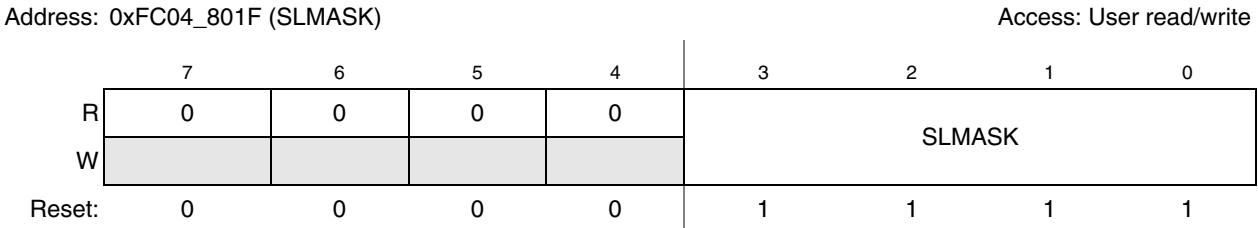


Figure 14-11. Saved Level Mask Register (SLMASK)

Table 14-13. SLMASK Field Descriptions

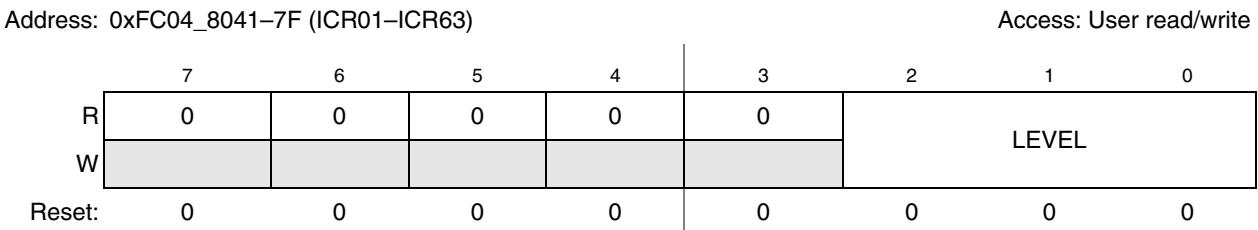
Field	Description
7–4	Reserved, should be cleared.
3–0 SLMASK	Saved level mask. Defines the saved level mask. See the CLMASK field definition for more information on the specific values.

14.2.9 Interrupt Control Register (ICR n , ($n = 01, 02, \dots, 63$))

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 14-12. Interrupt Control Registers (ICR n)Table 14-14. ICR n Field Descriptions

Field	Description
7–3	Reserved, should 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.

14.2.9.1 Interrupt Sources

Table 14-15 list the interrupt sources for each interrupt request line for INTC.

Table 14-15. Interrupt Source Assignment For INTC

Source	Module	Flag	Source Description	Flag Clearing Mechanism
0			Not Used	
1	EPORT	EPFR[EPF1]	Edge port flag 1	Write EPF1 = 1
2		EPFR[EPF4]	Edge port flag 4	Write EPF4 = 1
3		EPFR[EPF7]	Edge port flag 7	Write EPF7 = 1
4	PIT0	PCSR0[PIF]	PIT interrupt flag	Write PIF = 1 or write PMR
5	PIT1	PCSR0[PIF]	PIT interrupt flag	Write PIF = 1 or write PMR
6			Not Used	
7			Not Used	
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 CWIR[CWIC] = 1
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			Not Used	

Table 14-15. Interrupt Source Assignment For INTC (continued)

Source	Module	Flag	Source Description	Flag Clearing Mechanism
30	I ² C	I2SR[IIF]	I ² C Interrupt	Write I2SR[IIF] = 0
31	QSPI	QIR register	QSPI interrupt	Write 1 to appropriate QIR bit
32	DTIM0	DTER0 register	DMA Timer 0 interrupt	Write 1 to appropriate DTER0 bit
33	DTIM1	DTER1 register	DMA Timer 1 interrupt	Write 1 to appropriate DTER1 bit
34	DTIM2	DTER2 register	DMA Timer 2 interrupt	Write 1 to appropriate DTER2 bit
35	DTIM3	DTER3 register	DMA Timer 3 interrupt	Write 1 to appropriate DTER3 bit
36	FEC	EIR[TXF]	Transmit frame interrupt	Write EIR[TXF] = 1
37		EIR[TXB]	Transmit buffer interrupt	Write EIR[TXB] = 1
38		EIR[UN]	Transmit FIFO underrun	Write EIR[UN] = 1
39		EIR[RL]	Collision retry limit	Write EIR[RL] = 1
40		EIR[RXF]	Receive frame interrupt	Write EIR[RXF] = 1
41		EIR[RXB]	Receive buffer interrupt	Write EIR[RXB] = 1
42		EIR[MII]	MII interrupt	Write EIR[MII] = 1
43		EIR[LC]	Late collision	Write EIR[LC] = 1
44		EIR[HBERR]	Heartbeat error	Write EIR[HBERR] = 1
45		EIR[GRA]	Graceful stop complete	Write EIR[GRA] = 1
46		EIR[EBERR]	Ethernet bus error	Write EIR[EBERR] = 1
47		EIR[BABT]	Babbling transmit error	Write EIR[BABT] = 1
48		EIR[BABR]	Babbling receive error	Write EIR[BABR] = 1
49–61			Not Used	
62	SCM	SCMISR[CFEI]	Bus error interrupt	Write SCMISR[CFEI] = 1
63			Not Used	

14.2.10 Software and Level *n* IACK Registers (SWIACK*n*, L1IACK–L7IACK*n*)

The eight IACK registers 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: that is, 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 is a useful concept that 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.

Address: 0xFC04_80E0 (SWIACK) 0xFC04_80E0+4n (L_n IACK) $n=1:7$								Access: User read-only
R	7	6	5	4	3	2	1	0
W								
Reset (SWIACK):	0	0	0	0	0	0	0	0
Reset (L_n IACK):	0	0	0	1	1	0	0	0

Figure 14-13. Software and Level n IACK Registers (SWIACK n , L1IACK n –L7IACK n)

Table 14-16. SWIACK n and L x IACK 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 L_n IACK registers returns the highest priority unmasked interrupt source within the level. A write to any IACK register cause a bus error interrupt.

14.3 Functional Description

14.3.1 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 14.2.9.1, “Interrupt Sources”](#)). The higher numbered interrupt source has priority over the lower numbered interrupt source. See the below table for an example.

Table 14-17. 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 (ICR_n).

The operation of the interrupt controller can be broadly partitioned into three activities:

- Recognition
- Prioritization
- Vector determination during IACK

14.3.1.1 Interrupt Recognition

The interrupt controller continuously examines the request sources (IPR) and the interrupt mask register (IMR) to determine if there are active requests. This is the recognition phase. The interrupt force register (INTFRC) also factors into the generation of an active request.

14.3.1.2 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 to the processor core during this prioritization phase.

14.3.1.3 Interrupt Vector Determination

Once 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 INTC,

$$\text{vector_number} = 64 + \text{interrupt source number}$$

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 INTC:

```
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 = 24) is returned and it is the responsibility of the service routine to handle this error situation.

Note this protocol implies the interrupting peripheral is not accessed during the acknowledge cycle since 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 in conjunction with the processor's masking logic to form a dual-mask capability. In this mode of operation, 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.

14.3.2 Low-Power Wake-up Operation

The system control module (SCM) contains an 8-bit low-power control register (LPCR) used for controlling the low-power stop mode. This register must explicitly be programmed by software to enter low-power mode. It also contains a wake-up control register (WCR) that sets the priority level of the interrupt that is necessary to bring the device out of the specified low power mode. Refer to [Chapter 8, “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. Once 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 an interrupt request with a priority level greater than the value programmed in WCR[PRILVL].
5. Once 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 8.2.1, “Wake-up Control Register.”](#)

14.4 Initialization/Application Information

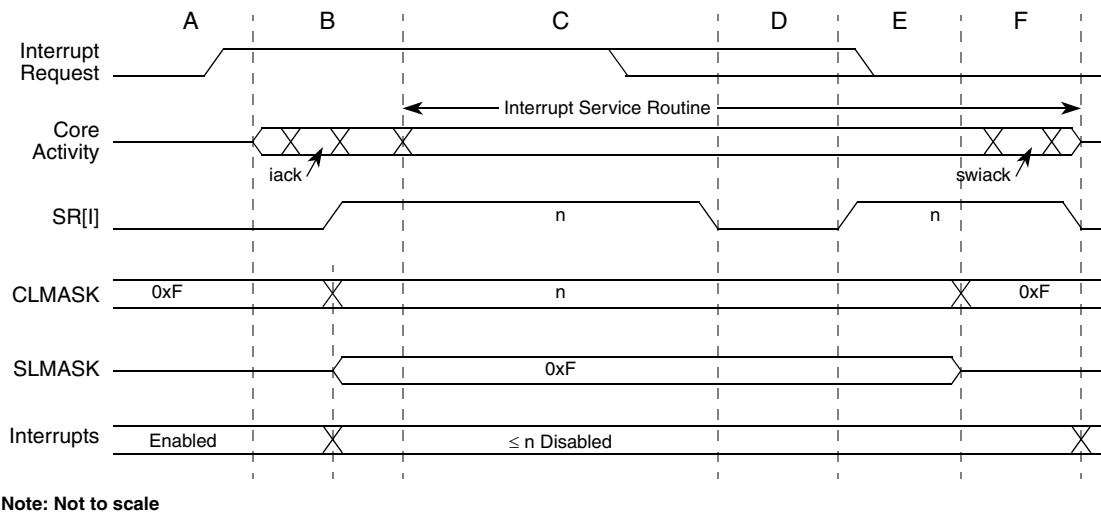
The interrupt controller’s reset state has all requests masked via the IMR. Before any interrupt requests are enabled, the following steps must be taken:

1. The ICONFIG register needs to be set to the desired system configuration.
2. All the ICR_n registers need to be programmed with the appropriate interrupt levels.
3. The reset value for the level mask registers (CLMASK and SLMASK) are set to -1, the value with no levels masked. Typically, these registers do not need to be modified before interrupts are enabled.
4. The appropriate interrupt vector tables and interrupt service routines must be loaded 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 0x0.

14.4.1 Interrupt Service Routines

This section focuses on the interaction of the interrupt masking functionality with the service routine.

[Figure 14-14](#) presents a timing diagram showing various phases during the execution of an interrupt service routine with the controller level masking functionality enabled. It is important to note the time scale in this diagram is not meant to be accurate.

**Figure 14-14. 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 is performed and the interrupt controller returns the appropriate vector number. As the interrupt acknowledge read is performed, the vector number is returned to the core. The contents of the CLMASK register are loaded into the SLMASK register, and the CLMASK register is updated to the level of the interrupt being acknowledged. 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 is executed. 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 then 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 that is ready to execute.

Obviously, there are many variations to the handling of the SR[I] and the CLMASK values to create a flexible, responsive system for handling interrupt requests within the device.

Chapter 15

Edge Port Module (EPORT)

15.1 Introduction

The edge port module (EPORT) has seven external interrupt pins, $\overline{\text{IRQ7}}\text{--}\overline{\text{IRQ1}}$. Each pin can be configured individually as a level-sensitive interrupt pin, an edge-detecting interrupt pin (rising edge, falling edge, or both), or a general-purpose input/output (I/O) pin.

NOTE

Not all EPORT signals may be output from the device. See [Chapter 2, “Signal Descriptions,”](#) to determine which signals are available.

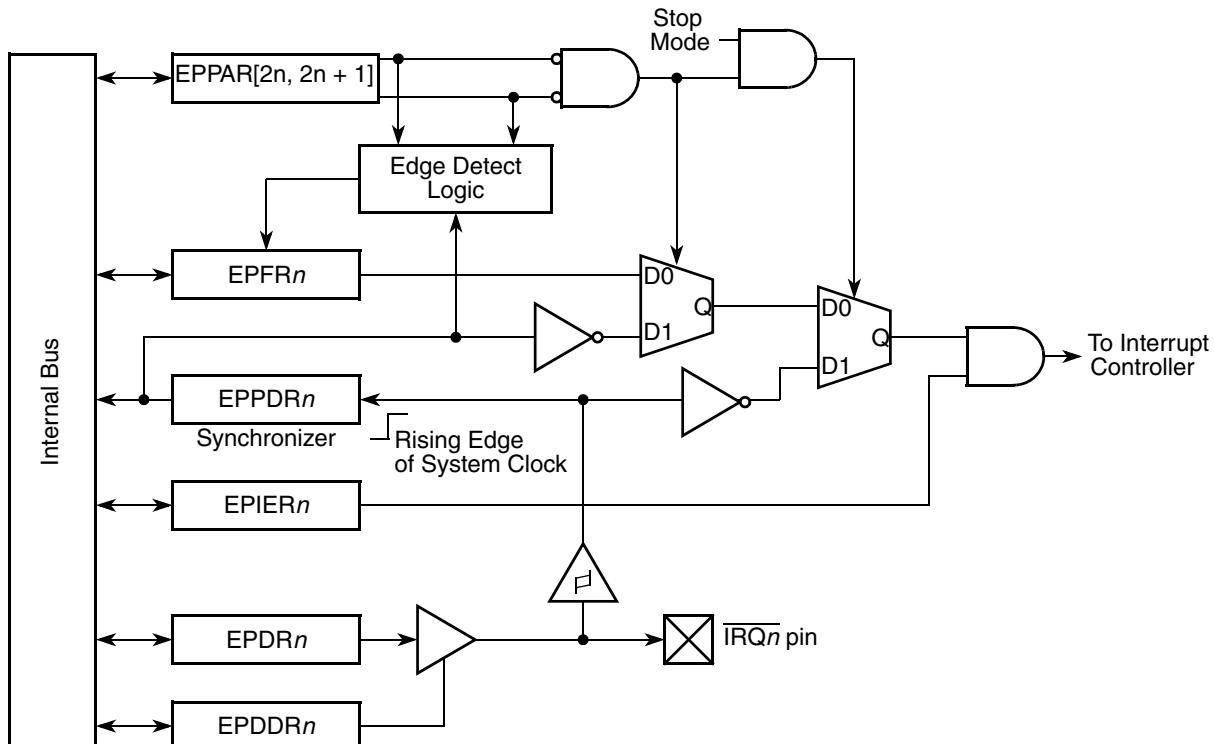


Figure 15-1. EPORT Block Diagram

NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to [Chapter 13, “General Purpose I/O Module”](#)) prior to configuring the edge port module.

15.2 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 8, “Power Management.”](#) Table 15-1 shows EPORT module operation in low-power modes, and describes how this module may exit from each mode.

NOTE

The wakeup control register (WCR) in the system control module specifies the interrupt level at or above which is needed to bring the device out of a low-power mode.

Table 15-1. Edge Port Module Operation in Low-power Modes

Low-power Mode	EPORT Operation	Mode Exit
Wait	Normal	Any \overline{IRQn} interrupt at or above level in WCR
Doze	Normal	Any \overline{IRQn} interrupt at or above level in WCR
Stop	Level-sensing only	Any \overline{IRQn} 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 either a selected edge or a low level on an external pin. In stop mode, there are no clocks 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 since no clocks are available.

15.3 Interrupt/GPIO Pin Descriptions

All EPORT pins default to general-purpose input pins at reset. The pin value is synchronized to the rising edge of FB_CLK when read from the EPORT pin data register (EPPDR). The values used in the edge/level detect logic are also synchronized to the rising edge of FB_CLK. These pins use Schmitt-triggered input buffers that have built-in hysteresis designed to decrease the probability of generating false, edge-triggered interrupts for slow rising and falling input signals.

When a pin is configured as an output, it is driven to a state whose level is determined by the corresponding bit in the EPORT data register (EPDR). All bits in the EPDR are set at reset.

15.4 Memory Map/Register Definition

This subsection describes the memory map and register structure. Refer to [Table 15-2](#) for a description of the EPORT memory map.

NOTE

Longword accesses to any of the edge port registers will result in a bus error.
Only byte and word accesses are allowed.

Table 15-2. Edge Port Module Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
Supervisor Access Only Registers¹					
0xFC08_8000	EPORT Pin Assignment Register (EPPAR)	16	R/W	0x0000	15.4.1/15-3
0xFC08_8002	EPORT Data Direction Register (EPDDR)	8	R/W	0x00	15.4.2/15-4
0xFC08_8003	EPORT Interrupt Enable Register (EPIER)	8	R/W	0x00	15.4.3/15-4
Supervisor/User Access Registers					
0xFC08_8004	EPORT Data Register (EPDR)	8	R/W	0xFF	15.4.4/15-5
0xFC08_8005	EPORT Pin Data Register (EPPDR)	8	R	See Section	15.4.5/15-5
0xFC08_8006	EPORT Flag Register (EPFR)	8	R/W	0x00	15.4.6/15-6

¹ User access to supervisor only address locations have no effect and result in a bus error

15.4.1 EPORT Pin Assignment Register (EPPAR)

The EPORT pin assignment register (EPPAR) controls the function of each pin individually.

Address: 0xFC08_8000 (EPPAR)								Access: Supervisor read/write								
R								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 15-2. EPORT Pin Assignment Register (EPPAR)

Table 15-3. EPPAR Field Descriptions

Field	Description
15–2 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{IRQn} interrupt. Pins configured as edge-triggered are latched and need not remain asserted for interrupt generation. A pin configured for edge detection can trigger an interrupt regardless of its configuration as input or output. Interrupt requests generated in the EPORT module can be masked by the interrupt controller module. EPPAR functionality is independent of the selected pin direction. Reset clears the EPPAn fields. 00 Pin \overline{IRQn} level-sensitive 01 Pin \overline{IRQn} rising edge triggered 10 Pin \overline{IRQn} falling edge triggered 11 Pin \overline{IRQn} both falling edge and rising edge triggered
1–0	Reserved, should be cleared.

15.4.2 EPORT Data Direction Register (EPDDR)

The EPORT data direction register (EPDDR) controls the direction of each one of the pins individually.

Address: 0xFC08_8002 (EPDDR)										Access: Supervisor read/write						
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	EPDD7	EPDD6	EPDD5	EPDD4	EPDD3	EPDD2	EPDD1									
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 15-3. EPORT Data Direction Register (EPDDR)**Table 15-4. EPDDR Field Descriptions**

Field	Description
7–1 EPDDn	Setting any bit in the EPDDR configures the corresponding pin as an output. Clearing any bit in EPDDR configures the corresponding pin as an input. Pin direction is independent of the level/edge detection configuration. Reset clears EPDD7–EPDD1. To use an EPORT pin as an external interrupt request source, its corresponding bit in EPDDR must be clear. Software can generate interrupt requests by programming the EPORT data register when the EPDDR selects output. 0 Corresponding EPORT pin configured as input 1 Corresponding EPORT pin configured as output
0	Reserved, should be cleared.

15.4.3 Edge Port Interrupt Enable Register (EPIER)

The EPORT interrupt enable register (EPIER) enables interrupt requests for each pin individually.

: 0xFC08_8003 (EPIER) Access: User read/write

	7	6	5	4	3	2	1	0
R	EPIE7	EPIE6	EPIE5	EPIE4	EPIE3	EPIE2	EPIE1	0
W	0	0	0	0	0	0	0	0
Reset:	0	0	0	0	0	0	0	0

Figure 15-4. EPORT Port Interrupt Enable Register (EPIER)

Table 15-5. EPIER Field Descriptions

Field	Description
7–1 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"> • The corresponding bit in the EPORT flag register (EPFR) is set or later becomes set. • The corresponding pin level is low and the pin is configured for level-sensitive operation. Clearing a bit in EPIER negates any interrupt request from the corresponding EPORT pin. Reset clears EPIE7–EPIE1. 0 Interrupt requests from corresponding EPORT pin disabled 1 Interrupt requests from corresponding EPORT pin enabled
0	Reserved, should be cleared.

15.4.4 Edge Port Data Register (EPDR)

The EPORT data register (EPDR) holds the data to be driven to the pins.

: 0xFC08_8004 (EPDR) Access: User read/write

	7	6	5	4	3	2	1	0
R	EPD7	EPD6	EPD5	EPD4	EPD3	EPD2	EPD1	0
W	1	1	1	1	1	1	1	1
Reset:	1	1	1	1	1	1	1	1

Figure 15-5. EPORT Port Data Register (EPDR)

Table 15-6. EPDR Field Descriptions

Field	Description
7–1 EPDn	Edge port data bits. Data written to EPDR is stored in an internal register; if any pin of the port is configured as an output, the bit stored for that pin is driven onto the pin. Reading EPDR returns the data stored in the register. Reset sets EPD7–EPD1.
0	Reserved, should be cleared.

15.4.5 Edge Port Pin Data Register (EPPDR)

The EPORT pin data register (EPPDR) reflects the current state of the pins.

: 0xFC08_8005 (EPPDR)								Access: User read-only
R	7	6	5	4	3	2	1	0
EPPD7	EPPD6	EPPD5	EPPD4	EPPD3	EPPD2	EPPD1	0	
W								

Reset: [IRQ7] [IRQ6] [IRQ5] [IRQ4] [IRQ3] [IRQ2] [IRQ1] 0

Figure 15-6. EPORT Port Pin Data Register (EPPDR)

Table 15-7. EPPDR Field Descriptions

Field	Description
7–1 EPPD n	Edge port pin data bits. The read-only EPPDR reflects the current state of the EPORT pins IRQ7–IRQ1. Writing to EPPDR has no effect, and the write cycle terminates normally. Reset does not affect EPPDR.
0	Reserved, should be cleared.

15.4.6 Edge Port Flag Register (EPFR)

The EPORT flag register (EPFR) individually latches EPORT edge events.

: 0xFC08_8006 (EPFR)								Access: User read/write
R	7	6	5	4	3	2	1	0
EPF7	EPF6	EPF5	EPF4	EPF3	EPF2	EPF1	0	

Reset: 0 0 0 0 0 0 0 0 0

Figure 15-7. EPORT Port Flag Register (EPFR)

Table 15-8. EPFR Field Descriptions

Field	Description
7–1 EPF n	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–EPF1. 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 (EPPAR n = 00), pin transitions do not affect this register. 0 Selected edge for IRQ n pin has not been detected. 1 Selected edge for IRQ n pin has been detected.
0	Reserved, should be cleared.

Chapter 16

Enhanced Direct Memory Access (eDMA)

16.1 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 which performs source and destination address calculations, and the actual data movement operations, along with local memory containing transfer control descriptors for each channel.

16.2 Block Diagram

Figure 16-1 is a block diagram of the eDMA module.

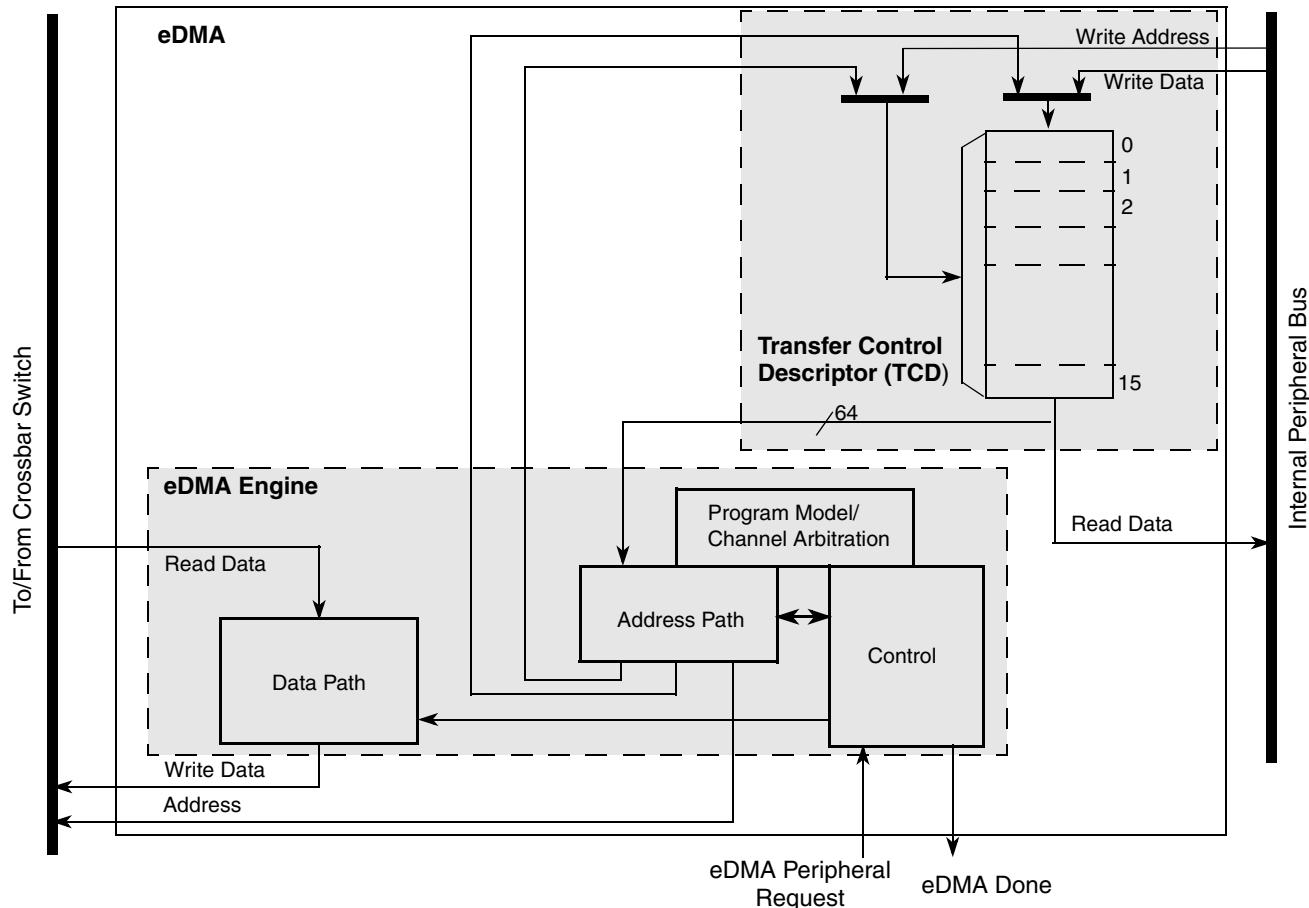


Figure 16-1. eDMA Block Diagram

16.3 Features

The eDMA is a highly-programmable data transfer engine, which has been 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 is not defined within the data packet itself. The eDMA module features the following:

- 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
- 16-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

Throughout this chapter, n is used to reference the channel number.

16.4 eDMA Modes of Operation

16.4.1 Normal Mode

In normal mode, the eDMA is used to transfer data between a source and a destination. The source and destination can be a memory block or an I/O block capable of operation with the eDMA.

16.4.2 Debug Mode

In debug mode, the eDMA will stop transferring data. If debug mode is entered during transfer of a block of data described by a minor loop in the current active channel's TCD, the eDMA will continue operation until completion of the minor loop.

16.5 External Signal Description

This section describes the external signals of the eDMA controller.

Table 16-1. External Signal List

Signal Name	I/O	Description
DREQ0	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.
DACK0	O	Indicates when the external DMA request has been acknowledged.

16.6 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.

Reading reserved bits in a register will return the value of zero and writes to reserved bits in a register will be ignored. Reading or writing to a reserved memory location will generate a bus error.

Table 16-2. 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	16.6.1/16-4
0xFC04_4004	eDMA Error Status Register (EDMA_ES)	32	R	0x0000_0000	16.6.2/16-4
0xFC04_400E	eDMA Enable Request Register (EDMA_ERQ)	16	R/W	0x0000	16.6.3/16-7
0xFC04_4016	eDMA Enable Error Interrupt Register (EDMA_EEI)	16	R/W	0x0000	16.6.4/16-8
0xFC04_4018	eDMA Set Enable Request (EDMA_SERQ)	8	W	0x00	16.6.5/16-8
0xFC04_4019	eDMA Clear Enable Request (EDMA_CERQ)	8	W	0x00	16.6.6/16-9
0xFC04_401A	eDMA Set Enable Error Interrupt Register (EDMA_SEEI)	8	W	0x00	16.6.7/16-10
0xFC04_401B	eDMA Clear Enable Error Interrupt Register (EDMA_CEEI)	8	W	0x00	16.6.8/16-10
0xFC04_401C	eDMA Clear Interrupt Request Register (EDMA_CINT)	8	W	0x00	16.6.9/16-11
0xFC04_401D	eDMA Clear Error Register (EDMA_CERR)	8	W	0x00	16.6.10/16-12
0xFC04_401E	eDMA Set START Bit Register (EDMA_SSRT)	8	W	0x00	16.6.11/16-12
0xFC04_401F	eDMA Clear DONE Status Bit Register (EDMA_CDNE)	8	W	0x00	16.6.12/16-13
0xFC04_4026	eDMA Interrupt Request Register (EDMA_INT)	32	R/W	0x0000	16.6.13/16-14
0xFC04_402E	eDMA Error Register (EDMA_ERR)	32	R/W	0x0000	16.6.14/16-14
0xFC04_4100 + hex(n)	eDMA Channel n Priority Register (DCHPRIn) for $n = 0\text{--}15$	8	R/W	See Register Description	16.6.15/16-15
0xFC04_5000 + hex($32 \times n$)	Transfer Control Descriptor (TCDn) for $n = 0\text{--}15$	256	R/W	See Register Description	16.6.16/16-16

16.6.1 eDMA Control Register (EDMA_CR)

The EDMA_CR defines the basic operating configuration of the eDMA. Arbitration can be configured to use either a fixed priority or a round robin scheme. In fixed priority arbitration, the highest priority channel requesting service is selected to execute. The priorities are assigned by the channel priority registers (see [Section 16.6.15, “eDMA Channel n Priority Registers \(DCHPRIn\)”\).](#) In round robin arbitration mode, the channel priorities are ignored and the channels are cycled through without regard to priority.

Address: 0xFC04_4000																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	0	0	0	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
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
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		
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		
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 16-2. eDMA Control Register (EDMA_CR)

Table 16-3. eDMA_CR Field Descriptions

Field	Description
31–3	Reserved, should be cleared.
2 ERCA	Enable round robin channel arbitration. 0 Fixed priority arbitration is used for channel selection. 1 Round robin arbitration is used for channel selection.
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 will resume when either the system exits debug mode or the EDBG bit is cleared.
0	Reserved, should be cleared.

16.6.2 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. All channel priority levels 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.

All configuration error conditions except the scatter/gather and minor loop link errors are reported as the channel is activated and assert an error interrupt request if enabled. 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 is terminated 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 will execute 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 will execute before the channel is terminated due to the destination bus error.

The occurrence of any type of error causes the eDMA engine to immediately stop the active channel, and the appropriate channel bit in the eDMA error register to be asserted. At the same time, the details of the error condition are loaded into the EDMA_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. Once the error status has been updated, the eDMA engine continues to operate by servicing the next appropriate channel. A channel that experiences an error condition is not automatically disabled. If a channel is terminated by an error and then issues another service request before the error is fixed, that channel will execute and terminate with the same error condition.

Address: 0xFC04_4004

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	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	CPE	0	0	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 16-3. eDMA Error Status Register (EDMA_ES)

Table 16-4. EDMA_ES Field Descriptions

Field	Description
31 VLD	Logical OR of all EDMA_ERR 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.
30–15	Reserved, should be cleared.
14 CPE	Channel priority error 0 No channel priority error. 1 The last recorded error was a configuration error in the channel priorities. Channel priorities are not unique.
13–12	Reserved, should be cleared.
11–8 ERRCHN	Error channel number. The channel number of the last recorded error. (excluding CPE errors)
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].
3 NCE	NBYTES/CITER configuration error. 0 No NBYTES/CITER configuration error. 1 The last recorded error was a configuration error detected in the TCDn_NBYTES or TCDn_CITER fields. <ul style="list-style-type: none">• TCDn_NBYTES is not a multiple of TCDn_ATTR[SSIZE] and TCDn_ATTR[DSIZE], or• TCDn_CITER[CITER] is equal to zero, or• TCDn_CITER[E_LINK] is not equal to TCDn_BITER[E_LINK].
2 SGE	Scatter/gather configuration error. 0 No scatter/gather configuration error. 1 The last recorded error was a configuration error detected in the TCDn_DLAST_SGA field. This field is checked at the beginning of a scatter/gather operation after major loop completion if TCDn_CSR[E_SG] is enabled. TCDn_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.

16.6.3 eDMA Enable Request Register (EDMA_ERQ)

The EDMA_ERQ register provides a bit map for the 16 implemented channels to enable the request signal for each channel. 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 that the request enable for a single channel can easily be modified without the need to perform a read-modify-write sequence to the EDMA_ERQ.

Both the DMA request input signal and this enable request flag must be asserted before a channel's hardware service request is accepted. The state of the eDMA enable request flag does not affect a channel service request made explicitly through software or a linked channel request.

Address: 0xFC04_400E

Access: User read/write

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

Figure 16-4. eDMA Enable Request Register (EDMA_ERQ)

Table 16-5. EDMA_ERQ Field Descriptions

Field	Description
15–0 ERQ _n	Enable DMA Request <i>n</i> . 0 The DMA request signal for channel <i>n</i> is disabled. 1 The DMA request signal for channel <i>n</i> is enabled.

The assignments between the DMA requests from the peripherals to the channels of the eDMA are shown in [Table 16-6](#).

Table 16-6. DMA Request Summary for eDMA

Channel	Source	Description
0	DREQ0	External DMA request 0
1	UISR0[FFULL/RXRDY]	UART0 Receive
2	UISR0[TXRDY]	UART0 Transmit
3	UISR1[FFULL/RXRDY]	UART1 Receive
4	UISR1[TXRDY]	UART1 Transmit
5	UISR2[FFULL/RXRDY]	UART2 Receive
6	UISR2[TXRDY]	UART2 Transmit
7	DTER0[CAP] or DTER0[REF]	Timer 0
8	DTER1[CAP] or DTER1[REF]	Timer 1
9	DTER2[CAP] or DTER2[REF]	Timer 2

Table 16-6. DMA Request Summary for eDMA (continued)

Channel	Source	Description
10	DTER3[CAP] or DTER3[REF]	Timer 3
11–15	Software Activated	Activated explicitly by setting the TCDn_CSR[START] bit

As a given channel completes the processing of its major iteration count, there is a flag in the transfer control descriptor that may affect the ending state of the EDMA_ERQ bit for that channel. If the TCDn_CSR[D_REQ] bit is set, then the corresponding EDMA_ERQ bit is cleared, disabling the DMA request. If the D_REQ bit is cleared, the state of the EDMA_ERQ bit is unaffected.

16.6.4 eDMA Enable Error Interrupt Registers (EDMA_EEI)

The EDMA_EEI register provides a bit map for the 16 channels to enable the error interrupt signal for each channel. 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 that 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 register.

Both 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.

Address: 0xFC04_4016

Access: User read/write

R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W	EEI15	EEI14	EEI13	EEI12	EEI11	EEI10	EEI9	EEI8	EEI7	EEI6	EEI5	EEI4	EEI3	EEI2	EEI1	EEI0
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-5. eDMA Enable Error Interrupt Register (EDMA_EEI)**Table 16-7. EDMA_EEI Field Descriptions**

Field	Description
15–0 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.

16.6.5 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 to enable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQ to be set. Setting the SAER bit provides a global set function, forcing the entire contents of EDMA_ERQ to be set. Reads of this register return all zeroes.

Address: 0xFC04_4018

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		SAER				SERQ		
Reset	0	0	0	0	0	0	0	0

Figure 16-6. eDMA Set Enable Request Register (EDMA_SERQ)

Table 16-8. EDMA_SERQ Field Descriptions

Field	Description
7	Reserved, should be cleared.
6 SAER	Set all enable requests. 0 Set only those EDMA_ERQ bits specified in the SERQ field. 1 Set all bits in EDMA_ERQ.
5–4	Reserved, should be cleared.
3–0 SERQ	Set enable request. Sets the corresponding bit in EDMA_ERQ

16.6.6 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 to disable the DMA request for a given channel. The data value on a register write causes the corresponding bit in the EDMA_ERQ to be cleared. Setting the CAER bit provides a global clear function, forcing the entire contents of the EDMA_ERQ to be cleared, disabling all DMA request inputs. Reads of this register return all zeroes.

Address: 0xFC04_4019

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W		CAER				CERQ		
Reset	0	0	0	0	0	0	0	0

Figure 16-7. eDMA Clear Enable Request Register (EDMA_CERQ)

Table 16-9. EDMA_CERQ Field Descriptions

Field	Description
7	Reserved, should be cleared.
6 CAER	Clear all enable requests. 0 Clear only those EDMA_ERQ bits specified in the CERQ field. 1 Clear all bits in EDMA_ERQ.

Table 16-9. EDMA_CERQ Field Descriptions (continued)

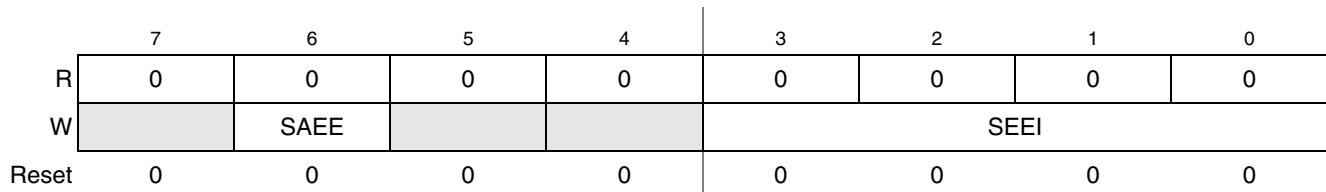
Field	Description
5–4	Reserved, should be cleared.
3–0 CERQ	Clear enable request. Clears the corresponding bit in EDMA_ERQ.

16.6.7 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 to enable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_EEI to be set. Setting the SAEE bit provides a global set function, forcing the entire contents of EDMA_EEI to be set. Reads of this register return all zeroes.

Address: 0xFC04_401A

Access: User write-only

**Figure 16-8. eDMA Set Enable Error Interrupt Register (EDMA_SEEI)****Table 16-10. EDMA_SEEI Field Descriptions**

Field	Description
7	Reserved, should be cleared.
6 SAEE	Sets all enable error interrupts. 0 Set only those EDMA_EEI bits specified in the SEEI field. 1 Sets all bits in EDMA_EEI.
5–4	Reserved, should be cleared.
3–0 SEEI	Set enable error interrupt. Sets the corresponding bit in EDMA_EEI.

16.6.8 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 to disable the error interrupt for a given channel. The data value on a register write causes the corresponding bit in the EDMA_EEI to be cleared. Setting the CAEE bit provides a global clear function, forcing the entire contents of the EDMA_EEI to be cleared, disabling all DMA request inputs. Reads of this register return all zeroes.

Address: 0xFC04_401B

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0		
W		CAEE					CEEI	
Reset	0	0	0	0	0	0	0	0

Figure 16-9. eDMA Clear Enable Error Interrupt Register (EDMA_CEEI)**Table 16-11. EDMA_CEEI Field Descriptions**

Field	Description
7	Reserved, should be cleared.
6 CAEE	Clear all enable error interrupts. 0 Clear only those EDMA_EEI bits specified in the CEEI field. 1 Clear all bits in EDMA_EEI.
5–4	Reserved, should be cleared.
3–0 CEEI	Clear enable error interrupt. Clears the corresponding bit in EDMA_EEI.

16.6.9 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 to disable the interrupt request for a given channel. The given value on a register write causes the corresponding bit in the EDMA_INT to be cleared. Setting the CAIR bit provides a global clear function, forcing the entire contents of the EDMA_INT to be cleared, disabling all DMA interrupt requests. Reads of this register return all zeroes.

Address: 0xFC04_401C

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0		
W		CAIR					CINT	
Reset	0	0	0	0	0	0	0	0

Figure 16-10. eDMA Clear Interrupt Request (EDMA_CINT)**Table 16-12. EDMA_CINT Field Descriptions**

Field	Description
7	Reserved, should be cleared.
6 CAIR	Clear all interrupt requests. 0 Clear only those EDMA_INT bits specified in the CINT field. 1 Clear all bits in EDMA_INT.

Table 16-12. EDMA_CINT Field Descriptions (continued)

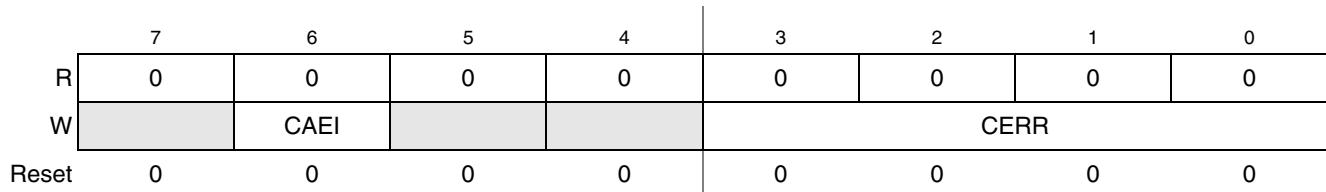
Field	Description
5–4	Reserved, should be cleared.
3–0 CINT	Clear interrupt request. Clears the corresponding bit in EDMA_INT.

16.6.10 eDMA Clear Error Register (EDMA_CERR)

The EDMA_CERR provides a simple memory-mapped mechanism to clear a given bit in the EDMA_ERR 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 to be cleared. Setting the CAEI bit provides a global clear function, forcing the entire contents of the EDMA_ERR to be cleared, clearing all channel error indicators. Reads of this register return all zeroes.

Address: 0xFC04_401D

Access: User write-only

**Figure 16-11. eDMA Clear Error Register (EDMA_CERR)****Table 16-13. EDMA_CERR Field Descriptions**

Field	Description
7	Reserved, should be cleared.
6 CAEI	Clear all error indicators. 0 Clear only those EDMA_ERR bits specified in the CERR field. 1 Clear all bits in EDMA_ERR.
5–4	Reserved, should be cleared.
3–0 CERR	Clear error indicator. Clears the corresponding bit in EDMA_ERR.

16.6.11 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. Reads of this register return all zeroes.

Address: 0xFC04_401E

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0	0	0
W	SAST						SSRT	
Reset	0	0	0	0	0	0	0	0

Figure 16-12. eDMA Set START Bit Register (EDMA_SSRT)

Table 16-14. EDMA_SSRT Field Descriptions

Field	Description
7	Reserved, should be cleared.
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].
5–4	Reserved, should be cleared.
3–0 SSRT	Set START bit. Sets the corresponding bit in TCDn_CSR[START].

16.6.12 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. Reads of this register return all zeroes.

Address: 0xFC04_401F

Access: User write-only

	7	6	5	4	3	2	1	0
R	0	0	0	0	0	0		
W	CADN						CDNE	
Reset	0	0	0	0	0	0	0	0

Figure 16-13. eDMA Clear DONE Status Bit Register (EDMA_CDNE)

Table 16-15. EDMA_CDNE Field Descriptions

Field	Description
7	Reserved, should be cleared.
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]
5–4	Reserved, should be cleared.
3–0 CDNE	Clear DONE bit. Clears the corresponding bit in TCDn_CSR[DONE].

16.6.13 eDMA Interrupt Request Register (EDMA_INT)

The EDMA_INT provide a bit map for the 16 channels signaling the presence of an interrupt request for each channel. The eDMA engine signals the occurrence of a programmed interrupt upon the completion of a data transfer as defined in the transfer control descriptor by setting the appropriate bit in this register. The outputs of this register are directly routed to the interrupt controller (INTC). During the execution of the interrupt service routine associated with any given channel, it is 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 one 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.

Address: 0xFC04_4026

Access: User read/write

	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	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-14. eDMA Interrupt Request Register (EDMA_INT)

Table 16-16. EDMA_INT Field Descriptions

Field	Description
15–0 INT n	eDMA interrupt request n 0 The interrupt request for channel n is cleared. 1 The interrupt request for channel n is active.

16.6.14 eDMA Error Register (EDMA_ERR)

The EDMA_ERR provide a bit map for the 16 channels signaling the presence of an error for each channel. 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, and 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, since 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_402E

Access: User read/write

	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	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 16-15. eDMA Error (EDMA_ERR) Register

Table 16-17. EDMA_ERR Field Descriptions

Field	Description
15–0 <i>Earn</i>	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.

16.6.15 eDMA Channel *n* Priority Registers (DCHPRI*n*)

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. The channel priorities are evaluated by numeric value; i.e., 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 will be reported. The range of the priority value is limited to the values of 0 through 15.

Channel preemption is enabled on a per-channel basis by setting the DCHPRI*n*[ECP] bit. Channel preemption allows the executing channel's data transfers to be temporarily suspended in favor of starting a higher priority channel. Once the preempting channel has completed all of 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 will be suspended and the higher priority channel will be serviced. Nested preemption (attempting to preempt a preempting channel) is not supported. Once a preempting channel begins execution, it cannot be preempted. Preemption is only available when fixed arbitration is selected.

Address: 0xFC04_4100 + *n*, where *n* = 0–15

Access: User read/write

	7	6	5	4	3	2	1	0
R	ECP	0	0	0	CHPRI			
W					-1	-1	-1	-1
Reset	0	0	0	0	-1	-1	-1	-1

¹ Reset value for the channel priority fields, CHPRI, is equal to the corresponding channel number for each priority register, i.e., DCHPRI15[CHPRI] = 0b1111.

Figure 16-16. eDMA Channel *n* Priority Register (DCHPRI*n*)

Table 16-18. DCHPRI_n Field Descriptions

Field	Description
7 ECP	Enable channel preemption. 0 Channel n cannot be suspended by a higher priority channel's service request. 1 Channel n can be temporarily suspended by the service request of a higher priority channel.
6-4	Reserved, should be cleared.
3-0 CHPRI	Channel n arbitration priority. Channel priority when fixed-priority arbitration is enabled.

16.6.16 Transfer Control Descriptors (TCDn)

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 15. The definitions of each TCD n are presented as 11 registers of 16 or 32 bits. [Table 16-19](#) is a register list of the basic TCD structure.

Table 16-19. TCD*n* Memory Structure

eDMA Offset	TCD n Register Name	Abbreviation	Width (bits)
0xFC04_5000 + (0x20 × n)	Source Address	TCD n _SADDR	32
0xFC04_5004 + (0x20 × n)	Transfer Attributes	TCD n _ATTR	16
0xFC04_5006 + (0x20 × n)	Signed Source Address Offset	TCD n _SOFF	16
0xFC04_5008 + (0x20 × n)	Minor Byte Count	TCD n _NBYTES	32
0xFC04_500C + (0x20 × n)	Last Source Address Adjustment	TCD n _SLAST	32
0xFC04_5010 + (0x20 × n)	Destination Address	TCD n _DADDR	32
0xFC04_5014 + (0x20 × n)	Current Minor Loop Link, Major Loop Count	TCD n _CITER	16
0xFC04_5016 + (0x20 × n)	Signed Destination Address Offset	TCD n _DOFF	16
0xFC04_5018 + (0x20 × n)	Last Destination Address Adjustment/Scatter Gather Address	TCD n _DLAST_SGA	32
0xFC04_501C + (0x20 × n)	Beginning Minor Loop Link, Major Loop Count	TCD n _BITER	16
0xFC04_501E + (0x20 × n)	Control and Status	TCD n _CSR	16

The following figures and tables define the fields of the TCD*n* structure.

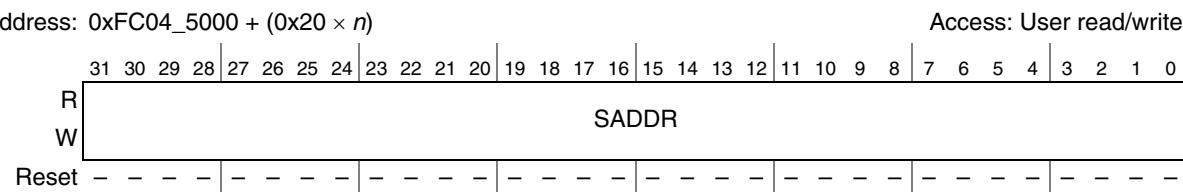
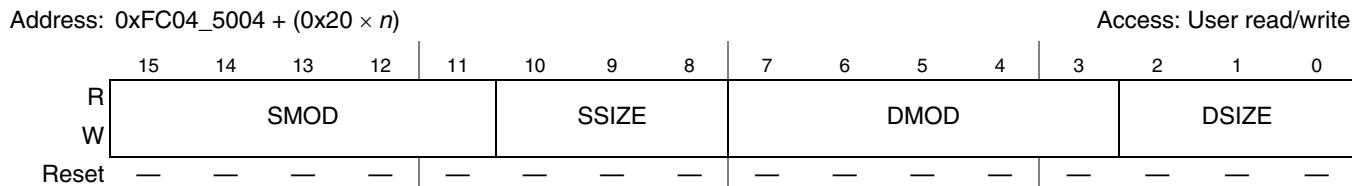


Figure 16-17. TCDn Source Address (TCDn_SADDR)

Table 16-20. TCDn_SADDR Field Descriptions

Field	Description
31–0 SADDR	Source address. Memory address pointing to the source data.

**Figure 16-18. TCDn Transfer Attributes (TCDn_ATTR)****Table 16-21. TCDn_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 which is specified to be either the value after SADDR + SOFF calculation is performed or the original register value. The setting of this field provides the ability to easily implement a circular data queue. For data queues requiring power-of-2 "size" bytes, the queue should start at a 0-modulo-size address and the SMOD field should be set to the appropriate value for the queue, freezing the desired number of upper address bits. The value programmed into this field specifies the number of lower address bits that are allowed to change. For 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 will cause a configuration error.
7–3 DMOD	Destination address modulo. See the SMOD definition.
2–0 DSIZE	Destination data transfer size. See the SSIZE definition.

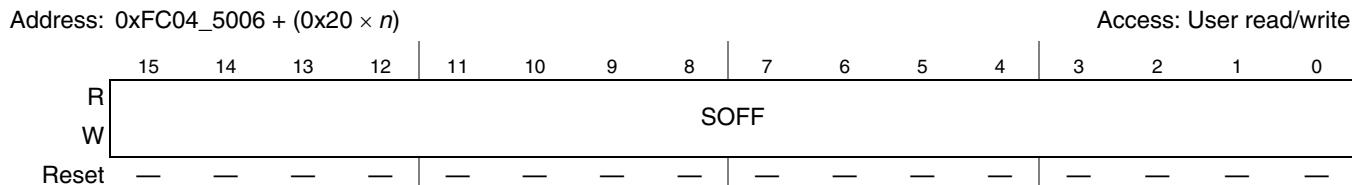
**Figure 16-19. TCDn Signed Source Address Offset (TCDn_SOFF)**

Table 16-22. 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.

Address: 0xFC04_5008 + (0x20 × n) Access: User read/write

Figure 16-20. TCDn Minor Byte Count (TCDn_NBYTES)

Table 16-23. 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 is activated, the contents of the appropriate TCD are loaded into the eDMA engine, and the appropriate reads and writes performed until the minor byte transfer count has been transferred. This is an indivisible operation and cannot be halted. (Although, it may be stalled by using the bandwidth control field, or via preemption.) Once the minor count is exhausted, the current values of the SADDR and DADDR 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>Note: An NBYTES value of 0x0000_0000 is interpreted as a 4 GB transfer.</p>

Address: 0xFC04_500C + (0x20 × n) Access: User read/write

Figure 16-21. TCDn Source Last Address Adjustment (TCDn_SLAST)

Table 16-24. TCDn_SLAST Field Descriptions

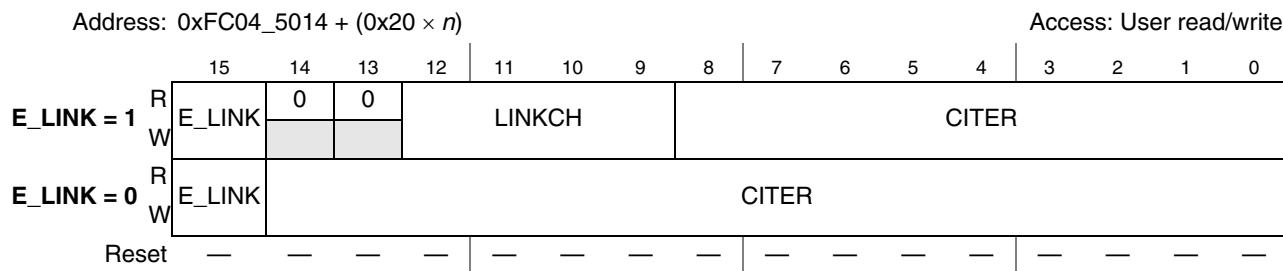
Field	Description
31–0 SLAST	Last source address adjustment. Adjustment value added to the source address at the completion of the major iteration count. This value can be applied to “restore” the source address to the initial value, or adjust the address to reference the next data structure.

Address: 0xFC04_5010 + (0x20 × n) Access: User read/write

Figure 16-22. TCD n Destination Address (TCD n .DADDR)

Table 16-25. TCD n _DADDR Field Descriptions

Field	Description
31–0 DADDR	Destination address. Memory address pointing to the destination data.

**Figure 16-23. TCD n Current Major Iteration Count (TCD n _CITER)****Table 16-26. 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 TCDn_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>Note: This bit must be equal to the BITER.E_LINK bit otherwise a configuration error will be reported.</p>
14–13	Reserved, should be cleared.
12–9 LINKCH	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 four bits by setting that channel's TCD n _CSR[START] bit.
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. Once 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>Note: When the CITER field is initially loaded by software, it must be set to the same value as that contained in the BITER field.</p> <p>Note: If the channel is configured to execute a single service request, the initial values of BITER and CITER should be 0x0001.</p>

Enhanced Direct Memory Access (eDMA)

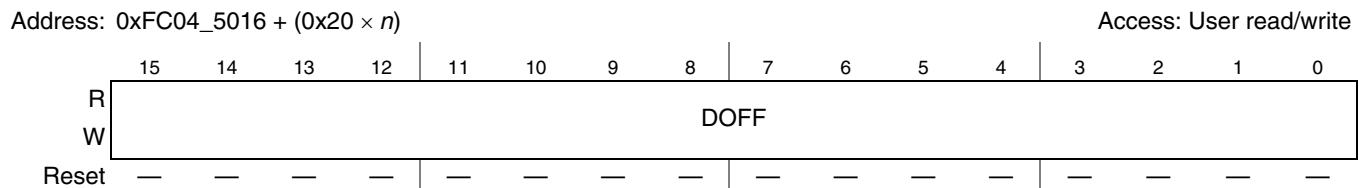


Figure 16-24. TCD n Destination Address Signed Offset (TCD n _DOFF)

Table 16-27. 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.

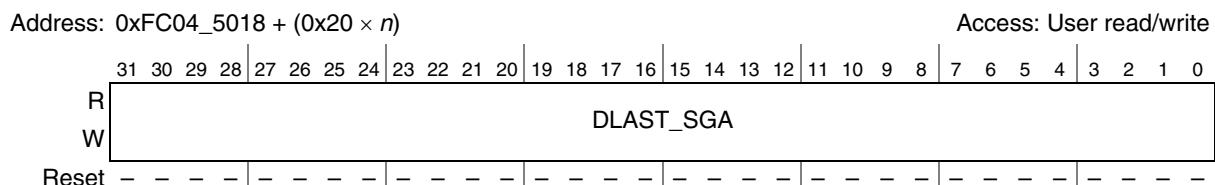


Figure 16-25. TCD n Destination Last Address Adjustment (TCD n _DLAST_SGA)

Table 16-28. 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">Adjustment value added to the destination address at the completion of the major iteration count. This value can be applied to restore the destination address to the initial value or adjust the address to reference the next data structure. else <ul style="list-style-type: none">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.

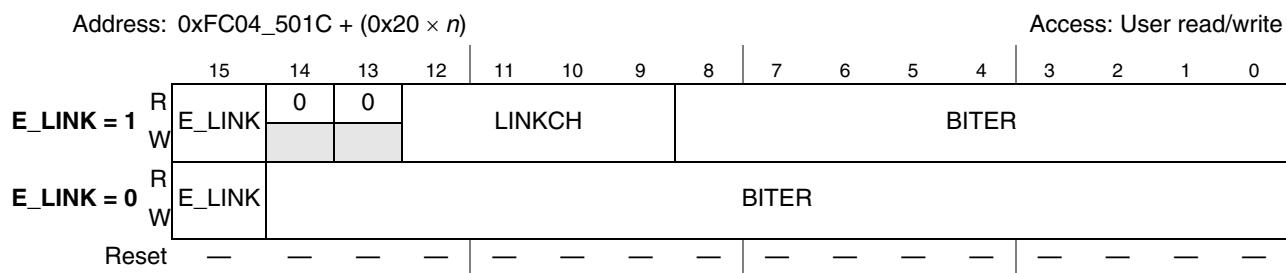


Figure 16-26. TCD n Beginning Major Iteration Count (TCD n _BITER)

Table 16-29. 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 is disabled, the BITER value is extended to 15 bits in place of a link channel number. If the major loop is exhausted, this link mechanism is suppressed in favor of the MAJOR_E_LINK channel linking.</p> <p>0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.</p> <p>Note: When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field, otherwise a configuration error will be reported. As the major iteration count is exhausted, the contents of this field is reloaded into the CITER field.</p>
14–13	Reserved, should be cleared.
12–9 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 four bits by setting that channel's TCDn_CSR[START] bit.</p> <p>0–15 Link to DMA channel 0–15</p> <p>Note: When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field, otherwise a configuration error will be 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>Note: When the TCD is first loaded by software, this field must be set equal to the corresponding CITER field, otherwise a configuration error will be 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)

Access: User read/write

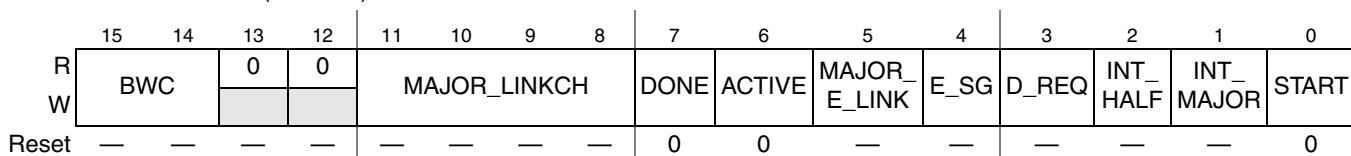
**Figure 16-27. TCDn Control and Status (TCDn_CSR)**

Table 16-30. TCDn_CSR Field Descriptions

Field	Description
15–14 BWC	<p>Bandwidth control. This two-bit field provides a mechanism to effectively throttle the amount of bus bandwidth consumed by the eDMA. In general, as the eDMA processes the 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"> 00 No eDMA engine stalls 01 Reserved 10 eDMA engine stalls for 4 cycles after each r/w 11 eDMA engine stalls for 8 cycles after each r/w
13–12	Reserved, should be cleared.
11–8 MAJOR_LINKCH	<p>Link channel number.</p> <p>If (MAJOR_E_LINK = 0) then</p> <ul style="list-style-type: none"> • No channel-to-channel linking (or chaining) is performed after the major loop counter is exhausted. <p>else</p> <ul style="list-style-type: none"> • After the major loop counter is exhausted, the eDMA engine initiates a channel service request at the channel defined by these four bits by setting that channel's TCDn_CSR[START] bit. <p>0–15 Link to DMA channel 0–15</p>
7 DONE	<p>Channel done. This flag indicates the eDMA has completed the major loop. It is set by the eDMA engine as the CITER count reaches zero; it is cleared by software, or the hardware when the channel is activated.</p> <p>Note: This bit must be cleared in order to write the MAJOR_E_LINK or E_SG bits.</p>
6 ACTIVE	Channel active. This flag signals the channel is currently in execution. It is set when channel service begins, and is cleared by the eDMA engine as the minor loop completes or if any error condition is detected.
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>Note: 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"> 0 The channel-to-channel linking is disabled. 1 The channel-to-channel linking is enabled.
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 which is loaded as the transfer control descriptor into the local memory.</p> <p>Note: 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"> 0 The current channel's TCD is “normal” format. 1 The current channel's TCD specifies a scatter gather format. The DLAST_SGA field provides a memory pointer to the next TCD to be loaded into this channel after the major loop completes its execution.
3 D_REQ	<p>Disable request. If this flag is set, the eDMA hardware automatically clears the corresponding DMAERQ bit when the current major iteration count reaches zero.</p> <ul style="list-style-type: none"> 0 The channel's DMAERQ bit is not affected. 1 The channel's DMAERQ bit is cleared when the major loop is complete.

Table 16-30. TCDn_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 is disabled 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.

16.7 Functional Description

This section provides an overview of the microarchitecture and functional operation of the eDMA module.

16.7.1 eDMA Microarchitecture

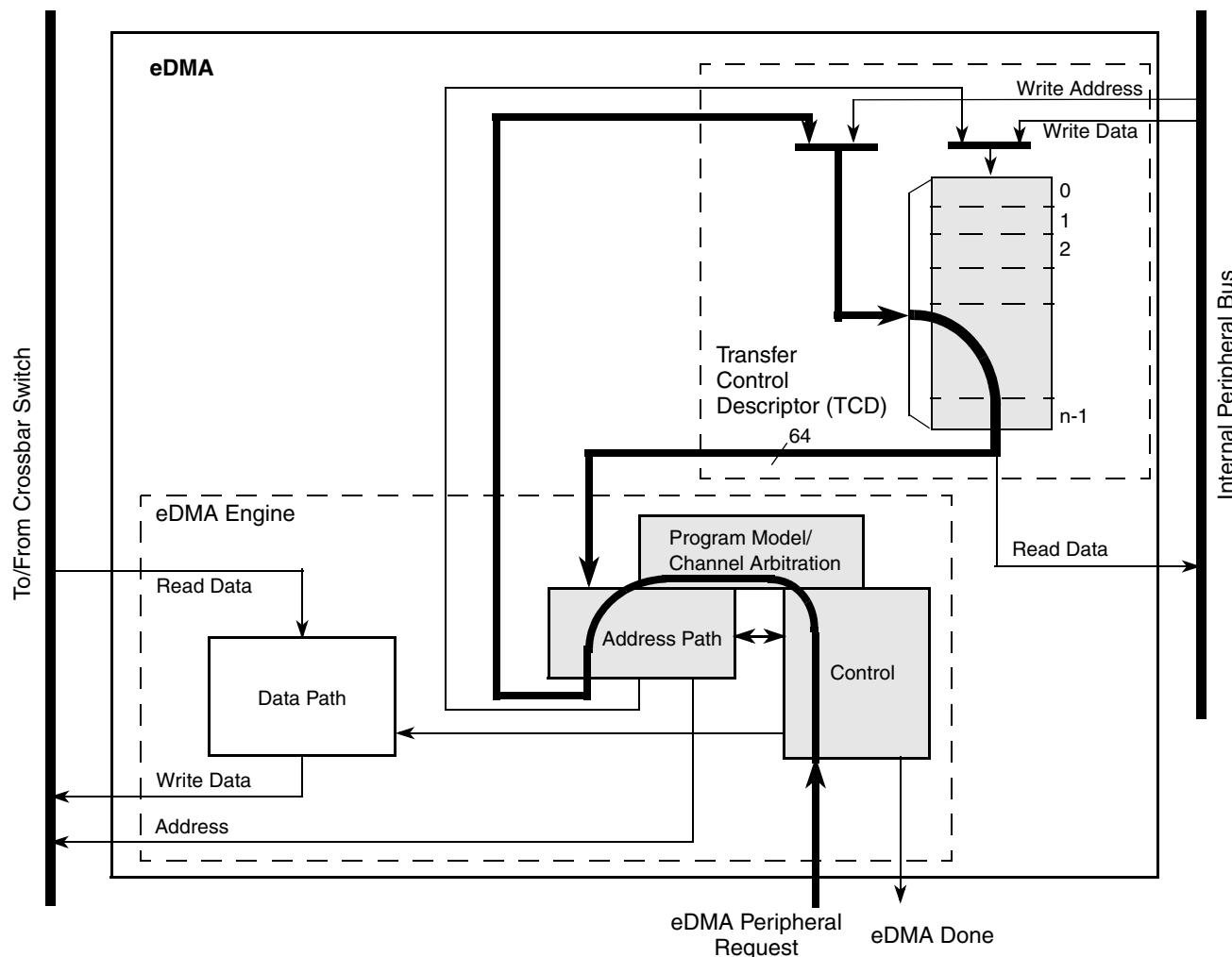
The eDMA module is partitioned into two major modules: the eDMA engine and the transfer control descriptor local memory. Additionally, the eDMA engine is further partitioned into four submodules, which are detailed below.

- eDMA Engine
 - Address Path: This block implements registered versions of two channel transfer control descriptors: channel x and channel y, and handles all master bus address calculations. All the channels provide the exact 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. Once 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 DCHPRIIn[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 is 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. Once the minor loop completes execution, the address path hardware writes the new values for the TCDn_{SADDR, DADDR, CITER} back to local memory. If the major iteration count is exhausted, additional processing is performed, including the final address pointer updates, reloading the TCDn_CITER field, and a possible fetch of the next TCDn 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 (the address phase), while the data path module implements the 2nd stage of the pipeline (the 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 the 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 been 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, handling accesses from both 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.

16.7.2 eDMA Basic Data Flow

The basic flow of a data transfer can be partitioned into three segments. As shown in [Figure 16-28](#), 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 is performed, either using the fixed-priority or round-robin algorithm. After the arbitration is complete, the activated channel number is sent through the address path and converted into the required address to access the 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 16-28. eDMA Operation, Part 1**

In the second part of the basic data flow as shown in [Figure 16-29](#), 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 been transferred.

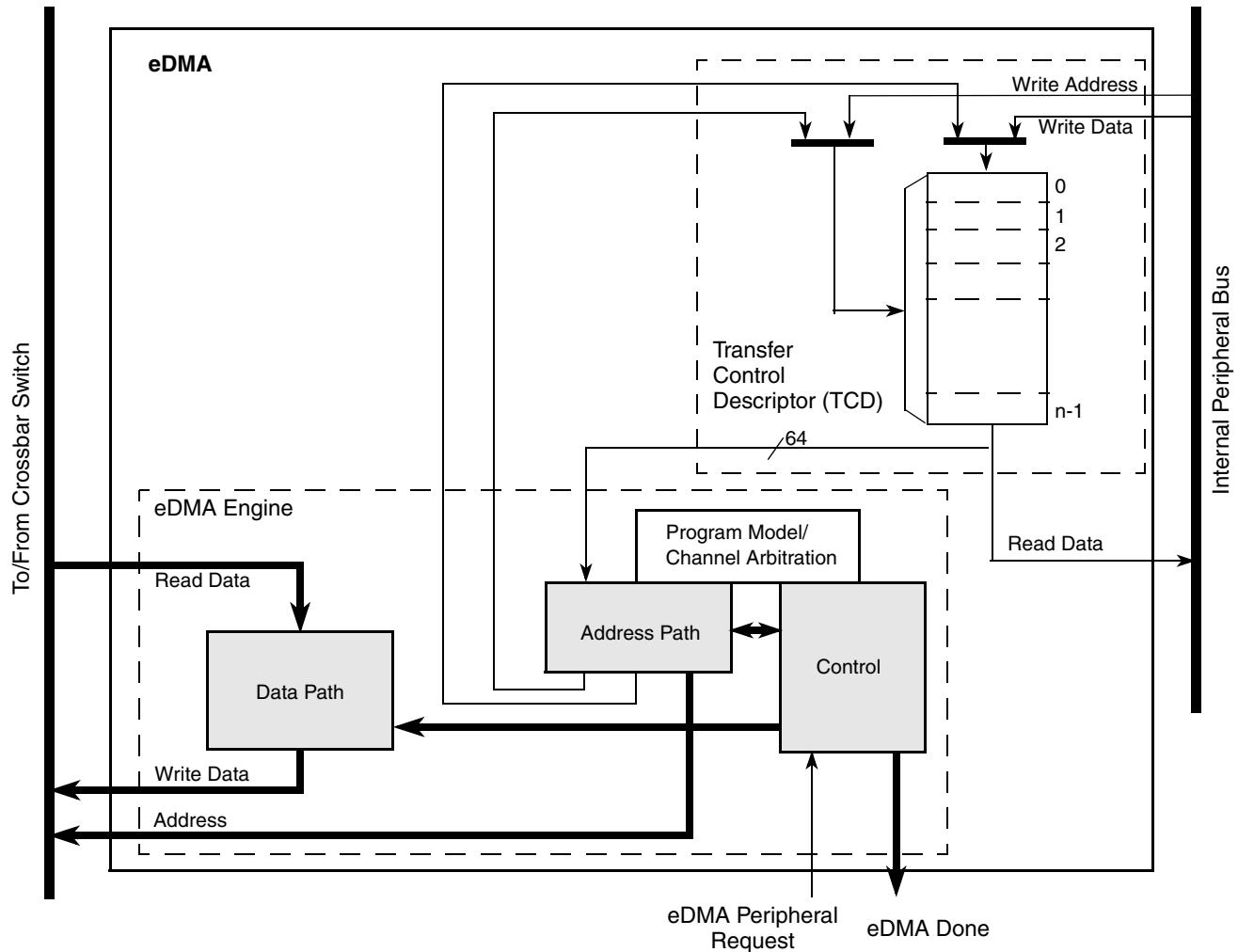


Figure 16-29. eDMA Operation, Part 2

Once the minor byte count has been moved, the final phase of the basic data flow is performed. In this segment, the address path logic performs the required updates to certain fields in the appropriate TCD, e.g., SADDR, DADDR, CITER. If the major iteration count is exhausted, then there are additional operations which are performed. These include the final address adjustments and reloading of the BITER field into the CITER. Additionally, assertion of an optional interrupt request occurs at this time, as does a possible fetch of a new TCD from memory using the scatter/gather address pointer included in the descriptor. The updates to the TCD memory and the assertion of an interrupt request are shown in [Figure 16-30](#).

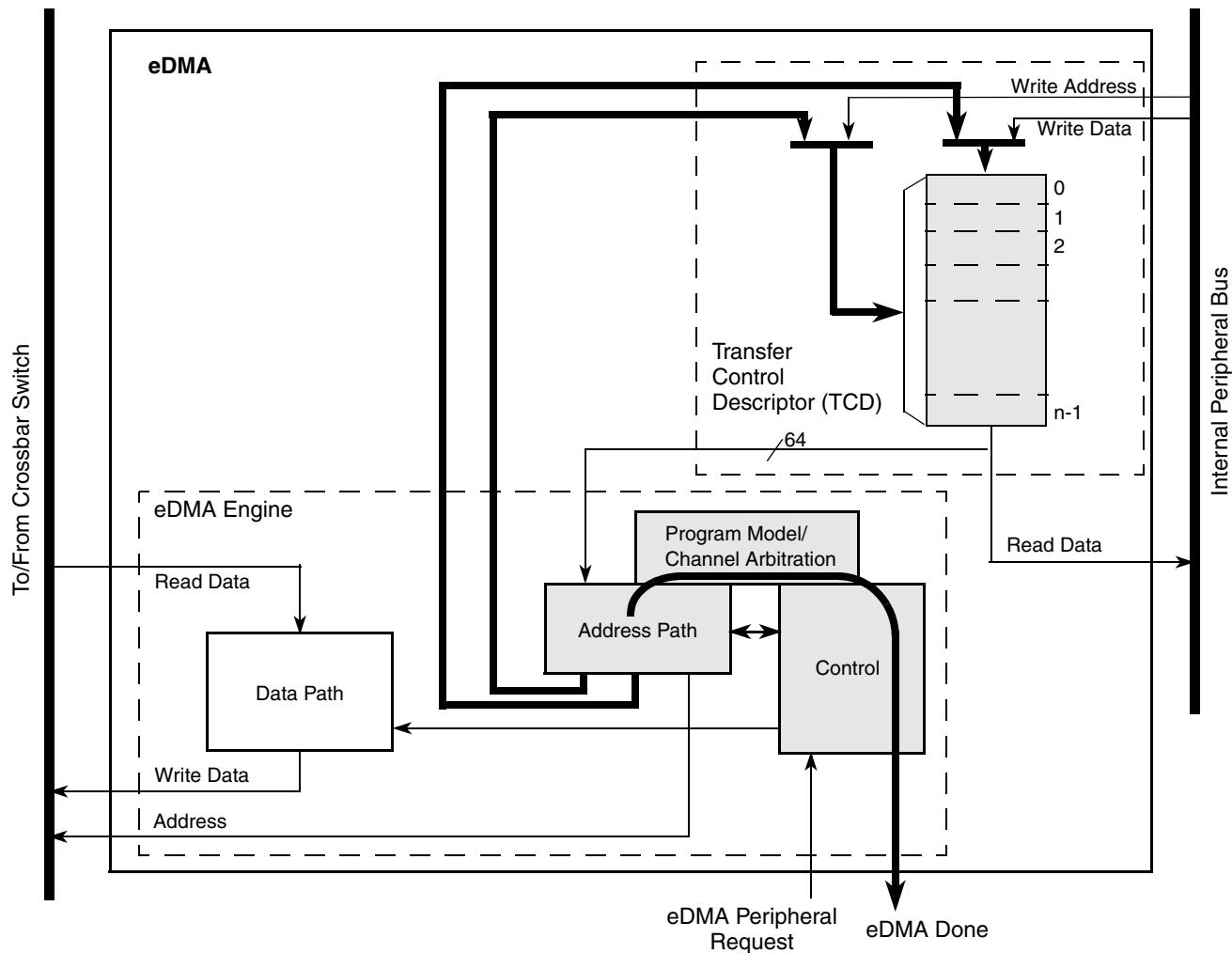


Figure 16-30. eDMA Operation, Part 3

16.8 Initialization/Application Information

16.8.1 eDMA Initialization

A typical initialization of the eDMA would have 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_EEI 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 either software (setting the TCD n _CSR[START] bit) or by hardware (slave device asserting its eDMA peripheral request signal).

Once 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 will read the entire TCD, including the TCD control and status fields shown in [Table 16-31](#), for the selected channel into its internal address path module. As the TCD is being 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 ($TCDn_NBYTES$) have been transferred. When the 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 is executed, i.e. interrupts, major loop channel linking, and scatter/gather operations, if enabled.

Table 16-31. TCD Control and Status Fields

$TCDn_CSR$ Field Name	Description
START	Control bit to explicitly start channel when using a software initiated DMA service (Automatically cleared by hardware)
ACTIVE	Status bit indicating the channel is currently in execution
DONE	Status bit indicating major loop completion (Cleared by software when using a software initiated DMA service)
D_REQ	Control bit to disable DMA request at end of major loop completion when using a hardware initiated DMA service
BWC	Control bits for “throttling” bandwidth control of a channel
E_SG	Control bit to enable scatter-gather feature
INT_HALF	Control bit to enable interrupt when major loop is half complete
INT_MAJ	Control bit to enable interrupt when major loop completes

[Table 16-32](#) 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 16-32. 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 16-33 lists the memory array terms and how the TCD settings interrelate.

Table 16-33. 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)			

16.8.2 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 channel priority error (EDMA_ES[CPE]).

For all error types other than channel priority error, 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 will be detected and recorded again.

If priority levels are not unique, when any channel requests service, a channel priority error will be reported. The highest channel priority that has an active request will be selected, but the lowest numbered channel with that priority will be selected by arbitration and executed by the eDMA engine. The hardware service request handshake signals, error interrupts, and error reporting will be associated with the selected channel.

16.8.3 DMA Arbitration Mode Considerations

16.8.3.1 Fixed Channel Arbitration

In this mode, the channel service request from the highest priority channel will be selected to execute.

16.8.3.2 Round Robin Channel Arbitration

Channels are serviced starting with the highest channel number and rotating through to the lowest channel number without regard to the channel priority levels

16.8.4 DMA Transfer

16.8.4.1 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 will begin after the channel service request is acknowledged and the channel is selected to execute. Once the transfer is complete, the $TCDn_CSR[DONE]$ bit will be set and an interrupt will be generated if properly enabled.

For example, the following TCD entry is configured to transfer 16 bytes of data. The eDMA is programmed for one iteration of the major loop transferring 16 bytes per iteration. The source memory has a byte wide memory port located at 0x1000. The destination memory has a longword-wide port located at 0x2000. The address offsets are programmed in increments to match the size of the transfer; one byte for the source and four bytes for the destination. The final source and destination addresses are adjusted to return to their beginning values.

Example 16-1. Single Request DMA Transfer

```

TCDn_CITER = TCDn_BITER = 1
TCDn_NBYTES = 16
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 would generate the following sequence of events:

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.

16.8.4.2 Multiple Requests

The next example is the same as previous with the exception of transferring 32 bytes via two hardware requests. The only fields that change are the major loop iteration count and the final address offsets. The eDMA is programmed for two iterations of the major loop transferring 16 bytes per iteration. After the channel's hardware requests are enabled in the $EDMA_ERQ$, channel service requests are initiated by the slave device.

```

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.

16.8.4.3 Modulo Feature

The modulo feature of the eDMA provides the ability to easily implement a circular data queue in which the size of the queue is a power of 2. MOD is a 5-bit field for both the source and destination in the TCD, and it specifies which lower address bits are allowed to increment from their original value after the

address+offset calculation. All upper address bits remain the same as in the original value. A setting of 0 for this field disables the modulo feature.

Table 16-34 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 16-34. Modulo Feature Example

Transfer Number	Address
1	0x12345670
2	0x12345674
3	0x12345678
4	0x1234567C
5	0x12345670
6	0x12345674

16.8.5 eDMA TCD n Status Monitoring

16.8.5.1 Minor Loop Complete

There are two methods to test for minor loop completion when using software initiated service requests. The first method is to read the TCD n _CITER field and test for a change. Another method may be extracted from the sequence shown below. The second method is to test the TCD n _CSR[START] bit and the TCD n _CSR[ACTIVE] bit. The minor loop complete condition is indicated by both bits reading zero after the TCD n _CSR[START] was set. Polling the TCD n _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 TCD n _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:

	TCDn_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 was activated.

16.8.5.2 Active Channel TCDn Reads

The eDMA will read back the true TCDn_SADDR, TCDn_DADDR, and TCDn_NBYTES values if read while a channel is executing. The true values of the SADDR, DADDR, and NBYTES are the values the eDMA engine is currently using in its internal register file and not the values in the TCD local memory for that channel. The addresses (SADDR and DADDR) and NBYTES (decrements to zero as the transfer progresses) can give an indication of the progress of the transfer. All other values are read back from the TCD local memory.

16.8.5.3 Preemption Status

Preemption is only available when fixed arbitration is selected as the channel arbitration mode. A preemptive situation is one in which a preempt-enabled channel is running and a higher priority request becomes active. When the eDMA engine is not operating in fixed channel arbitration mode, the determination of the relative priority of the actively running and the outstanding requests become undefined. Channel priorities are treated as equal (or more exactly, constantly rotating) when round-robin arbitration mode is selected.

The TCDn_CSR[ACTIVE] bit for the preempted channel remains asserted throughout the preemption. The preempted channel is temporarily suspended while the preempting channel executes one iteration of the major loop. If two TCDn_CSR[ACTIVE] bits are set at the same time in the global TCD map, a higher priority channel is actively preempting a lower priority channel.

16.8.6 Channel Linking

Channel linking (or chaining) is a mechanism where one channel sets the TCDn_CSR[START] bit of another channel (or itself), thus initiating a service request for that channel. This operation is automatically performed by the eDMA engine at the conclusion of the major or minor loop when properly enabled.

The minor loop channel linking occurs at the completion of the minor loop (or one iteration of the major loop). The TCDn_CITER[E_LINK] field is used to determine 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:

```
TCDn_CITER[E_LINK] = 1
TCDn_CITER[LINKCH] = 0xC
TCDn_CITER[CITER] value = 0x4
TCDn_CSR[MAJOR_E_LINK] = 1
TCDn_CSR[MAJOR_LINKCH] = 0x7
```

will execute 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 ($TCDn_CITER[E_LINK] = 1$), the $TCDn_CITER[CITER]$ field uses a nine bit vector to form the current iteration count. When minor loop linking is disabled ($TCDn_CITER[E_LINK] = 0$), the $TCDn_CITER[CITER]$ field uses a 15-bit vector to form the current iteration count. The bits associated with the $TCDn_CITER[LINKCH]$ field are concatenated onto the CITER value to increase the range of the CITER.

NOTE

The $TCDn_CITER[E_LINK]$ bit and the $TCDn_BITER[E_LINK]$ bit must equal or a configuration error will be reported. The CITER and BITER vector widths must be equal in order to calculate the major loop, half-way done interrupt point.

Table 16-35 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 16-35. 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

16.8.7 Dynamic Programming

This section provides recommended methods to change the programming model during channel execution.

16.8.7.1 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, thus allowing software to enable either feature during channel execution.

Because software is allowed to change the configuration during execution, a coherency sequence must be followed. Consider the scenario where the user attempts to execute a dynamic channel link by enabling the TCD n _CSR[MAJOR_E_LINK] bit at the same time the eDMA engine is retiring 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 TCD n once the TCD n _CSR[DONE] bit for that channel is set, indicating the major loop is complete.

NOTE

Software must clear the TCD n _CSR[DONE] bit before writing the TCD n _CSR[MAJOR_E_LINK] or TCD n _CSR[E_SG] bits. The TCD n _CSR[DONE] bit is cleared automatically by the eDMA engine once a channel begins execution.

Chapter 17

FlexBus

17.1 Introduction

This chapter describes data transfer operations and error conditions. It describes transfers initiated by the ColdFire processor and includes detailed timing diagrams showing the interaction of signals in supported bus operations.

NOTE

Unless otherwise noted, in this chapter ‘clock’ refers to the FB_CLK used for the bus ($f_{sys/2}$).

The external data bus is shared between the FlexBus module and the SDRAM controller. When the SDRAM controller is in SDR mode (DRAMSEL = 1), the data bus is switched dynamically between the SDRAM controller and the FlexBus module. However, when the SDRAM controller is in DDR mode (DRAMSEL = 0), D[31:16] is dedicated to the SDRAM data bus and D[15:0] is dedicated to the FlexBus data bus. In this case, external pins D[15:0], are mapped internally to the upper two bytes of the FlexBus data bus, FB_D[31:16]. This chapter will only use D[31:0] or D[31:X] to designate the data bus, but the actual pins used will depend on the DRAMSEL setting. Take this into consideration throughout this chapter.

17.1.1 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 with a maximum bus frequency of up to 83.33 MHz. It can be directly connected to asynchronous or synchronous devices, such as external boot ROMs, Flash memories, gate-array logic, or other simple target (slave) devices with little or no additional circuitry. 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 pin configuration, see [Table 13-1](#) for more details.) Chip-select $\overline{FB_CS0}$ can be dedicated to boot ROM access and can be programmed to be byte (8 bits), word (16 bits), or longword (32 bits) wide. Control signal timing is compatible with common ROM / Flash memories.

17.1.2 Features

The following list summarizes the key FlexBus features:

- Six independent, user-programmable chip-select signals ($\overline{\text{FB_CS}}[5:0]$) that can interface with SRAM, PROM, EPROM, EEPROM, Flash, and other peripherals
- 8-, 16-, and 32-bit port sizes
- Byte-, word-, longword-, and 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

17.2 External Signals

This section describes the external signals that are involved in data transfer operations. [Table 17-1](#) summarizes the FlexBus signals.

Table 17-1. FlexBus Signal Summary

Signal Name	Direction	Description	Reset State
A[23:0]	O	Address bus	Hi-Z
D[31:0]	I/O	Data bus	Hi-Z
$\overline{\text{FB_CS}}[5:0]$	O	General purpose chip-selects	Hi-Z
BE/BWE[3:0]	O	Byte Selects	Hi-Z
$\overline{\text{OE}}$	O	Output Enable	Hi-Z
R/W	O	Read/Write. 1 = Read, 0 = Write	Hi-Z
$\overline{\text{TS}}$	O	Transfer Start	Hi-Z
$\overline{\text{TA}}$	I	Transfer Acknowledge	—

17.2.1 Address & Data Buses (A[23:0], D[31:0])

The A[23:0] and D[31:0] buses carry the address and data respectively. The number of byte lanes used to carry the data is determined by the port size associated with the matching chip select.

17.2.2 Chip-Selects ($\overline{\text{FB_CS}}[5:0]$)

The chip-select signal indicates which device is being 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 (see [Section 17.3.2, “Memory Map/Register Definition”](#)).

17.2.3 Byte Enables/Byte Write Enables (BE/BWE[3:0])

When driven low, the byte enable ($\overline{\text{BE/BWE}}[3:0]$) outputs indicate that data is to be latched or driven onto a byte of the data bus. BE/BWE_n signals are asserted only to the memory bytes used during read or write accesses. A configuration option is provided to assert these signals on both reads and writes (byte enable) or writes only (byte write enable).

The $\overline{BE/BWE}[3:0]$ signals are asserted during accesses to on-chip peripherals but not to on-chip SRAM or cache. For SRAM or Flash devices, the $\overline{BE/BWE}[3:0]$ outputs should be connected to individual byte strobe signals.

17.2.4 Output Enable (\overline{OE})

The output enable signal (\overline{OE}) is sent to the interfacing memory and/or peripheral to enable a read transfer. \overline{OE} is asserted only when a chip select matches the current address decode.

17.2.5 Read/Write (R/\overline{W})

The ColdFire device drives the R/\overline{W} signal to indicate the direction of the current bus operation. It is driven high during read bus cycles and low during write bus cycles.

17.2.6 Transfer Start (\overline{TS})

The assertion of \overline{TS} indicates that the device has begun a bus transaction and the address and attributes are valid. \overline{TS} is asserted for one bus clock cycle.

17.2.7 Transfer Acknowledge (\overline{TA})

This signal indicates that the 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.

If auto-acknowledge is disabled, the external device drives \overline{TA} to terminate the bus transfer; if auto-acknowledge is enabled, the \overline{TA} is generated internally after a specified number of wait states, or the external device may assert external \overline{TA} before the wait-state countdown, terminating the cycle early. The device negates $\overline{FB_CSn}$ a cycle after the last \overline{TA} asserts. During read cycles, the peripheral must continue to drive data until \overline{TA} is recognized. For write cycles, the processor continues to drive data one clock after $\overline{FB_CSn}$ is negated.

The number of wait states is determined by either internally programmed auto acknowledgement or the external \overline{TA} input. If the external \overline{TA} is used, the peripheral has total control on the number of wait states.

NOTE

External devices should only assert \overline{TA} while the $\overline{FB_CSn}$ signal to the external device is asserted.

17.3 Chip-Select Operation

Each chip-select has a dedicated set of the following registers for configuration and control:

- Chip-select address registers (CSARn) control the base address space of the chip-select. See [Section 17.3.2.1, “Chip-Select Address Registers \(CSAR0–CSAR5\).”](#)
- Chip-select mask registers (CSMRn) provide 16-bit address masking and access control. See [Section 17.3.2.2, “Chip-Select Mask Registers \(CSMR0–CSMR5\).”](#)

- Chip-select control registers (CSCRn) provide port size and burst capability indication, wait-state generation, address setup and hold times, and automatic acknowledge generation features. See [Section 17.3.2.3, “Chip-Select Control Registers \(CSCR0–CSCR5\).”](#)

$\overline{\text{FB_CS}0}$ is a global chip-select after reset and provides re-locatable boot ROM capability.

17.3.1 General Chip-Select Operation

When a bus cycle is initiated, the device first compares its address with the base address and mask configurations programmed for chip-selects 0–5 (configured in CSCR0–CSCR5). If the driven address matches a programmed chip-select, the appropriate chip-select is asserted fulfilling the requirements as programmed in the respective configuration register.

If the address does not match a CSAR register, the internal bus cycle is terminated and no chip select is asserted.

Should an address match multiple CSARs, the matching chip select signals are driven; however, the chip select signals are driven during an external burst-inhibited bus cycle with external termination on a 32-bit port.

17.3.1.1 8-, 16-, and 32-Bit Port Sizing

Static bus sizing is programmable through the port size bits, CSCR[PS]. (See [Section 17.3.2.3, “Chip-Select Control Registers \(CSCR0–CSCR5\).”](#)) Note that the processor always drives a 24-bit address on the A bus regardless of the external device’s address size. The external device must connect its address lines to the appropriate A bits starting from A0 and upward. It must also connect its data lines to the D bus starting from D31 and downward. No bit ordering is required when connecting address and data lines to the A and D buses. For example, a 16-bit address/16-bit data device would connect its addr[15:0] to A[15:0] and data[15:0] to D[31:16]. See [Figure 17-4](#) for graphical connection.

17.3.1.2 Global Chip-Select Operation

$\overline{\text{FB_CS}0}$, the global (boot) chip-select, allows address decoding for boot ROM before system initialization. Its operation differs from other external chip-select outputs after system reset.

After system reset, $\overline{\text{FB_CS}0}$ is asserted for every external access. No other chip-select can be used until the valid bit CSMR0[V] is set, at which point $\overline{\text{FB_CS}0}$ functions as configured. After this, $\overline{\text{FB_CS}}[5:1]$ can be used as well. At reset, the port size of the global chip-select is determined by the logic levels on the D[4:3] signals, [Table 17-2, “Chip Configuration Module \(CCM\),”](#) for more information.

Table 17-2. D[4:3], Port Size of Boot $\overline{\text{FB_CS}0}$

D[4:3]	Boot $\overline{\text{FB_CS}0}$ Port Size at Reset
00	32-bit port
01	8-bit port
10	16-bit port
11	16-bit port

17.3.2 Memory Map/Register Definition

The following tables describe in detail the registers and bit meanings for configuring chip-select operation. [Table 17-3](#) shows the chip-select register memory map. Reading unused or reserved locations terminates normally and returns zeros.

Table 17-3. Chip Select Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/ Page
0xFC00_8000 + n^*0xC	Chip-Select Address Register (CSAR n) $n = 0\text{--}5$	32	R/W	0x0000_0000	17.3.2.1/17-5
0xFC00_8004 + n^*0xC	Chip-Select Mask Register (CSMR n) $n = 0\text{--}5$	32	R/W	0x0000_0000	17.3.2.2/17-6
0xFC00_8008 + n^*0xC	Chip-Select Control Register (CSCR n) $n = 0\text{--}5$	32	R/W	See Section	17.3.2.3/17-6

17.3.2.1 Chip-Select Address Registers (CSAR0–CSAR5)

The CSAR n registers specify the chip-select base addresses.

NOTE

Since the FlexBus module is one of the slaves connected to the cross-bar 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`. Be sure to set the CSAR n registers appropriately.

Address: 0xFC00_8000 (CSAR0)
0xFC00_800C (CSAR1)
0xFC00_8018 (CSAR2)
0xFC00_8024 (CSAR3)
0xFC00_8030 (CSAR4)
0xFC00_803C (CSAR5)

Access: User read/write

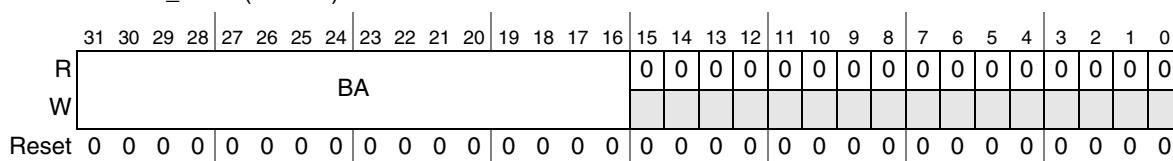


Figure 17-1. Chip-Select Address Registers (CSAR n)

Table 17-4. CSAR n 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, should be cleared.

17.3.2.2 Chip-Select Mask Registers (CSMR0–CSMR5)

CSMR_n registers are used to specify the address mask and allowable access types for the respective chip-selects.

Figure 17-2. Chip-Select Mask Registers (CSMR n)

Table 17-5. 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 $\overline{FB_CS}_n$ is 2^n; $n = (\text{number of bits set in respective CSMR[BAM]}) + 16$. For example, if CSAR0 = 0x0000 and CSMR0[BAM] = 0x0008, $\overline{FB_CS}_0$ would address two discontinuous 64-Kbyte memory blocks: one from 0x0_0000–0x0_FFFF and one from 0x8_0000–0x8_FFFF.</p>
	<p>Likewise, for $\overline{FB_CS}_0$ to access 32 Mbytes of address space starting at location 0x0, $\overline{FB_CS}_1$ must begin at the next byte after $\overline{FB_CS}_0$ for a 16-Mbyte address space. Then CSAR0 = 0x0000, CSMR0[BAM] = 0x01FF, CSAR1 = 0x0200, and CSMR1[BAM] = 0x00FF.</p>
15–9	Reserved, should 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 $\text{CSAR}_n[\text{WP}] = 1$ results in the appropriate chip-select not being selected, as well as a bus error sent to the interrupt controller.</p> <p>0 Both read and write accesses are allowed 1 Only read accesses are allowed</p>
7–1	Reserved, should 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 $\overline{FB_CS}_0$, which acts as the global chip-select). Reset clears each $\text{CSMR}_n[\text{V}]$. At reset, no chip-select other than $\overline{FB_CS}_0$ can be used until the CSMR0[V] is set. At which point $\overline{FB_CS}[5:0]$ functions as configured.</p> <p>0 Chip-select invalid 1 Chip-select valid</p>

17.3.2.3 Chip-Select Control Registers (CSCR0–CSCR5)

Each CSCR n , Figure 17-3, controls the auto acknowledge, address setup and hold times, port size, burst capability, and activation of each chip-select. Note that to support the global chip-select, FB CS0, the

CSCR0 reset values differ from the other CSCRs. $\overline{\text{FB_CS0}}$ allows address decoding for boot ROM before system initialization.

Address:	0xFC00_8008 (CSCR0)	Access:	User
	0xFC00_8014 (CSCR1)		read/write
	0xFC00_8020 (CSCR2)		
	0xFC00_802C (CSCR3)		
	0xFC00_8038 (CSCR4)		
	0xFC00_8044 (CSCR5)		
R	31 30 29 28 27 26	25 24 23 22	21 20 19 18 17 16
W	SWS	0 0 SWSEN 0	ASET RDAH WRAH
Reset: CSCR0	0 0 0 0 0 0	0 0 0 0 1 1	1 1 1 1
Reset: CSCRs	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	WS	SBM AA PS BEM BSTR	BSTW 0 0 0
Reset: CSCR0	1 1 1 1 1 1 [DRAMSEL]	0 D4 D3	0 0 0 0
Reset: CSCRs	0 0 0 0 0 0 [DRAMSEL]	0 0 0 0	0 0 0 0

Figure 17-3. Chip-Select Control Registers (CSCR_n)

Table 17-6. 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 only used if the SWSEN bit is set, otherwise the WS value is used for all burst transfers.
25–24	Reserved, should be cleared
23 SWSEN	Secondary wait state enable. 0 The WS value is used to insert wait states before an internal transfer acknowledge is generated for all transfers. 1 The SWS value is used to insert wait states before an internal transfer acknowledge is generated for burst transfer secondary terminations.
22	Reserved, should be cleared
21–20 ASET	Address setup. This field controls the assertion of the chip-select with respect to assertion of a valid address and attributes. Note that the address and attributes are considered valid at the same time <u>TS</u> asserts. 00 Assert <u>FB_CSn</u> on first rising clock edge after address is asserted. (Default <u>FB_CSn</u>) 01 Assert <u>FB_CSn</u> on second rising clock edge after address is asserted. 10 Assert <u>FB_CSn</u> on third rising clock edge after address is asserted. 11 Assert <u>FB_CSn</u> on fourth rising clock edge after address is asserted.(Reset <u>FB_CS0</u>)
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. The hold time only applies at the end of a transfer. Therefore, a burst transfer only has a hold time added after the last bus cycle. 00 Hold address and attributes one cycle after <u>FB_CSn</u> negates on reads. (Default <u>FB_CSn</u>) 01 Hold address and attributes two cycles after <u>FB_CSn</u> negates on reads. 10 Hold address and attributes three cycles after <u>FB_CSn</u> negates on reads. 11 Hold address and attributes four cycles after <u>FB_CSn</u> negates on reads. (Reset <u>FB_CS0</u>)

Table 17-6. CSCR_n Field Descriptions (continued)

Field	Description
17–16 WRAH	<p>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. The hold time only applies at the end of a transfer. Therefore, a burst transfer only has a hold time added after the last bus cycle.</p> <p>00 Hold address and attributes one cycle after $\overline{\text{FB_CS}n}$ negates on writes. (Default $\overline{\text{FB_CS}n}$)</p> <p>01 Hold address and attributes two cycles after $\overline{\text{FB_CS}n}$ negates on writes.</p> <p>10 Hold address and attributes three cycles after $\overline{\text{FB_CS}n}$ negates on writes.</p> <p>11 Hold address and attributes four cycles after $\overline{\text{FB_CS}n}$ negates on writes. (Reset $\overline{\text{FB_CS}0}$)</p>
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 = 0, $\overline{\text{TA}}$ must be asserted by the external system regardless of the number of wait states generated. In that case, the external transfer acknowledge ends the cycle. An external $\overline{\text{TA}}$ supersedes the generation of an internal $\overline{\text{TA}}$.
9 SBM	<p>Split bus mode. For proper operation of the chip select signals, this bit must be set when the SDRAM controller is in DDR mode (DRAMSEL signal is negated). The reset value of SBM is the opposite of the DRAMSEL signal.</p> <p>0 Device is not in split bus mode (SDRAM controller is in SDR mode, DRAMSEL = 1).</p> <p>1 Device is in split bus mode (SDRAM controller is in DDR mode, DRAMSEL = 0).</p> <p>Note: Placing the device in split bus mode is only controlled by the DRAMSEL signal. This bit is only used to provide correct operation of the chip select signals.</p>
8 AA	<p>Auto-acknowledge enable. Determines the assertion of the internal transfer acknowledge for accesses specified by the chip-select address.</p> <p>0 No internal $\overline{\text{TA}}$ is asserted. Cycle is terminated externally.</p> <p>1 Internal $\overline{\text{TA}}$ is asserted as specified by WS. Note that if AA = 1 for a corresponding $\overline{\text{FB_CS}n}$ and the external system asserts an external $\overline{\text{TA}}$ before the wait-state countdown asserts the internal $\overline{\text{TA}}$, the cycle is terminated. Burst cycles increment the address bus between each internal termination.</p>
7–6 PS	<p>Port size. Specifies the width of the data port associated with each chip-select. It determines where data is driven during write cycles and where data is sampled during read cycles.</p> <p>00 32-bit port size. Valid data sampled and driven on D[31:0]</p> <p>01 8-bit port size. Valid data sampled and driven on D[31:24]</p> <p>1x 16-bit port size. Valid data sampled and driven on D[31:16]</p>
5 BEM	<p>Byte enable mode. Specifies the byte enable operation. Certain SRAMs have byte enables that must be asserted during reads as well as writes. BEM can be set in the relevant CSCR to provide the appropriate mode of byte enable support for these SRAMs.</p> <p>0 $\overline{\text{BE/BWE}}$ is not asserted for reads. $\overline{\text{BE/BWE}}$ is asserted for data write only.</p> <p>1 $\overline{\text{BE/BWE}}$ is asserted for read and write accesses.</p>
4 BSTR	Burst read enable. Specifies whether burst reads are used for memory associated with each $\overline{\text{FB_CS}n}$.
	<p>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.</p> <p>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.</p>
3 BSTW	Burst write enable. Specifies whether burst writes are used for memory associated with each $\overline{\text{FB_CS}n}$.
	<p>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.</p> <p>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.</p>
2–0	Reserved, should be cleared.

17.4 Functional Description

17.4.1 Data Transfer Operation

Data transfers between the chip and other devices involve the following signals:

- Address / data bus (A[23:0], D[31:0])
- Control signals (\overline{TS} and \overline{TA})
- $\overline{FB_CSn}$, \overline{OE} , $\overline{BE/BWE}[3:0]$
- Attribute signals (R/ \overline{W})

The address, write data, \overline{TS} , $\overline{FB_CSn}$, and all attribute signals change on the rising edge of the clock. Read data is latched into the device on the rising edge of the clock.

The FlexBus supports byte, word, longword, and line operand transfers and allows accesses to 8-, 16-, and 32-bit data ports. Transfer parameters such as 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 17.3.2.3, “Chip-Select Control Registers \(CSCR0–CSCR5\).](#)

17.4.2 Data Byte Alignment and Physical Connections

The device aligns data transfers in FlexBus byte lanes, with the number of lanes depending on the width of the data port. [Figure 17-4](#) shows the byte lanes that external memory should be connected to and the sequential transfers if a longword is transferred for three port sizes. For example, an 8-bit memory should be connected to the single lane D[31:24] (BE/BWE0). A longword transfer through this 8-bit port takes four transfers on D[31:24], starting with the MSB and going to the LSB. A longword transfer through a 32-bit port requires one transfer on each of the four byte lanes of the FlexBus.

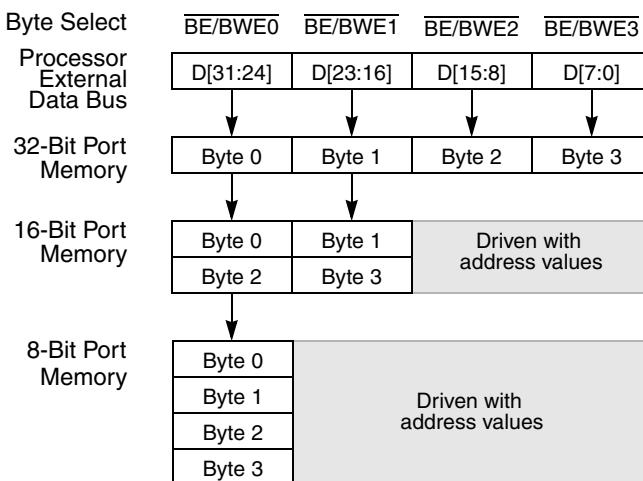


Figure 17-4. Connections for External Memory Port Sizes

17.4.3 Bus Cycle Execution

As shown in [Figure 17-7](#) and [Figure 17-9](#), basic bus operations occur in four clocks, as follows:

1. At the first clock edge, the address, attributes, and \overline{TS} are driven.
2. $\overline{FB_CSn}$ is asserted at the second rising clock edge to indicate which device has been selected and by that time the address and attributes are valid and stable. \overline{TS} 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 returned at this cycle.
External slave asserts \overline{TA} at this clock edge.
3. Read data and \overline{TA} are sampled on the third clock edge. \overline{TA} can be negated after this edge and read data can then be tristated.
4. $\overline{FB_CSn}$ 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.

17.4.3.1 Data Transfer Cycle States

The data transfer operation in the device is controlled by an on-chip state machine. The state transition diagram for basic read and write cycles is shown in [Figure 17-5](#).

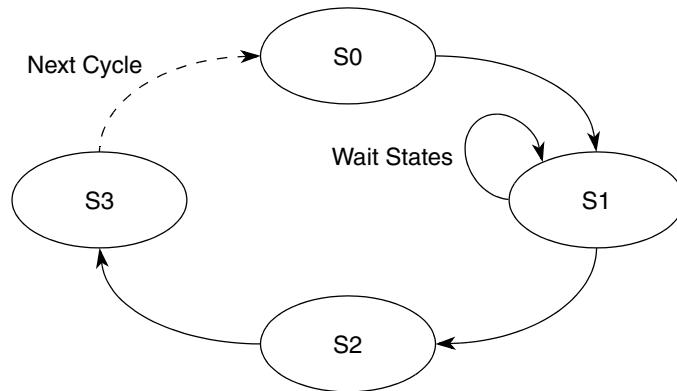


Figure 17-5. Data Transfer State Transition Diagram

[Table 17-7](#) describes the states as they appear in subsequent timing diagrams.

Table 17-7. 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 A[23:0], asserts \overline{TS} , and drives R/W high for a read and low for a write, if these signals are not already in the appropriate state.

Table 17-7. Bus Cycle States (continued)

State	Cycle	Description
S1	All	\overline{TS} is negated on the rising edge of FB_CLK, and $\overline{FB_CSn}$ is asserted. Data is driven on D[31:X] for writes, and D[31:X] is tristated for reads. Address continues to be driven on A[23:0] pins. If \overline{TA} is recognized asserted, then the cycle moves on to S2. If \overline{TA} is not asserted either internally or externally, then the S1 state continues to repeat.
	Read	Data is made available by the external device before the rising edge of FB_CLK with \overline{TA} asserted. The device will latch data on this rising clock edge.
S2	All	For internal termination, both the $\overline{FB_CSn}$ and internal \overline{TA} will be negated. For external termination, the external device should negate \overline{TA} , and $\overline{FB_CSn}$ select is negated after the rising edge of FB_CLK at the end of S2.
	Read	The external device can stop driving data after the rising edge of FB_CLK at the beginning of S2. However, data can be driven until the end of S3 or any additional address hold cycles.
S3	All	Address, data, and R/W go invalid off the rising edge of FB_CLK at the end of S3, terminating the read or write cycle.

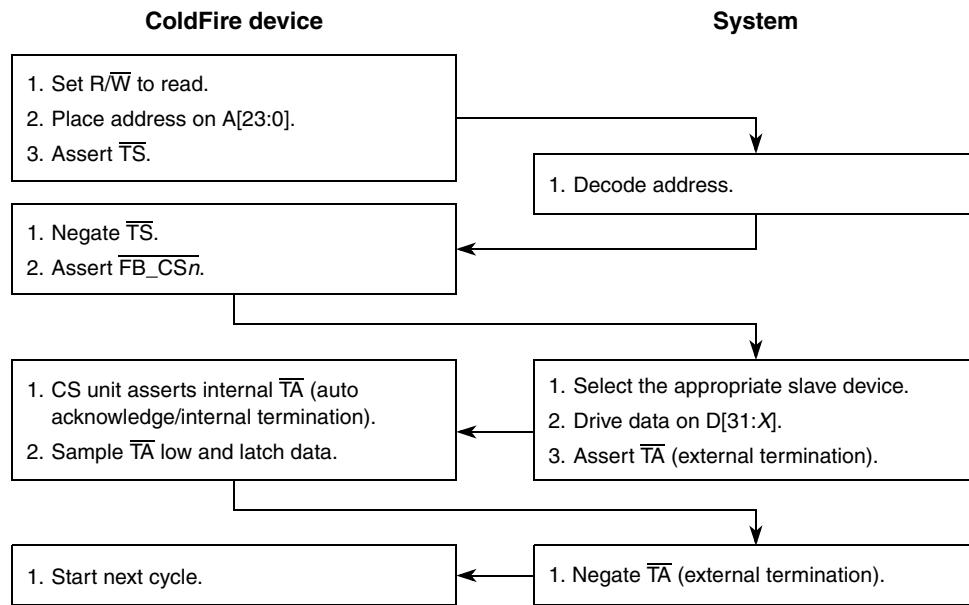
17.4.4 FlexBus Timing Examples

17.4.4.1 Basic Read Bus Cycle

During a read cycle, the ColdFire device receives data from memory or a peripheral device. [Figure 17-6](#) is a read cycle flowchart.

NOTE

Throughout this chapter D[31:X] is a data bus that can be 32-, 16-, or 8-bits wide.

**Figure 17-6. Read Cycle Flowchart**

The read cycle timing diagram is shown in [Figure 17-7](#).

NOTE

In the following timing diagrams, the dotted lines indicate $\overline{\text{TA}}$, $\overline{\text{OE}}$, and $\overline{\text{FB_CS}_n}$ timing when internal termination is used ($\text{CSCR}[\text{AA}] = 1$). The external and internal $\overline{\text{TA}}$ assert at the same time; however, 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. However, this should be ignored by the connected device.

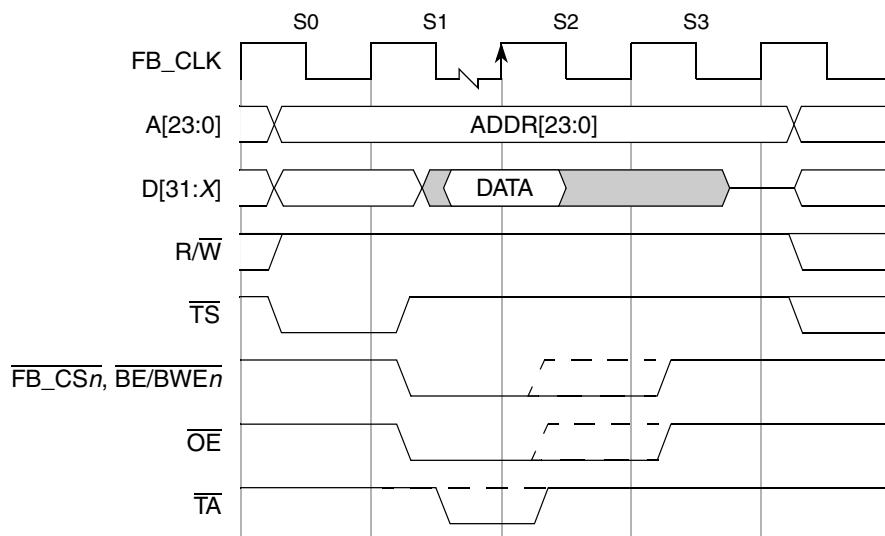
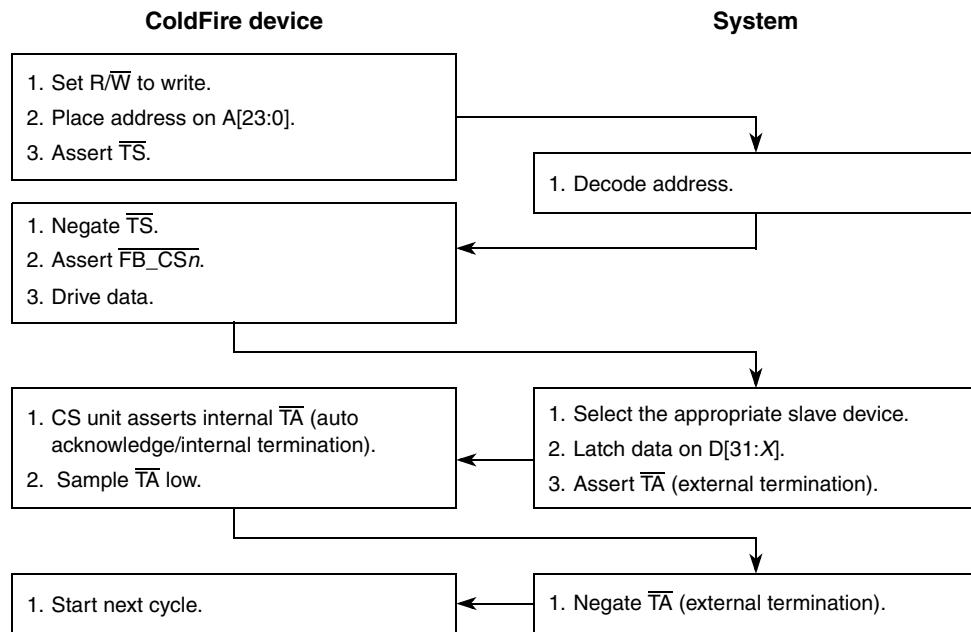


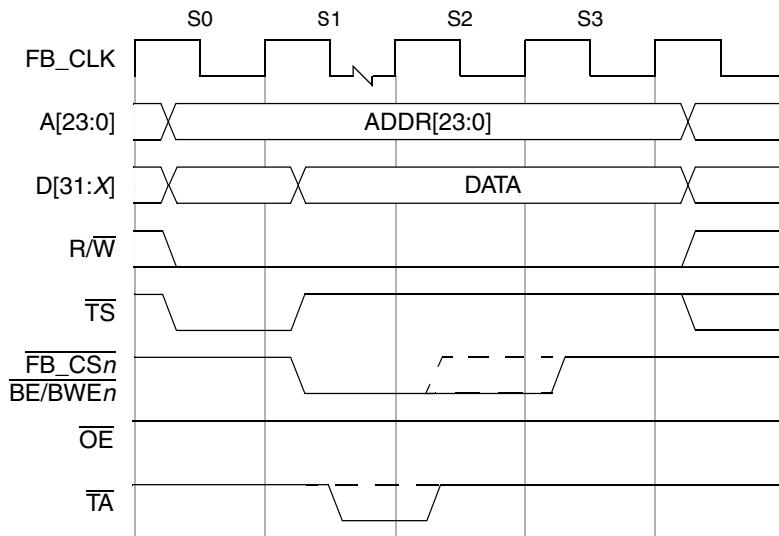
Figure 17-7. Basic Read Bus Cycle

17.4.4.2 Basic Write Bus Cycle

During a write cycle, the device sends data to memory or to a peripheral device. The write cycle flowchart is shown in [Figure 17-8](#).

**Figure 17-8. Write Cycle Flowchart**

The write cycle timing diagram is shown in [Figure 17-9](#).

**Figure 17-9. Basic Write Bus Cycle**

17.4.4.3 Bus Cycle Sizing

This section shows timing diagrams for various port size scenarios. [Figure 17-10](#) illustrates the basic byte read transfer to an 8-bit device with no wait states. The address is driven on the A[23:0] bus throughout the bus cycle. The external device returns the read data on D[31:24], and may tristate the data line or continue to drive the data one clock after $\overline{\text{TA}}$ is sampled asserted.

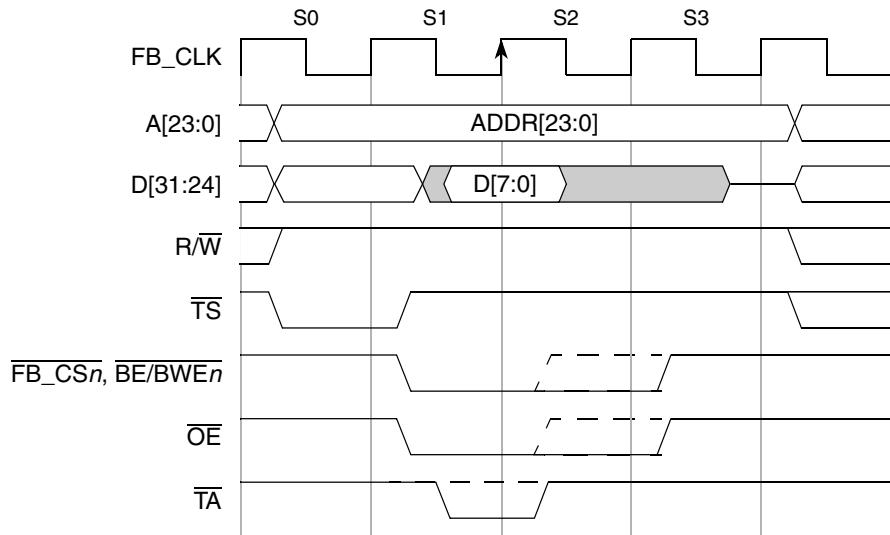


Figure 17-10. Single Byte Read Transfer

[Figure 17-11](#) shows the similar configuration for a write transfer. The data is driven from the second clock on D[31:24].

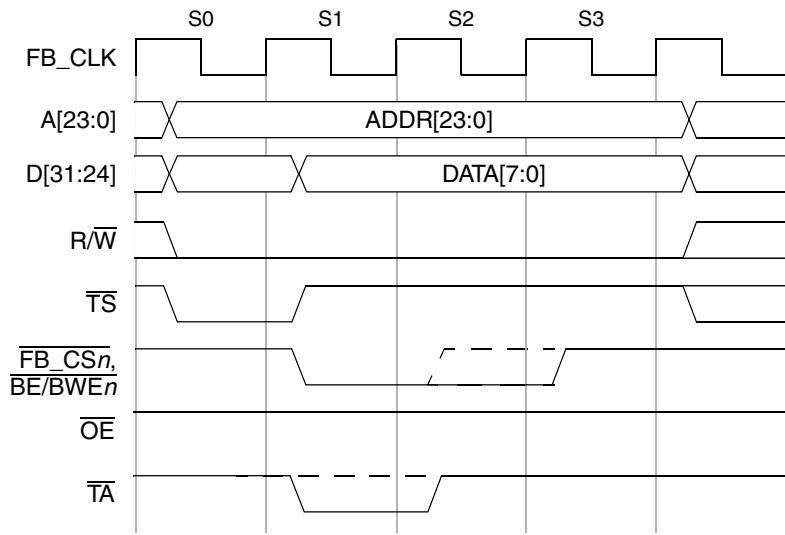


Figure 17-11. Single Byte Write Transfer

[Figure 17-12](#) illustrates the basic word read transfer to a 16-bit device with no wait states. The address is driven on the A[23:0] bus throughout the bus cycle. The external device returns the read data on D[31:16] and may tristate the data line or continue to drive the data one clock after $\overline{\text{TA}}$ is sampled asserted.

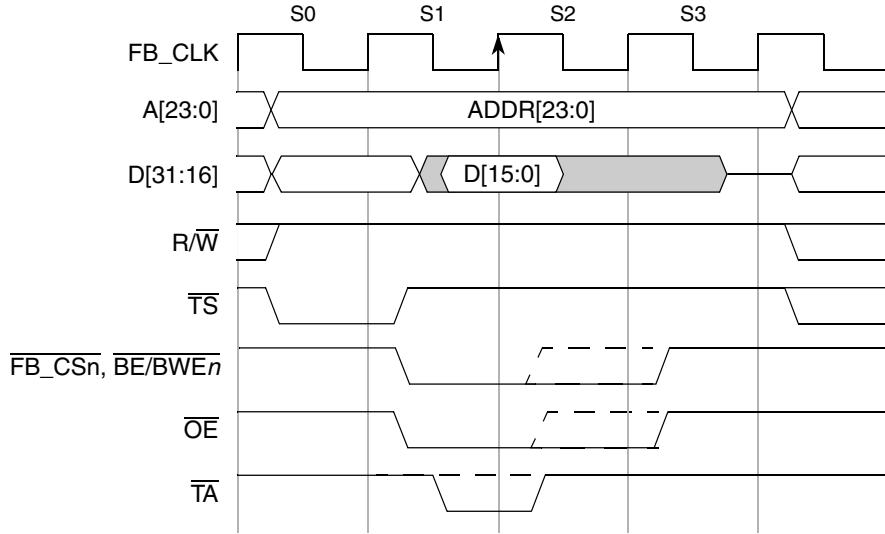


Figure 17-12. Single Word Read Transfer

Figure 17-13 shows the similar configuration for a write transfer. The data is driven from the second clock on D[31:16].

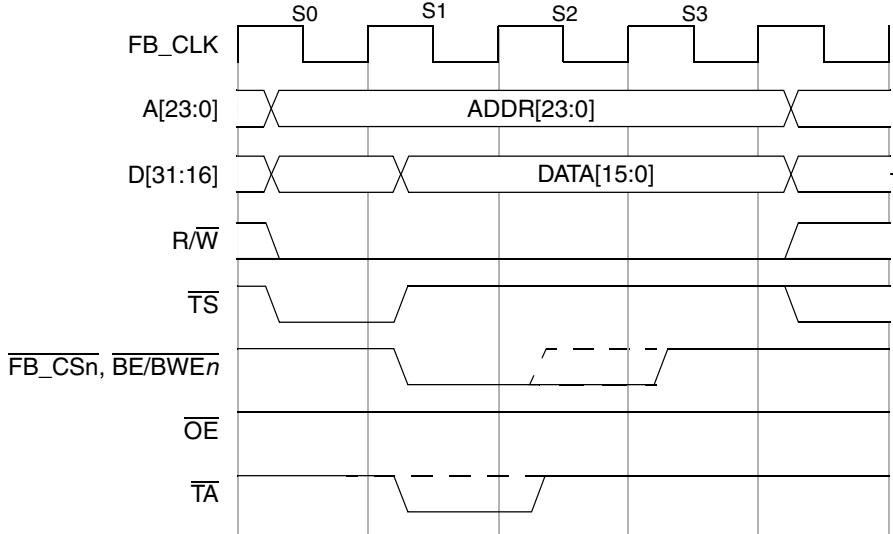


Figure 17-13. Single Word Write Transfer

Figure 17-14 depicts a longword read through a 32-bit device.

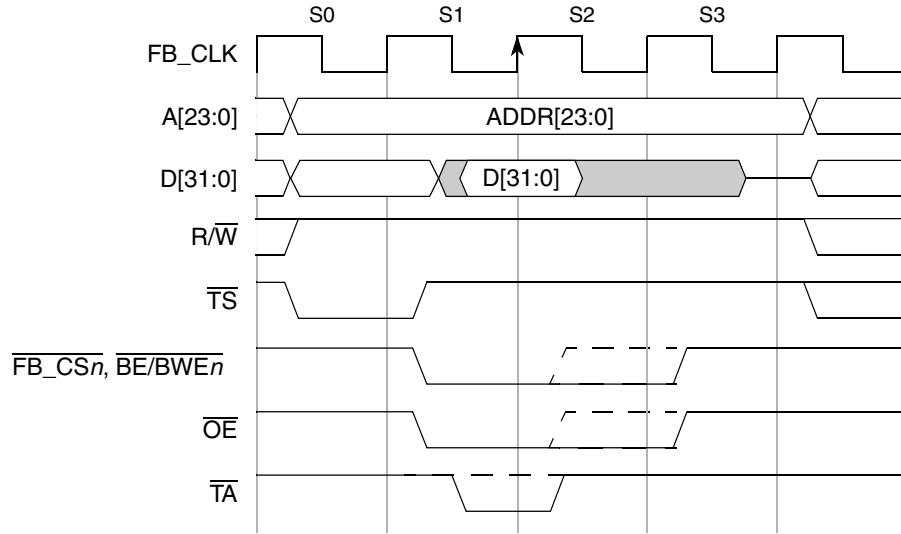
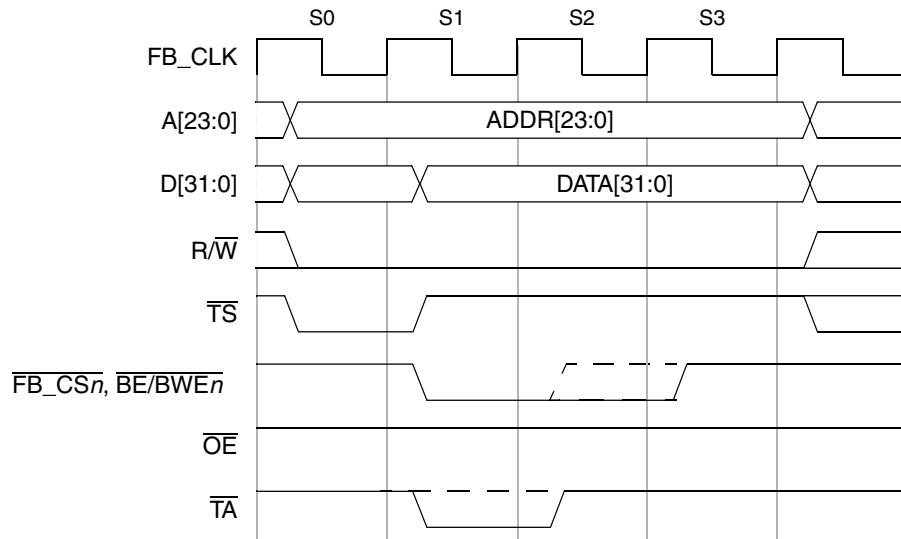
**Figure 17-14. Longword Read Transfer**

Figure 17-15 illustrates the longword write to a 32-bit device.

**Figure 17-15. Longword Write Transfer**

17.4.4.4 Timing Variations

The ColdFire device has several features that can be used to 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.

17.4.4.4.1 Wait States

Wait states can be inserted before each beat of a transfer by programming the $CSCR_n$ registers. Wait states can be used to give the peripheral or memory more time to return read data or sample write data.

Figure 17-16 and Figure 17-17 show the basic read and write bus cycles (also shown in Figure 17-7 and Figure 17-12). This is the default case with no wait states.

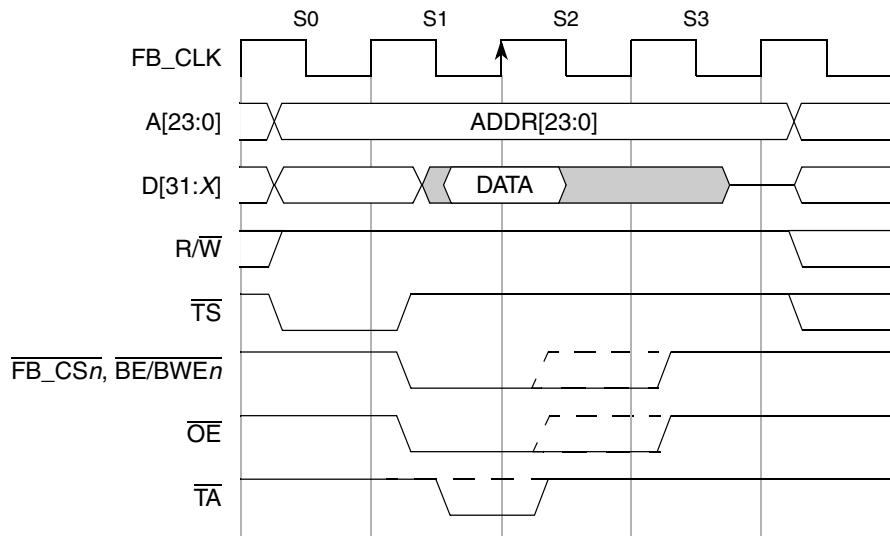


Figure 17-16. Basic Read Bus Cycle (No Wait States)

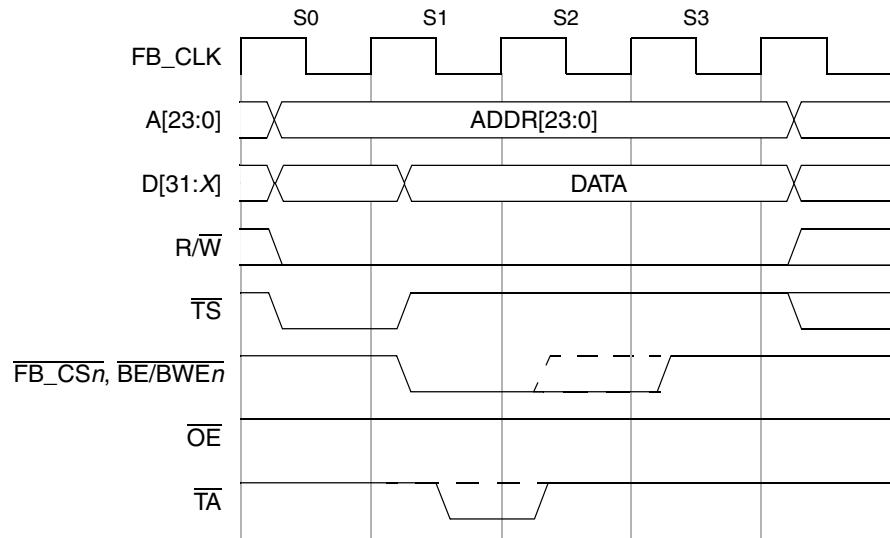


Figure 17-17. Basic Write Bus Cycle (No Wait States)

If wait states are used, then the S1 state will repeat continuously until either the internal **TA** is asserted by the chip select auto-acknowledge unit or the external **TA** is recognized as asserted. Figure 17-18 and Figure 17-19 show a read and write cycle with one wait state.

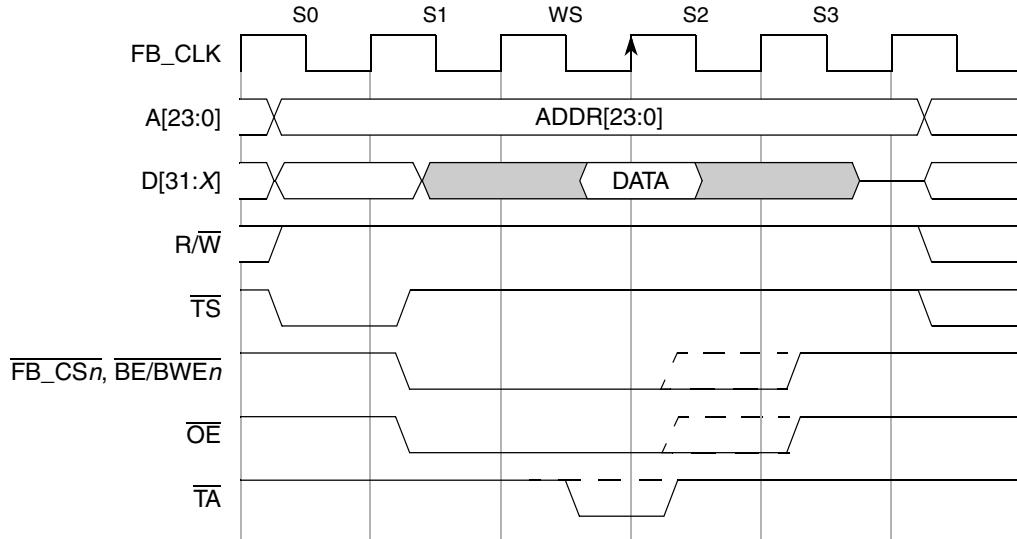


Figure 17-18. Read Bus Cycle (One Wait State)

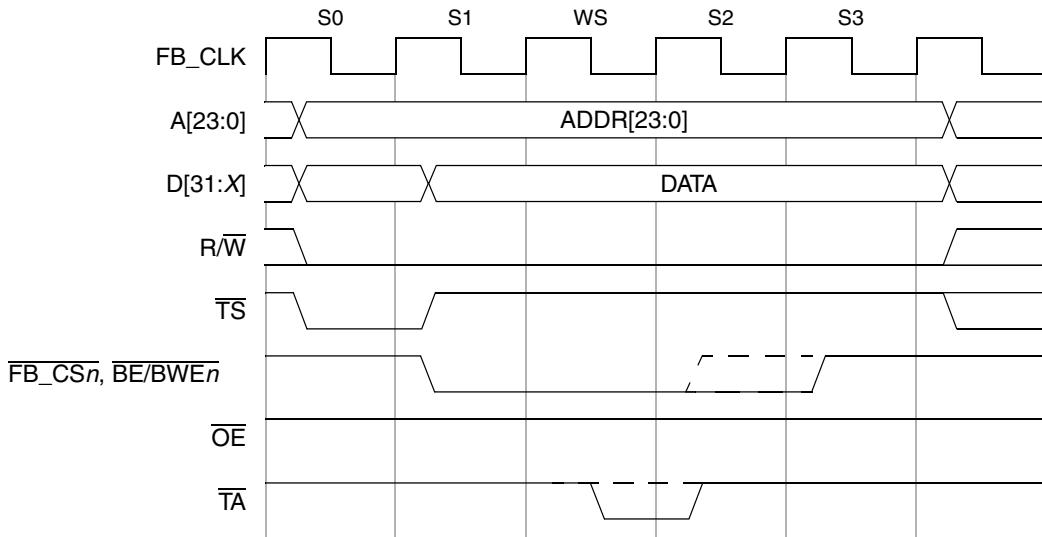


Figure 17-19. Write Bus Cycle (One Wait State)

17.4.4.4.2 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 transfer start (\overline{TS}) is asserted. [Figure 17-20](#) and [Figure 17-21](#) show read and write bus cycles with two clocks of address setup.

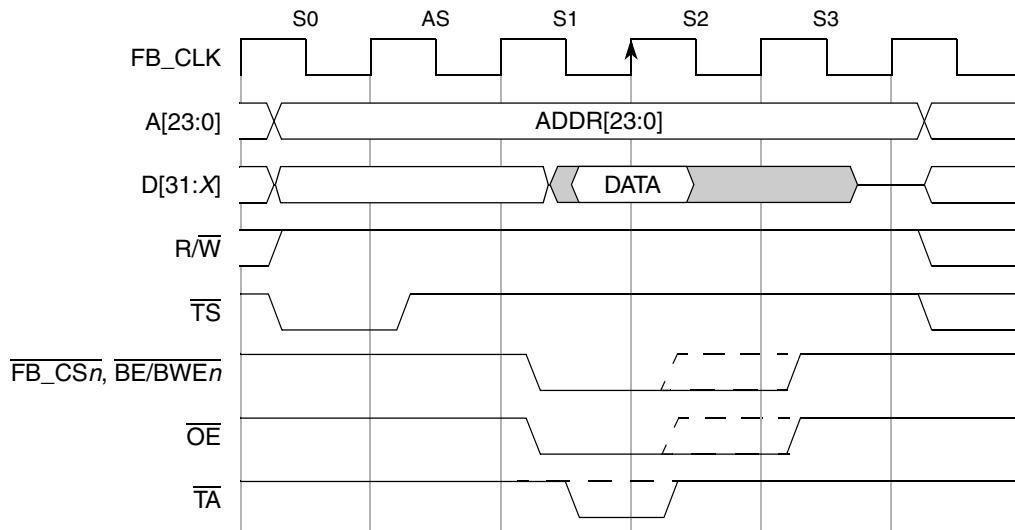


Figure 17-20. Read Bus Cycle with Two Clock Address Setup (No Wait States)

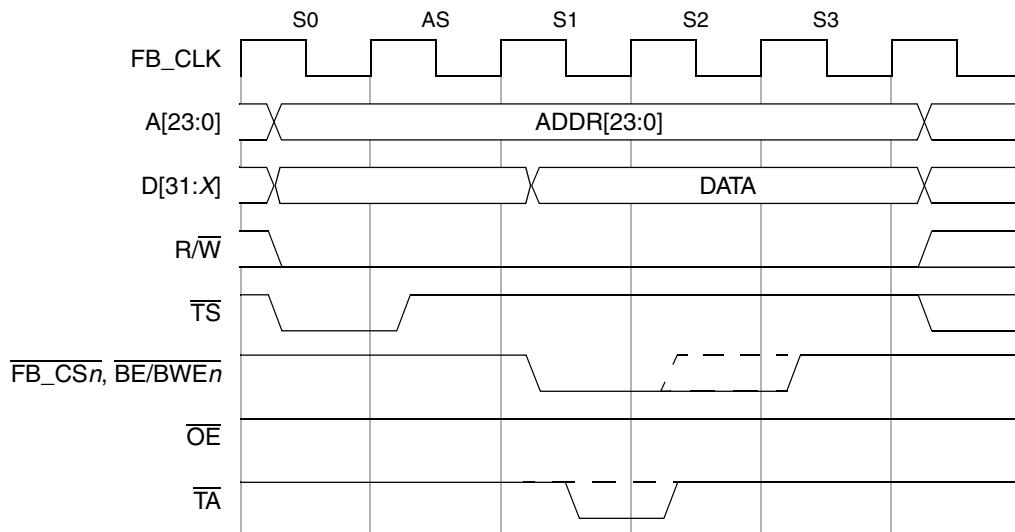


Figure 17-21. Write Bus Cycle with Two Clock Address Setup (No Wait States)

In addition to address setup, there is also a programmable address hold option for each chip select. Address and attributes can be held one to four clocks after chip select, byte selects, and output enable negate. Figure 17-22 and Figure 17-23 show read and write bus cycles with two clocks of address hold.

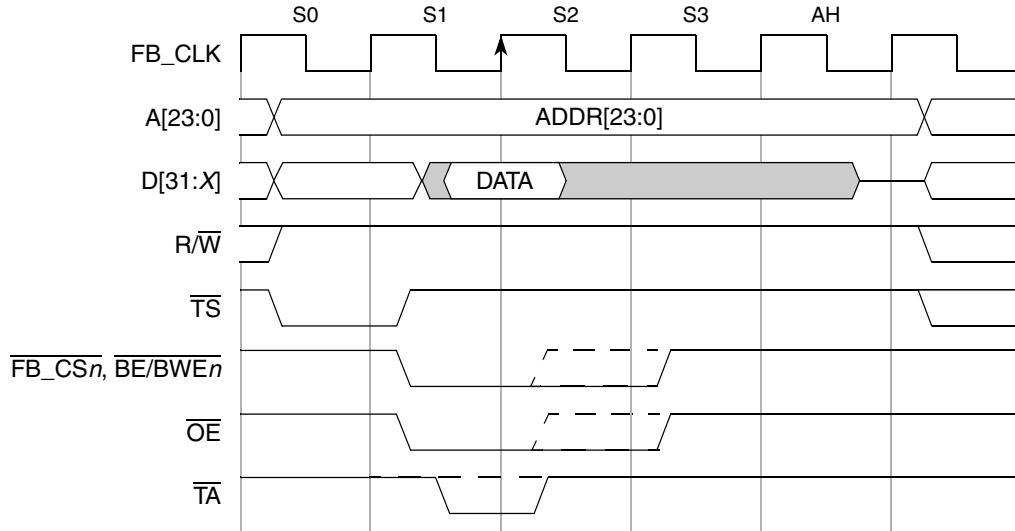


Figure 17-22. Read Cycle with Two Clock Address Hold (No Wait States)

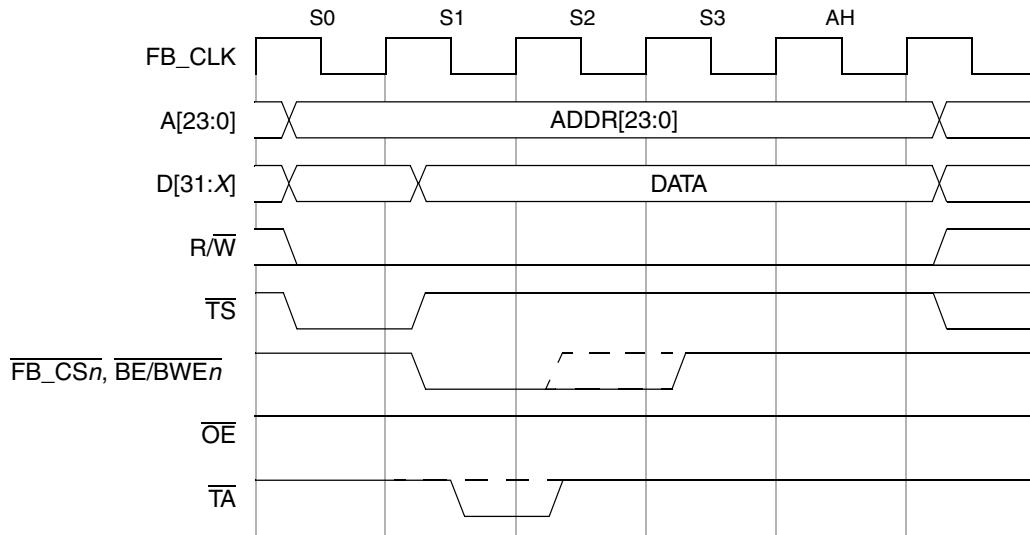


Figure 17-23. Write Cycle with Two Clock Address Hold (No Wait States)

Figure 17-24 shows a bus cycle that uses address setup, wait states, and address hold.

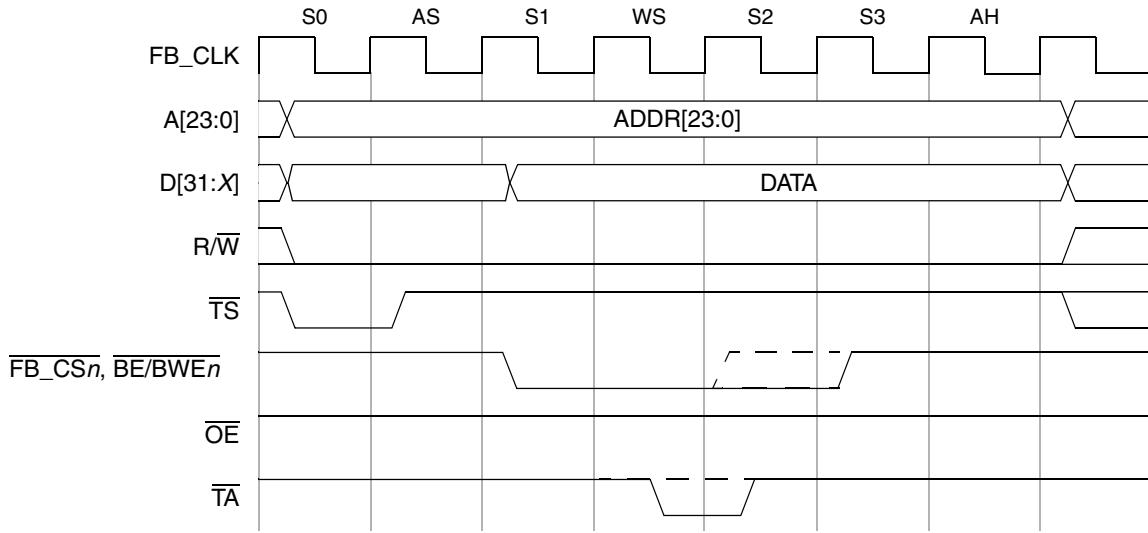


Figure 17-24. Write Cycle with Two Clock Address Setup and Two Clock Hold (One Wait State)

17.4.5 Burst Cycles

The device can be programmed to initiate burst cycles if its transfer size exceeds the size of the port it is transferring to. With bursting disabled, any transfer that is larger than the port size is broken into multiple individual transfers. With bursting enabled, an access that is larger than port size would result a burst cycle of multiple beats. [Table 17-8](#) shows the result of such transfer translations.

Table 17-8. Transfer Size and Port Size Translation

Port Size PS[1:0]	Transfer Size	Burst-inhibited: number of transfers Burst enabled: number of beats
01 (8-bit)	word	2
	longword	4
	line	16
1x (16-bit)	longword	2
	line	8
00 (32-bit)	line	4

The FlexBus can support 3-1-1-1 burst cycles and optimize DMA transfers. A user can add wait states by delaying termination of the cycle. If internal termination is used, different wait state counters can be used for the first access and the following beats.

The CSCR n registers are used to enable bursting for reads, writes, or both. Memory spaces can be declared burst-inhibited for reads and writes by clearing the appropriate CSCR n [BSTR,BSTW] bits.

[Figure 17-25](#) shows a longword read through an 8-bit device programmed for burst enable. The transfer results in a 4-beat burst and the data is driven on D[31:24].

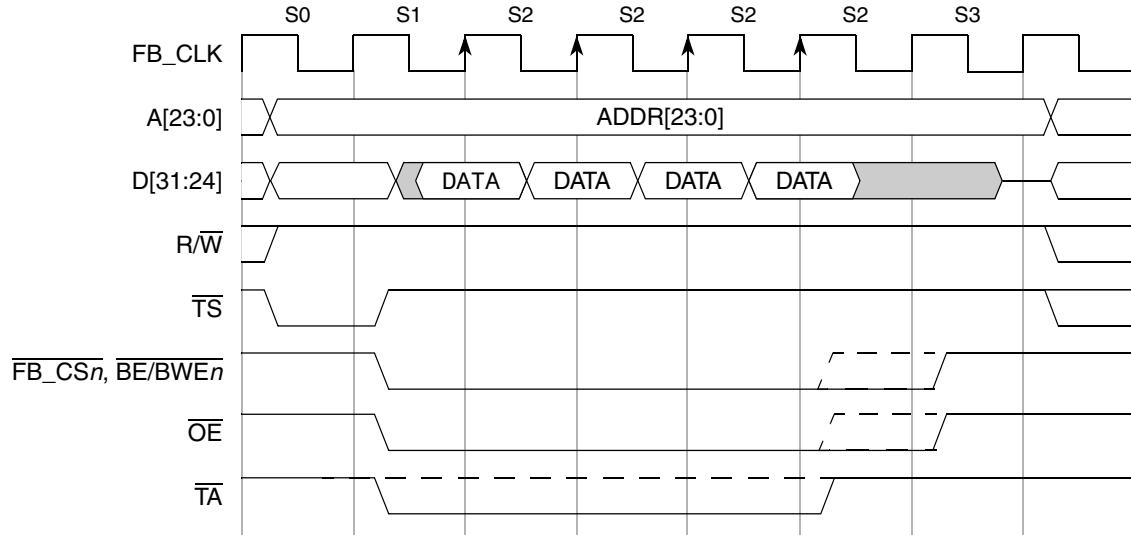


Figure 17-25. Longword Read Burst from 8-Bit Port 3-1-1-1 (No Wait States)

Figure 17-26 shows a longword write through an 8-bit device programmed for burst enable. The transfer results in a 4-beat burst and the data is driven on D[31:24].

NOTE

The first beat of any write burst cycle will have at least one wait state. If the bus cycle is programmed for zero wait states (CSCR_n[WS] = 0), then one wait state will be added. Otherwise the programmed number of wait states will be used.

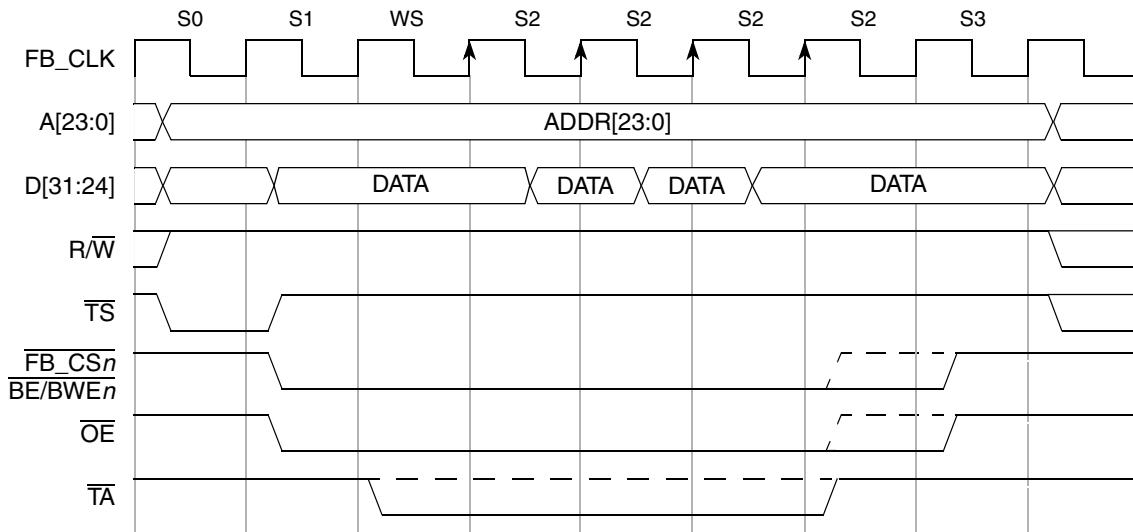


Figure 17-26. Longword Write Burst to 8-Bit Port 4-1-1-1 (No Wait States)

Figure 17-27 shows a longword read through an 8-bit device with burst inhibited. The transfer results in four individual transfers.

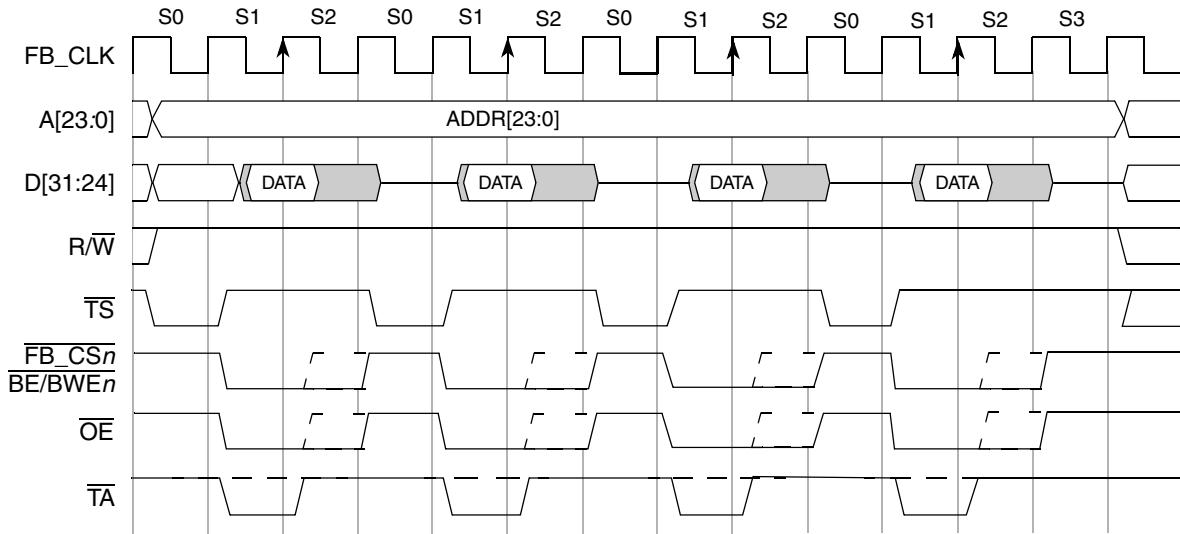


Figure 17-27. Longword Read Burst-Inhibited from 8-Bit Port (No Wait States)

Figure 17-28 shows a longword write through an 8-bit device with burst inhibited. The transfer results in four individual transfers.

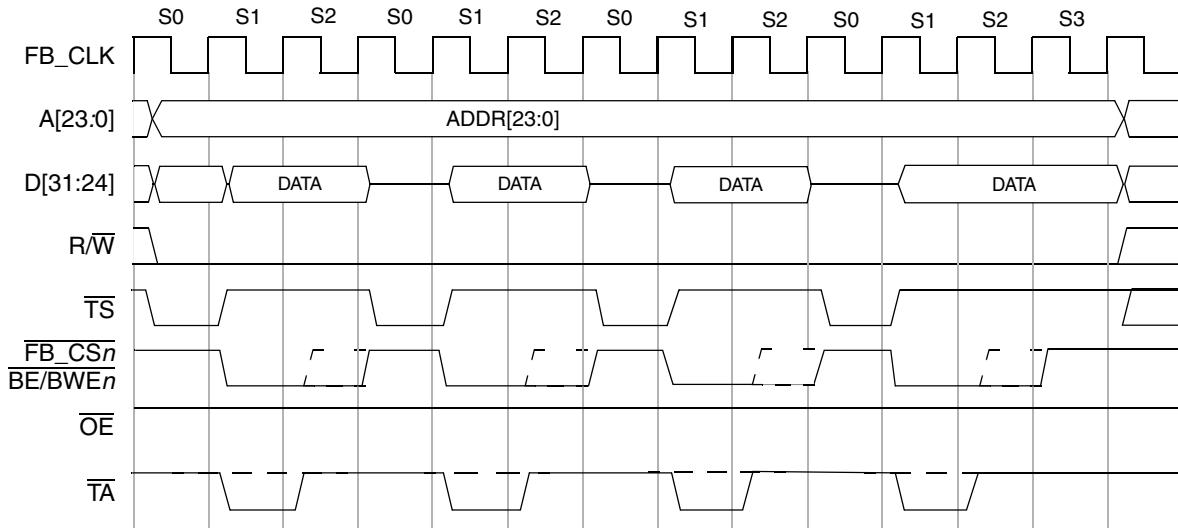


Figure 17-28. Longword Write Burst-Inhibited to 8-Bit Port (No Wait States)

Figure 17-29 illustrates another read burst transfer, but in this case a wait state is added between individual beats.

NOTE

The number of wait states in the first beat is determined by CSCR n [WS], but for subsequent beats, the number of wait states is determined by CSCR n [WS] or CSCR n [SWS] based on the value of CSCR n [SWSEN].

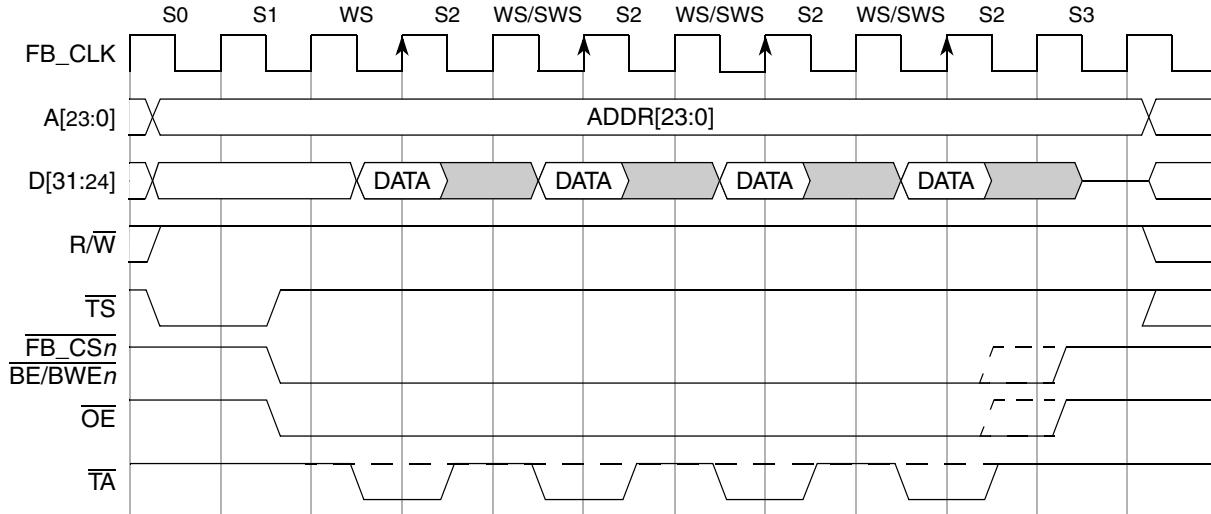


Figure 17-29. Longword Read Burst from 8-Bit Port 4-2-2-2 (One Wait State)

Figure 17-29 illustrates a write burst transfer with one wait state.

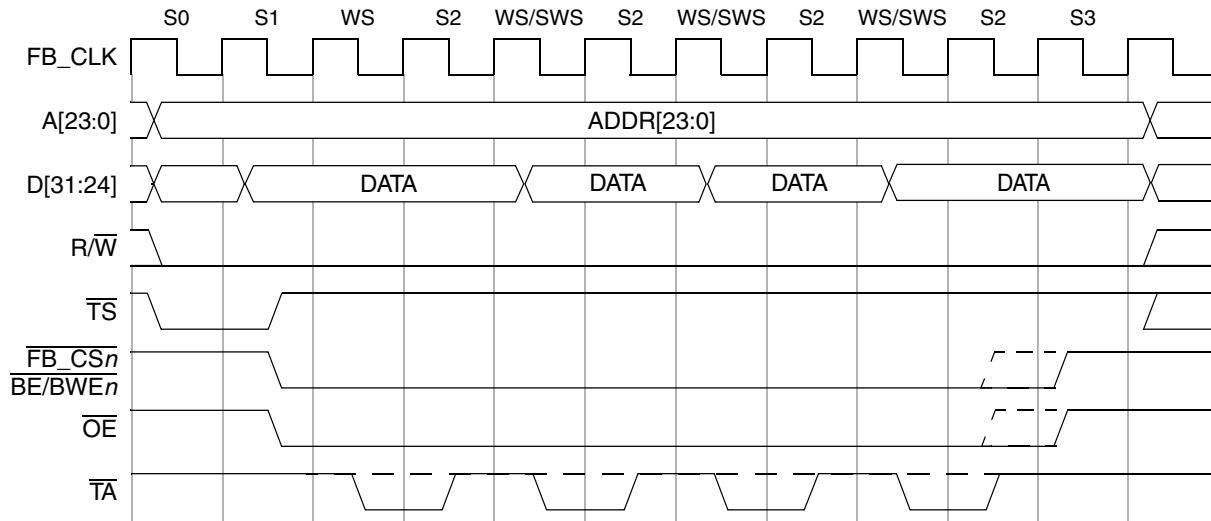


Figure 17-30. Longword Write Burst to 8-Bit Port 4-2-2-2 (One Wait State)

If address setup and hold are used, only the first and last beat of the burst cycle will be affected as shown in Figure 17-31.

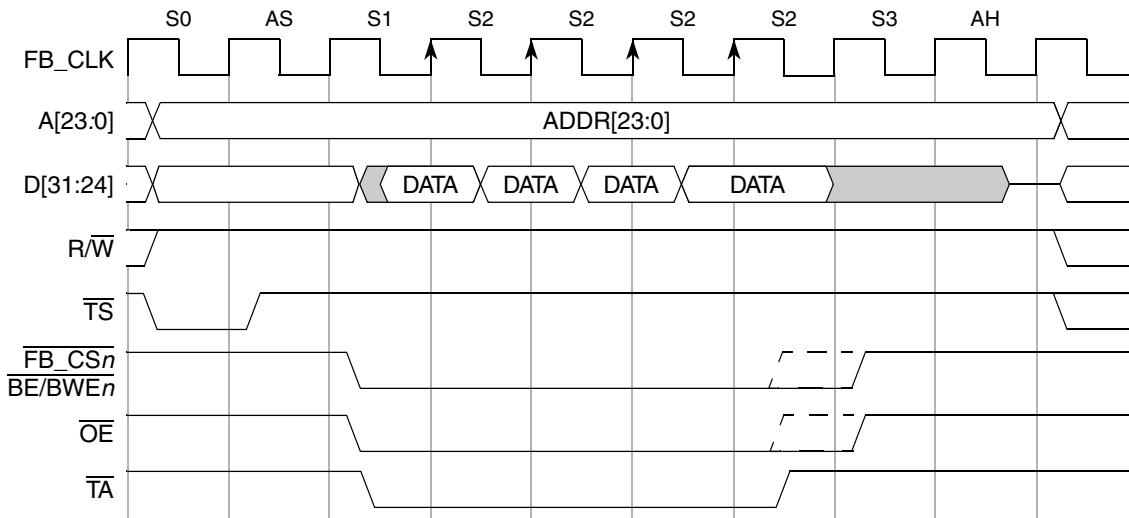


Figure 17-31. Longword Read Burst from 8-Bit Port 4-1-1-1 (Address Setup and Hold)

Figure 17-32 shows a write cycle with one clock of address setup and address hold.

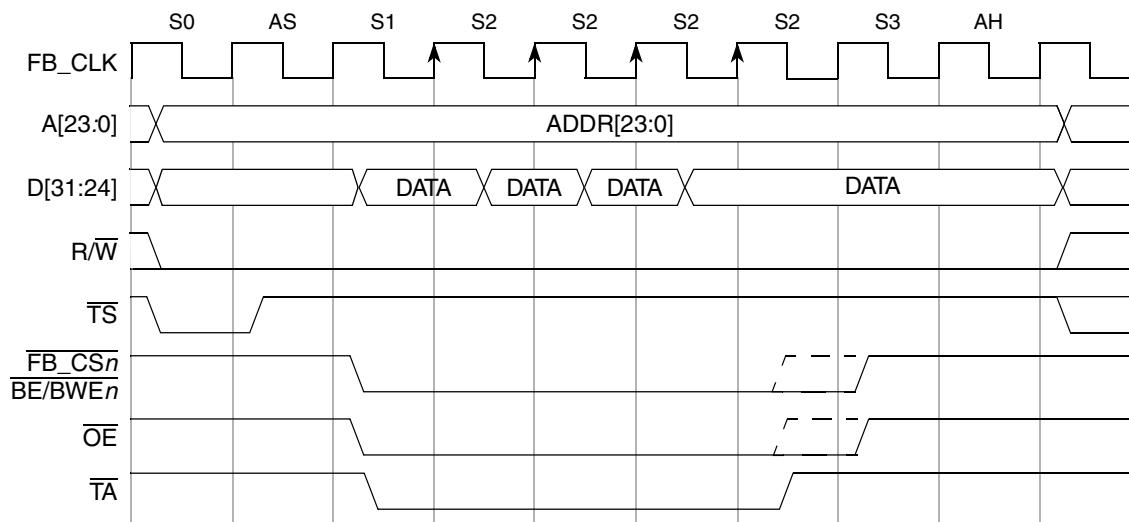


Figure 17-32. Longword Write Burst to 8-Bit Port 4-1-1-1 (Address Setup and Hold)

17.4.6 Misaligned Operands

Because operands, unlike opcodes, can reside at any byte boundary, they are allowed to be misaligned. A byte operand is properly aligned at any address, a word operand is misaligned at an odd address, and a longword is misaligned at an address not a multiple of four. Although the processor enforces no alignment restrictions for data operands (including program counter (PC) relative data addressing), additional bus cycles are required for misaligned operands.

Instruction words and extension words (opcodes) must reside on word boundaries. Attempting to prefetch a misaligned instruction word causes an address error exception.

The device converts misaligned, cache-inhibited operand accesses to multiple aligned accesses.

[Figure 17-33](#) shows the transfer of a longword operand from a byte address to a 32-bit port. First, a byte is transferred 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 is transferred with a byte offset of 0x2. The next two bytes are transferred in this cycle. In the third cycle, byte 3 is transferred. The byte offset is now 0x0, the port supplies the final byte, and the operation is complete.

	31	24 23	16 15	8 7	0	A[2:0]
Transfer 1	—	Byte 0	—	—	—	001
Transfer 2	—	—	Byte 1	Byte 2	—	010
Transfer 3	Byte 3	—	—	—	—	100

Figure 17-33. Example of 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 [Figure 17-34](#) differs from the one in [Figure 17-33](#) because the operand is word-sized and the transfer takes only two bus cycles.

	31	24 23	16 15	8 7	0	A[2:0]
Transfer 1	—	—	—	Byte 0	—	001
Transfer 2	Byte 0	—	—	—	—	100

Figure 17-34. Example of a Misaligned Word Transfer (32-Bit Port)

17.4.7 Bus Errors

The ColdFire device has no bus monitor. If the auto-acknowledge feature is not enabled for the address that generates the error, the bus cycle can be terminated by asserting \overline{TA} or by using the software watchdog timer. If it is required that the processor handle a bus error differently, an interrupt handler can be invoked by asserting an interrupt to the core along with \overline{TA} when the bus error occurs.

Chapter 18

SDRAM Controller (SDRAMC)

18.1 Introduction

This chapter describes configuration and operation of the synchronous DRAM (SDRAM) controller. It begins with a general description and brief glossary, and includes a description of signals involved in DRAM operations. The remainder of the chapter describes the programming model and signal timing, as well as the command set required for synchronous operations. It also includes examples that the designer can follow to better understand how to configure the DRAM controller for synchronous operations.

NOTE

Unless otherwise noted, in this chapter ‘clock’ refers to the FB_CLK used for the bus ($f_{sys/2}$).

The external data bus is shared between the FlexBus module and the SDRAM controller. When the SDRAM controller is in SDR mode (DRAMSEL = 1), the data bus is switched dynamically between the SDRAM controller and the FlexBus module. However, when the SDRAM controller is in DDR mode (DRAMSEL = 0), D[31:16] is dedicated to the SDRAM data bus and D[15:0] is dedicated to the FlexBus data bus.

In this chapter, the SDRAM data bus signals named SD_D[31:0]. However, since these signals share external pins with the FlexBus, the pin names on the device are D[31:0].

18.1.1 Block Diagram

The figure below is a block diagram of the SDRAM controller.

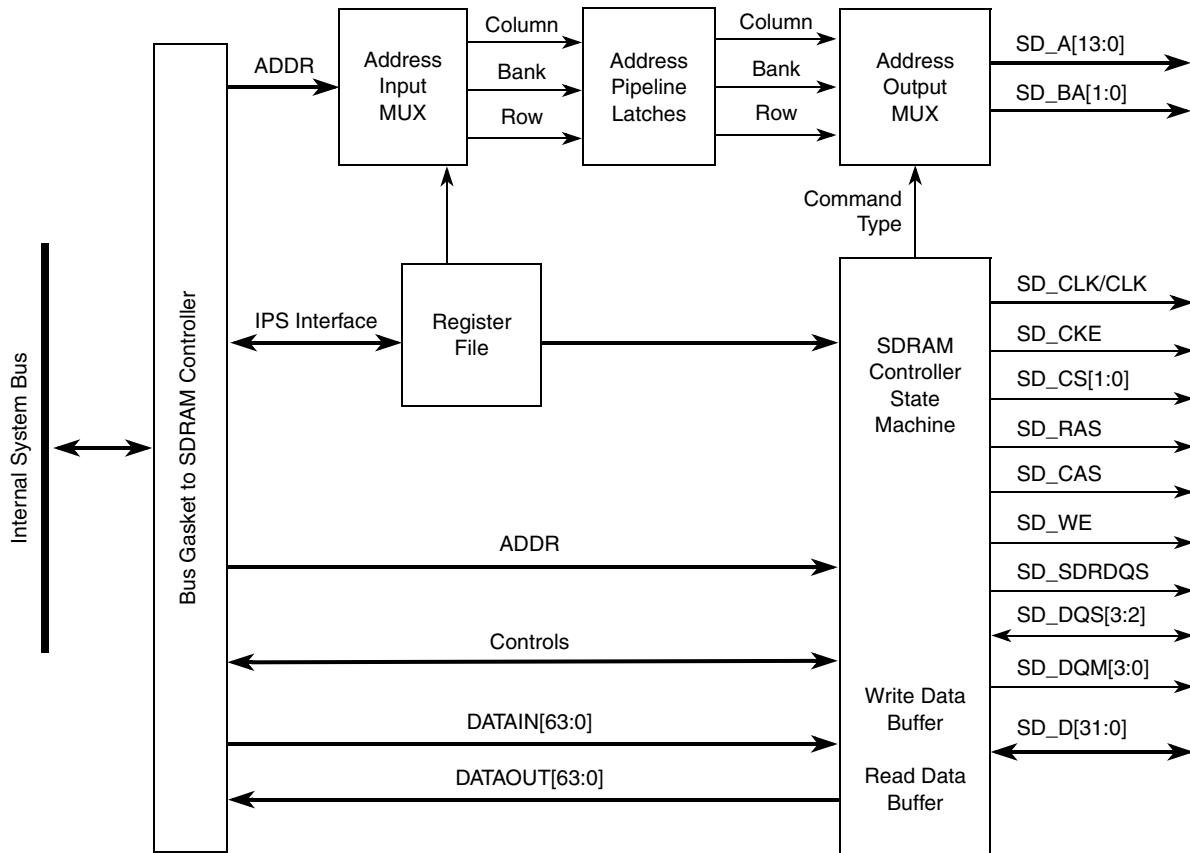


Figure 18-1. SDRAM Controller Block Diagram

18.1.2 Features

The SDRAM controller contains the following features:

- Supports standard SDRAM (single data rate, or SDR) and dual data rate (DDR) SDRAM; one or the other, not mixed.
- Support for lower-power/mobile DDR SDRAM.
- Dynamic 16- or 32-bit fixed memory data port width.
- 16 bytes critical word first burst transfer. Supports sequential address order only.
- Up to 14 lines of row address, up to 12 (in 32-bit bus mode) or 13 (in 16-bit bus mode) column address lines, 2 bits of bank address, and two pinned-out chip selects. The maximum row bits plus column bits equals 24 in 32-bit bus mode or 25 in 16-bit bus mode.

- Minimum memory configuration of 8 MByte
 - 11 bit row address (RA), 8 bit column address (CA), 2 bit bank address (BA), 32-bit bus, one chip select
 - 11 bit row address (RA), 9 bit column address (CA), 2 bit bank address (BA), 16-bit bus, one chip select
- Supports up to 512MByte of memory.
 - 24 bits RA+CA, 2 bits BA, 32-bit bus, two chip selects
 - 25 bits RA+CA, 2 bits BA, 16-bit bus, two chip selects
- Supports page mode for decreased latency and higher bandwidth; remembers one active row for each bank; four independent active rows per each chip select.
- Programmable refresh interval timer.
- Supports sleep mode and self-refresh mode.
- Error detect and parity check are not supported.
- The SDRAM controller does not include a dedicated I²C interface to access memory module (DIMM) serial presence detect EEPROM. If needed, this must be handled by one of the on-chip I²C channels external to the SDRAM controller.

18.1.3 Terminology

The following terminology is used in this chapter:

- SDRAM block: Any group of DRAM memories selected by one of the $\overline{\text{SD_CS}}$ signals. Thus, the SDRAMC can support up to two independent memory blocks. The base address of each block is programmed in the SDRAM chip-select configuration registers.
- SDRAM bank: An internal partition in an SDRAM device. For example, a 64-Mbit SDRAM component might be configured as four 512K x 32 banks. Banks are selected through the SD_BA[1:0] signals.
- SDRAM: RAMs that operate like asynchronous DRAMs but with a synchronous clock, a pipelined, multiple-bank architecture, and a faster speed.

18.2 External Signal Description

This section introduces the signal names that will be used throughout the remainder of the SDRAM controller documentation.

Table 18-1. External Signal List

Signal Name	I/O	Active	Description
SD_CLK	O	Rising	Output clock for SDRAM cycles. 50% duty cycle.
$\overline{\text{SD_CLK}}$	O	SDR:None DDR:Rising	Inverted version of the SDRAM clock. Used with SD_CLK to provide the differential clocks for DDR SDRAM. This clock output is not used by SDR memory.
SD_CKE	O	High	This output is the SDRAM clock enable. SD_CKE negates to put the SDRAM into low-power, self-refresh mode.

Table 18-1. External Signal List (continued)

Signal Name	I/O	Active	Description
<code>SD_CS[1:0]</code>	O	Low	These signals interface to the chip select line of the SDRAMs within a memory block. Thus, there is one <code>SD_CS</code> line for each memory block.
<code>SD_RAS</code>	O	Low	SDRAM synchronous row address strobe.
<code>SD_CAS</code>	O	Low	SDRAM synchronous column address strobe.
<code>SD_WE</code>	O	Low	Memory “WE” command bit. Name is legacy from async DRAM “write enable”; function is similar but not identical.
<code>SD_A[13:0]</code>	O	Valid	Memory multiplexed row/column address. This bus carries mode code during memory initialization.
<code>SD_BA[1:0]</code>	O	Valid	Memory bank address. Each SDRAM module has four internal row banks. It is also used to select the SDRAM internal mode register during power-up initialization.
<code>SD_D[31:0]</code>	I/O	Valid	Memory read and write data. In 16-bit DDR configuration, the memory device data bus is connected to <code>SD_D[31:16]</code> bits.
<code>SD_DQM[3:0]</code>	O	High	Memory write data byte mask. Note the address correspondence shown below: <code>SD_DM3</code> - <code>SD_D[31:24]</code> <code>SD_DM2</code> - <code>SD_D[23:16]</code> <code>SD_DM1</code> - <code>SD_D[15:8]</code> <code>SD_DM0</code> - <code>SD_D[7:0]</code>
<code>SD_DQS[3:2]</code>	I/O	SDR:None DDR:Rising and Falling	Data strobe output to clock write data at memory. Note the address correspondence shown below: <code>SD_DQS3</code> - <code>SD_D[31:24]</code> <code>SD_DQS2</code> - <code>SD_D[23:16]</code>
<code>SD_SDRDQS</code>	O	Toggle	SDR data strobe. Generated by the memory controller in SDR mode, to mimic the DQS generated by DDR memories during reads. It should be routed out and connected back to <code>SD_DQS</code> inputs.

18.3 Interface Recommendations

18.3.1 Supported Memory Configurations

The SDRAM controller supports up to 14 row addresses and up to 12 (13 in 16-bit bus mode) column addresses. However, the maximum row and column addresses are not supported at the same time. The number of row and column addresses must be less than or equal to 24 (25 in 16-bit bus mode). In addition to row/column address lines, there are always two row bank address bits. Therefore, the greatest possible address space which can be accessed using a single chip select is $2^{26} \times 32$ bit ($2^{27} \times 16$ bit) or 256 MBytes.

Table 18-3 and **Table 18-5** show the address multiplexing used by the memory controller for different configurations. When the SDRAM controller receives the internal module enable, it latches the internal bus address lines `A[27:0]` and multiplexes them into row, column, and bank addresses (RA, CA, and BA respectfully). In 32-bit bus mode, `A[9:2]` are used for `CA[7:0]`. In 16-bit mode, `A[9:1]` are used for `CA[8:0]`. `A[11:10]` are always used for `BA[1:0]`, and `A[23:12]` are always used for `RA[11:0]`. `A[27:24]` can

be used for additional row or column address bits, as needed. The additional row or column address bits are programmed via SDCR[addr_mux] registers.

NOTE

When the SDRAMC is configured to support an external 32-bit data bus. It is not possible to connect a smaller device(s) to only part of the SDRAM's data bus. For example, if 16-bit wide devices are used, then user must use two 16-bit devices connected as a 32-bit port.

Table 18-2. Address Multiplexing for 32-bit Bus Mode

addr_mux[1:0]	Internal Address Bits [27:24]			
	A[27]	A[26]	A[25]	A[24]
0x00	CA12	CA11	CA9	CA8
0x01	CA11	CA9	CA8	RA12
0x10	CA9	CA8	RA13	RA12
0x11	Reserved. Do Not Use.			

Table 18-3. SDRAM Address Multiplexing in 32-bit Bus Mode

Device	Configuration	Row bit x Col bit x Banks	addr_m ux[1:0]	Internal Address						
				27	26	25	24	23-12	11-10	9-2
64 Mbits	2M x 32 bit	11 x 8 x 4	00	— ^{1,2}	—	—	—	RA11-0	BA1-0	CA7-0
	4M x 16 bit	12 x 8 x 4	00	—	—	—	—			
	8M x 8 bit	12 x 9 x 4	00	—	—	—	CA8			
		13 x 8 x 4	01	—	—	—	RA12			
	16M x 4 bit	12 x 10 x 4	00	—	—	CA9	CA8			
		13 x 9 x 4	01	—	—	CA8	RA12			
128 Mbits	4M x 32 bit	12 x 8 x 4	00	—	—	—	—	RA11-0	BA1-0	CA7-0
	8M x 16 bit	12 x 9 x 4	00	—	—	—	CA8			
		13 x 8 x 4	01	—	—	—	RA12			
	16M x 8 bit	12 x 10 x 4	00	—	—	CA9	CA8			
		13 x 9 x 4	01	—	—	CA8	RA12			
		14 x 8 x 4	10	—	—	RA13	RA12			
	32M x 4 bit	12 x 11 x 4	00	—	CA11	CA9	CA8			
		13 x 10 x 4	01	—	CA9	CA8	RA12			
		14 x 9 x 4	10	—	CA8	RA13	RA12			

Table 18-3. SDRAM Address Multiplexing in 32-bit Bus Mode (continued)

Device	Configuration	Row bit x Col bit x Banks	addr_m ux[1:0]	Internal Address						
				27	26	25	24	23-12	11-10	9-2
256 Mbits	8M x 32 bit	12 x 9 x 4	00	—	—	—	CA8	RA11-0	BA1-0	CA7-0
		13 x 8 x 4	01	—	—	—	RA12			
	16M x 16 bit	12 x 10 x 4	00	—	—	CA9	CA8			
		13 x 9 x 4	01	—	—	CA8	RA12			
		14 x 8 x 4	10	—	—	RA13	RA12			
		12 x 11 x 4	00	—	CA11	CA9	CA8			
	32M x 8 bit	13 x 10 x 4	01	—	CA9	CA8	RA12			
		14 x 9 x 4	10	—	CA8	RA13	RA12			
		12 x 12 x 4	00	CA12	CA11	CA9	CA8			
	64M x 4 bit	13 x 11 x 4	01	CA11	CA9	CA8	RA12			
		14 x 10 x 4	10	CA9	CA8	RA13	RA12			
		12 x 10 x 4	00	—	—	CA9	CA8			
512 Mbits	16M x 32 bit	13 x 9 x 4	01	—	—	CA8	RA12	RA11-0	BA1-0	CA7-0
		14 x 8 x 4	10	—	—	RA13	RA12			
		12 x 11 x 4	00	—	CA11	CA9	CA8			
	32 M x 16 bit	13 x 10 x 4	01	—	CA9	CA8	RA12			
		14 x 9 x 4	10	—	CA8	RA13	RA12			
		12 x 12 x 4	00	CA12	CA11	CA9	CA8			
	64M x 8 bit	13 x 11 x 4	01	CA11	CA9	CA8	RA12			
		14 x 10 x 4	10	CA9	CA8	RA13	RA12			
		12 x 11 x 4	00	—	CA11	CA9	CA8			
1 Gbits	32M x 32 bit	13 x 10 x 4	01	—	CA9	CA8	RA12	RA11-0	BA1-0	CA7-0
		14 x 9 x 4	10	—	CA8	RA13	RA12			
		12 x 12 x 4	00	CA12	CA11	CA9	CA8			
	64M x 16 bit	13 x 11 x 4	01	CA11	CA9	CA8	RA12			
		14 x 10 x 4	10	CA9	CA8	RA13	RA12			
		12 x 12 x 4	00	CA12	CA11	CA9	CA8			
2 Gbits	64M x 32 bit	13 x 11 x 4	01	CA11	CA9	CA8	RA12	RA11-0	BA1-0	CA7-0
		14 x 10 x 4	10	CA9	CA8	RA13	RA12			

1- All SD_A[13:0] bits are generated on every access, but only the bits actually used by the memory are shown.

2- All column address (CA) bits in this table are physical column address lines. The SDRAM controller inserts an extra bit CA10 to control the precharge option.

Table 18-4. Address Multiplexing for 16-bit Bus Mode

addr_mux[1:0]	Internal Address Bits [27:24]			
	A[27]	A[26]	A[25]	A[24]
0x00	CA13	CA12	CA11	CA9
0x01	CA12	CA11	CA9	RA12
0x10	CA11	CA9	RA13	RA12
0x11	Reserved. Do Not Use.			

Table 18-5. SDRAM Address Multiplexing in 16-bit Bus Mode

Device	Configuration	Row bit x Col bit x Banks	addr_m ux[1:0]	Internal Address							
				27	26	25	24	23-12	11-10	9-1	
64 Mbits	4M x 16 bit	11 x 9 x 4	00	— ^{1,2}	—	—	—	RA11-0	BA1-0	CA8-0	
	8M x 8 bit	12 x 9 x 4	00	—	—	—	—				
	16M x 4 bit	12 x 10 x 4	00	—	—	—	CA9				
		13 x 9 x 4	01	—	—	—	RA12				
128 Mbits	8M x 16 bit	12 x 9 x 4	00	—	—	—	—	RA11-0	BA1-0	CA8-0	
	16M x 8 bit	12 x 10 x 4	00	—	—	—	CA9				
		13 x 9 x 4	01	—	—	—	RA12				
		12 x 11 x 4	00	—	—	CA11	CA9				
	32M x 4 bit	13 x 10 x 4	01	—	—	CA9	RA12				
		14 x 9 x 4	10	—	—	RA13	RA12				
256 Mbits	16M x 16 bit	12 x 10 x 4	00	—	—	—	CA9	RA11-0	BA1-0	CA8-0	
		13 x 9 x 4	01	—	—	—	RA12				
	32M x 8 bit	12 x 11 x 4	00	—	—	CA11	CA9				
		13 x 10 x 4	01	—	—	CA9	RA12				
		14 x 9 x 4	10	—	—	RA13	RA12				
	64M x 4 bit	12 x 12 x 4	00	—	CA12	CA11	CA9				
		13 x 11 x 4	01	—	CA11	CA9	RA12				
		14 x 10 x 4	10	—	CA9	RA13	RA12				

Table 18-5. SDRAM Address Multiplexing in 16-bit Bus Mode (continued)

Device	Configuration	Row bit x Col bit x Banks	addr_m ux[1:0]	Internal Address							
				27	26	25	24	23-12	11-10	9-1	
512 Mbits	32 M x 16 bit	12 x 11 x 4	00	—	—	CA11	CA9	RA11-0	BA1-0	CA8-0	
		13 x 10 x 4	01	—	—	CA9	RA12				
		14 x 9 x 4	10	—	—	RA13	RA12				
	64M x 8bit	12 x 12 x 4	00	—	CA12	CA11	CA9				
		13 x 11 x 4	01	—	CA11	CA9	RA12				
		14 x 10 x 4	10	—	CA9	RA13	RA12				
1 Gbits	64M x 16bit	12 x 12 x 4	00	—	CA12	CA11	CA9	RA11-0	BA1-0	CA8-0	
		13 x 11 x 4	01	—	CA11	CA9	RA12				
		14 x 10 x 4	10	—	CA9	RA13	RA12				
2 Gbits	128M x16bit	12 x 13 x 4	00	CA13	CA12	CA11	CA9	RA11-0	BA1-0	CA8-0	
		13 x 12 x 4	01	CA12	CA11	CA9	RA12				
		14 x 11 x 4	10	CA11	CA9	RA13	RA12				

¹ All SD_A[13:0] bits are generated on every access, but only the bits actually used by the memory are shown.

² All column address (CA) bits in this table are physical column address lines. The SDRAM controller inserts an extra bit CA10 to control the precharge option.

All memory devices of a single chip select block must have the same configuration and row/col address width; however, this is not necessary between different blocks. If mixing different memory organizations in different blocks, the following guidelines will ensure that every block is fully contiguous.

For 32-bit data bus configuration:

- If all devices' row address width is 12 bits, the column address can be ≥ 8 bits.
- If all devices' row address width is 13 bits, the column address can be ≥ 8 bits.
- If all devices' column address width is 8 bits, the row address can be ≥ 11 bits.
- The maximum row bits plus column bits equals 24.
- x8 and x16 data width memory devices can be mixed (but not in the same space).
- x32 data width memory devices cannot be mixed with any other width.

For 16-bit data bus configuration:

- If all devices' row address width is 12 bits, the column address can be ≥ 9 bits.
- If all devices' row address width is 13 bits, the column address can be ≥ 9 bits.
- If all devices' column address width is 9 bits, the row address can be ≥ 11 bits.
- The maximum row bits plus column bits equals 25.
- x16 data width memory devices cannot be mixed with any other width.

18.3.2 SDRAM SDR Connections

Figure 18-2 shows a block diagram of the connections between the memory controller and SDR SDRAM components. SDR design requires special timing consideration for the SD_DQS[3:2] signals. For reads from DDR SDRAMs, the memory will drive the DQS pins so that the data lines and DQS signals have concurrent edges. The SDRAMC is designed to latch data 1/4 clock after the SD_DQS[3:2] edge. For DDR SDRAM, this ensures that the latch time is in the middle of the data valid window.

The SDRAMC also uses the SD_DQS[3:2] signals to determine when read data can be latched for SDR SDRAM; however, SDR memories do not provide DQS outputs. Instead the SDRAMC provides a SD_SDRDQS output that is routed back into the controller as SD_DQS[3:2]. The SD_SDRDQS signal should be routed such that the valid data from the SDRAM reaches the controller at the same time or just before the SD_SDRDQS reaches the SD_DQS[3:2] inputs.

When routing SD_SDRDQS the outbound trace length should be matched to the SD_CLK trace length. This will align SD_SDRDQS to the SD_CLK as if the memory had generated the DQS pulse. The inbound trace should be routed along the data path, which should synchronize the SD_DQS so that the data is latched in the middle of the data valid window.

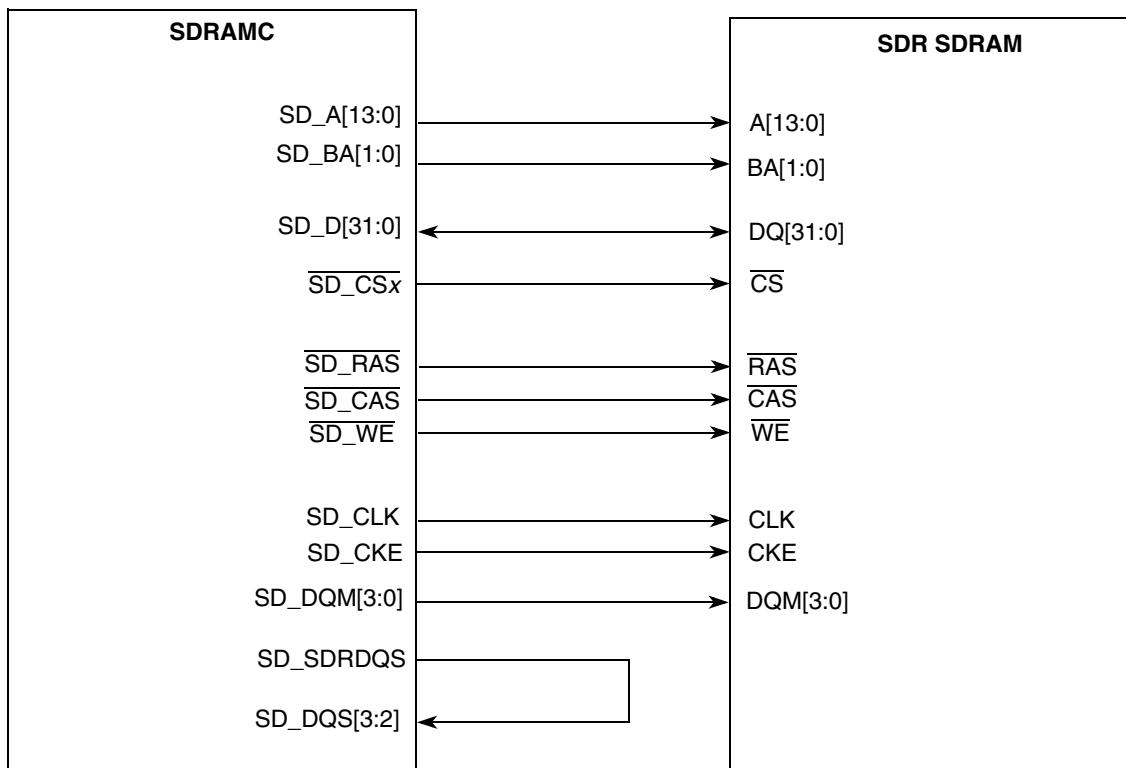


Figure 18-2. Connections to SDR SDRAM

18.3.3 SDRAM DDR Component Connections

Figure 18-3 shows a block diagram of the connections between the controller and DDR SDRAM components.

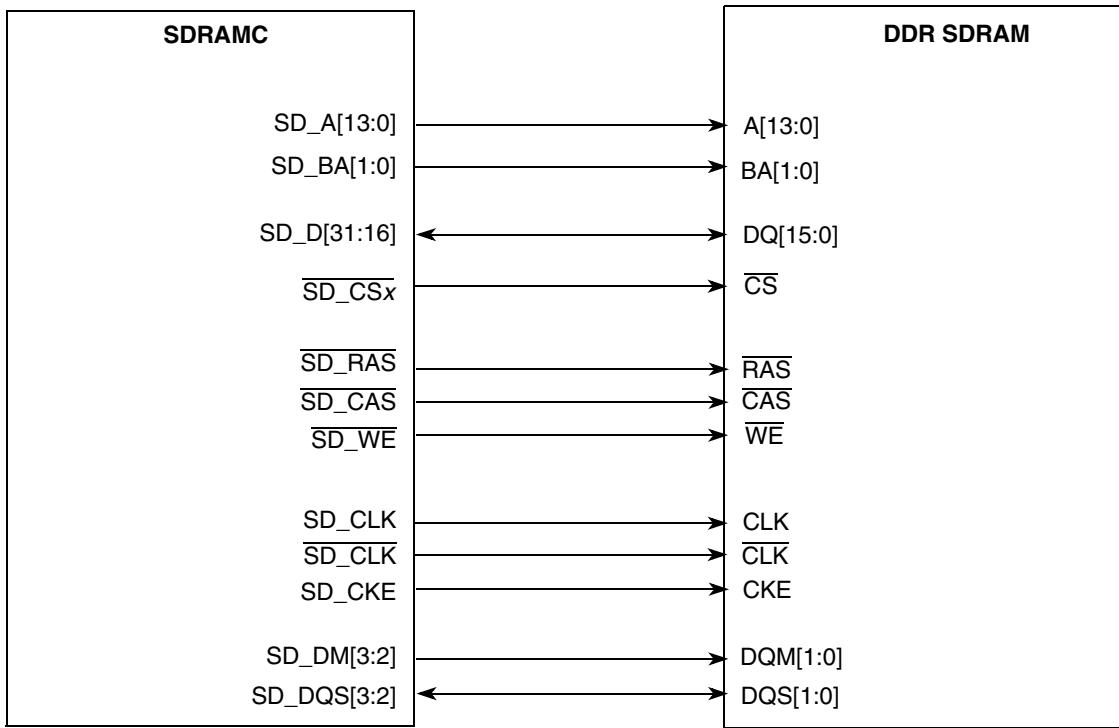


Figure 18-3. Connections to DDR SDRAM

18.3.4 DDR SDRAM Layout Considerations

Due to the critical timing for DDR SDRAM, there are a number of considerations that should be taken into account during PCB layout:

- Minimize overall trace lengths.
- Each DQS, DM, and DQ group must have identical loading and similar routing to maintain timing integrity.
- The loading and routing of SD_DQS must match those of SD_D.
- Control and clock signals are routed point-to-point.
- Trace length for clock, address, and command signals should match.
- Route DDR signals on layers adjacent to the ground plane.
- Use a VREF plane under the SDRAM.
- VREF is decoupled from both SDVDD and VSS.
- To avoid crosstalk, keep address and command signals separate from data and data strobes.
- Use different resistor packs for command/address and data/data strobes.

- Use single series termination (25 Ω value is recommended, but standard resistor packs with similar values can be substituted).
- Series termination should be between the processor and memory, but closest to the processor.
- The SD_CLK and $\overline{\text{SD}_\text{CLK}}$ signals require a 100 Ω parallel terminal to each other.
- 0.1 μF decoupling for every termination resistor pack.

NOTE

Only series termination is supported on this device, which is different than typical DDR designs.

18.3.4.1 Termination Example

Figure 18-4 shows the recommended termination circuitry for DDR SDRAM signals.

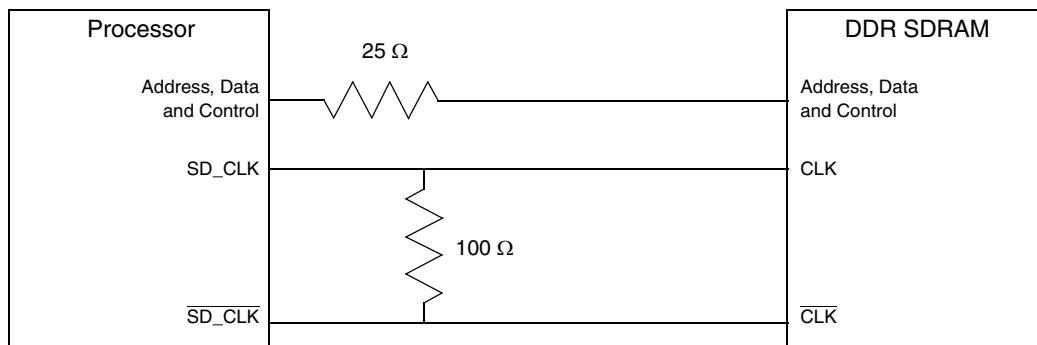


Figure 18-4. DDR SDRAM Termination Circuit

18.4 Memory Map/Register Definition

The SDRAM controller and its associated logic contain two sets of programming registers:

- SDRAM controller's control and configuration registers
- Chip select configuration control registers

Table 18-6 shows the SDRAM controller control and configuration registers. Unspecified memory spaces are reserved for future use. Access to reserved space is prohibited. It is recommended to write 0 to reserved space. Reads from a write-only bit will return 0.

Table 18-6. SDRAMC Memory Map

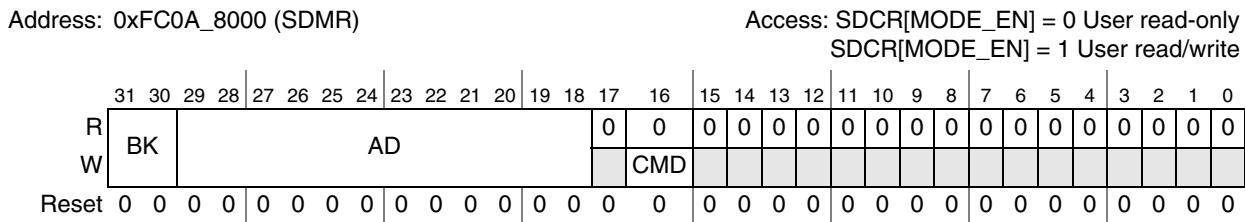
Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_8000	SDRAM Mode/Extended Mode Register (SDMR)	32	R/W	0x0000_0000	18.4.1/18-12
0xFC0A_8004	SDRAM Control Register (SDCR)	32	R/W	0x0000_0000	18.4.2/18-13
0xFC0A_8008	SDRAM Configuration Register 1 (SDCFG1)	32	R/W	0x0000_0000	18.4.3/18-14
0xFC0A_800C	SDRAM Configuration Register 2 (SDCFG2)	32	R/W	0x0000_0000	18.4.4/18-16

Table 18-6. SDRAMC Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC0A_8110	SDRAM Chip Select 0 Configuration (SDCS0)	32	R/W	0x0000_0000	18.4.5/18-17
0xFC0A_8114	SDRAM Chip Select 1 Configuration (SDCS1)	32	R/W	0x0000_0000	18.4.5/18-17

18.4.1 SDRAM Mode/Extended Mode Register (SDMR)

The SDMR, shown in [Figure 18-5](#), is used to write to the mode and extended mode registers that physically reside within the SDRAM chips. These registers must be programmed during SDRAM initialization. See [Section 18.5.2, “Power-Up Initialization”](#) for more information on the initialization sequence.

**Figure 18-5. SDRAM Mode/Extended Mode Register (SDMR)****Table 18-7. SDMR Field Descriptions**

Field	Description
31–30 BK	Bank address. Driven onto SD_BA[1:0] along with a LMR/LEMR command. All SDRAM chip selects are asserted simultaneously. SDCR[CKE] must be set before attempting to generate an LMR/LEMR command. The SD_BA[1:0] value is used to select between LMR and LEMR commands. 00 Load mode register command (LMR) 01 Load extended mode register command (LEMR) for non-mobile DDR devices 10 Load extended mode register command (LEMR) for mobile DDR devices 11 Reserved
29–18 AD	Address. Driven onto SD_A[11:0] along with an LMR/LEMR command. The AD value is stored as the mode (or extended mode) register data.
17	Reserved, should be cleared.
16 CMD	Command. This bit is write-only and always returns a ‘0’ when read. 1 Generate an LMR/LEMR command 0 Do not generate any command
15–0	Reserved, should be cleared.

18.4.2 SDRAM Control Register (SDCR)

The SDCR, shown in [Figure 18-6](#), control SDRAMC operating modes including the refresh count and address line muxing.

Address: 0xFC0A_8004 (SDCR)

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	MOD	CKE	DDR_MODE	REF_EN	0	0	ADDR_MUX		0	OE_RULE	REF_CNT					
W	E_EN															
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	MEM_PS	0	DQS_OE	0	0	0	0	0	0	0	0	0	0	0
W														IREF	IPALL	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 18-6. SDRAM Control Register (SDCR)

Table 18-8. SDCR Field Descriptions

Field	Description
31 MODE_EN	SDRAM mode register programming enable. 0 SDMR locked, cannot be written. 1 SDMR enabled, can be written.
30 CKE	Clock enable. CKE must be set to perform normal read and write operations. Clear CKE to put the memory in self-refresh or power-down mode. 0 SD_CKE is negated (low) 1 SD_CKE is asserted (high)
29 DDR_MODE	DDR mode select. 0 SDR mode 1 DDR mode
28 REF_EN	Refresh enable. 0 Automatic refresh disabled 1 Automatic refresh enabled
27–26	Reserved, should be cleared.
25–24 ADDR_MUX	Controls the use of internal address bits A[27:24] as row or column bits on the SD_A bus. See Table 18-2 , Table 18-3 , Table 18-4 , and Table 18-5 .
23	Reserved, should be cleared.
22 OE_RULE	Drive rule selection. 0 Tri-state except to write. SD_D and SD_DQS are only driven when necessary to perform a write command. 1 Drive except to read. SD_D and SD_DQS are only tristated when necessary to perform a read command. When not being driven for a write cycle, SD_D hold the most recent value and SD_DQS are driven low. This mode is intended for minimal applications only, to prevent floating signals and allow unterminated board traces. However, terminated wiring is always recommended over unterminated.

Table 18-8. SDCR Field Descriptions (continued)

Field	Description
21–16 REF_CNT	The average periodic interval at which the controller generates refresh commands to memory; measured in increments of $64 \times SD_CLK$ period. $REF_CNT = (t_{REFI} / (t_{CK} \times 64)) - 1$, rounded down to the next integer value. If the SDRAM data sheet does not define t_{REFI} , it can be calculated by $t_{REFI} = t_{REF} / \#rows$.
15–14	Reserved, should be cleared.
13 MEM_PS	Memory data port size. 0 32-bit data bus 1 16-bit data bus
12	Reserved, should be cleared.
11–10 DQS_OE	DQS output enable. Each DQS_OE bit is a master enable for the corresponding SD_DQSn signal. The DSQ_OE[1] bit enables SD_DQS3 and the DSQ_OE[0] bit enables SD_DQS2. 0 SD_DQSn can never drive. Use this value in SDR mode or in DDR mode with a “single DQS” memory. Some 32-bit DDR devices only have a single DQS pin. Enable one of the SD_DQSn signals and disable the other. Then short both pins external to the device. 1 SD_DQSn can drive as necessary, depending on commands and SDCR[OE_RULE] setting. DDR only.
9–3	Reserved, should be cleared.
2 IREF	Initiate Refresh command. Used to force a software initiated Refresh command. This bit is write-only, reads return zero. 0 Do not generate a Refresh command. 1 Generate a Refresh command. All $\overline{SD_CSn}$ signals are asserted simultaneously. SDCR[CKE] must be set before attempting to generate a software refresh command. Note: A software requested refresh is completely independent of the periodic refresh interval counter. Software refresh is only possible when MODE_EN = 1.
1 IPALL	Initiate Precharge All command. Used to force a software initiated precharge all command. This bit is write-only, reads return zero. 0 Do not generate a precharge command. 1 Generate a precharge all command. All $\overline{SD_CSn}$ signals are asserted simultaneously. SDCR[CKE] must be set before attempting to generate a software precharge command. Note: Software precharge is only possible when MODE_EN = 1. Note: Do not set IREF and IPALL at the same time.
0	Reserved, should be cleared.

18.4.3 SDRAM Configuration Register 1 (SDCFG1)

The 32-bit read/write SDRAM configuration register 1 (SDCFG1) stores delay values necessary between specific SDRAM commands. During initialization, software loads values to the register according to the selected SD_CLK frequency and SDRAM information obtained from the data sheet. This register is reset only by a power-up reset signal.

The read and write latency fields govern the relative timing of commands and data, and must be exact values. All other fields govern the relative timing from one command to another, they have minimum values but any larger value is also legal (but with decreased performance).

NOTE

In all calculations for setting the fields of this register, convert time units to clock units and always round up to the nearest integer.

Address: 0xFC0A_8008 (SDCFG1)												Access: User read/write							
R				W				R				W				R			
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	0	0	0	0
R	0	PRE2ACT		REF2ACT				0	WT_LAT				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 18-7. SDRAM Configuration Register 1 (SDCFG1)

Table 18-9. SDCFG1 Field Descriptions

Field	Description
31–28 SRD2RWP	Single Read to Read/Write/Precharge delay. Limiting case is Read to Write. SDR: SRD2RWP = CL + t_{HZ} + 2 DDR: SRD2RWP = CL + 1 t_{HZ} is the time it takes the data bus to return to hi-impedance after a read and is found in the SDRAM device specifications Note: Count value is in SD_CLK periods for both SDR and DDR mode.
27	Reserved, should be cleared.
26–24 SWT2RWP	Single Write to Read/Write/Precharge delay. Limiting case is Write to Precharge. SDR: SWT2RWP = t_{WR} DDR: SWT2RWP = t_{WR} + 1 clk Note: Count value is in SD_CLK periods for both SDR and DDR mode.
23–20 RD_LAT	Read CAS Latency. Read command to read data available delay counter. For DDR: If CL = 2, write 0x6 If CL = 2.5, write 0x7 For SDR: If CL = 2, write 0x2 If CL = 3, write 0x3 Note: CL = 2.5 is not supported for SDR. SDR: Count value is in SD_CLK periods. DDR: Count value is in SD_CLK2 periods.
19	Reserved, should be cleared.

Table 18-9. SDCFG1 Field Descriptions (continued)

Field	Description
18–16 ACT2RW	Active to Read/Write delay. Active command to any following read or write delay counter. Suggested value = $t_{RCD} - 1$ (Round up to nearest integer) Example: If $t_{RCD} = 20\text{ns}$ and SD_CLK = 99 MHz $20\text{ns} / 10.1\text{ ns} = 1.98$; round to 2. Note: Count value is in SD_CLK periods for both SDR and DDR mode.
15	Reserved, should be cleared.
14–12 PRE2ACT	Precharge to Active delay. Precharge command to following Active command delay counter. Suggested value = $t_{RP} - 1$ (Round up to nearest integer) Example: If $t_{RP} = 20\text{ns}$ and SD_CLK = 99MHz $20\text{ns} / 10.1\text{ns} = 1.98$; round to 2. Note: Count value is in SD_CLK periods for both SDR and DDR mode.
11–8 REF2ACT	Refresh to Active delay. Refresh command to following Active or Refresh command delay counter. Suggested value = $t_{RFC} - 1$ (Round up to nearest integer) Example: If $t_{RFC} = 75\text{ns}$ and SD_CLK = 99MHz $75\text{ns} / 10.1\text{ns} = 7.425$; round to 8. Note: Count value is in SD_CLK periods for both SDR and DDR mode.
7	Reserved, should be cleared.
6–4 WT_LAT	Write latency. Write command to write data delay counter. SDR: write 0x0 DDR: write 0x3 Note: SDR mode: Count value is in SD_CLK periods. DDR mode: Count value is in SD_CLK2 periods.
3–0	Reserved, should be cleared.

18.4.4 SDRAM Configuration Register 2 (SDCFG2)

The 32-bit read/write configuration register 2 stores delay values necessary between specific SDRAM commands. During initialization, software loads values to the register according to the SDRAM information obtained from the data sheet. This register is reset only by a power-up reset signal.

The burst length (BL) field must be exact. All other fields govern the relative timing from one command to another, they have minimum values but any larger value is also legal (but with decreased performance).

All delays in this register are expressed in SD_CLK. In all calculations for setting the fields of this register, convert time units to clock units and always round up to the nearest integer.

Address: 0xFC0A_800C (SCFG2)																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	BRD2RP				BWT2RWP				BRD2W				BL				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 18-8. SDRAM Configuration Register 2 (SDCFG2)

Table 18-10. SDCFG2 Field Descriptions

Field	Description
31–28 BRD2RP	Burst Read to Read/Precharge delay. Limiting case is Read to Read. SDR: BRD2RP = BurstLength + 1 DDR: BRD2RP = BurstLength/2 + 1
27–24 BWT2R WP	Burst Write to Read/Write/Precharge delay. Limiting case is Write to Precharge. SDR: BWT2RWP = BurstLength + t_{WR} - 2 DDR: BWT2RWP = BurstLength/2 + t_{WR}
23–20 BRD2W	Burst Read to Write delay. SDR: BRD2W = CL + BurstLength + t_{HZ} DDR: BRD2W = CL + BurstLength/2 - 1
19–16 BL	Burst Length. BL = BurstLength - 1 Note: Burst length depends on port size. If 32-bit bus (SDCR[MEM_PS] = 0), burst length is 4. Write BL = 3. If 16-bit bus (SDCR[MEM_PS] = 1), burst length is 8. Write BL = 7.
15–0	Reserved, should be cleared.

18.4.5 SDRAM Chip Select Configuration Registers (SDCSn)

These registers define base address and space size of each chip select.

NOTE

Since the SDRAM module is one of the slaves connected to the cross-bar switch, it is only accessible within a certain memory range. The only applicable address ranges for which the chip-selects can be active are 0x4000_0000–0x7FFF_FFFF. Be sure to set the SD**C**Sn registers appropriately.

Address: 0xFC0A_8110 (SDCS0) 0xFC0A_8114 (SDCS1)																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	CSBA								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 18-9. SRAM Chip Select Configuration Register (SDCSn)

Table 18-11. SDCS n Field Descriptions

Field	Description		
31–20 CSBA	Chip select base address. Since the SDRAM module is one of the slaves connected to the cross-bar switch, it is only accessible within a certain memory range. The only applicable address ranges for which the chip-selects can be active are 0x4000_0000–0x7FFF_FFFF. Therefore, the possible range for this field is 0x400–0x7FF.		
19–5	Reserved, should be cleared.		
4–0 CSSZ	Chip select size.		
CSSZ	Size	Address Lines to Compare	
00000	Disabled	—	
00001–10010	Reserved	Reserved	
10011	1 MByte	A[31:20]	
10100	2 MByte	A[31:21]	
10101	4 MByte	A[31:22]	
10110	8 MByte	A[31:23]	
10111	16 MByte	A[31:24]	
11000	32 MByte	A[31:25]	
11001	64 MByte	A[31:26]	
11010	128 MByte	A[31:27]	
11011	256 MByte	A[31:28]	
11100	512 MByte	A[31:29]	
11101	1 GByte	A[31:30]	
11110	2 GByte	A31	
11111	4 GByte	Ignore A[31:20]	

Any chip select can be enabled or disabled, independent of others. Any chip select can be allocated any size of address space from 1M to 4G, independent of others. Any chip select address space can begin at any size-aligned base address, independent of others.

For contiguous memory with different sizes of memory blocks, place largest block at lowest address and place smaller blocks in descending size order at ascending base addresses.

For example, assume a system with 2 chip selects: CS0=16M, CS1=256M:

CS0CFG = 0x4F000017 = enable 16M @ 0x4F000000-0x4FFFFFFF

CS1CFG = 0x5000001B = enable 256M @ 0x50000000-0x5FFFFFFF

This gives 272MB total memory, at 0x4F000000-0x5FFFFFFF.

18.5 Functional Overview

18.5.1 SDRAM Commands

When an internal bus master accesses SDRAM address space, the memory controller generates the corresponding SDRAM command. [Table 18-12](#) lists SDRAM commands supported by the memory controller.

Table 18-12. SDRAM Commands

Function	Symbol	CKE	\overline{CS}	\overline{RAS}	\overline{CAS}	\overline{WE}	$BA[1:0]$	$A[10]$	Other A
Command Inhibit	INH	H	H	X	X	X	X	X	X
No Operation	NOP	H	L	H	H	H	X	X	X
Row and Bank Active	ACTV	H	L	L	H	H	V	V	V
Read	READ	H	L	H	L	H	V	L	V
Write	WRITE	H	L	H	L	L	V	L	V
Burst Terminate	BST	H	L	H	H	L	X	X	X
Precharge All Banks	PALL	H	L	L	H	L	X	H	X
Precharge Selected Bank	PRE	H	L	L	H	L	V	L	X
Load Mode Register	LMR	H	L	L	L	L	LL	V	V
Load Extended Mode Register	LEMR	H	L	L	L	L	LH	V	V
Auto Refresh	REF	H	L	L	L	H	X	X	X
Self Refresh	SREF	H→L	L	L	L	H	X	X	X
Power Down	PDWN	H→L	H	X	X	X	X	X	X
H = High L = Low V = Valid X = Don't care									

Many commands require a delay before the next command may be issued; sometimes the delay depends on the type of the next command. These delay requirements are managed by the values programmed in the memory controller configuration registers (SDCFG1, SDCFG2).

18.5.1.1 Row and Bank Active Command (ACTV)

The ACTV command is responsible for latching the row and bank address and activating the specified row bank of a memory block. Once the row is activated, it can be accessed using subsequent read and write commands.

NOTE

The SDRAMC will support one active row for each chip select block. See [Section 18.5.3, “Page Management,”](#) for more information.

18.5.1.2 Read Command (READ)

When the SDRAMC receives a read request via the internal bus, it first checks the row and bank of the new access. If the address falls within the active row of an active bank, it is a page hit, and the READ is issued as soon as possible (pending any delays required by previous commands). If the address is within an inactive bank, the memory controller will issue an ACTV followed by the READ command. If the address is not within the active row of an active bank, the memory controller will issue a PRE command to close the active row. Then the SDRAMC issues ACTV to activate the necessary row and bank for the new access, followed finally by the READ to the SDRAM.

The PALL/PRE and ACTV commands (if necessary) can sometimes be issued in parallel with an on-going data movement.

To truncate a burst read when only a single read is needed, the memory controller issues the burst terminate command. With SDR memory, the data masks are negated throughout the entire read size. With DDR memory, the data masks are asserted high throughout the entire read size; but DDR memory ignores the data masks during reads.

18.5.1.3 Write Command (WRITE)

When the memory controller receives a write request via the internal bus, it first checks the row and bank of the new access. If the address falls within the active row of an active bank, it is a page hit, and the WRITE is issued as soon as possible (pending any delays required by previous commands). If the address is within the inactive bank, the memory controller will issue an ACTV followed by the WRITE command. If the address is not within the active row of an active bank, the memory controller will issue a PRE command to close the active row. Then the SDRAMC issues ACTV to activate the necessary row and bank for the new access, followed finally by the WRITE command to the SDRAM.

The PALL/PRE and ACTV commands (if necessary) can sometimes be issued in parallel with an on-going data movement.

18.5.1.4 Burst Terminate Command (BST)

SDRAMs are burst-only devices, but they provide mechanisms to truncate a burst if not all of the beats are needed. The burst terminate command is used to truncate read bursts (SDR and DDR) and write bursts (SDR). In order to truncate a burst write for DDR, the read command can be used to abort the remaining unnecessary write beats. This method also works when in SDR mode. The most recently registered read or write command prior to the burst terminate command will be truncated. The active page remains open.

18.5.1.5 Precharge All Banks (PALL) and Selected Bank (PRE) Commands

The Precharge command puts SDRAM into an idle state. The SDRAM must be in this idle state before a REF, LMR, LEMR, or ACTV command to open a new row within a particular bank can be issued.

The memory controller issues the precharge command only when necessary for one of the following conditions:

- Access to a new row
- Refresh interval elapsed
- Software commanded precharge during device initialization

NOTE

DRAMs also have a maximum bank open period, after which a precharge is required. The memory controller does not time the bank open period because the refresh interval is always less.

18.5.1.6 Load Mode/Extended Mode Register Command (LMR, LEMR)

All SDRAM devices contain mode registers that are used to configure the timing and burst mode for the SDRAM. These commands are used to access the mode registers that physically reside within the SDRAM devices. During the LMR or LEMR command the SDRAM will latch the address and bank buses and load the values into the selected mode register.

NOTE

The LMR and LEMR commands are only used during SDRAM initialization.

The following steps should be used to write the mode register and extended mode register:

1. Set the SDCR[MODE_EN] bit.
2. Write the SDMR[BA] bits to select the mode register.
3. Write the desired mode register value to the SDMR[ADDR]. Do not overwrite the SDMR[BA] values. This step can be performed in the same register write in step 2.
4. Set the SDMR[CMD] bit.
5. For DDR, step 2 to 4 should be performed twice. The first is for the extended mode register. The later is for the mode register.
6. Clear the SDCR[MODE_EN] bit.

18.5.1.6.1 Mode Register Definition

Figure 18-10 shows the mode register definition. Note that this is the SDRAM's mode register not the SDRAMC's mode/extended mode register (SDMR) defined in [Section 18.4.1, “SDRAM Mode/Extended Mode Register \(SDMR\).”](#) Please refer to the device data sheet for a detailed description.

Field	BA1	BA0	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
	0	0			OP_MODE				CL		BT		BLEN	

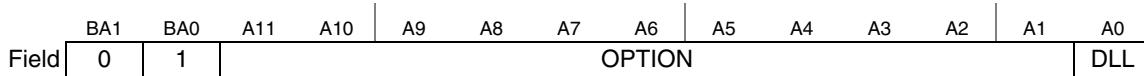
Figure 18-10. Mode Register

Table 18-13. Mode Register Field Descriptions

Field	Description
BA1-BA0	Bank Address. These must both be zero to select the mode register.
A11-A7 OP_MO DE	Operating Mode. xx00 Standard Operation (SDR only) 00000 Normal Operation (DDR) 00010 Reset DLL (DDR) Else Reserved
A6-A4 CL	CAS Latency. Delay in clocks from issuing a READ to valid data out. Check the SDRAM manufacturer's spec as the CL settings supported can vary from memory to memory.
A3 BT	Burst Type. 0 Sequential 1 Interleaved. This setting should not be used since the SDRAMC does not support interleaved bursts.
A2-A0 BLEN	Burst Length. Determines the number of column locations that are accessed for a given READ or WRITE command. 000 One. This setting is not valid for DDR. 001 Two 010 Four 011 Eight Else Reserved

18.5.1.6.2 Extended Mode Register Definition

Figure 18-11 shows the extended mode register used by DDR SDRAMs. Note that this is the SDRAM's extended mode register not the SDRAMC's mode/extended mode register (SDMR) defined in Section 18.4.1, “SDRAM Mode/Extended Mode Register (SDMR).” Please refer to the device data sheet for a detailed description.

**Figure 18-11. Extended Mode Register****Table 18-14. Extended Mode Register Field Descriptions**

Field	Description
BA1-BA0	Bank Address. These must be set to '01' to select the extended mode register.
A11-A1 OPTION	Option. These bits are not defined by the DDR specification. Each DDR SDRAM manufacturer can use these bits to implement optional features. Check with the SDRAM manufacturer to determine if any optional features have been implemented. For normal operation all bits should be cleared.
A0 DLL	Delay Locked Loop. Controls enabling of the delay locked loop circuitry used for DDR timing. 0 Enabled 1 Disabled

18.5.1.7 Auto-Refresh Command (REF)

The memory controller issues auto-refresh commands according to the SDCR[REF_CNT] value. Each time the programmed refresh interval elapses, the memory controller issues a PALL command followed by a REF command.

If a memory access is in progress at the time the refresh interval elapses, the memory controller schedules the refresh after the transfer is finished; but the interval timer continues counting so that the average refresh rate is constant.

After REF command, the SDRAM is in an idle state and waits for an ACTV command.

18.5.1.8 Self-Refresh (SREF) and Power Down (PDWN) Commands

The memory controller issues either a PDWN or a SREF command if the SDCR[CKE] bit is cleared. If the SDCR[REF_EN] bit is set when CKE is negated, the controller issues a SREF command; if the REF_EN bit is cleared, the controller issues a PDWN command. The REF_EN bit may be changed in the same register write that changes the CKE bit; the controller will act upon the new value of the REF_EN bit.

Just like an auto-refresh command, the controller automatically issues a PALL command before the self refresh command.

The memory is reactivated from power down or self refresh mode by setting the CKE bit.

If a normal refresh interval elapses while the memory is in self refresh mode, a PALL and REF will be performed as soon as the memory is reactivated. If the memory is put into and brought out of self refresh all within a single refresh interval, the next automatic refresh will occur on schedule.

In self refresh mode, the memory does not require an external clock. The SD_CLK can be stopped for maximum power savings. If the memory controller clock is stopped, the refresh interval timer must be reset before the memory is reactivated (if periodic refresh is to be resumed). The refresh interval timer is reset by clearing the REF_EN bit. This can be done at any time while the memory is in self refresh mode, before or after the memory controller clock is stopped/restarted, but *not* with the same control register write that clears CKE; this would put the memory in power down mode. To restart periodic refresh when the memory is reactivated, the REF_EN bit must be reassigned; this can be done before the memory is reactivated, or in the same control register write that sets CKE to exit self refresh mode.

18.5.2 Power-Up Initialization

SDRAMs have a prescribed initialization sequence. The following section details the memory initialization steps for DDR SDRAM. The sequence might change slightly from device-to-device. Refer to the device datasheet as the most relevant reference.

1. After reset is deactivated, pause for the amount of time indicated in the SDRAM specification. Usually 100 μ s or 200 μ s.
2. Configure the pin multiplex control for the shared $\overline{\text{SD_CS}}$ pins in the GPIO module.
3. Write the base address and mask registers (SDBAR0, SDBAR1, SDMR0, and SDMR1) to setup the address space for each chip-select.
4. Program the SDRAM configuration registers (SDCFG1 and SDCFG2) with the correct delay and timing values.
5. Issue a PALL command. Initialize the SDRAM control register (SDCR) with SDCR[IPALL] set. The SDCR[REF and IREF] bits should remain cleared for this step. Perform a single read/write to any of the SDRAM chip select address spaces to issue the command.

6. Initialize the SDRAM's extended mode register to enable the DLL. See [Section 18.5.1.6, “Load Mode/Extended Mode Register Command \(LMR, LEMR\),”](#) for instructions on issuing an LEMR command.
7. Initialize the SDRAM's mode register and reset the DLL using the LMR command. See [Section 18.5.1.6, “Load Mode/Extended Mode Register Command \(LMR, LEMR\),”](#) for more instruction on issuing an LMR command. During this step the OP_MODE field of the mode register should be set to normal operation/reset DLL.
8. Pause for the DLL lock time specified by the memory.
9. Issue a second PALL command. Initialize the SDRAM control register (SDCR) with SDCR[IPALL] set. The SDCR[REF and IREF] bits should remain cleared for this step. Perform a single read/write to any of the SDRAM chip select address spaces to issue the command.
10. Refresh the SDRAM. The SDRAM spec should indicate a number of refresh cycles to be performed before issuing an LMR command. Write to the SDCR with the IREF bit set (SDCR[REF and IPALL] should be cleared). This will force a refresh of the SDRAM each time the IREF bit is set. Repeat this step until the specified number of refresh cycles have been completed.
11. Initialize the SDRAM's mode register using the LMR command. See [Section 18.5.1.6, “Load Mode/Extended Mode Register Command \(LMR, LEMR\),”](#) for more instruction on issuing an LMR command. During this step the OP_MODE field of the mode register should be set to normal operation.
12. Set SDCR[REF] to enable automatic refreshing, and clear SDCR[MODE_EN] to lock the SDMR. SDCR[IREF and IPALL] remain cleared.

18.5.3 Page Management

SDRAM devices have four internal banks. A particular row and bank of memory must be activated to allow read and write accesses. The SDRAM controller supports paging mode to maximize the memory access throughput. During operation, the SDRAM controller maintains an open page for each SD_CS block. An open page is composed of the active rows in the internal banks. Each internal bank has its own active row.

The physical page size of a SD_CS block is equal to the space size divided by the number of rows; but the page may not be contiguous in the internal address space because SDRAMs can have a different row address open in each bank and the internal address bits (A[27:24] and A[9:2]) or (A[27:24] and A[9:1]) used for memory column addresses are not consecutive.

Because the column address may be split across two portions of the internal address, the contiguous page size is (number of contiguous columns per bank) × (number of bits). This gives a contiguous page size of 1 KBytes. However, the total (possibly fragmented) page size is (number of banks) × (number of columns) × (number of bits).

If a new access does not fall in the open row of an open bank of a SD_CS block, the open row must be closed (PRE) and the new row must be opened (ACTV), then the READ or WRITE command can proceed. An ACTV command only activates one bank at one time. If another read or write falls in an inactive bank, another ACTV is needed but no precharge is needed. If a read or write falls in any open row of any active banks of a page, no PRE or ACTV is needed; the read or write command can be issued immediately.

A page is kept open until one of the following conditions occurs:

- An access outside the open page.
- A refresh cycle is started.

All $\overline{SD_CS}$ blocks are refreshed at the same time. The refresh closes all banks of every SDRAM block.

18.5.4 Transfer Size

In the SDRAMC, the internal data bus is 32 bits wide, while the SDRAM external interface bus is either 32 or 16 bits wide. Therefore, each internal data beat requires one or two memory data beats. The SDRAM controller manages the size translation (packing/unpacking) between internal and external DRAM buses.

The burst size is the processor standard 16 bytes: Four beats of 4 bytes on the internal bus, four beats of 4 bytes (32-bit mode), or eight beats of 2 bytes (16-bit mode) on the memory bus. The SDRAM controller follows the critical beat first, sequential transfer format required.

The burst size and transfer order must be programmed in the SDRAM Mode registers during initialization; the burst size also must be programmed in the memory controller (SDCFG2 register).

Chapter 19

Fast Ethernet Controller (FEC)

19.1 Introduction

This chapter provides a feature-set overview, a functional block diagram, and transceiver connection information for both the 10 and 100 Mbps MII (Media Independent Interface), as well as the 7-wire serial interface. Additionally, detailed descriptions of operation and the programming model are included.

19.1.1 Overview

The Ethernet media access controller (MAC) is designed to support both 10 and 100 Mbps Ethernet/IEEE 802.3 networks. An external transceiver interface and transceiver function are required to complete the interface to the media. The FEC supports three different standard MAC-PHY (physical) interfaces for connection to an external Ethernet transceiver. The FEC supports the 10/100 Mbps MII and the 10 Mbps-only 7-wire interface, which uses a subset of the MII pins.

NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to [Chapter 13, “General Purpose I/O Module”](#)) prior to configuring the FEC.

19.1.2 Block Diagram

The block diagram of the FEC is shown below. The FEC is implemented with a combination of hardware and microcode. The off-chip (Ethernet) interfaces are compliant with industry and IEEE 802.3 standards.

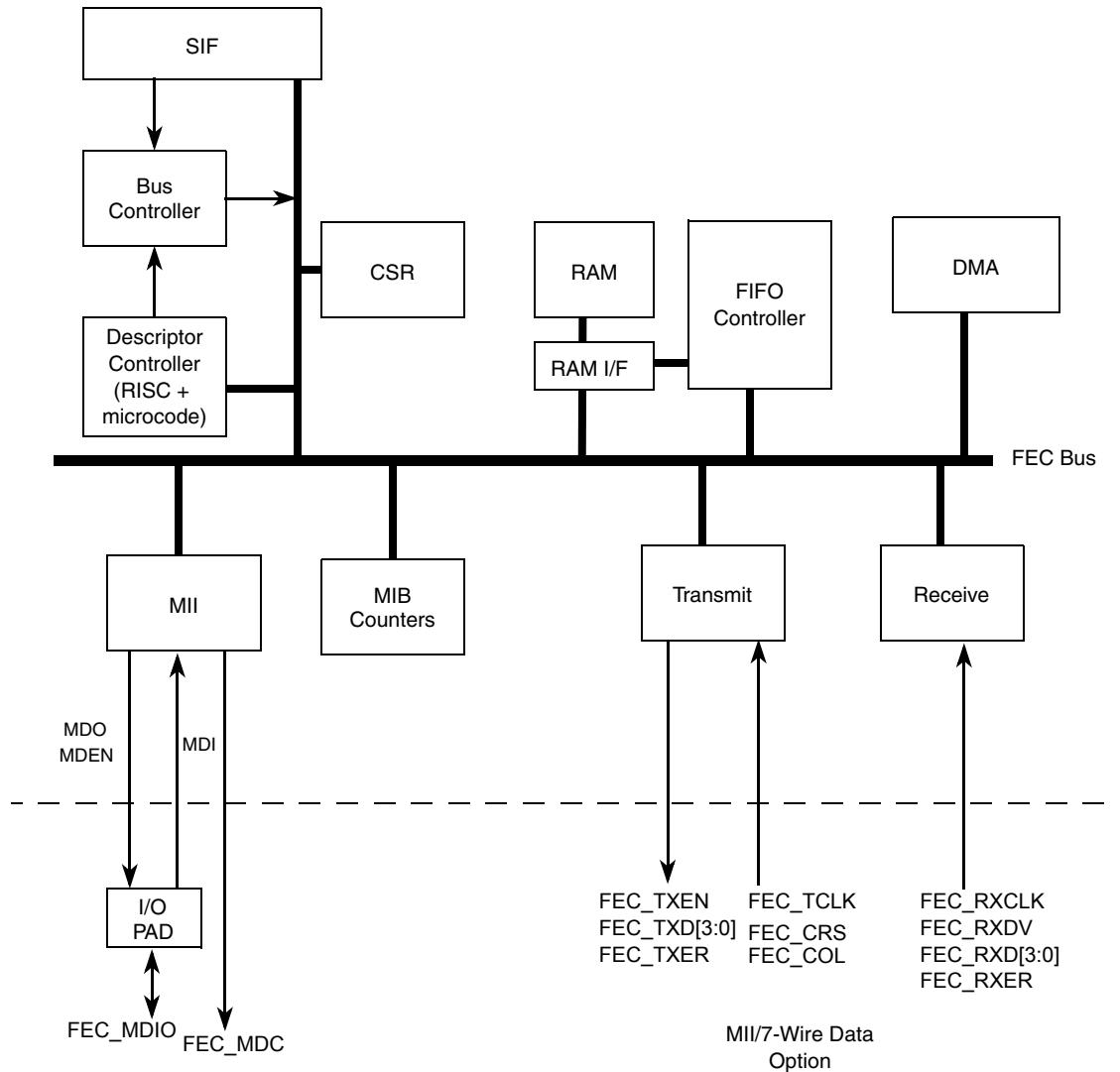


Figure 19-1. FEC Block Diagram

The descriptor controller is a RISC-based controller that provides the following functions in the FEC:

- Initialization (those internal registers not initialized by the user or hardware)
- High level control of the DMA channels (initiating DMA transfers)
- Interpreting buffer descriptors
- Address recognition for receive frames
- Random number generation for transmit collision backoff timer

NOTE

DMA references in this section refer to the FEC’s DMA engine. This DMA engine is for the transfer of FEC data only, and is not related to the DMA controller described in [Chapter 16, “Enhanced Direct Memory Access \(eDMA\)”](#) nor to the DMA timers described in [Chapter 22, “DMA Timers \(DTIM0–DTIM3\)”](#).

The RAM is the focal point of all data flow in the fast Ethernet controller and is divided into transmit and receive FIFOs. The FIFO boundaries are programmable using the FRSR register. User data flows to/from the DMA block from/to the receive/transmit FIFOs. Transmit data flows from the transmit FIFO into the transmit block and receive data flows from the receive block into the receive FIFO.

The user controls the FEC by writing, through the SIF (slave interface) module, into control registers located in each block. The CSR (control and status register) block provides global control (e.g. Ethernet reset and enable) and interrupt handling registers.

The MII block provides a serial channel for control/status communication with the external physical layer device (transceiver). This serial channel consists of the FEC_MDC (management data clock) and FEC_MDIO (management data input/output) lines of the MII interface.

The DMA block provides multiple channels allowing transmit data, transmit descriptor, receive data and receive descriptor accesses to run independently.

The transmit and receive blocks provide the Ethernet MAC functionality (with some assist from microcode).

The message information block (MIB) maintains counters for a variety of network events and statistics. It is not necessary for operation of the FEC but provides valuable counters for network management. The counters supported are the RMON (RFC 1757) Ethernet Statistics group and some of the IEEE 802.3 counters. See [Section 19.2.3, “MIB Block Counters Memory Map”](#) for more information.

19.1.3 Features

The FEC incorporates the following features:

- Support for three different Ethernet physical interfaces:
 - 100-Mbps IEEE 802.3 MII
 - 10-Mbps IEEE 802.3 MII
 - 10-Mbps 7-wire interface (industry standard)
- IEEE 802.3 full duplex flow control
- Programmable max frame length supports IEEE 802.1 VLAN tags and priority
- Support for full-duplex operation (200 Mbps throughput) with a minimum internal bus clock rate of 50 MHz
- Support for half-duplex operation (100 Mbps throughput) with a minimum internal bus clock rate of 50 MHz
- Retransmission from transmit FIFO following a collision (no processor bus utilization)

- Automatic internal flushing of the receive FIFO for runts (collision fragments) and address recognition rejects (no processor bus utilization)
- Address recognition
 - Frames with broadcast address may be always accepted or always rejected
 - Exact match for single 48-bit individual (unicast) address
 - Hash (64-bit hash) check of individual (unicast) addresses
 - Hash (64-bit hash) check of group (multicast) addresses
 - Promiscuous mode

19.1.4 Modes of Operation

The primary operational modes are described in this section.

19.1.4.1 Full and Half Duplex Operation

Full duplex mode is intended for use on point to point links between switches or end node to switch. Half duplex mode is used in connections between an end node and a repeater or between repeaters. Selection of the duplex mode is controlled by TCR[FDEN].

When configured for full duplex mode, flow control may be enabled. Refer to the TCR[RFC_PAUSE] and TCR[TFC_PAUSE] bits, the RCR[FCE] bit, and [Section 19.3.10, “Full Duplex Flow Control,”](#) for more details.

19.1.5 Interface Options

The following interface options are supported. A detailed discussion of the interface configurations is provided in [Section 19.3.5, “Network Interface Options”](#).

19.1.5.1 10 Mbps and 100 Mbps MII Interface

MII is the Media Independent Interface defined by the IEEE 802.3 standard for 10/100 Mbps operation. The MAC-PHY interface may be configured to operate in MII mode by setting RCR[MII_MODE].

The speed of operation is determined by the FEC_TXCLK and FEC_RXCLK pins which are driven by the external transceiver. The transceiver will either auto-negotiate the speed or it may be controlled by software via the serial management interface (FEC_MDC/FEC_MDIO pins) to the transceiver. Refer to the MMFR and MSCR register descriptions, as well as the section on the MII, for a description of how to read and write registers in the transceiver via this interface.

19.1.5.2 10 Mbps 7-Wire Interface Operation

The FEC supports a 7-wire interface as used by many 10 Mbps Ethernet transceivers. The RCR[MII_MODE] bit controls this functionality. If this bit is cleared, the MII mode is disabled and the 10 Mbps, 7-wire mode is enabled.

19.1.6 Address Recognition Options

The address options supported are promiscuous, broadcast reject, individual address (hash or exact match), and multicast hash match. Address recognition options are discussed in detail in [Section 19.3.8, “Ethernet Address Recognition.”](#)

19.1.7 Internal Loopback

Internal loopback mode is selected via RCR[LOOP]. Loopback mode is discussed in detail in [Section 19.3.13, “Internal and External Loopback.”](#)

19.2 Memory Map/Register Definition

This section gives an overview of the registers, followed by a description of the buffers.

The FEC is programmed by a combination of control/status registers (CSRs) and buffer descriptors. The CSRs are used for mode control and to extract global status information. The descriptors are used to pass data buffers and related buffer information between the hardware and software.

19.2.1 High-Level Module Memory Map

The FEC implementation requires a 1-Kbyte memory map space. This is divided into 2 sections of 512 bytes each. The first is used for control/status registers. The second contains event/statistic counters held in the MIB block. [Table 19-1](#) defines the top level memory map.

Table 19-1. Module Memory Map

Address	Function
0xFC03_0000–FC03_01FF	Control/Status Registers
0xFC03_0200–FC03_03FF	MIB Block Counters

19.2.2 Register Memory Map

[Table 19-2](#) shows the FEC register memory map with each register address, name, and a brief description.

Table 19-2. FEC Register Memory Map

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC03_0004	Interrupt Event Register (EIR)	32	R/W	0x0000_0000	19.2.4.1/19-8
0xFC03_0008	Interrupt Mask Register (EIMR)	32	R/W	0x0000_0000	19.2.4.2/19-10
0xFC03_0010	Receive Descriptor Active Register (RDAR)	32	R/W	0x0000_0000	19.2.4.3/19-11
0xFC03_0014	Transmit Descriptor Active Register (TDAR)	32	R/W	0x0000_0000	19.2.4.4/19-11
0xFC03_0024	Ethernet Control Register (ECR)	32	R/W	0x0000_0000	19.2.4.5/19-12
0xFC03_0040	MII Management Frame Register (MMFR)	32	R/W	Undefined	19.2.4.6/19-13
0xFC03_0044	MII Speed Control Register (MSCR)	32	R/W	0x0000_0000	19.2.4.7/19-14

Table 19-2. FEC Register Memory Map (continued)

Address	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC03_0064	MIB Control/Status Register (MIBC)	32	R/W	0x0000_0000	19.2.4.8/19-15
0xFC03_0084	Receive Control Register (RCR)	32	R/W	0x05EE_0001	19.2.4.9/19-15
0xFC03_00C4	Transmit Control Register (TCR)	32	R/W	0x0000_0000	19.2.4.10/19-16
0xFC03_00E4	Physical Address Low Register (PALR)	32	R/W	Undefined	19.2.4.11/19-17
0xFC03_00E8	Physical Address High Register (PAUR)	32	R/W	See Section	19.2.4.12/19-18
0xFC03_00EC	Opcode/Pause Duration (OPD)	32	R/W	See Section	19.2.4.13/19-18
0xFC03_0118	Descriptor Individual Upper Address Register (IAUR)	32	R/W	Undefined	19.2.4.14/19-19
0xFC03_011C	Descriptor Individual Lower Address Register (IALR)	32	R/W	Undefined	19.2.4.15/19-19
0xFC03_0120	Descriptor Group Upper Address Register (GAUR)	32	R/W	Undefined	19.2.4.16/19-20
0xFC03_0124	Descriptor Group Upper Address Register (GALR)	32	R/W	Undefined	19.2.4.17/19-20
0xFC03_0144	Transmit FIFO Watermark (TFWR)	32	R/W	0x0000_0001	19.2.4.18/19-21
0xFC03_014C	FIFO Receive Bound Register (FRBR)	32	R	0x0000_0600	19.2.4.19/19-21
0xFC03_0150	FIFO Receive FIFO Start Register (FRSR)	32	R	0x0000_0500	19.2.4.20/19-22
0xFC03_0180	Pointer to Receive Descriptor Ring (ERDSR)	32	R/W	Undefined	19.2.4.21/19-22
0xFC03_0184	Pointer to Transmit Descriptor Ring (ETDSR)	32	R/W	Undefined	19.2.4.22/19-23
0xFC03_0188	Maximum Receive Buffer Size (EMRBR)	32	R/W	Undefined	19.2.4.23/19-23

19.2.3 MIB Block Counters Memory Map

Table 19-3 defines the MIB counters memory map which defines the locations in the MIB RAM space where hardware maintained counters reside. These fall in the 0xFC03_0200-0xFC03_03FF address offset range. The counters are divided into two groups.

RMON counters are included which cover the Ethernet statistics counters defined in RFC 1757. In addition to the counters defined in the Ethernet statistics group, a counter is included to count truncated frames as the FEC only supports frame lengths up to 2032 bytes. The RMON counters are implemented independently for transmit and receive to insure accurate network statistics when operating in full duplex mode.

IEEE counters are included which support the mandatory and recommended counter packages defined in Section 5 of ANSI/IEEE Std. 802.3 (1998 edition). The IEEE Basic Package objects are supported by the FEC but do not require counters in the MIB block. In addition, some of the recommended package objects which are supported do not require MIB counters. Counters for transmit and receive full duplex flow control frames are included as well.

Table 19-3. MIB Counters Memory Map

Address	Register
0xFC03_0200	Count of frames not counted correctly (RMON_T_DROP)
0xFC03_0204	RMON Tx packet count (RMON_T_PACKETS)
0xFC03_0208	RMON Tx Broadcast Packets (RMON_T_BC_PKT)
0xFC03_020C	RMON Tx Multicast Packets (RMON_T_MC_PKT)
0xFC03_0210	RMON Tx Packets w CRC/Align error (RMON_T_CRC_ALIGN)
0xFC03_0214	RMON Tx Packets < 64 bytes, good CRC (RMON_T_UNDERSIZE)
0xFC03_0218	RMON Tx Packets > MAX_FL bytes, good CRC (RMON_T_OVERSIZE)
0xFC03_021C	RMON Tx Packets < 64 bytes, bad CRC (RMON_T_FRAG)
0xFC03_0220	RMON Tx Packets > MAX_FL bytes, bad CRC (RMON_T_JAB)
0xFC03_0224	RMON Tx collision count (RMON_T_COL)
0xFC03_0228	RMON Tx 64 byte packets (RMON_T_P64)
0xFC03_022C	RMON Tx 65 to 127 byte packets (RMON_T_P65TO127)
0xFC03_0230	RMON Tx 128 to 255 byte packets (RMON_T_P128TO255)
0xFC03_0234	RMON Tx 256 to 511 byte packets (RMON_T_P256TO511)
0xFC03_0238	RMON Tx 512 to 1023 byte packets (RMON_T_P512TO1023)
0xFC03_023C	RMON Tx 1024 to 2047 byte packets (RMON_T_P1024TO2047)
0xFC03_0240	RMON Tx packets w > 2048 bytes (RMON_T_P_GTE2048)
0xFC03_0244	RMON Tx Octets (RMON_T_OCTETS)
0xFC03_0248	Count of frames not counted correctly (IEEE_T_DROP)
0xFC03_024C	Frames Transmitted OK (IEEE_T_FRAME_OK)
0xFC03_0250	Frames Transmitted with Single Collision (IEEE_T_1COL)
0xFC03_0254	Frames Transmitted with Multiple Collisions (IEEE_T_MCOL)
0xFC03_0258	Frames Transmitted after Deferral Delay (IEEE_T_DEF)
0xFC03_025C	Frames Transmitted with Late Collision (IEEE_T_LCOL)
0xFC03_0260	Frames Transmitted with Excessive Collisions (IEEE_T_EXCOL)
0xFC03_0264	Frames Transmitted with Tx FIFO Underrun (IEEE_T_MACERR)
0xFC03_0268	Frames Transmitted with Carrier Sense Error (IEEE_T_CSERR)
0xFC03_026C	Frames Transmitted with SQE Error (IEEE_T_SQE)
0xFC03_0270	Flow Control Pause frames transmitted (IEEE_T_FDXFC)
0xFC03_0274	Octet count for Frames Transmitted w/o Error (IEEE_T_OCTETS_OK)
0xFC03_0284	RMON Rx packet count (RMON_R_PACKETS)
0xFC03_0288	RMON Rx Broadcast Packets (RMON_R_BC_PKT)
0xFC03_028C	RMON Rx Multicast Packets (RMON_R_MC_PKT)

Table 19-3. MIB Counters Memory Map (continued)

Address	Register
0xFC03_0290	RMON Rx Packets w CRC/Align error (RMON_R_CRC_ALIGN)
0xFC03_0294	RMON Rx Packets < 64 bytes, good CRC (RMON_R_UNDERSIZE)
0xFC03_0298	RMON Rx Packets > MAX_FL bytes, good CRC (RMON_R_OVERSIZE)
0xFC03_029C	RMON Rx Packets < 64 bytes, bad CRC (RMON_R_FRAG)
0xFC03_02A0	RMON Rx Packets > MAX_FL bytes, bad CRC (RMON_R_FRAG)
0xFC03_02A4	Reserved (RMON_R_RESVD_0)
0xFC03_02A8	RMON Rx 64 byte packets (RMON_R_P64)
0xFC03_02AC	RMON Rx 65 to 127 byte packets (RMON_R_P65TO127)
0xFC03_02B0	RMON Rx 128 to 255 byte packets (RMON_R_P128TO255)
0xFC03_02B4	RMON Rx 256 to 511 byte packets (RMON_R_P256TO511)
0xFC03_02B8	RMON Rx 512 to 1023 byte packets (RMON_R_P512TO1023)
0xFC03_02BC	RMON Rx 1024 to 2047 byte packets (RMON_R_P1024TO2047)
0xFC03_02C0	RMON Rx packets w > 2048 bytes (RMON_R_P_GTE2048)
0xFC03_02C4	RMON Rx Octets (RMON_R_OCTETS)
0xFC03_02C8	Count of frames not counted correctly (IEEE_R_DROP)
0xFC03_02CC	Frames Received OK (IEEE_R_FRAME_OK)
0xFC03_02D0	Frames Received with CRC Error (IEEE_R_CRC)
0xFC03_02D4	Frames Received with Alignment Error (IEEE_R_ALIGN)
0xFC03_02D8	Receive Fifo Overflow count (IEEE_R_MACERR)
0xFC03_02DC	Flow Control Pause frames received (IEEE_R_FDXFC)
0xFC03_02E0	Octet count for Frames Rcvd w/o Error (IEEE_R_OCTETS_OK)

19.2.4 Register Description

The following sections describe each register in detail.

19.2.4.1 Ethernet Interrupt Event Register (EIR)

When an event occurs that sets a bit in the EIR, an interrupt will be generated if the corresponding bit in the interrupt mask register (EIMR) is also set. The bit in the EIR is cleared if a one is written to that bit position; writing zero has no effect. This register is cleared upon hardware reset.

These interrupts can be divided into operational interrupts, transceiver/network error interrupts, and internal error interrupts. Interrupts which may occur in normal operation are GRA, TXF, TXB, RXF, RXB, and MII. Interrupts resulting from errors/problems detected in the network or transceiver are HBERR, BABR, BABT, LC, and RL. Interrupts resulting from internal errors are HBERR and UN.

Some of the error interrupts are independently counted in the MIB block counters:

- HBERR - IEEE_T_SQE
- BABR - RMON_R_OVERSIZE (good CRC), RMON_R_JAB (bad CRC)
- BABT - RMON_T_OVERSIZE (good CRC), RMON_T_JAB (bad CRC)
- LATE_COL - IEEE_T_LCOL
- COL_RETRY_LIM - IEEE_T_EXCOL
- XFIFO_UN - IEEE_T_MACERR

Software may choose to mask off these interrupts since these errors will be visible to network management via the MIB counters.

Address: 0xFC03_0004

Access: User read/write

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	HB ERR	BABR	BABT	GRA	TXF	TXB	RXF	RXB	MII	EB ERR	LC	RL	UN	0	0	0
W	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
	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 19-2. Ethernet Interrupt Event Register (EIR)

Table 19-4. EIR Field Descriptions

Field	Description
31 HBERR	Heartbeat error. This interrupt indicates that HBC is set in the TCR register and that the COL input was not asserted within the Heartbeat window following a transmission.
30 BABR	Babbling receive error. This bit indicates a frame was received with length in excess of RCR[MAX_FL] bytes.
29 BABT	Babbling transmit error. This bit indicates that the transmitted frame length has exceeded RCR[MAX_FL] bytes. This condition is usually caused by a frame that is too long being placed into the transmit data buffer(s). Truncation does not occur.
28 GRA	Graceful stop complete. This interrupt will be asserted for one of three reasons. Graceful stop means that the transmitter is put into a pause state after completion of the frame currently being transmitted. 1) A graceful stop, which was initiated by the setting of the TCR[GTS] bit is now complete. 2) A graceful stop, which was initiated by the setting of the TCR[TFC_PAUSE] bit is now complete. 3) A graceful stop, which was initiated by the reception of a valid full duplex flow control “pause” frame is now complete. Refer to Section 19.3.10, “Full Duplex Flow Control
27 TXF	Transmit frame interrupt. This bit indicates that a frame has been transmitted and that the last corresponding buffer descriptor has been updated.
26 TXB	Transmit buffer interrupt. This bit indicates that a transmit buffer descriptor has been updated.
25 RXF	Receive frame interrupt. This bit indicates that a frame has been received and that the last corresponding buffer descriptor has been updated.

Table 19-4. EIR Field Descriptions (continued)

Field	Description
24 RXB	Receive buffer interrupt. This bit indicates that a receive buffer descriptor has been updated that was not the last in the frame.
23 MII	MII interrupt. This bit indicates that the MII has completed the data transfer requested.
22 EBERR	Ethernet bus error. This bit indicates that a system bus error occurred when a DMA transaction was underway. When the EBERR bit is set, ECR[ETHER_EN] will be cleared, halting frame processing by the FEC. When this occurs software will need to insure that the FIFO controller and DMA are also soft reset.
21 LC	Late collision. This bit indicates that a collision occurred beyond the collision window (slot time) in half duplex mode. The frame is truncated with a bad CRC and the remainder of the frame is discarded.
20 RL	Collision retry limit. This bit indicates that 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 will commence. This error can only occur in half duplex mode.
19 UN	Transmit FIFO underrun. This bit indicates that 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.
18-0	Reserved, should be cleared.

19.2.4.2 Interrupt Mask Register (EIMR)

The EIMR register controls which interrupt events are allowed to generate actual interrupts. All implemented bits in this CSR are read/write. This register is cleared upon a hardware reset. If the corresponding bits in both the EIR and EIMR registers are set, the interrupt will be signalled to the interrupt controller. The interrupt signal will remain asserted until a 1 is written to the EIR bit (write 1 to clear) or a 0 is written to the EIMR bit.

Address: 0xFC03_0008

Access: User read/write

R				W				R				W				
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
R	HB ERR	BABR	BABT	GRA	TXF	TXB	RXF	RXB	MII	EB ERR	LC	RL	UN	0	0	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
	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 19-3. Interrupt Mask Register (EIMR)

Table 19-5. EIMR Field Descriptions

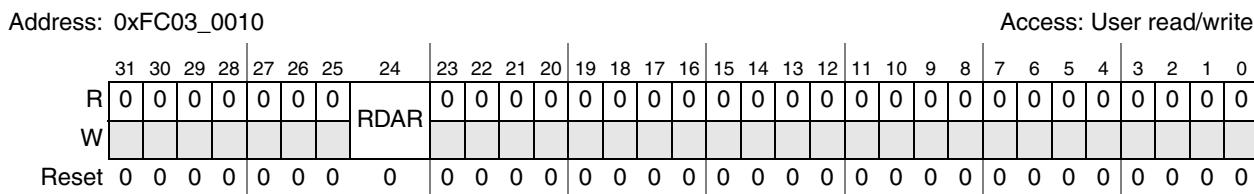
Field	Description
31–19 See Figure 19-3 and Table 19-4	Interrupt mask. Each bit corresponds to an interrupt source defined by the EIR register. The corresponding EIMR bit determines whether an interrupt condition can generate an interrupt. At every processor clock, the EIR samples the signal generated by the interrupting source. The corresponding EIR bit reflects the state of the interrupt signal even if the corresponding EIMR bit is set. 0 The corresponding interrupt source is masked. 1 The corresponding interrupt source is not masked.
18–0	Reserved, should be cleared.

19.2.4.3 Receive Descriptor Active Register (RDAR)

RDAR is a command register, written by the user, that indicates that the receive descriptor ring has been updated (empty receive buffers have been produced by the driver with the empty bit set).

Whenever the register is written, the RDAR bit is set. This is independent of the data actually written by the user. When set, the FEC will poll the receive descriptor ring and process receive frames (provided ECR[ETHER_EN] is also set). Once the FEC polls a receive descriptor whose empty bit is not set, then the FEC will clear the RDAR bit and cease receive descriptor ring polling until the user sets the bit again, signifying that additional descriptors have been placed into the receive descriptor ring.

The RDAR register is cleared at reset and when ECR[ETHER_EN] is cleared.

**Figure 19-4. Receive Descriptor Active Register (RDAR)****Table 19-6. RDAR Field Descriptions**

Field	Description
31–25	Reserved, should be cleared.
24 RDAR	Set to one when this register is written, regardless of the value written. Cleared by the FEC device whenever no additional “empty” descriptors remain in the receive ring. Also cleared when ECR[ETHER_EN] is cleared.
23–0	Reserved, should be cleared.

19.2.4.4 Transmit Descriptor Active Register (TDAR)

The TDAR is a command register which should be written by the user to indicate that the transmit descriptor ring has been updated (transmit buffers have been produced by the driver with the ready bit set in the buffer descriptor).

Whenever the register is written, the TDAR bit is set. This value is independent of the data actually written by the user. When set, the FEC will poll the transmit descriptor ring and process transmit frames (provided ECR[ETHER_EN] is also set). Once the FEC polls a transmit descriptor whose ready bit is not set, then

the FEC will clear the TDAR bit and cease transmit descriptor ring polling until the user sets the bit again, signifying additional descriptors have been placed into the transmit descriptor ring.

The TDAR register is cleared at reset, when ECR[ETHER_EN] is cleared, or when ECR[RESET] is set.

Address: 0xFC03_0014																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	0	0	0	0	0	0	0	0	TDAR	0	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	0	0	0	0	0	

Figure 19-5. Transmit Descriptor Active Register (TDAR)

Table 19-7. TDAR Field Descriptions

Field	Description																														
31–25	Reserved, should be cleared.																														
24 TDAR	Set to one when this register is written, regardless of the value written. Cleared by the FEC device whenever no additional ready descriptors remain in the transmit ring. Also cleared when ECR[ETHER_EN] is cleared.																														
23–0	Reserved, should be cleared.																														

19.2.4.5 Ethernet Control Register (ECR)

ECR is a read/write user register, though both fields in this register may be altered by hardware as well. The ECR is used to enable/disable the FEC.

Address: 0xFC03_0024																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	0	0	0	0	0	0	0	0	0	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	0	0	0	0	0

Figure 19-6. Ethernet Control Register (ECR)

Table 19-8. ECR Field Descriptions

Field	Description																														
31–2	Reserved, should be cleared.																														
1 ETHER_EN	When this bit is set, the FEC is enabled, and reception and transmission are possible. When this bit is cleared, reception is immediately stopped and transmission is stopped after a bad CRC is appended to any currently transmitted frame. The buffer descriptor(s) for an aborted transmit frame are not updated after clearing this bit. When ETHER_EN is cleared, the DMA, buffer descriptor, and FIFO control logic are reset, including the buffer descriptor and FIFO pointers. The ETHER_EN bit is altered by hardware under the following conditions:																														
• ECR[RESET] is set by software, in which case ETHER_EN will be cleared																															
0 RESET	When this bit is set, the equivalent of a hardware reset is performed but it is local to the FEC. ETHER_EN is cleared and all other FEC registers take their reset values. Also, any transmission/reception currently in progress is abruptly aborted. This bit is automatically cleared by hardware during the reset sequence. The reset sequence takes approximately 8 internal bus clock cycles after RESET is set.																														

19.2.4.6 MII Management Frame Register (MMFR)

The MMFR is accessed by the user and does not reset to a defined value. The MMFR register is used to communicate with the attached MII compatible PHY device(s), providing read/write access to their MII registers. Performing a write to the MMFR will cause a management frame to be sourced unless the MSCR has been programmed to 0. In the case of writing to MMFR when MSCR = 0, if the MSCR register is then written to a non-zero value, an MII frame will be generated with the data previously written to the MMFR. This allows MMFR and MSCR to be programmed in either order if MSCR is currently zero.

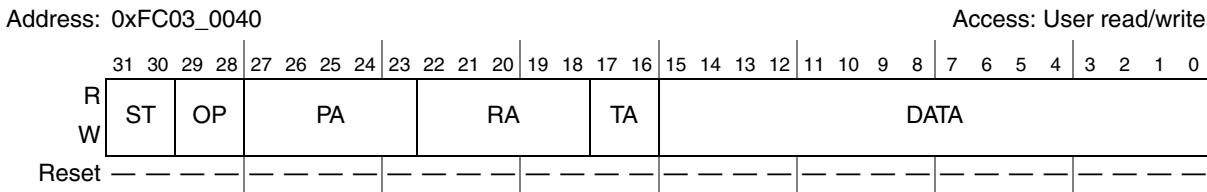


Figure 19-7. MII Management Frame Register (MMFR)

Table 19-9. MMFR Field Descriptions

Field	Description
31–30 ST	Start of frame delimiter. These bits must be programmed to 0b01 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. This field specifies one of up to 32 attached PHY devices.
22–18 RA	Register address. This field 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.

To perform a read or write operation on the MII Management Interface, the MMFR register must be written by the user. To generate a valid read or write management frame, the ST field must be written with a 01 pattern, and the TA field must be written with a 10. If other patterns are written to these fields, a frame will be generated but will not comply with the IEEE 802.3 MII definition.

To generate an IEEE 802.3-compliant MII Management Interface write frame (write to a PHY register), the user must write {01 01 PHYAD REGAD 10 DATA} to the MMFR register. Writing this pattern will cause the control logic to shift out the data in the MMFR register following a preamble generated by the control state machine. During this time the contents of the MMFR register will be altered as the contents are serially shifted and will be unpredictable if read by the user. Once the write management frame operation has completed, the MII interrupt will be generated. At this time the contents of the MMFR register will match the original value written.

To generate an MII Management Interface read frame (read a PHY register), the user must write {01 10 PHYAD REGAD 10 XXXX} to the MMFR register (the content of the DATA field is a don't care). Writing this pattern will cause the control logic to shift out the data in the MMFR register following a preamble generated by the control state machine. During this time the contents of the MMFR register will be altered as the contents are serially shifted, and will be unpredictable if read by the user. Once the read management frame operation has completed, the MII interrupt will be generated. At this time the contents of the MMFR register will match the original value written except for the DATA field whose contents have been replaced by the value read from the PHY register.

If the MMFR register is written while frame generation is in progress, the frame contents will be altered. Software should use the MII_STATUS register and/or the MII interrupt to avoid writing to the MMFR register while frame generation is in progress.

19.2.4.7 MII Speed Control Register (MSCR)

The MSCR provides control of the MII clock (FEC_MDC pin) frequency and allows a preamble drop on the MII management frameFEC_.

Address: 0xFC03_0044																																	
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																									DIS_PRE	MII_SPEED							
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 19-8. MII Speed Control Register (MSCR)

Table 19-10. MSCR Field Descriptions

Field	Description
31–8	Reserved, should be cleared.
7 DIS_PRE	Asserting this bit will cause preamble (32 1's) not to be prepended to the MII management frame. The MII standard allows the preamble to be dropped if the attached PHY device(s) does not require it.
6–1 MII_SPEED	MII_SPEED controls the frequency of the MII management interface clock (FEC_MDC) relative to the internal bus clock. A value of 0 in this field will "turn off" the FEC_MDC and leave it in low voltage state. Any non-zero value will result in the FEC_MDC frequency of $1/(MII_SPEED \times 2)$ of the internal bus frequency.
0	Reserved, should be cleared.

The MII_SPEED field must be programmed with a value to provide an FEC_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 in order to source a read or write management frame. After the management frame is complete, the MSCR register may optionally be set to zero to turn off the FEC_MDC. The FEC_MDC generated will have a 50% duty cycle except when MII_SPEED is changed during operation (change will take effect following either a rising or falling edge of FEC_MDC).

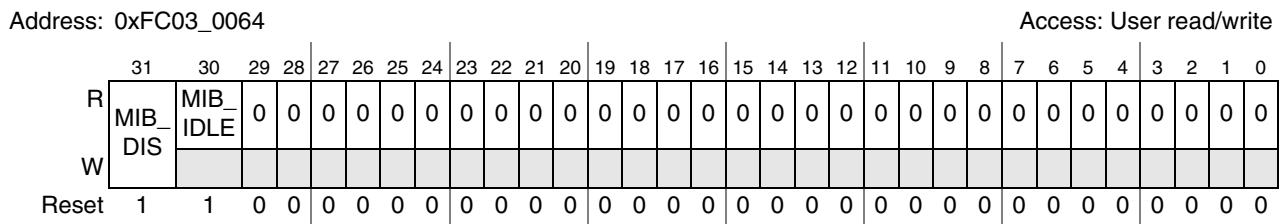
If the internal bus clock is 25 MHz, programming this register to 0x0000_0005 will result in an FEC_MDC frequency of $25\text{ MHz} \times 1/(5 \times 2) = 2.5\text{ MHz}$. A table showing optimum values for MII_SPEED as a function of internal bus clock frequency is provided below.

Table 19-11. Programming Examples for MSCR

internal Bus Clock Frequency	MSCR[MII_SPEED]	FEC_MDC frequency
25 MHz	0x5	2.50 MHz
33 MHz	0x7	2.36 MHz
40 MHz	0x8	2.50 MHz
50 MHz	0xA	2.50 MHz
66 MHz	0xE	2.36 MHz

19.2.4.8 MIB Control Register (MIBC)

The MIBC is a read/write register that controls and observes the state of the MIB block. This register is accessed by user software if there is a need to disable the MIB block operation. For example, in order to clear all MIB counters in RAM the user should disable the MIB block, then clear all the MIB RAM locations, then enable the MIB block. The MIB_DIS bit is reset to 1. See [Table 19-3](#) for the locations of the MIB counters.

**Figure 19-9. MIB Control Register (MIBC)****Table 19-12. MIBC Field Descriptions**

Field	Description
31 MIB_DIS	A read/write control bit. If set, the MIB logic will halt 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.
29–0	Reserved.

19.2.4.9 Receive Control Register (RCR)

The RCR, programmed by the user, controls the operational mode of the receive block and should be written only when ECR[ETHER_EN] = 0 (initialization time).

Address: 0xFC03_0084 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	MAX_FL										
W																
Reset	0	0	0	0	0	1	0	1	1	1	1	0	1	1	1	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	FCE	BC_REJ	PROM	MII_MODE	DRT	LOOP
W													0	0	0	1
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 19-10. Receive Control Register (RCR)

Table 19-13. RCR Field Descriptions

Field	Description
31–27	Reserved, should be cleared.
26–16 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 will cause the BABT interrupt to occur. Receive Frames longer than MAX_FL will cause the BABR interrupt to occur and will set the LG bit in the end of frame receive buffer descriptor. The recommended default value to be programmed by the user is 1518 or 1522 (if VLAN Tags are supported).
15–6	Reserved, should be cleared.
5 FCE	Flow control enable. If asserted, the receiver will detect PAUSE frames. Upon PAUSE frame detection, the transmitter will stop transmitting data frames for a given duration.
4 BC_REJ	Broadcast frame reject. If asserted, frames with DA (destination address) = FF_FF_FF_FF_FF_FF will be rejected unless the PROM bit is set. If both BC_REJ and PROM = 1, then frames with broadcast DA will be accepted and the M (MISS) bit will be 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. Selects external interface mode. Setting this bit to one selects MII mode, setting this bit equal to zero selects 7-wire mode (used only for serial 10 Mbps). This bit controls the interface mode for both transmit and receive blocks.
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. If set, transmitted frames are looped back internal to the device and the transmit output signals are not asserted. The internal bus clock is substituted for the FEC_TXCLK when LOOP is asserted. DRT must be set to zero when setting LOOP.

19.2.4.10 Transmit Control Register (TCR)

The TCR is read/write and is written by the user to configure the transmit block. This register is cleared at system reset. Bits 2 and 1 should be modified only when ECR[ETHER_EN] = 0.

Address: 0xFC03_00C4

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	RFC_PAUSE	TFC_PAUSE	FEDN	HBC	GTS		
W																												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		

Figure 19-11. Transmit Control Register (TCR)

Table 19-14. TCR Field Descriptions

Field	Description
31–5	Reserved, should be cleared.
4 RFC_PAUSE	Receive frame control pause. This read-only status bit will be asserted when a full duplex flow control pause frame has been received and the transmitter is paused for the duration defined in this pause frame. This bit will automatically clear when the pause duration is complete.
3 TFC_PAUSE	Transmit frame control pause. Transmits a PAUSE frame when asserted. When this bit is set, the MAC will stop transmission of data frames after the current transmission is complete. At this time, the GRA interrupt in the EIR register will be asserted. With transmission of data frames stopped, the MAC will transmit a MAC Control PAUSE frame. Next, the MAC will clear the TFC_PAUSE bit and resume transmitting data frames. Note that if the transmitter is paused due to user assertion of GTS or reception of a PAUSE frame, the MAC may still transmit a MAC Control PAUSE frame.
2 FEDN	Full duplex enable. If set, frames are transmitted independent of carrier sense and collision inputs. This bit should only be modified when ETHER_EN is cleared.
1 HBC	Heartbeat control. If set, the heartbeat check is performed following end of transmission and the HB bit in the status register will be set if the collision input does not assert within the heartbeat window. This bit should only be modified when ETHER_EN is cleared.
0 GTS	Graceful transmit stop. When this bit is set, the MAC will stop transmission after any frame that is currently being transmitted is complete and the GRA interrupt in the EIR register will be asserted. If frame transmission is not currently underway, the GRA interrupt will be asserted immediately. Once transmission has completed, a “restart” can be accomplished by clearing the GTS bit. The next frame in the transmit FIFO will then be transmitted. If an early collision occurs during transmission when GTS = 1, transmission will stop after the collision. The frame will be transmitted again once GTS is cleared. Note that there may be old frames in the transmit FIFO that will be transmitted when GTS is reassigned. To avoid this clear ECR[ETHER_EN] following the GRA interrupt.

19.2.4.11 Physical Address Low Register (PALR)

The PALR is written by the user. This register 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 6-byte source address field when transmitting PAUSE frames. This register is not reset and must be initialized by the user.

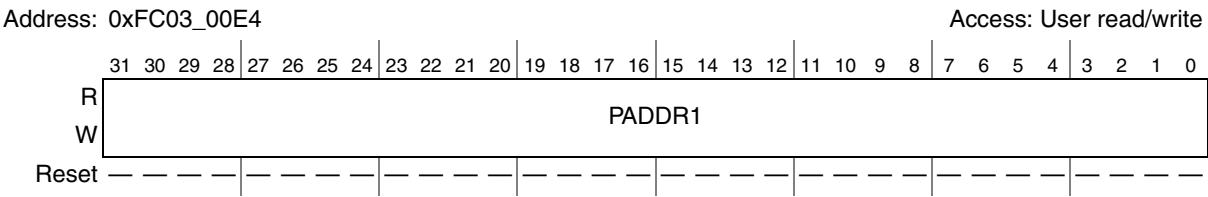


Figure 19-12. Physical Address Low Register (PALR)

Table 19-15. 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 to be used for exact match, and the source address field in PAUSE frames.

19.2.4.12 Physical Address High Register (PAUR)

The PAUR is written by the user. This register 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 6-byte Source Address field when transmitting PAUSE frames. Bits 15:0 of PAUR contain a constant type field (0x8808) used for transmission of PAUSE frames. The upper 16 bits of this register are not reset and must be initialized by the user.

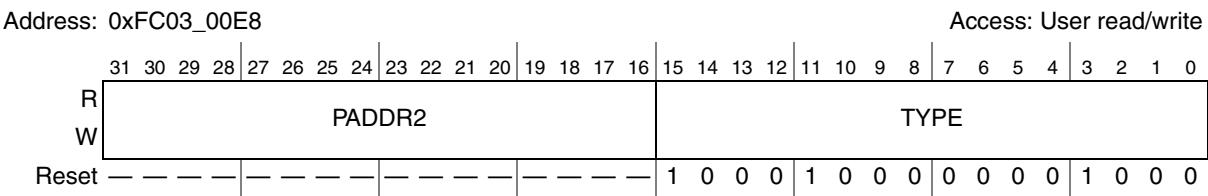


Figure 19-13. Physical Address High Register (PAUR)

Table 19-16. PAUR Field Descriptions

Field	Description
31–16 PADDR2	Bytes 4 (bits 31:24) and 5 (bits 23:16) of the 6-byte individual address to be used for exact match, and the Source Address field in PAUSE frames.
15–0 TYPE	Type field in PAUSE frames. These 16-bits are a constant value of 0x8808.

19.2.4.13 Opcode/Pause Duration Register (OPD)

The 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 will pause transmission for the duration specified in the pause duration field. The lower 16 bits of this register are not reset and must be initialized by the user.

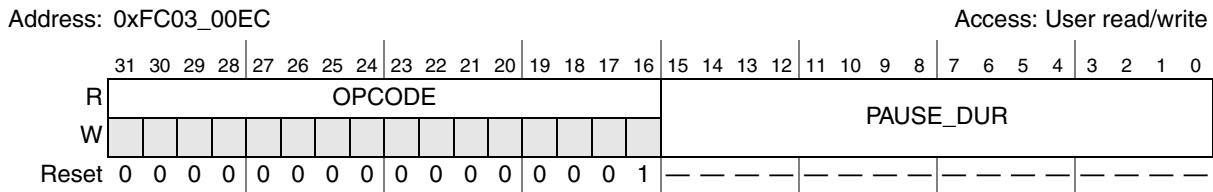


Figure 19-14. Opcode/Pause Duration Register (OPD)

Table 19-17. OPD Field Descriptions

Field	Description
31–16 OPCODE	Opcode field used in PAUSE frames. These bits are a constant, 0x0001.
15–0 PAUSE_DUR	Pause Duration field used in PAUSE frames.

19.2.4.14 Descriptor Individual Upper Address Register (IAUR)

The IAUR is written by the user. This register contains the upper 32 bits of the 64-bit individual address hash table used in the address recognition process to check for possible match with the DA field of receive frames with an individual DA. This register is not reset and must be initialized by the user.

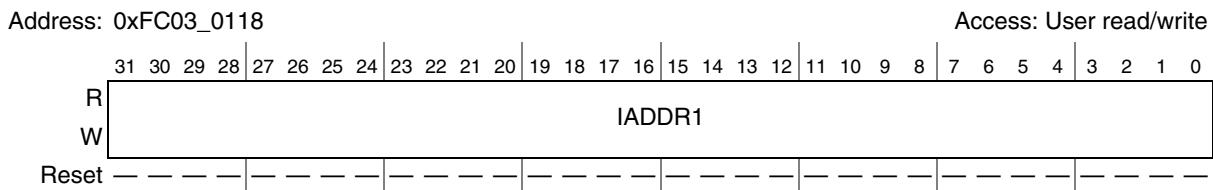


Figure 19-15. Descriptor Individual Upper Address Register (IAUR)

Table 19-18. 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.

19.2.4.15 Descriptor Individual Lower Address Register (IALR)

The IALR register is written by the user. This register contains the lower 32 bits of the 64-bit individual address hash table used in the address recognition process to check for possible match with the DA field of receive frames with an individual DA. This register is not reset and must be initialized by the user.

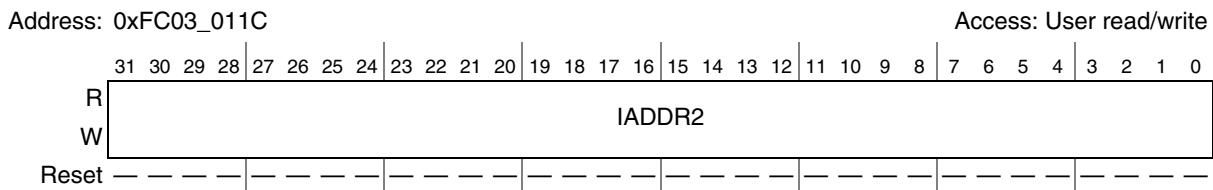


Figure 19-16. Descriptor Individual Lower Address Register (IALR)

Table 19-19. 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.

19.2.4.16 Descriptor Group Upper Address Register (GAUR)

The GAUR is written by the user. This register contains the upper 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. This register must be initialized by the user.

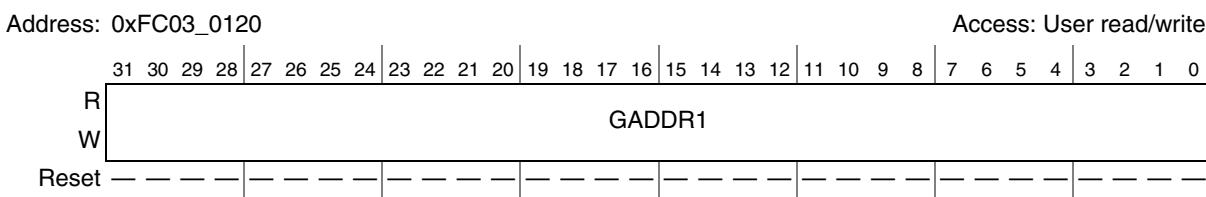


Figure 19-17. Descriptor Group Upper Address Register (GAUR)

Table 19-20. GAUR Field Descriptions

Field	Description
31–0 GADDR1	The GADDR1 register 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.

19.2.4.17 Descriptor Group Lower Address Register (GALR)

The GALR register is written by the user. This register contains the lower 32 bits of the 64-bit hash table used in the address recognition process for receive frames with a multicast address. This register must be initialized by the user.

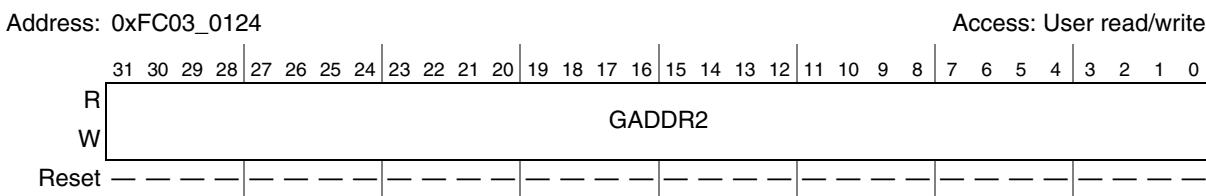


Figure 19-18. Descriptor Group Lower Address Register (GALR)

Table 19-21. GALR Field Descriptions

Field	Description
31–0 GADDR2	The GADDR2 register 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.

19.2.4.18 FIFO Transmit FIFO Watermark Register (TFWR)

The TFWR is a 2-bit read/write register programmed by the user to control the amount of data required in the transmit FIFO before transmission of a frame can begin. This allows the user 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 will minimize 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).

Address: 0xFC03_0144																																
	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	TFWR	
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 19-19. FIFO Transmit FIFO Watermark Register (TFWR)

Table 19-22. TFWR Field Descriptions

Field	Description
31–2	Reserved, should be cleared.
1–0 TFWR	Number of bytes written to transmit FIFO before transmission of a frame begins 00 64 bytes written 01 64 bytes written 10 128 bytes written 11 192 bytes written

19.2.4.19 FIFO Receive Bound Register (FRBR)

The FRBR is an 8-bit register that the user can read to determine the upper address bound of the FIFO RAM. Drivers can use this value, along with the FRSR to appropriately divide the available FIFO RAM between the transmit and receive data paths.

Address: 0xFC03_014C																																
	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	1	R_BOUND				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	0	0	0	0	0	0	0	0	

Figure 19-20. FIFO Receive Bound Register (FRBR)

Table 19-23. FRBR Field Descriptions

Field	Description
31–10	Reserved, read as 0 (except bit 10, which is read as 1).
9–2 R_BOUND	Read-only. Highest valid FIFO RAM address.
1–0	Reserved, read as 0.

19.2.4.20 FIFO Receive Start Register (FRSR)

The FRSR is programmed by the user to indicate the starting address of the receive FIFO. FRSR marks the boundary between the transmit and receive FIFOs. The transmit FIFO uses addresses from the start of the FIFO to the location four bytes before the address programmed into the FRSR. The receive FIFO uses addresses from FRSR to FRBR inclusive.

The FRSR register is initialized by hardware at reset. FRSR only needs to be written to change the default value.

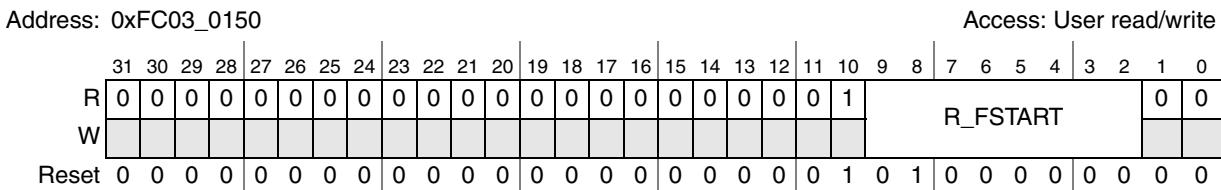


Figure 19-21. FIFO Receive Start Register (FRSR)

Table 19-24. FRSR Field Descriptions

Field	Description
31–11	Reserved, should be cleared.
10	Reserved, should be set.
9–2 R_FSTART	Address of first receive FIFO location. Acts as delimiter between receive and transmit FIFOs. For proper operation, ensure that R_FSTART is set to 0x48 or greater.
1–0	Reserved, should be cleared.

19.2.4.21 Receive Descriptor Ring Start Register (ERDSR)

The ERDSR is written by the user. It provides a pointer 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 by the user prior to operation.

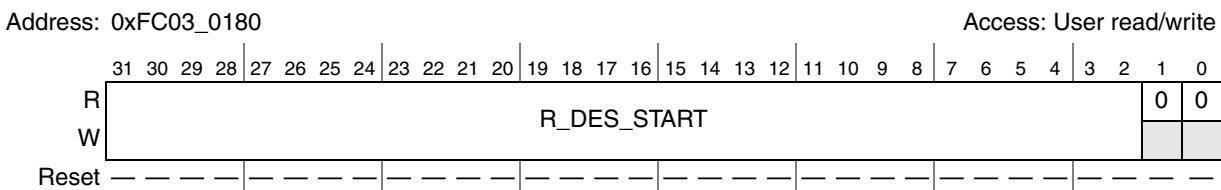


Figure 19-22. Receive Descriptor Ring Start Register (ERDSR)

Table 19-25. ERDSR Field Descriptions

Field	Description
31–2 R_des_start	Pointer to start of receive buffer descriptor queue.
1–0	Reserved, should be cleared.

19.2.4.22 Transmit Buffer Descriptor Ring Start Registers (ETSDR)

The ETSDR is written by the user. It 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). Bits 1 and 0 should be written to 0 by the user. Non-zero values in these two bit positions are ignored by the hardware.

This register is not reset and must be initialized by the user prior to operation.



Figure 19-23. Transmit Buffer Descriptor Ring Start Register (ETSDR)

Table 19-26. ETSDR Field Descriptions

Field	Description
31–2 X_DES_START	Pointer to start of transmit buffer descriptor queue.
1–0	Reserved, should be cleared.

19.2.4.23 Receive Buffer Size Register (EMRBR)

The EMRBR is a user-programmable register that dictates the maximum size of all receive buffers. Note that because receive frames will be truncated at 2k-15 bytes, only bits 10–4 are used. 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, EMRBR must be set to RCR[MAX_FL] or larger. The EMRBR must be evenly divisible by 16. To insure this, bits 3-0 are forced low. To minimize bus utilization (descriptor fetches), it is recommended that EMRBR be greater than or equal to 256 bytes.

The EMRBR register does not reset, and must be initialized by the user.

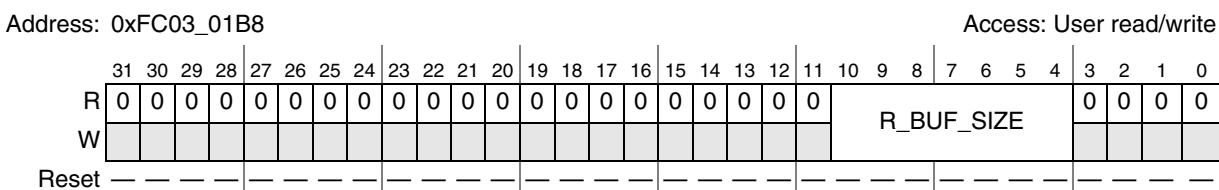


Figure 19-24. Receive Buffer Size Register (EMRBR)

Table 19-27. EMRBR Field Descriptions

Field	Description
31–11	Reserved, should be cleared.
10–4 R_BUF_SIZE	Receive buffer size in bytes. 0x00 0 bytes 0x01 16 bytes 0x02 32 bytes ... 0x7F 2032 bytes
3–0	Reserved, should be cleared.

19.2.5 Buffer Descriptors

This section provides a description of the operation of the driver/DMA via the buffer descriptors. It is followed by a detailed description of the receive and transmit descriptor fields.

19.2.5.1 Driver/DMA Operation with Buffer Descriptors

The data for the FEC frames must reside in memory external to the FEC. The data for a frame is placed in one or more buffers. Associated with each buffer is a buffer descriptor (BD) which contains a starting address (32-bit aligned pointer), data length, and status/control information (which contains the current state for the buffer). To permit maximum user flexibility, the BDs are also located in external memory and are read in by the FEC DMA engine.

Software produces buffers by allocating/initializing memory and initializing buffer descriptors. Setting the RxBD[E] or TxBD[R] bit produces the buffer. Software writing to either the TDAR or RDAR tells the FEC that a buffer has been placed in external memory for the transmit or receive data traffic, respectively. The hardware reads the BDs and consumes the buffers after they have been produced. After the data DMA is complete and the buffer descriptor status bits have been written by the DMA engine, the RxBD[E] or TxBD[R] bit will be cleared by hardware to signal the buffer has been consumed. Software may poll the BDs to detect when the buffers have been consumed or may rely on the buffer/frame interrupts. These buffers may then be processed by the driver and returned to the free list.

The ECR[ETHER_EN] signal operates as a reset to the BD/DMA logic. When ECR[ETHER_EN] is cleared, the DMA engine BD pointers are reset to point to the starting transmit and receive BDs. The buffer descriptors are not initialized by hardware during reset. At least one transmit and receive buffer descriptor must be initialized by software before the ECR[ETHER_EN] bit is set.

The buffer descriptors operate as two separate rings. ERDSR defines the starting address for receive BDs and ETDSR defines the starting address for transmit BDs. The last buffer descriptor in each ring is defined by the wrap (W) bit. When set, W indicates that the next descriptor in the ring is at the location pointed to by ERDSR and ETDSR for the receive and transmit rings, respectively. Buffer descriptor rings must start on a 32-bit boundary; however, it is recommended they are made 128-bit aligned.

19.2.5.1.1 Driver/DMA Operation with Transmit BDs

Typically a transmit frame will be divided between multiple buffers. An example is to have an application payload in one buffer, TCP header in a 2nd buffer, IP header in a 3rd buffer, Ethernet/IEEE 802.3 header in a 4th buffer. The Ethernet MAC does not prepend the Ethernet header (destination address, source address, length/type field(s)), so this must be provided by the driver in one of the transmit buffers. The Ethernet MAC can append the Ethernet CRC to the frame. Whether the CRC is appended by the MAC or by the driver is determined by the TC bit in the transmit BD which must be set by the driver.

The driver (TxBD software producer) should set up Tx BDs in such a way that a complete transmit frame is given to the hardware at once. If a transmit frame consists of three buffers, the BDs should be initialized with pointer, length and control (W, L, TC, ABC) and then the TxBD[R] bits should be set = 1 in reverse order (3rd, 2nd, 1st BD) to insure that the complete frame is ready in memory before the DMA begins. If the TxBDs are set up in order, the DMA Controller could DMA the first BD before the 2nd was made available, potentially causing a transmit FIFO underrun.

In the FEC, the DMA is notified by the driver that new transmit frame(s) are available by writing to the TDAR register. When this register is written to (data value is not significant) the FEC RISC will tell the DMA to read the next transmit BD in the ring. Once started, the RISC + DMA will continue to read and interpret transmit BDs in order and DMA the associated buffers, until a transmit BD is encountered with the R bit = 0. At this point the FEC will poll this BD one more time. If the R bit = 0 the second time, then the RISC will stop the transmit descriptor read process until software sets up another transmit frame and writes to TDAR.

When the DMA of each transmit buffer is complete, the DMA writes back to the BD to clear the R bit, indicating that the hardware consumer is finished with the buffer.

19.2.5.1.2 Driver/DMA Operation with Receive BDs

Unlike transmit, the length of the receive frame is unknown by the driver ahead of time. Therefore the driver must set a variable to define the length of all receive buffers. In the FEC, this variable is written to the EMRBR register.

The driver (RxBD software producer) should set up some number of empty buffers for the Ethernet by initializing the address field and the E and W bits of the associated receive BDs. The hardware (receive DMA) will consume these buffers by filling them with data as frames are received and clearing the E bit and writing to the L (1 indicates last buffer in frame) bit, the frame status bits (if L = 1) and the length field.

If a receive frame spans multiple receive buffers, the L bit is only set for the last buffer in the frame. For non-last buffers, the length field in the receive BD will be written by the DMA (at the same time the E bit is cleared) with the default receive buffer length value. For end of frame buffers, the receive BD will be written with L = 1 and information written to the status bits (M, BC, MC, LG, NO, CR, OV, TR). Some of the status bits are error indicators which, if set, indicate the receive frame should be discarded and not given to higher layers. The frame status/length information is written into the receive FIFO following the end of the frame (as a single 32-bit word) by the receive logic. The length field for the end of frame buffer will be written with the length of the entire frame, not just the length of the last buffer.

For simplicity the driver may assign the default receive buffer length to be large enough to contain an entire frame, keeping in mind that a malfunction on the network or out of spec implementation could result in

giant frames. Frames of 2k-15 (2033) bytes or larger are truncated by the FEC at 2032 bytes so software is guaranteed never to see a receive frame larger than 2032 bytes.

Similar to transmit, the FEC will poll the receive descriptor ring after the driver sets up receive BDs and writes to the RDAR register. As frames are received, the FEC will fill receive buffers and update the associated BDs, then read the next BD in the receive descriptor ring. If the FEC reads a receive BD and finds the E bit = 0, it will poll this BD once more. If the BD = 0 a second time, the FEC will stop reading receive BDs until the driver writes to RDAR.

19.2.5.2 Ethernet Receive Buffer Descriptor (RxBD)

In the RxBD, the user initializes the E and W bits in the first longword and the pointer in second longword. When the buffer has been DMA'd, the Ethernet controller will modify the E, L, M, BC, MC, LG, NO, CR, OV, and TR bits and write the length of the used portion of the buffer in the first longword. The M, BC, MC, LG, NO, CR, OV, and TR bits in the first longword of the buffer descriptor are only modified by the Ethernet controller when the L bit is set.

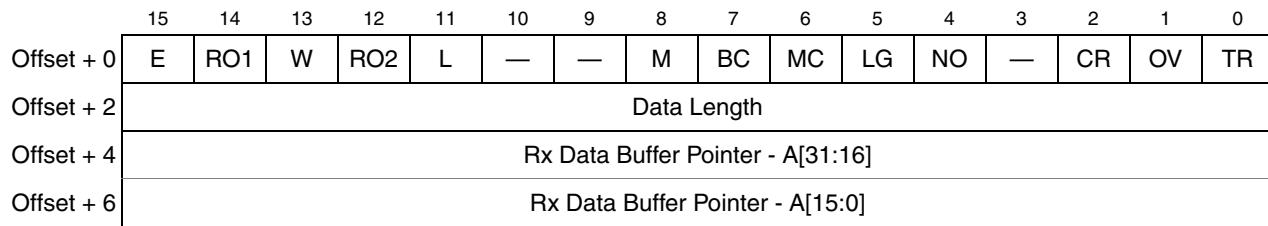


Figure 19-25. Receive Buffer Descriptor (RxBD)

Table 19-28. Receive Buffer Descriptor Field Definitions

Word	Field	Description
Offset + 0	15 E	Empty. Written by the FEC (=0) and user (=1). 0 The data buffer associated with this BD has been filled with received data, or data reception has been 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 will not be modified by hardware, nor will 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 ERDSR.
Offset + 0	12 RO2	Receive software ownership. This field is reserved for use by software. This read/write bit will not be modified by hardware, nor will its value affect hardware.
Offset + 0	11 L	Last in frame. Written by the FEC. 0 The buffer is not the last in a frame. 1 The buffer is the last in a frame.
Offset + 0	10–9	Reserved, should be cleared.

Table 19-28. Receive Buffer Descriptor Field Definitions (continued)

Word	Field	Description
Offset + 0	8 M	Miss. Written by the FEC. This bit is set by the FEC for frames that were accepted in promiscuous mode, but were flagged as a “miss” by the internal address recognition. Thus, while in promiscuous mode, the user can use the M-bit to quickly determine whether the frame was destined to this station. This bit is valid only if the L-bit is set and the PROM bit is 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	Will be set if the DA is broadcast (FF-FF-FF-FF-FF-FF).
Offset + 0	6 MC	Will be set if the DA is multicast and not BC.
Offset + 0	5 LG	Rx frame length violation. Written by the FEC. 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 2032 bytes.
Offset + 0	4 NO	Receive non-octet aligned frame. Written by the FEC. 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. This bit is valid only if the L-bit is set. If this bit is set the CR bit will not be set.
Offset + 0	3	Reserved, should be cleared.
Offset + 0	2 CR	Receive CRC error. Written by the FEC. This frame contains a 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 FEC. 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 will be zero. This bit is valid only if the L-bit is set.
Offset + 0	0 TR	Will be set if the receive frame is truncated (frame length > 2032 bytes). If the TR bit is set the frame should be discarded and the other error bits should be ignored as they may be incorrect.
Offset + 2	15–0 Data Length	Data length. Written by the FEC. Data length is the number of octets written by the FEC into this BD’s data buffer if L = 0 (the value will be equal to EMRBR), or the length of the frame including CRC if L = 1. It is written by the FEC once as the BD is closed.
Offset + 4	15–0 A[31:16]	RX data buffer pointer, bits [31:16] ¹
Offset + 6	15–0 A[15:0]	RX data buffer pointer, bits [15:0]

¹ The receive buffer pointer, which contains the address of the associated data buffer, must always be evenly divisible by 16. The buffer must reside in memory external to the FEC. This value is never modified by the Ethernet controller.

NOTE

Whenever the software driver sets an E bit in one or more receive descriptors, the driver should follow that with a write to RDAR.

19.2.5.3 Ethernet Transmit Buffer Descriptor (TxBD)

Data is presented to the FEC for transmission by arranging it in buffers referenced by the channel’s TxBDs. The Ethernet controller confirms transmission by clearing the ready bit (R bit) when DMA of the buffer is

complete. In the TxBD the user initializes the R, W, L, and TC bits and the length (in bytes) in the first longword, and the buffer pointer in the second longword.

The FEC will set the R bit = 0 in the first longword of the BD when the buffer has been transferred. Status bits for the buffer/frame are not included in the transmit buffer descriptors. Transmit frame status is indicated via individual interrupt bits (error conditions) and in statistic counters in the MIB block. See [Section 19.2.3, “MIB Block Counters Memory Map”](#) for more details.

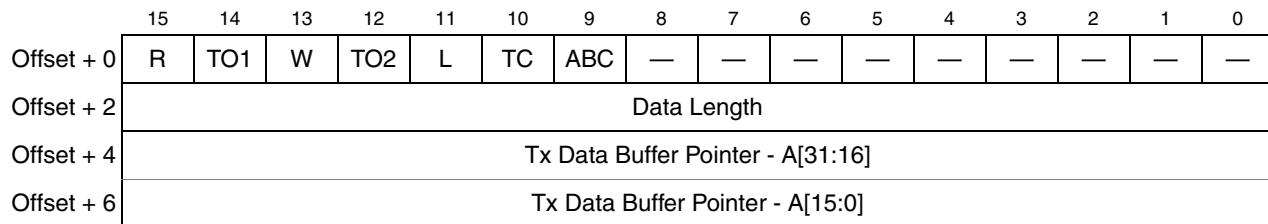


Figure 19-26. Transmit Buffer Descriptor (TxBD)

Table 19-29. Transmit Buffer Descriptor Field Definitions

Word	Field	Description
Offset + 0	15 R	Ready. Written by the FEC and the user. 0 The data buffer associated with this BD is not ready for transmission. The user is free to manipulate this BD or its associated data buffer. The FEC clears this bit after the buffer has been transmitted or after an error condition is encountered. 1 The data buffer, which has been prepared for transmission by the user, has not been transmitted or is currently being transmitted. No fields of this BD may be written by the user once this bit is set.
Offset + 0	14 TO1	Transmit software ownership. This field is reserved for software use. This read/write bit will not be modified by hardware, nor will 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 will not be modified by hardware, nor will 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	Tx CRC. Written by user (only valid if L = 1). 0 End transmission immediately after the last data byte. 1 Transmit the CRC sequence after the last data byte.
Offset + 0	9 ABC	Append bad CRC. Written by user (only valid if L = 1). 0 No effect 1 Transmit the CRC sequence inverted after the last data byte (regardless of TC value).
Offset + 0	8–0	Reserved, should be cleared.
Offset + 2	15–0 Data Length	Data Length, written by user. Data length is the number of octets the FEC should transmit from this BD's data buffer. It is never modified by the FEC. Bits [15:5] are used by the DMA engine, bits[4:0] are ignored.

Table 19-29. Transmit Buffer Descriptor Field Definitions (continued)

Word	Field	Description
Offset + 4	15–0 A[31:16]	Tx data buffer pointer, bits [31:16] ¹
Offset + 6	15–0 A[15:0]	Tx data buffer pointer, bits [15:0].

¹ The transmit buffer pointer, which contains the address of the associated data buffer, must always be evenly divisible by 4. The buffer must reside in memory external to the FEC. This value is never modified by the Ethernet controller.

NOTE

Once the software driver has set up the buffers for a frame, it should set up the corresponding BDs. The last step in setting up the BDs for a transmit frame should be to set the R bit in the first BD for the frame. The driver should follow that with a write to TDAR which will trigger the FEC to poll the next BD in the ring.

19.3 Functional Description

This section describes the operation of the FEC, beginning with the hardware and software initialization sequence, then the software (Ethernet driver) interface for transmitting and receiving frames.

Following the software initialization and operation sections are sections providing a detailed description of the functions of the FEC.

19.3.1 Initialization Sequence

This section describes which registers are reset due to hardware reset, which are reset by the FEC RISC, and what locations the user must initialize prior to enabling the FEC.

19.3.1.1 Hardware Controlled Initialization

In the FEC, registers and control logic that generate interrupts are reset by hardware. A hardware reset negates output signals and resets general configuration bits.

Other registers reset when the ECR[ETHER_EN] bit is cleared. ECR[ETHER_EN] is cleared by a hard reset or may be cleared by software to halt operation. By clearing ECR[ETHER_EN], the configuration control registers such as the TCR and RCR will not be reset, but the entire data path will be reset.

Table 19-30. ECR[ETHER_EN] De-Assertion Effect on FEC

Register/Machine	Reset Value
XMIT block	Transmission is aborted (bad CRC appended)
RECV block	Receive activity is aborted
DMA block	All DMA activity is terminated
RDAR	Cleared

Table 19-30. ECR[ETHER_EN] De-Assertion Effect on FEC

Register/Machine	Reset Value
TDAR	Cleared
Descriptor Controller block	Halt operation

19.3.2 User Initialization (Prior to Setting ECR[ETHER_EN])

The user needs to initialize portions of the FEC prior to setting the ECR[ETHER_EN] bit. The exact values will depend on the particular application. The sequence is not important.

Ethernet MAC registers requiring initialization are defined in [Table 19-31](#).

Table 19-31. User Initialization (Before ECR[ETHER_EN])

Description
Initialize EIMR
Clear EIR (write 0xFFFF_FFFF)
TFWR (optional)
IALR / IAUR
GAUR / GALR
PALR / PAUR (only needed for full duplex flow control)
OPD (only needed for full duplex flow control)
RCR
TCR
MSCR (optional)
Clear MIB_RAM (locations 0xFC03_0200–0xFC03_02FC)

FEC FIFO/DMA registers that require initialization are defined in [Table 19-32](#).

Table 19-32. FEC User Initialization (Before ECR[ETHER_EN])

Description
Initialize FRSR (optional)
Initialize EMRBR
Initialize ERDSR
Initialize ETDSR
Initialize (Empty) Transmit Descriptor ring
Initialize (Empty) Receive Descriptor ring

19.3.3 Microcontroller Initialization

In the FEC, the descriptor control RISC initializes some registers after ECR[ETHER_EN] is asserted. After the microcontroller initialization sequence is complete, the hardware is ready for operation.

[Table 19-33](#) shows microcontroller initialization operations.

Table 19-33. Microcontroller Initialization

Description
Initialize BackOff Random Number Seed
Activate Receiver
Activate Transmitter
Clear Transmit FIFO
Clear Receive FIFO
Initialize Transmit Ring Pointer
Initialize Receive Ring Pointer
Initialize FIFO Count Registers

19.3.4 User Initialization (After Setting ECR[ETHER_EN])

After setting ECR[ETHER_EN], the user can set up the buffer/frame descriptors and write to the TDAR and RDAR. Refer to [Section 19.2.5, “Buffer Descriptors”](#) for more details.

19.3.5 Network Interface Options

The FEC supports both an MII interface for 10/100 Mbps Ethernet and a 7-wire serial interface for 10 Mbps Ethernet. The interface mode is selected by the RCR[MII_MODE] bit. In MII mode (RCR[MII_MODE] = 1), there are 18 signals defined by the IEEE 802.3 standard and supported by the EMAC. These signals are shown in [Table 19-34](#) below.

Table 19-34. MII Mode

Signal Description	EMAC pin
Transmit Clock	FEC_TXCLK
Transmit Enable	FEC_TXEN
Transmit Data	FEC_TXD[3:0]
Transmit Error	FEC_TXER
Collision	FEC_COL
Carrier Sense	FEC_CRS
Receive Clock	FEC_RXCLK
Receive Data Valid	FEC_RXDV
Receive Data	FEC_RXD[3:0]
Receive Error	FEC_RXER
Management Data Clock	FEC_MDC
Management Data Input/Output	FEC_MDIO

The 7-wire serial mode interface ($\text{RCR}[\text{MII_MODE}] = 0$) operates in what is generally referred to as the ‘AMD’ mode. 7-wire mode connections to the external transceiver are shown in [Table 19-35](#).

Table 19-35. 7-Wire Mode Configuration

SIGNAL DESCRIPTION	EMAC PIN
Transmit Clock	FEC_TXCLK
Transmit Enable	FEC_TXEN
Transmit Data	FEC_TXD[0]
Collision	FEC_COL
Receive Clock	FEC_RXCLK
Receive Data Valid	FEC_RXDV
Receive Data	FEC_RXD[0]

19.3.6 FEC Frame Transmission

The Ethernet transmitter is designed to work with almost no intervention from software. Once $\text{ECR}[\text{ETHER_EN}]$ is set and data appears in the transmit FIFO, the Ethernet MAC is able to transmit onto the network.

When the transmit FIFO fills to the watermark (defined by the TFWR), the MAC transmit logic will assert FEC_TXEN and start transmitting the preamble (PA) sequence, the start frame delimiter (SFD), and then the frame information from the FIFO. However, the controller defers the transmission if the network is busy (FEC_CRS asserts). Before transmitting, the controller waits for carrier sense to become inactive, then determines if carrier sense stays inactive for 60 bit times. If so, the transmission begins after waiting an additional 36 bit times (96 bit times after carrier sense originally became inactive). See [Section 19.3.14.1, “Transmission Errors”](#) for more details.

If a collision occurs during transmission of the frame (half duplex mode), the Ethernet controller follows the specified backoff procedures and attempts to retransmit the frame until the retry limit is reached. The transmit FIFO stores at least the first 64 bytes of the transmit frame, so that they do not have to be retrieved from system memory in case of a collision. This improves bus utilization and latency in case immediate retransmission is necessary.

When all the frame data has been transmitted, the FCS (frame check sequence) or 32-bit cyclic redundancy check (CRC) bytes are appended if the TC bit is set in the transmit frame control word. If the ABC bit is set in the transmit frame control word, a bad CRC will be appended to the frame data regardless of the TC bit value. Following the transmission of the CRC, the Ethernet controller writes the frame status information to the MIB block. Short frames are automatically padded by the transmit logic (if the TC bit in the transmit buffer descriptor for the end of frame buffer = 1).

Both buffer (TXB) and frame (TFINT) interrupts may be generated as determined by the settings in the EIMR.

The transmit error interrupts are HBERR, BABT, LATE_COL, COL_RETRY_LIM, and XFIFO_UN. If the transmit frame length exceeds MAX_FL bytes the BABT interrupt will be asserted, however the entire frame will be transmitted (no truncation).

To pause transmission, set the GTS (graceful transmit stop) bit in the TCR register. When the TCR[GTS] is set, the FEC transmitter stops immediately if transmission is not in progress; otherwise, it continues transmission until the current frame either finishes or terminates with a collision. After the transmitter has stopped the GRA (graceful stop complete) interrupt is asserted. If TCR[GTS] is cleared, the FEC resumes transmission with the next frame.

The Ethernet controller transmits bytes least significant bit first.

19.3.7 FEC Frame Reception

The FEC receiver is designed to work with almost no intervention from the host and can perform address recognition, CRC checking, short frame checking, and maximum frame length checking.

When the driver enables the FEC receiver by setting ECR[ETHER_EN], it will immediately start processing receive frames. When FEC_RXDV is asserted, the receiver will first check for a valid PA/SFD header. If the PA/SFD is valid, it will be stripped and the frame will be processed by the receiver. If a valid PA/SFD is not found, the frame will be ignored.

In serial mode, the first 16 bit times of RX_D0 following assertion of FEC_RXDV are ignored. Following the first 16 bit times the data sequence is checked for alternating 1/0s. If a 11 or 00 data sequence is detected during bit times 17 to 21, the remainder of the frame is ignored. After bit time 21, the data sequence is monitored for a valid SFD (11). If a 00 is detected, the frame is rejected. When a 11 is detected, the PA/SFD sequence is complete.

In MII mode, the receiver checks for at least one byte matching the SFD. Zero or more PA bytes may occur, but if a 00 bit sequence is detected prior to the SFD byte, the frame is ignored.

After the first 6 bytes of the frame have been received, the FEC performs address recognition on the frame.

Once a collision window (64 bytes) of data has been received and if address recognition has not rejected the frame, the receive FIFO is signalled that the frame is accepted and may be passed on to the DMA. If the frame is a runt (due to collision) or is rejected by address recognition, the receive FIFO is notified to reject the frame. Thus, no collision fragments are presented to the user except late collisions, which indicate serious LAN problems.

During reception, the Ethernet controller checks for various error conditions and once the entire frame is written into the FIFO, a 32-bit frame status word is written into the FIFO. This status word contains the M, BC, MC, LG, NO, CR, OV, and TR status bits, and the frame length. See [Section 19.3.14.2, “Reception Errors”](#) for more details.

Receive buffer (RXB) and frame interrupts (RFINT) may be generated if enabled by the EIMR register. A receive error interrupt is babbling receiver error (BABR). Receive frames are not truncated if they exceed the max frame length (MAX_FL); however, the BABR interrupt will occur and the LG bit in the receive buffer descriptor (RxBD) will be set. See [Section 19.2.5.2, “Ethernet Receive Buffer Descriptor \(RxBD\)”](#) for more details.

When the receive frame is complete, the FEC sets the L-bit in the RxBD, writes the other frame status bits into the RxBD, and clears the E-bit. The Ethernet controller next generates a maskable interrupt (RFINT bit in EIR, maskable by RFIEN bit in EIMR), indicating that a frame has been received and is in memory. The Ethernet controller then waits for a new frame.

The Ethernet controller receives serial data lsb first.

19.3.8 Ethernet Address Recognition

The FEC filters the received frames based on destination address (DA) type — individual (unicast), group (multicast), or broadcast (all-ones group address). The difference between an individual address and a group address is determined by the I/G bit in the destination address field. A flowchart for address recognition on received frames is illustrated in the figures below.

Address recognition is accomplished through the use of the receive block and microcode running on the microcontroller. The flowchart shown in [Figure 19-27](#) illustrates the address recognition decisions made by the receive block, while [Figure 19-28](#) illustrates the decisions made by the microcontroller.

If the DA is a broadcast address and broadcast reject (RCR[BC_REJ]) is cleared, then the frame will be accepted unconditionally, as shown in [Figure 19-27](#). Otherwise, if the DA is not a broadcast address, then the microcontroller runs the address recognition subroutine, as shown in [Figure 19-28](#).

If the DA is a group (multicast) address and flow control is disabled, then the microcontroller will perform a group hash table lookup using the 64-entry hash table programmed in GAUR and GALR. If a hash match occurs, the receiver accepts the frame.

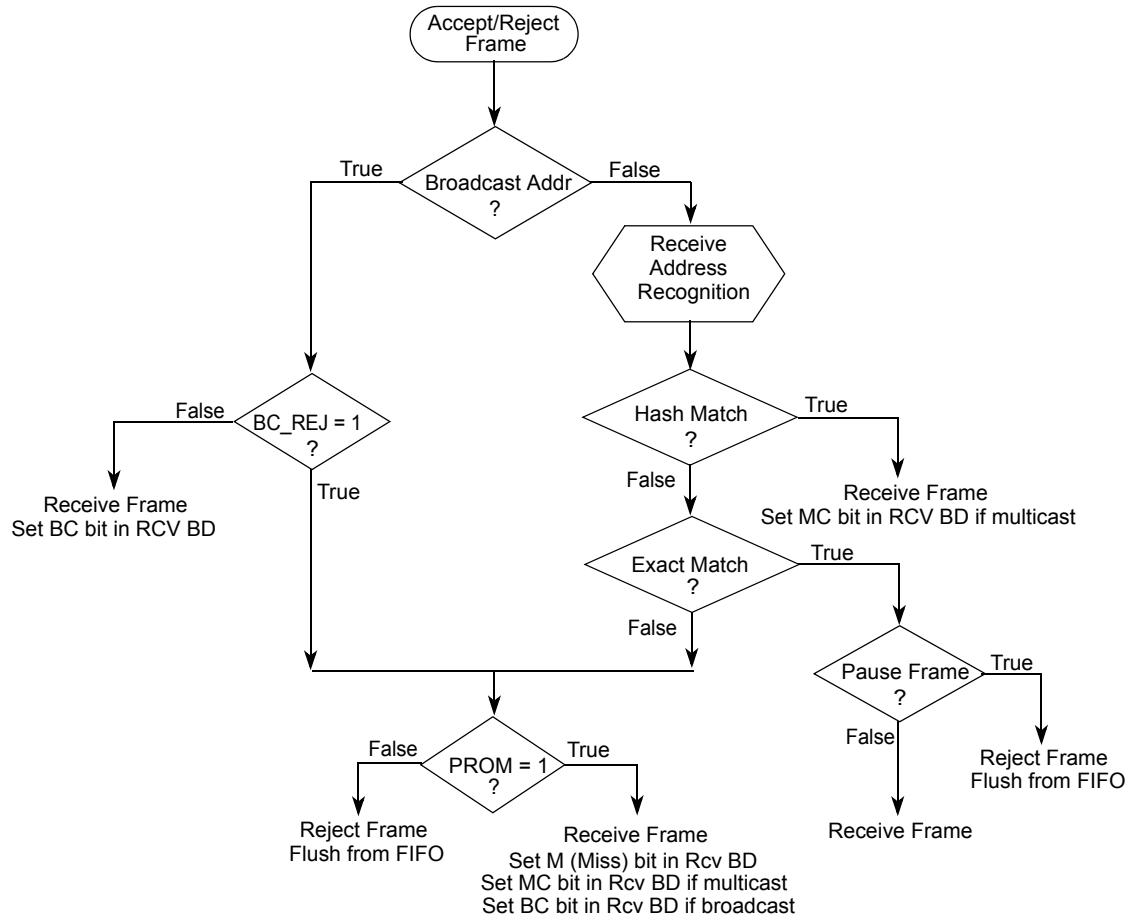
If flow control is enabled, the microcontroller will do an exact address match check between the DA and the designated PAUSE DA (01:80:C2:00:00:01). If the receive block determines that the received frame is a valid PAUSE frame, then the frame will be rejected. Note the receiver will detect a PAUSE frame with the DA field set to either the designated PAUSE DA or the unicast physical address.

If the DA is the individual (unicast) address, the microcontroller performs an individual exact match comparison between the DA and 48-bit physical address that the user programs in the PALR and PAUR registers. If an exact match occurs, the frame is accepted; otherwise, the microcontroller does an individual hash table lookup using the 64-entry hash table programmed in registers, IAUR and IALR. In the case of an individual hash match, the frame is accepted. Again, the receiver will accept or reject the frame based on PAUSE frame detection, shown in [Figure 19-27](#).

If neither a hash match (group or individual), nor an exact match (group or individual) occur, then if promiscuous mode is enabled (RCR[PROM] = 1), then the frame will be accepted and the MISS bit in the receive buffer descriptor is set; otherwise, the frame will be rejected.

Similarly, if the DA is a broadcast address, broadcast reject (RCR[BC_REJ]) is asserted, and promiscuous mode is enabled, then the frame will be accepted and the MISS bit in the receive buffer descriptor is set; otherwise, the frame will be rejected.

In general, when a frame is rejected, it is flushed from the FIFO.



NOTES:

BC_REJ - field in RCR register (BroadCast REject)
 PROM - field in RCR register (PROMiscous mode)
 Pause Frame - valid PAUSE frame received

Figure 19-27. Ethernet Address Recognition—Receive Block Decisions

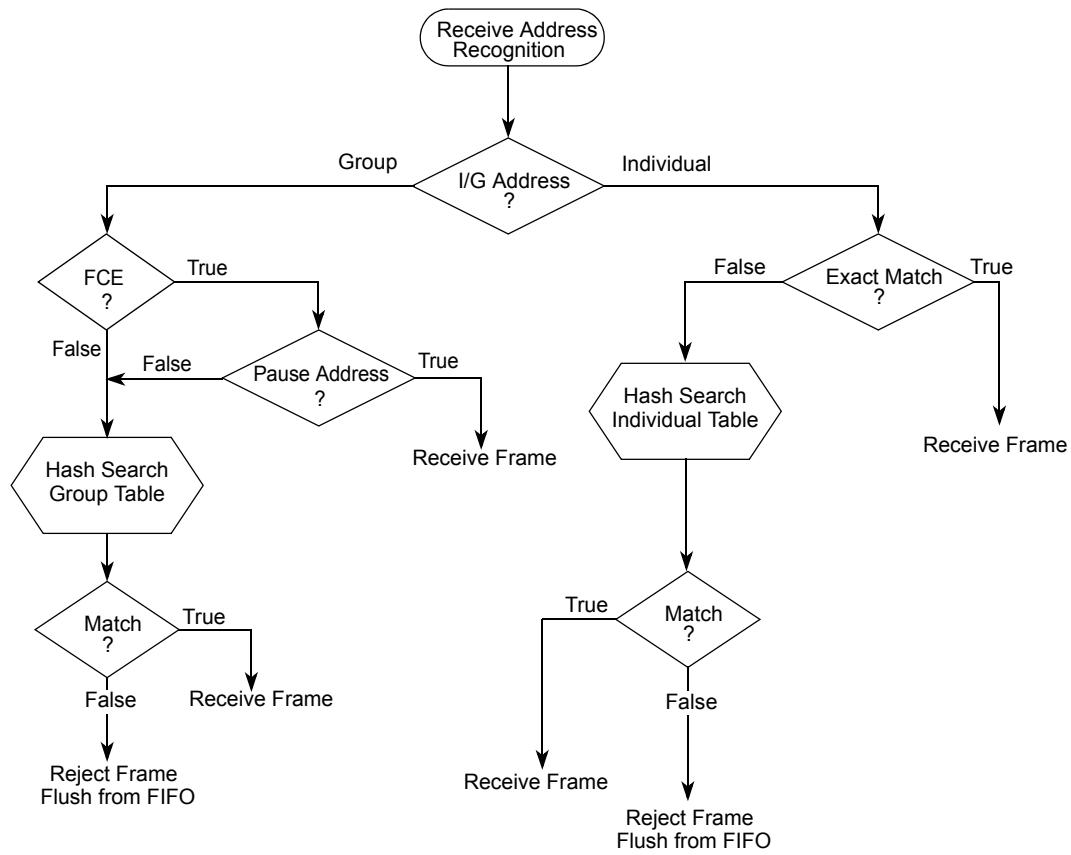


Figure 19-28. Ethernet Address Recognition—Microcode Decisions

19.3.9 Hash Algorithm

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, which are represented by 64 bits stored in GAUR, GALR (group address hash match) or 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 6 most significant bits of the CRC-encoded result to generate a number between 0 and 63. The msb of the CRC result selects GAUR (msb = 1) or GALR (msb = 0). The least significant 5 bits of the hash result select the bit within the selected register. If the CRC generator selects a bit that is set in the hash table, the frame is accepted; otherwise, 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 hash table registers must be initialized by the user. The CRC32 polynomial to use in computing the hash is:

$$X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X + 1$$

Eqn. 19-1

A table of example destination addresses and corresponding hash values is included below for reference.

Table 19-36. Destination Address to 6-Bit Hash

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
65:ff:ff:ff:ff:ff	0x0	0
55:ff:ff:ff:ff:ff	0x1	1
15:ff:ff:ff:ff:ff	0x2	2
35:ff:ff:ff:ff:ff	0x3	3
b5:ff:ff:ff:ff:ff	0x4	4
95:ff:ff:ff:ff:ff	0x5	5
d5:ff:ff:ff:ff:ff	0x6	6
f5:ff:ff:ff:ff:ff	0x7	7
db:ff:ff:ff:ff:ff	0x8	8
fb:ff:ff:ff:ff:ff	0x9	9
bb:ff:ff:ff:ff:ff	0xa	10
8b:ff:ff:ff:ff:ff	0xb	11
0b:ff:ff:ff:ff:ff	0xc	12
3b:ff:ff:ff:ff:ff	0xd	13
7b:ff:ff:ff:ff:ff	0xe	14
5b:ff:ff:ff:ff:ff	0xf	15
27:ff:ff:ff:ff:ff	0x10	16
07:ff:ff:ff:ff:ff	0x11	17
57:ff:ff:ff:ff:ff	0x12	18
77:ff:ff:ff:ff:ff	0x13	19
f7:ff:ff:ff:ff:ff	0x14	20
c7:ff:ff:ff:ff:ff	0x15	21
97:ff:ff:ff:ff:ff	0x16	22
a7:ff:ff:ff:ff:ff	0x17	23
99:ff:ff:ff:ff:ff	0x18	24
b9:ff:ff:ff:ff:ff	0x19	25
f9:ff:ff:ff:ff:ff	0x1a	26
c9:ff:ff:ff:ff:ff	0x1b	27

Table 19-36. Destination Address to 6-Bit Hash (continued)

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
59:ff:ff:ff:ff:ff	0x1c	28
79:ff:ff:ff:ff:ff	0x1d	29
29:ff:ff:ff:ff:ff	0x1e	30
19:ff:ff:ff:ff:ff	0x1f	31
d1:ff:ff:ff:ff:ff	0x20	32
f1:ff:ff:ff:ff:ff	0x21	33
b1:ff:ff:ff:ff:ff	0x22	34
91:ff:ff:ff:ff:ff	0x23	35
11:ff:ff:ff:ff:ff	0x24	36
31:ff:ff:ff:ff:ff	0x25	37
71:ff:ff:ff:ff:ff	0x26	38
51:ff:ff:ff:ff:ff	0x27	39
7f:ff:ff:ff:ff:ff	0x28	40
4f:ff:ff:ff:ff:ff	0x29	41
1f:ff:ff:ff:ff:ff	0x2a	42
3f:ff:ff:ff:ff:ff	0x2b	43
bf:ff:ff:ff:ff:ff	0x2c	44
9f:ff:ff:ff:ff:ff	0x2d	45
df:ff:ff:ff:ff:ff	0x2e	46
ef:ff:ff:ff:ff:ff	0x2f	47
93:ff:ff:ff:ff:ff	0x30	48
b3:ff:ff:ff:ff:ff	0x31	49
f3:ff:ff:ff:ff:ff	0x32	50
d3:ff:ff:ff:ff:ff	0x33	51
53:ff:ff:ff:ff:ff	0x34	52
73:ff:ff:ff:ff:ff	0x35	53
23:ff:ff:ff:ff:ff	0x36	54
13:ff:ff:ff:ff:ff	0x37	55
3d:ff:ff:ff:ff:ff	0x38	56
0d:ff:ff:ff:ff:ff	0x39	57
5d:ff:ff:ff:ff:ff	0x3a	58
7d:ff:ff:ff:ff:ff	0x3b	59

Table 19-36. Destination Address to 6-Bit Hash (continued)

48-bit DA	6-bit Hash (in hex)	Hash Decimal Value
fd:ff:ff:ff:ff:ff	0x3c	60
dd:ff:ff:ff:ff:ff	0x3d	61
9d:ff:ff:ff:ff:ff	0x3e	62
bd:ff:ff:ff:ff:ff	0x3f	63

19.3.10 Full Duplex Flow Control

Full-duplex flow control allows the user to transmit pause frames and to detect received pause frames. Upon detection of a pause frame, MAC data frame transmission stops for a given pause duration.

To enable pause frame detection, the FEC must operate in full-duplex mode (TCR[FDEN] asserted) and flow control enable (RCR[FCE]) must be asserted. The FEC detects a pause frame when the fields of the incoming frame match the pause frame specifications, as shown in the table below. In addition, the receive status associated with the frame should indicate that the frame is valid.

Table 19-37. PAUSE Frame Field Specification

48-bit Destination Address	0x0180_c200_0001 or Physical Address
48-bit Source Address	Any
16-bit Type	0x8808
16-bit Opcode	0x0001
16-bit PAUSE Duration	0x0000–0xFFFF

Pause frame detection is performed by the receiver and microcontroller modules. The microcontroller runs an address recognition subroutine to detect the specified pause frame destination address, while the receiver detects the type and opcode pause frame fields. On detection of a pause frame, TCR[GTS] is set by the FEC internally. When transmission has paused, the EIR[GRA] interrupt is asserted and the pause timer begins to increment. Note that the pause timer makes use of the transmit backoff timer hardware, which is used for tracking the appropriate collision backoff time in half-duplex mode. The pause timer increments once every slot time, until OPD[PAUSE_DUR] slot times have expired. On OPD[PAUSE_DUR] expiration, TCR[GTS] is cleared allowing MAC data frame transmission to resume. Note that the receive flow control pause (TCR[RFC_PAUSE]) status bit is set while the transmitter is paused due to reception of a pause frame.

To transmit a pause frame, the FEC must operate in full-duplex mode and the user must assert flow control pause (TCR[TFC_PAUSE]). On assertion of transmit flow control pause (TCR[TFC_PAUSE]), the transmitter sets TCR[GTS] internally. When the transmission of data frames stops, the EIR[GRA] (graceful stop complete) interrupt asserts. Following EIR[GRA] assertion, the pause frame is transmitted. On completion of pause frame transmission, flow control pause (TCR[TFC_PAUSE]) and TCR[GTS] are cleared internally.

The user must specify the desired pause duration in the OPD register.

Note that when the transmitter is paused due to receiver/microcontroller pause frame detection, transmit flow control pause (TCR[TFC_PAUSE]) still may be asserted and will cause the transmission of a single pause frame. In this case, the EIR[GRA] interrupt will not be asserted.

19.3.11 Inter-Packet Gap (IPG) Time

The minimum inter-packet gap time for back-to-back transmission is 96 bit times. After completing a transmission or after the backoff algorithm completes, the transmitter waits for carrier sense to be negated before starting its 96 bit time IPG counter. Frame transmission may begin 96 bit times after carrier sense is negated if it stays negated for at least 60 bit times. If carrier sense asserts during the last 36 bit times, it will be ignored and a collision will occur.

The receiver receives back-to-back frames with a minimum spacing of at least 28 bit times. If an inter-packet gap between receive frames is less than 28 bit times, the following frame may be discarded by the receiver.

19.3.12 Collision Handling

If a collision occurs during frame transmission, the Ethernet controller will continue the transmission for at least 32 bit times, transmitting a JAM pattern consisting of 32 ones. If the collision occurs during the preamble sequence, the JAM pattern will be sent after the end of the preamble sequence.

If a collision occurs within 512 bit times, the retry process is initiated. The transmitter waits a random number of slot times. A slot time is 512 bit times. If a collision occurs after 512 bit times, then no retransmission is performed and the end of frame buffer is closed with a Late Collision (LC) error indication.

19.3.13 Internal and External Loopback

Both internal and external loopback are supported by the Ethernet controller. In loopback mode, both of the FIFOs are used and the FEC actually operates in a full-duplex fashion. Both internal and external loopback are configured using combinations of the LOOP and DRT bits in the RCR register and the FDEN bit in the TCR register.

For both internal and external loopback set FDEN = 1.

For internal loopback, set RCR[LOOP] = 1 and RCR[DRT] = 0. FEC_TXEN and FEC_TXER will not assert during internal loopback. During internal loopback, the transmit/receive data rate is higher than in normal operation because the internal internal bus clock is used by the transmit and receive blocks instead of the clocks from the external transceiver. This will cause an increase in the required system bus bandwidth for transmit and receive data being DMA'd to/from external memory. It may be necessary to pace the frames on the transmit side and/or limit the size of the frames to prevent transmit FIFO underrun and receive FIFO overflow.

For external loopback, set RCR[LOOP] = 0, RCR[DRT] = 0 and configure the external transceiver for loopback.

19.3.14 Ethernet Error-Handling Procedure

The Ethernet controller reports frame reception and transmission error conditions using the FEC RxBDs, the EIR register, and the MIB block counters.

19.3.14.1 Transmission Errors

19.3.14.1.1 Transmitter Underrun

If this error occurs, the FEC sends 32 bits that ensure a CRC error and stops transmitting. All remaining buffers for that frame are then flushed and closed. The UN bit is set in the EIR. The FEC will then continue to the next transmit buffer descriptor and begin transmitting the next frame.

The UN interrupt will be asserted if enabled in the EIMR register.

19.3.14.1.2 Retransmission Attempts Limit Expired

When this error occurs, the FEC terminates transmission. All remaining buffers for that frame are flushed and closed, and the RL bit is set in the EIR. The FEC will then continue to the next transmit buffer descriptor and begin transmitting the next frame.

The RL interrupt will be asserted if enabled in the EIMR register.

19.3.14.1.3 Late Collision

When a collision occurs after the slot time (512 bits starting at the Preamble), the FEC terminates transmission. All remaining buffers for that frame are flushed and closed, and the LC bit is set in the EIR register. The FEC will then continue to the next transmit buffer descriptor and begin transmitting the next frame.

The LC interrupt will be asserted if enabled in the EIMR register.

19.3.14.1.4 Heartbeat

Some transceivers have a self-test feature called ‘heartbeat’ or ‘signal quality error.’ To signify a good self-test, the transceiver indicates a collision to the FEC within 4 microseconds after completion of a frame transmitted by the Ethernet controller. This indication of a collision does not imply a real collision error on the network, but is rather an indication that the transceiver still seems to be functioning properly. This is called the heartbeat condition.

If the HBC bit is set in the TCR register and the heartbeat condition is not detected by the FEC after a frame transmission, then a heartbeat error occurs. When this error occurs, the FEC closes the buffer, sets the HB bit in the EIR register, and generates the HBERR interrupt if it is enabled.

19.3.14.2 Reception Errors

19.3.14.2.1 Overrun Error

If the receive block has data to put into the receive FIFO and the receive FIFO is full, the FEC sets the OV bit in the RxBD. All subsequent data in the frame will be discarded and subsequent frames may also be discarded until the receive FIFO is serviced by the DMA and space is made available. At this point the receive frame/status word is written into the FIFO with the OV bit set. This frame must be discarded by the driver.

19.3.14.2.2 Non-Octet Error (Dribbling Bits)

The Ethernet controller handles up to seven dribbling bits when the receive frame terminates past an non-octet aligned boundary. Dribbling bits are not used in the CRC calculation. If there is a CRC error, then the frame non-octet aligned (NO) error is reported in the RxBD. If there is no CRC error, then no error is reported.

19.3.14.2.3 CRC Error

When a CRC error occurs with no dribble bits, the FEC closes the buffer and sets the CR bit in the RxBD. CRC checking cannot be disabled, but the CRC error can be ignored if checking is not required.

19.3.14.2.4 Frame Length Violation

When the receive frame length exceeds MAX_FL bytes the BABR interrupt will be generated, and the LG bit in the end of frame RxBD will be set. The frame is not truncated unless the frame length exceeds 2032 bytes.

19.3.14.2.5 Truncation

When the receive frame length exceeds 2032 bytes the frame is truncated and the TR bit is set in the receive BD.

Chapter 20

Watchdog Timer Module

20.1 Introduction

The watchdog timer (WDT) is a 16-bit timer used to help software recover from runaway code. The watchdog timer has a free-running down-counter (watchdog counter) that generates a reset on underflow. To prevent a reset, software must periodically restart the countdown by servicing the watchdog.

20.1.1 Low-Power Mode Operation

This subsection describes the operation of the watchdog module in low-power modes and halted mode of operation (by issuing a HALT instruction). Low-power modes are described in [Chapter 8, “Power Management.”](#) [Table 20-1](#) shows the watchdog module operation in the low-power modes, and shows how this module may facilitate exit from each mode.

Table 20-1. Watchdog Module Operation in Low-power Modes

Low-power Mode	Watchdog Operation	Mode Exit
Wait	Normal if WCR[WAIT] cleared, stopped otherwise	Upon Watchdog reset
Doze	Normal if WCR[DOZE] cleared, stopped otherwise	Upon Watchdog reset
Stop	Stopped	No, only via external interrupts

In wait mode, with the watchdog control register’s WAIT bit (WCR[WAIT]) set, watchdog timer operation stops. In wait mode with the WCR[WAIT] bit cleared, the watchdog timer continues to operate normally. In doze mode with the WCR[DOZE] bit set, the watchdog timer module operation stops. In doze mode with the WCR[DOZE] bit cleared, the watchdog timer continues to operate normally. Watchdog timer operation stops in stop mode. When stop mode is exited, the watchdog timer continues to operate in its pre-stop mode state.

In halted mode (entered by issuing a HALT instruction or asserting $\overline{\text{BKPT}}$) with the WCR[HALTED] bit set, watchdog timer module operation stops. When halted mode is exited, watchdog timer operation continues from the state it was in before entering halted mode, but any updates made in halted mode remain. If the WCR[HALTED] bit is cleared, the watchdog timer continues to operate normally after executing a HALT instruction. This is a debug feature available for the user

20.1.2 Block Diagram

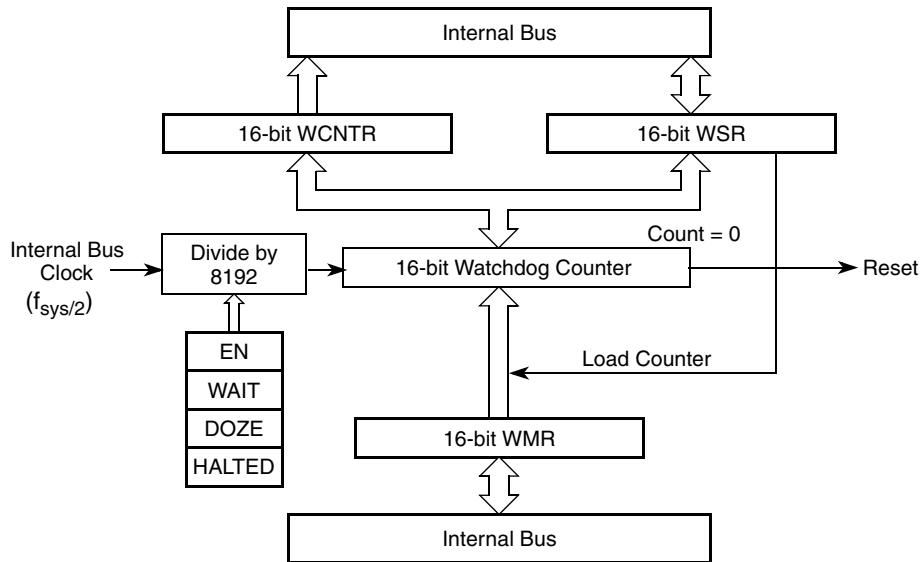


Figure 20-1. Watchdog Timer Block Diagram

20.2 Memory Map/Register Definition

This subsection describes the memory map and registers for the watchdog timer. Refer to [Table 20-2](#) for an overview of the watchdog memory map.

NOTE

Longword accesses to any of the watchdog timer registers will result in a bus error. Only byte and word accesses are allowed.

Table 20-2. Watchdog Timer Module Memory Map

Address ¹	Register	Width (bits)	Access	Reset Value	Section/Page
Supervisor Only Access					
0xFC08_C000	Watchdog Control Register (WCR)	16	R/W	0x000F	20.2.1/20-2
0xFC08_C002	Watchdog Modulus Register (WMR)	16	R/W	0xFFFF	20.2.2/20-3
Supervisor/User Access					
0xFC08_C004	Watchdog Count Register (WCNTR)	16	R	0xFFFF	20.2.3/20-4
0xFC08_C006	Watchdog Service Register (WSR)	16	R/W	0x0000	20.2.4/20-4

¹ Addresses not assigned to a register and undefined register bits are reserved for expansion. Write accesses to these reserved address spaces and reserved register bits have no effect.

20.2.1 Watchdog Control Register (WCR)

The 16-bit WCR configures watchdog timer operation.

Address: 0xFC08_C000

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	0	0	0	0	0	0	0	0	WAIT	DOZE	HALTED	EN
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1

Figure 20-2. Watchdog Control Register (WCR)

Table 20-3. WCR Field Descriptions

Field	Description
15–4	Reserved, should be cleared.
3 WAIT	Wait mode bit. Controls the function of the watchdog timer in wait mode. Once written, the WAIT bit is not affected by further writes except in halted mode. Reset sets WAIT. 0 Watchdog timer not affected in wait mode 1 Watchdog timer stopped in wait mode
2 DOZE	Doze mode bit. Controls the function of the watchdog timer in doze mode. Once written, the DOZE bit is not affected by further writes except in halted mode. Reset sets DOZE. 0 Watchdog timer not affected in doze mode 1 Watchdog timer stopped in doze mode
1 HALTED	Halted mode bit. Controls the function of the watchdog timer in halted/debug mode. Once written, the HALTED bit is not affected by further writes except in halted mode. During halted mode, watchdog timer registers can be written and read normally. When halted mode is exited, timer operation continues from the state it was in before entering halted mode, but any updates made in halted mode remain. If a write-once register is written for the first time in halted mode, the register is still writable when halted mode is exited. 0 Watchdog timer not affected in halted mode 1 Watchdog timer stopped in halted mode Note: Changing the HALTED bit from 1 to 0 during halted mode starts the watchdog timer. Changing the HALTED bit from 0 to 1 during halted mode stops the watchdog timer.
0 EN	Watchdog enable bit. Enables the watchdog timer. Once written, the EN bit is not affected by further writes except in halted mode. When the watchdog timer is disabled, the watchdog counter and prescaler counter are held in a stopped state. Once disabled, the watchdog cannot be re-enabled. 0 Watchdog timer disabled 1 Watchdog timer enabled

20.2.2 Watchdog Modulus Register (WMR)

The WMR register determines the timer modulus reload value.

Address: 0xFC08_C002

Access: Supervisor read/write

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

Figure 20-3. Watchdog Modulus Register (WMR)

Table 20-4. WMR Field Descriptions

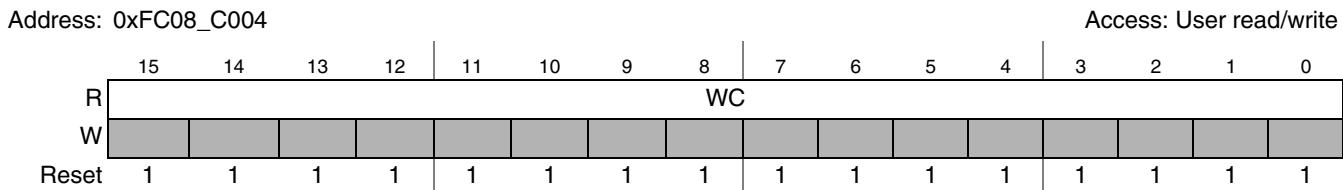
Field	Description
15–0 WM	Watchdog modulus. Contains the modulus that is reloaded into the watchdog counter by a service sequence. Once written, the WM[15:0] field is not affected by further writes except in halted mode. Writing to WMR immediately loads the new modulus value into the watchdog counter. The new value is also used at the next and all subsequent reloads. Reading WMR returns the value in the modulus register. Reset initializes the WM[15:0] field to 0xFFFF. Note: The prescaler counter is reset anytime a new value is loaded into the watchdog counter and also during reset.

20.2.3 Watchdog Count Register (WCNTR)

The WCNTR register provides visibility to the watchdog counter value. The timeout value for the watchdog timer is determined from the following equation:

$$\text{WDT timeout} = \frac{8192 \times \text{WCNTR}}{f_{\text{sys}/2}} \quad \text{Eqn. 20-1}$$

Thus, the maximum timeout will be $8192 \times 2^{16} / f_{\text{sys}/2} = 6.44$ seconds at 83.3 MHz internal bus frequency.

**Figure 20-4. Watchdog Count Register (WCNTR)****Table 20-5. WCNTR Field Descriptions**

Field	Description
15–0 WC	Watchdog count field. Reflects the current value in the watchdog counter. Reading the 16-bit WCNTR with two 8-bit reads is not guaranteed to return a coherent value. Writing to WCNTR has no effect, and write cycles are terminated normally.

20.2.4 Watchdog Service Register (WSR)

When the watchdog timer is enabled, writing 0x5555 and then 0xAAAA to WSR before the watchdog counter times out prevents a reset. If WSR is not serviced before the timeout, the watchdog timer sends a signal to the reset controller module that sets the RSR[WDR] bit and asserts a system reset.

Both writes must occur in the order listed before the timeout, but any number of instructions can be executed between the two writes. However, writing any value other than 0x5555 or 0xAAAA to WSR resets the servicing sequence, requiring both values to be written to keep the watchdog timer from causing a reset.

Address: 0xFC08_C006

Access: User read/write

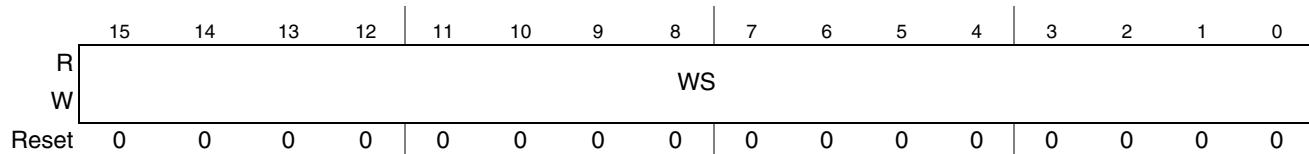


Figure 20-5. Watchdog Service Register (WSR)

Chapter 21

Programmable Interrupt Timer Modules (PIT0–PIT1)

21.1 Introduction

This chapter describes the operation of the two programmable interrupt timer modules: PIT0–PIT1.

21.1.1 Overview

Each PIT is a 16-bit timer that provides precise interrupts at regular intervals with minimal processor intervention. The timer can either count down from the value written in the modulus register, or it can be a free-running down-counter.

21.1.2 Block Diagram

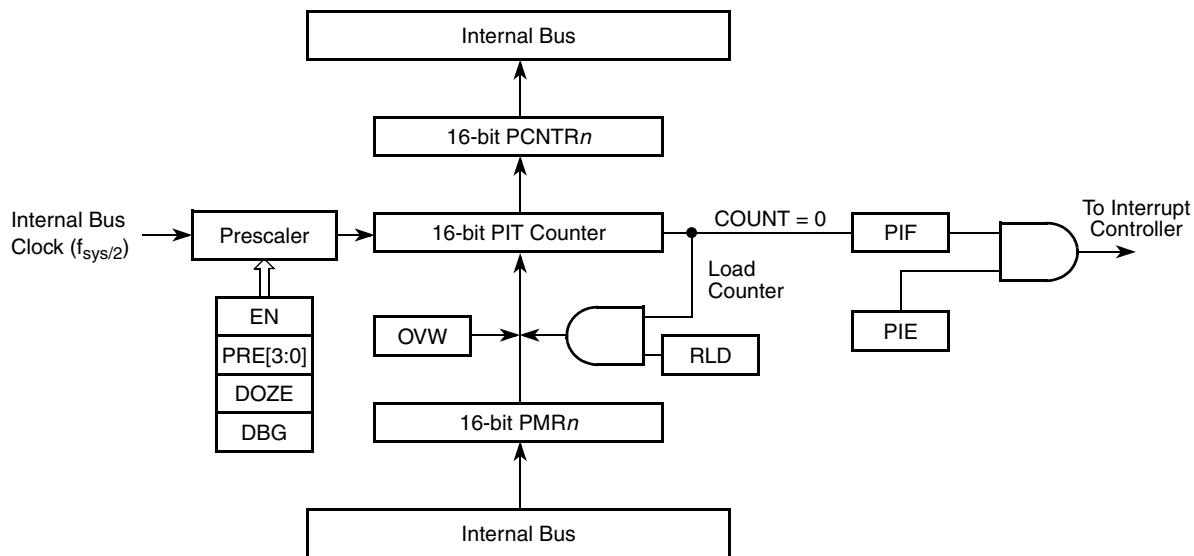


Figure 21-1. PIT Block Diagram

21.1.3 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 8, “Power Management.”](#) Table 21-1 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 21-1. 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, will 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 will be 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, the PIT continues to operate 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.

21.2 Memory Map/Register Definition

This section contains a memory map, shown in [Table 21-2](#), and describes the register structure for PIT0–PIT1.

NOTE

Longword accesses to any of the programmable interrupt timer registers will result in a bus error. Only byte and word accesses are allowed.

Table 21-2. Programmable Interrupt Timer Modules Memory Map

Address	Register	Width (bits)	Access ¹	Reset Value	Section/Page
PIT 0 PIT 1					
Supervisor Access Only Registers²					
0xFC08_0000 0xFC08_4000	PIT Control and Status Register (PCSR n)	16	R/W	0x0000	21.2.1/21-3
0xFC08_0002 0xFC08_4002	PIT Modulus Register (PMR n)	16	R/W	0xFFFF	21.2.2/21-4

Table 21-2. Programmable Interrupt Timer Modules Memory Map (continued)

Address	Register	Width (bits)	Access ¹	Reset Value	Section/Page
PIT 0 PIT 1					
User/Supervisor Access Registers					
0xFC08_0004 0xFC08_4004	PIT Count Register (PCNTR n)	16	R	0xFFFF	21.2.3/21-5

¹ Accesses to reserved address locations have no effect and result in a cycle termination transfer error.

² User mode accesses to supervisor only addresses have no effect and result in a cycle termination transfer error.

21.2.1 PIT Control and Status Register (PCSR n)

The PCSR n registers configure the corresponding timer's operation.

Address: 0xFC08_0000 (PCSR0)
0xFC08_4000 (PCSR1)

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	PRE				0	DOZE	DBG	OVW	PIE	PIF		
W					0	0	0	0	0	0	0	0	0	w1c	RLD	EN
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 21-2. PIT Control and Status Register (PCSR n)**Table 21-3. PCSR n Field Descriptions**

Field	Description
15–12	Reserved, should 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

Table 21-3. PCSR n Field Descriptions (continued)

Field	Description
7	Reserved, should 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 the 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 Note: 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 the 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 the 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.
1 RLD	Reload bit. The read/write reload bit enables loading the value of PMR n into the 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 the PIT is disabled, the counter and prescaler are held in a stopped state. This bit is read anytime, write anytime. 0 PIT disabled 1 PIT enabled

21.2.2 PIT Modulus Register (PMR n)

The 16-bit read/write PMR n contains the timer modulus value that is 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)	Access: Supervisor read/write																																																																												
<table border="1"> <tr> <td>15</td><td>14</td><td>13</td><td>12</td><td> </td><td>11</td><td>10</td><td>9</td><td>8</td><td> </td><td>7</td><td>6</td><td>5</td><td>4</td><td> </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></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>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><td></td><td></td><td></td> </tr> <tr> <td>Reset</td><td>1</td><td>1</td><td>1</td><td>1</td><td> </td><td>1</td><td>1</td><td>1</td><td></td><td>1</td><td>1</td><td>1</td><td>1</td><td> </td><td>1</td><td>1</td><td>1</td><td>1</td> </tr> </table>	15	14	13	12		11	10	9	8		7	6	5	4		3	2	1	0	R																			W																			Reset	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																																																																													
W																																																																													
Reset	1	1	1	1		1	1	1		1	1	1	1		1	1	1	1																																																											

Figure 21-3. PIT Modulus Register (PMR n)

21.2.3 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 to be coherent. Writing to PCNTR n has no effect, and write cycles are terminated normally.

Address: 0xFC08_0004 (PCNTR0) 0xFC08_4004 (PCNTR1)	Access: User read only																																																																												
<table border="1"> <tr> <td>15</td><td>14</td><td>13</td><td>12</td><td> </td><td>11</td><td>10</td><td>9</td><td>8</td><td> </td><td>7</td><td>6</td><td>5</td><td>4</td><td> </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></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>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><td></td><td></td><td></td> </tr> <tr> <td>Reset</td><td>1</td><td>1</td><td>1</td><td>1</td><td> </td><td>1</td><td>1</td><td>1</td><td></td><td>1</td><td>1</td><td>1</td><td>1</td><td> </td><td>1</td><td>1</td><td>1</td><td>1</td> </tr> </table>	15	14	13	12		11	10	9	8		7	6	5	4		3	2	1	0	R																			W																			Reset	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																																																																													
W																																																																													
Reset	1	1	1	1		1	1	1		1	1	1	1		1	1	1	1																																																											

Figure 21-4. PIT Count Register (PCNTR n)

21.3 Functional Description

This section describes the PIT functional operation.

21.3.1 Set-and-Forget Timer Operation

This mode of operation is selected when the RLD bit in the PCSR register is set.

When the PIT counter reaches a count of 0x0000, the PIF flag is set in PCSR n . The value in the modulus register is loaded into the counter, and the counter begins decrementing toward 0x0000. If the PCSR n [PIE] bit is set, the PIF flag issues an interrupt request to the CPU.

When the PCSR n [OVW] bit is set, the counter can be directly initialized by writing to PMR n without having to wait for the count to reach 0x0000.

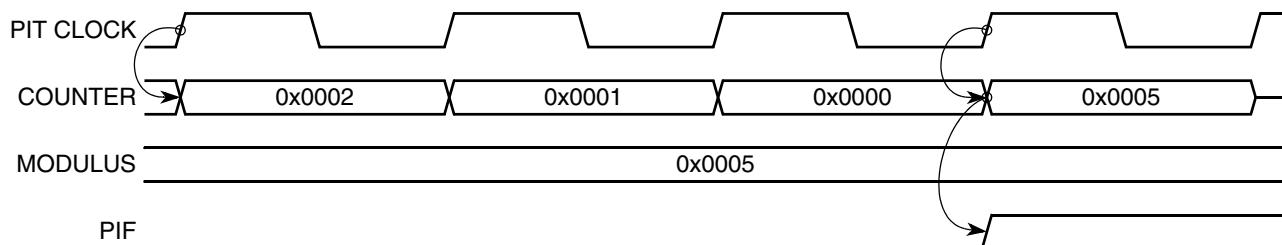


Figure 21-5. Counter Reloading from the Modulus Latch

21.3.2 Free-Running Timer Operation

This mode of operation is selected when the PCSR n [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, the PCSR n [PIF] flag is set. If the PCSR n [PIE] bit is set, the PIF flag issues an interrupt request to the CPU.

When the PCSR n [OVW] bit is set, the counter can be directly initialized by writing to PMR n without having to wait for the count to reach 0x0000.

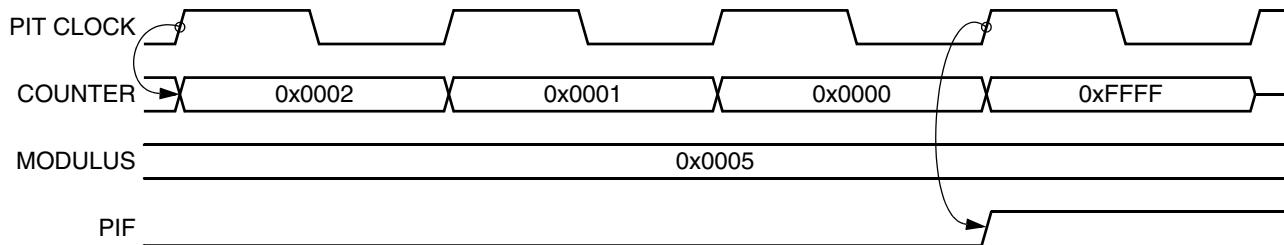


Figure 21-6. Counter in Free-Running Mode

21.3.3 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.

$$\text{Timeout period} = \text{PRE}[3:0] \times (\text{PM}[15:0] + 1) \times f_{\text{sys}/2}$$

Eqn. 21-1

21.3.4 Interrupt Operation

Table 21-4 shows the interrupt request generated by the PIT.

Table 21-4. 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 22

DMA Timers (DTIM0–DTIM3)

22.1 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*’ is used throughout this section to refer to registers or signals associated with one of the four identical timer modules: DTIM0, DTIM1, DTIM2, or DTIM3.

22.1.1 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 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*}). 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 initiate 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 13, “General Purpose I/O Module”](#)) prior to configuring the DMA Timers.

Figure 22-1 is a block diagram of one of the four identical timer modules.

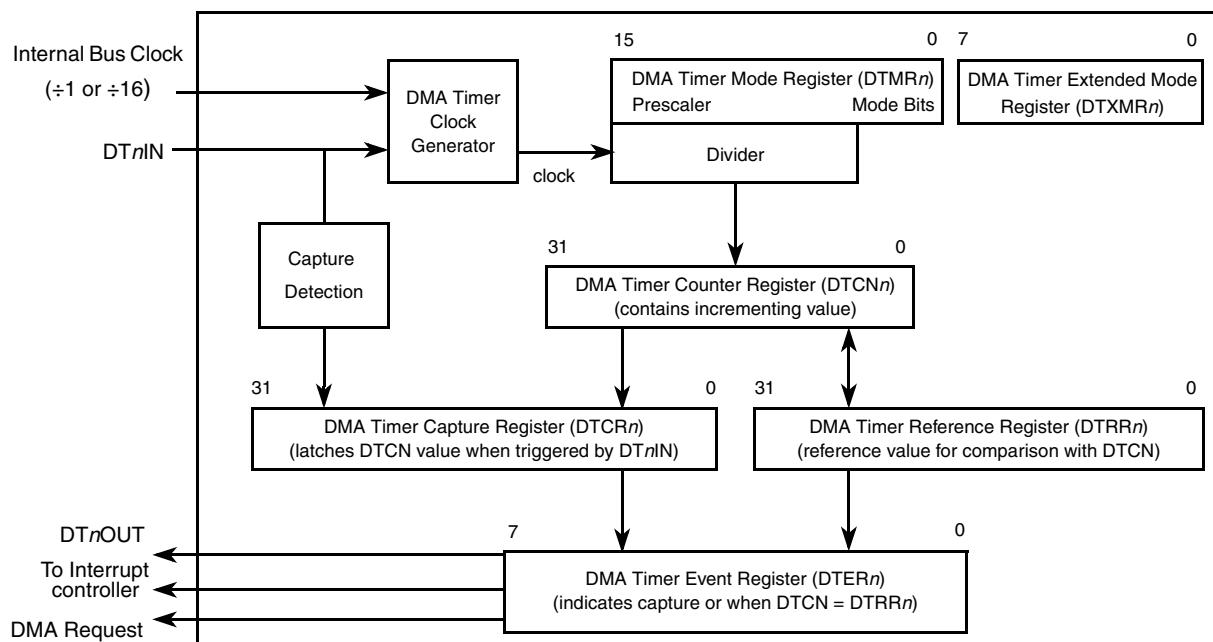


Figure 22-1. DMA Timer Block Diagram

22.1.2 Features

Each DMA timer module has the following features:

- Maximum timeout period of 211,190 seconds at 83.33 MHz (~58 hours)
- 12-ns resolution at 83.33 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

22.2 Memory Map/Register Definition

The timer module registers, shown in Table 22-1, can be modified at any time.

Table 22-1. 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 <i>n</i> Mode Register (DTMR <i>n</i>)	16	R/W	0x0000	22.2.1/22-3
0xFC07_0002 0xFC07_4002 0xFC07_8002 0xFC07_C002	DMA Timer <i>n</i> Extended Mode Register (DTXMR <i>n</i>)	8	R/W	0x00	22.2.2/22-4
0xFC07_0003 0xFC07_4003 0xFC07_8003 0xFC07_C003	DMA Timer <i>n</i> Event Register (DTER <i>n</i>)	8	R/W	0x00	22.2.3/22-5
0xFC07_0004 0xFC07_4004 0xFC07_8004 0xFC07_C004	DMA Timer <i>n</i> Reference Register (DTRR <i>n</i>)	32	R/W	0x1111_1111	22.2.4/22-6
0xFC07_0008 0xFC07_4008 0xFC07_8008 0xFC07_C008	DMA Timer <i>n</i> Capture Register (DTCR <i>n</i>)	32	R/W	0x0000_0000	22.2.5/22-7
0xFC07_000C 0xFC07_400C 0xFC07_800C 0xFC07_C00C	DMA Timer <i>n</i> Counter Register (DTCN <i>n</i>)	32	R	0x0000_0000	22.2.6/22-7

22.2.1 DMA Timer Mode Registers (DTMR*n*)

DTMRs, shown in [Figure 22-2](#), program the prescaler and various timer modes.

Address 0xFC07_0000 (DTMR0)
 0xFC07_4000 (DTMR1)
 0xFC07_8000 (DTMR2)
 0xFC07_C000 (DTMR3)

Access: User read/write

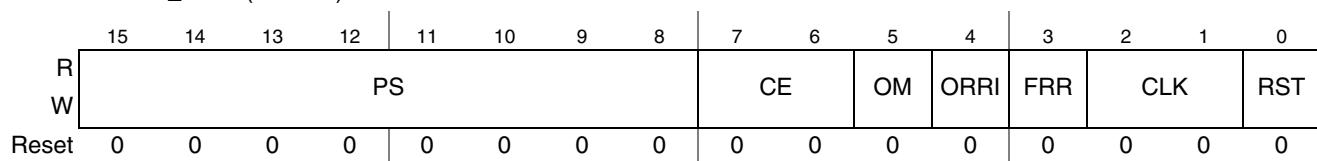
Figure 22-2. DMA Timer Mode Registers (DTMR*n*)

Table 22-2. DTMR n Field Descriptions

Field	Description
15–8 PS	Prescaler value. The prescaler is programmed to divide the clock input (internal bus clock/(16 or 1) or clock on DT n IN) by values from 1 (PS = 0x00) to 256 (PS = 0xFF).
7–6 CE	Capture edge. 00 Disable capture event output 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 (12-ns resolution at 83.33 MHz). 1 Toggle output.
4 ORRI	Output reference request, interrupt enable. If ORRI is set when DTER n [REF] = 1, a DMA request or an interrupt occurs, depending on the value of DTXMR n [DMAEN] (DMA request if =1, interrupt if =0). 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 0 Free run. Timer count continues to increment after reaching the reference value. 1 Restart. Timer count is reset immediately after reaching the reference value.
2–1 CLK	Input clock source for the timer 00 Stop count 01 Internal bus clock divided by 1 10 Internal bus clock divided by 16. Note that this clock source is not synchronized with the timer; thus 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 still be written while RST = 0. A transition of RST from 1 to 0 resets register values. The timer counter is not clocked unless the timer is enabled. 0 Reset timer (software reset) 1 Enable timer

22.2.2 DMA Timer Extended Mode Registers (DTXMR n)

The DTXMR n register programs 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

	7	6	5	4	3	2	1
R	DMAEN	0	0	0	0	0	0
W							
Reset:	0	0	0	0	0	0	0

Figure 22-3. DMA Timer Extended Mode Registers (DTXMR n)

Table 22-3. 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
6–1	Reserved, should be cleared.
0 MODE16	Selects the increment mode for the timer. MODE16 = 1 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 are still compared to the reference value. 0 Increment timer by 1 1 Increment timer by 65,537

22.2.3 DMA Timer Event Registers (DTER n)

DTER n , shown in [Figure 22-4](#), reports capture or reference events by setting DTER n [CAP] or DTER n [REF]. This reporting is done regardless of the corresponding DMA request or interrupt enable values, DTXMR n [DMAEN] and DTMR n [ORRI,CE].

Writing a 1 to either 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, the REF and CAP bits should be cleared 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, the processing of the DMA data transfer automatically clears both 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 0 6 0 5 0 4 0	3 0 2 0 1 REF 0 0	0 0 0 w1c 0 0 0 0
W			
Reset:	0 0 0 0	0 0 0 0	w1c 0 0 0

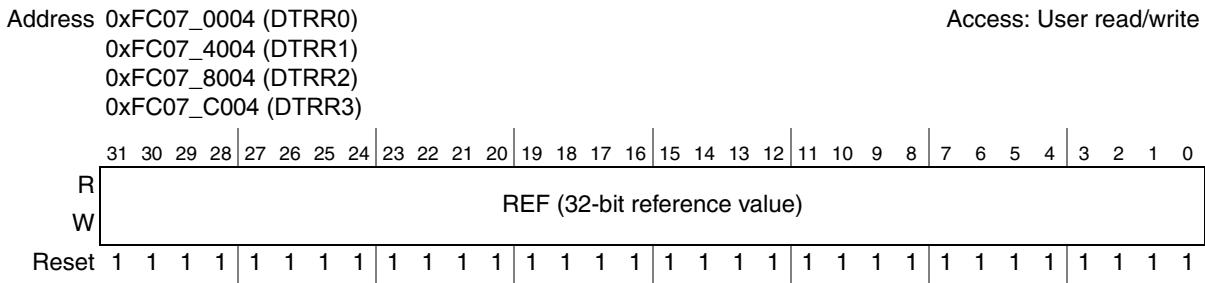
Figure 22-4. DMA Timer Event Registers (DTER n)

Table 22-4. DTER n Field Descriptions

Field	Description			
7–2	Reserved, should be cleared.			
1 REF	Output reference event. The counter value, DTCN n , equals the reference value, DT R R n . Writing a one to REF clears the event condition. Writing a zero has no effect.			
REF	DTMR n [ORRI]	DTXMR n [DMAEN]		
0	X	X	No event	
1	0	0	No request asserted	
1	0	1	No request asserted	
1	1	0	Interrupt request asserted	
1	1	1	DMA request asserted	
0 CAP	Capture event. The counter value has been latched into DTCR n . Writing a one to CAP clears the event condition. Writing a zero has no effect.			
CAP	DTMR n [CE]	DTXMR n [DMAEN]		
0	XX	X	No event	
1	00	0	Disable capture event output	
1	00	1	Disable capture event output	
1	01	0	Capture on rising edge & trigger interrupt	
1	01	1	Capture on rising edge & trigger DMA	
1	10	0	Capture on falling edge & trigger interrupt	
1	10	1	Capture on falling edge & trigger DMA	
1	11	0	Capture on any edge & trigger interrupt	
1	11	1	Capture on any edge & trigger DMA	

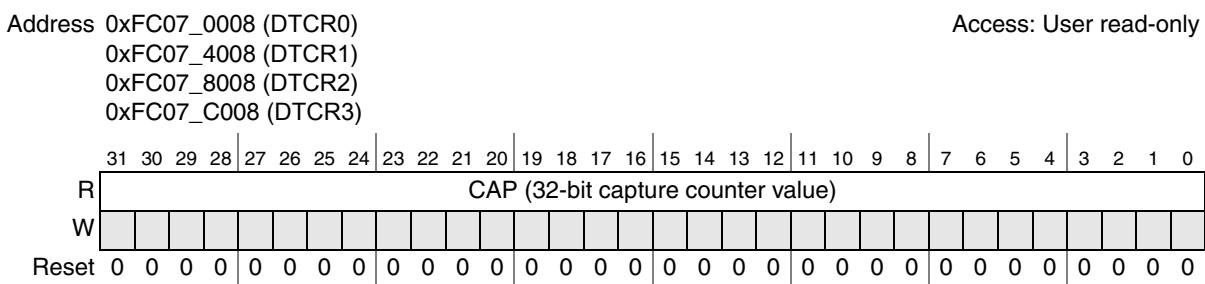
22.2.4 DMA Timer Reference Registers (DT R R n)

Each DT R R n , shown in [Figure 22-5](#), contains the reference value compared with the respective free-running timer counter (DTCN n) as part of the output-compare function. The reference value is not matched until DTCN n equals DT R R n , and the prescaler indicates that DTCN n should be incremented again. Thus, the reference register is matched after DT R R n +1 time intervals.

Figure 22-5. DMA Timer Reference Registers (DTRR n)

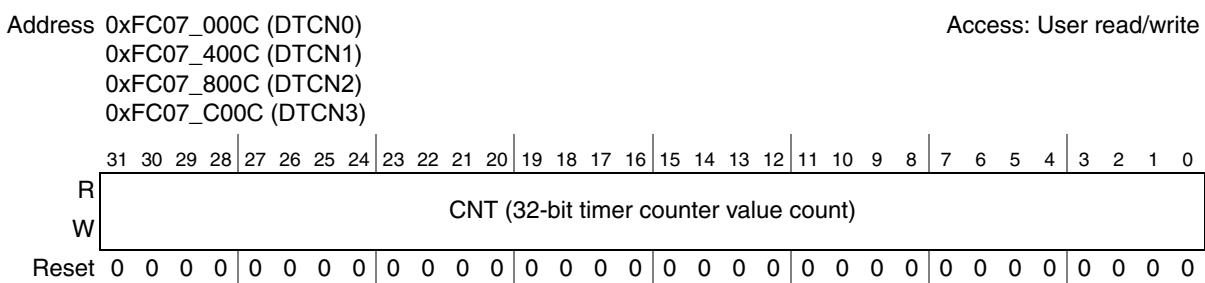
22.2.5 DMA Timer Capture Registers (DTCR n)

Each DTCR n latches the corresponding DTCN n 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 will result if DT n IN is set as the clock source when the input capture mode is used.

Figure 22-6. DMA Timer Capture Registers (DTCR n)

22.2.6 DMA Timer Counters (DTCN n)

The current value of the 32-bit DTCNs can be read at anytime without affecting counting. Any write to DTCN n clears it. The timer counter increments on the clock source rising edge (internal bus clock $\div 1$, internal bus clock $\div 16$, or DT n IN).

Figure 22-7. DMA Timer Counters (DTCN n)

22.3 Functional Description

22.3.1 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, DT_nIN. DT_nIN is synchronized to the internal bus clock, and the synchronization delay is between two and three internal bus clocks. The corresponding DTMR_n[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, DTC_n.

22.3.2 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 DT_nIN 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] is set and DTXMR_n[DMAEN] is one, a DMA request is asserted. If DTER_n[CAP] is set and DTXMR_n[DMAEN] is zero, an interrupt is asserted.

22.3.3 Reference Compare

Each DMA timer can be configured to count up to a reference value, at which point DTER_n[REF] is set. If DTMR_n[ORRI] is one and DTXMR_n[DMAEN] is zero, an interrupt is asserted. If DTMR_n[ORRI] is one and DTXMR_n[DMAEN] is one, 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.

22.3.4 Output Mode

When a timer reaches the reference value selected by DTRR, it can send an output signal on DT_nOUT. DT_nOUT can be an active-low pulse or a toggle of the current output, as selected by the DTMR_n[OM] bit.

22.4 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_nIN
 - Configure DT_nOUT 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_nIN, the maximum value of DT_nIN 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 will result.

- The 8-bit DTMR n [PS] prescaler value is set.
- Using DTMR n [RST], the counter is cleared and started.
- Timer events are either handled with an interrupt service routine, a DMA request, or by a software polling mechanism.

22.4.1 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,TMRO
move.l #0x0000,D0;writing to the timer counter with any
move.l D0,TCN0 ;value resets it to zero
move.l #0xAFAF,D0 ;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 TMRO,D0          ;save the contents of TMRO while setting
    bset #0,D0              ;the 0 bit. This enables timer 0 and starts counting
    move.w D0,TMRO          ;load the value back into the register, setting TMRO[RST]

```

DMA Timers (DTIM0–DTIM3)

```
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
```

22.4.2 Calculating Time-Out Values

The formula below determines time-out periods for various reference values:

$$\text{Timeout period} = (1/\text{clock frequency}) \times (1 \text{ or } 16) \times (\text{DTMR}_n[\text{PS}] + 1) \times (\text{DTRR}_n[\text{REF}] + 1) \quad \text{Eqn. 22-1}$$

When calculating time-out periods, add 1 to the prescaler to simplify calculating, because $\text{DTMR}_n[\text{PS}] = 0x00$ yields a prescaler of 1, and $\text{DTMR}_n[\text{PS}] = 0xFF$ yields a prescaler of 256.

For example, if a 83.33-MHz timer clock is divided by 16, $\text{DTMR}_n[\text{PS}] = 0x7F$, and the timer is referenced at $0x13DC3$ (81347 decimal), the time-out period is as follows:

$$\text{Timeout period} = \frac{1}{83.3 \times 10^6} \times 16 \times (127 + 1) \times (81347 + 1) = 2.00\text{s} \quad \text{Eqn. 22-2}$$

Chapter 23

Queued Serial Peripheral Interface (QSPI)

23.1 Introduction

This chapter describes the queued serial peripheral interface (QSPI) module. Following a feature set overview is a description of operation including details of the QSPI's internal RAM organization. The chapter concludes with the programming model and a timing diagram.

23.1.1 Block Diagram

Figure 23-1 illustrates the QSPI module.

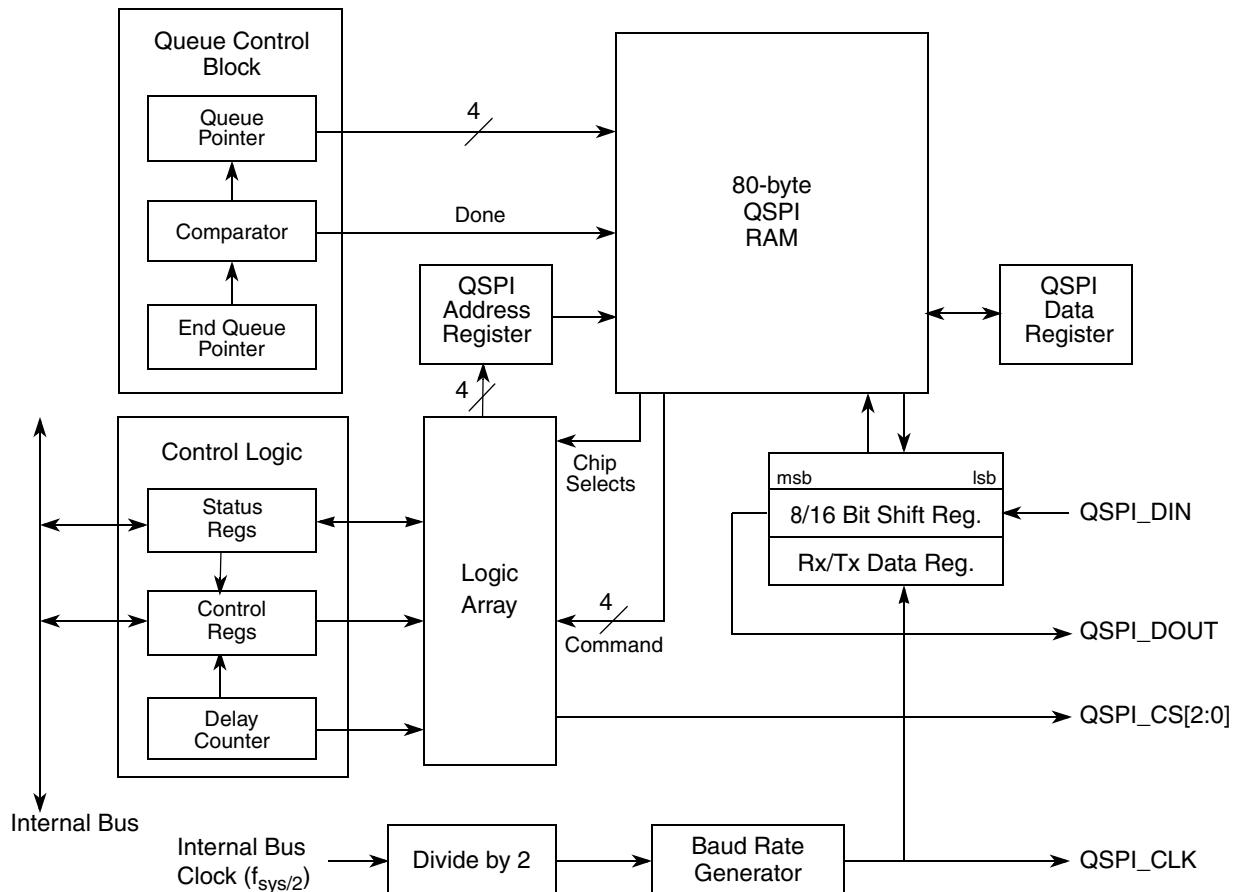


Figure 23-1. QSPI Block Diagram

23.1.2 Overview

The queued serial peripheral interface module provides a serial peripheral interface with queued transfer capability. It allows users to queue up to 16 transfers at once, eliminating CPU intervention between transfers. Transfer RAM in the QSPI is indirectly accessible using address and data registers.

NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to [Chapter 13, “General Purpose I/O Module”](#)) prior to configuring the QSPI Module.

23.1.3 Features

Features include the following:

- Programmable queue to support up to 16 transfers without user intervention
- Supports transfer sizes of 8 to 16 bits in 1-bit increments
- Four peripheral chip-select lines for control of up to 15 devices (All four chip selects may not be available on all devices. See [“](#) for details on which chip-selects are pinned-out.)
- Baud rates from 162.7 Kbps to 20.75 Mbps at 83.33 MHz internal bus frequency
- Programmable delays before and after transfers
- Programmable QSPI clock phase and polarity
- Supports wraparound mode for continuous transfers

23.1.4 Modes of Operation

Because the QSPI module only operates in master mode, the master bit in the QSPI mode register (QMR[MSTR]) must be set for the QSPI to function properly. If the master bit is not set, QSPI activity will be indeterminate. The QSPI can initiate serial transfers but cannot respond to transfers initiated by other QSPI masters.

23.2 External Signal Description

The module provides access to as many as 15 devices with a total of seven signals: QSPI_DOUT, QSPI_DIN, QSPI_CLK, QSPI_CS0, QSPI_CS1, and QSPI_CS2, and QSPI_CS3.

Peripheral chip-select signals, QSPI_CS n , are used to select an external device as the source or destination for serial data transfer. Signals are asserted whenever a command in the queue is executed. More than one chip-select signal can be asserted simultaneously.

Although QSPI_CS n will function as simple chip selects in most applications, up to 15 devices can be selected by decoding them with an external 4-to-16 decoder.

Table 23-1. QSPI Input and Output Signals and Functions

Signal Name	Hi-Z or Actively Driven	Function
QSPI Data Output (QSPI_DOUT)	Configurable	Serial data output from QSPI
QSPI Data Input (QSPI_DIN)	N/A	Serial data input to QSPI
Serial Clock (QSPI_CLK)	Actively driven	Clock output from QSPI
Peripheral Chip Selects (QSPI_CS _n)	Actively driven	Peripheral selects

23.3 Memory Map/Register Definition

Table 23-2 is the QSPI register memory map. Reading reserved locations returns zeros.

Table 23-2. QSPI Memory Map

Address ¹	Register	Width (bits)	Access	Reset Value	Section/Page
0xFC05_C000	QSPI Mode Register (QMR)	16	R/W	0x0104	23.3.1/23-3
0xFC05_C004	QSPI Delay Register (QDLYR)	16	R/W	0x0404	23.3.2/23-5
0xFC05_C008	QSPI Wrap Register (QWR)	16	R/W ²	0x0000	23.3.3/23-6
0xFC05_C00C	QSPI Interrupt Register (QIR)	16	R/W ²	0x0000	23.3.4/23-6
0xFC05_C010	QSPI Address Register (QAR)	16	R/W ²	0x0000	23.3.5/23-8
0xFC05_C014	QSPI Data Register (QDR)	16	R/W	0x0000	23.3.6/23-8

¹ Addresses not assigned to a register and undefined register bits are reserved for expansion.

² See the register description for special cases. Some bits may be read- or write-only.

23.3.1 QSPI Mode Register (QMR)

The QMR, shown in Figure 23-2, determines the basic operating modes of the QSPI module. Parameters such as QSPI_CLK polarity and phase, baud rate, master mode operation, and transfer size are determined by this register. The data output high impedance enable, DOHIE, controls the operation of QSPI_DOUT between data transfers. When DOHIE is cleared, QSPI_DOUT is actively driven between transfers. When DOHIE is set, QSPI_DOUT assumes a high impedance state.

NOTE

Because the QSPI does not operate in slave mode, the master mode enable bit (QMR[MSTR]) must be set for the QSPI module to operate correctly.

Address: 0xFC05_C000																Access: User read/write								
15	14	13	12		11	10	9	8		7	6	5	4		3	2	1	0						
R W	MSTR	DOHIE			BITS		CPOL	CPHA																

Figure 23-2. QSPI Mode Register (QMR)

Table 23-3. QMR Field Descriptions

Field	Description																						
15 MSTR	Master mode enable. 0 Reserved, do not use. 1 The QSPI is in master mode. Must be set for the QSPI module to operate correctly.																						
14 DOHIE	Data output high impedance enable. Selects QSPI_DOUT mode of operation. 0 Default value after reset. QSPI_DOUT is actively driven between transfers. 1 QSPI_DOUT is high impedance between transfers.																						
13–10 BITS	Transfer size. Determines the number of bits to be transferred for each entry in the queue. <table border="1" data-bbox="669 559 1068 1098"> <thead> <tr> <th>BITS</th><th>Bits per Transfer</th></tr> </thead> <tbody> <tr><td>0000</td><td>16</td></tr> <tr><td>0001–0111</td><td>Reserved</td></tr> <tr><td>1000</td><td>8</td></tr> <tr><td>1001</td><td>9</td></tr> <tr><td>1010</td><td>10</td></tr> <tr><td>1011</td><td>11</td></tr> <tr><td>1100</td><td>12</td></tr> <tr><td>1101</td><td>13</td></tr> <tr><td>1110</td><td>14</td></tr> <tr><td>1111</td><td>15</td></tr> </tbody> </table>	BITS	Bits per Transfer	0000	16	0001–0111	Reserved	1000	8	1001	9	1010	10	1011	11	1100	12	1101	13	1110	14	1111	15
BITS	Bits per Transfer																						
0000	16																						
0001–0111	Reserved																						
1000	8																						
1001	9																						
1010	10																						
1011	11																						
1100	12																						
1101	13																						
1110	14																						
1111	15																						
9 CPOL	Clock polarity. Defines the clock polarity of QSPI_CLK. 0 The inactive state value of QSPI_CLK is logic level 0. 1 The inactive state value of QSPI_CLK is logic level 1.																						
8 CPHA	Clock phase. Defines the QSPI_CLK clock-phase. 0 Data captured on the leading edge of QSPI_CLK and changed on the following edge of QSPI_CLK. 1 Data changed on the leading edge of QSPI_CLK and captured on the following edge of QSPI_CLK.																						
7–0 BAUD	Baud rate divider. The baud rate is selected by writing a value in the range 2–255. A value of zero disables the QSPI. A value of 1 is an invalid setting. The desired QSPI_CLK baud rate is related to the internal bus clock and QMR[BAUD] by the following expression: $\text{QMR[BAUD]} = f_{\text{sys}/2} / (2 \times [\text{desired QSPI_CLK baud rate}])$																						

Figure 23-3 shows an example of a QSPI clocking and data transfer.

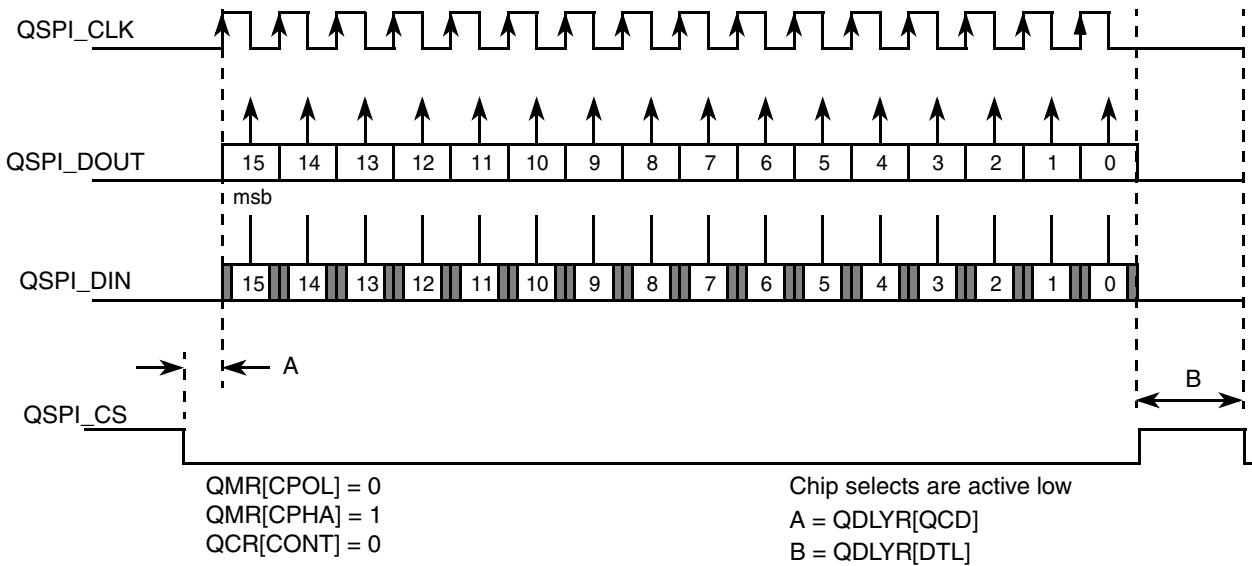


Figure 23-3. QSPI Clocking and Data Transfer Example

23.3.2 QSPI Delay Register (QDLYR)

Address: 0xFC05_C004

Access: User read/write

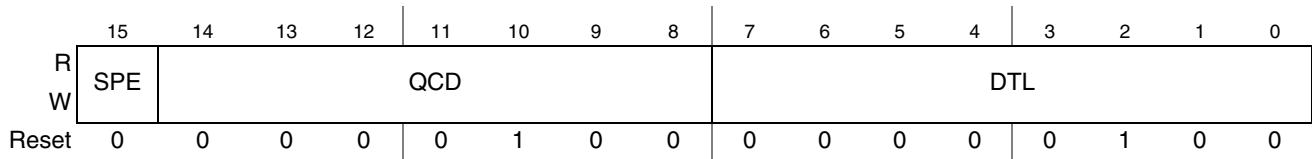


Figure 23-4. QSPI Delay Register (QDLYR)

Table 23-4. QDLYR Field Descriptions

Field	Description
15 SPE	QSPI enable. When set, the QSPI initiates transfers in master mode by executing commands in the command RAM. Automatically cleared by the QSPI when a transfer completes. The user can also clear this bit to abort transfer unless QIR[ABRTL] is set. The recommended method for aborting transfers is to set QWR[HALT].
14–8 QCD	QSPI_CLK delay. When the DSCK bit in the command RAM is set this field determines the length of the delay from assertion of the chip selects to valid QSPI_CLK transition. See Section 23.4.3, “Transfer Delays” for information on setting this bit field.
7–0 DTL	Delay after transfer. When the DT bit in the command RAM is set this field determines the length of delay after the serial transfer.

23.3.3 QSPI Wrap Register (QWR)

Address: 0xFC05_C008 Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	HALT	WREN	WRTO	CSIV	ENDQP				CPTQP				NEWQP			
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 23-5. QSPI Wrap Register (QWR)

Table 23-5. QWR Field Descriptions

Field	Description
15 HALT	Halt transfers. Assertion of this bit causes the QSPI to stop execution of commands once it has completed execution of the current command.
14 WREN	Wraparound enable. Enables wraparound mode. 0 Execution stops after executing the command pointed to by QWR[ENDQP]. 1 After executing command pointed to by QWR[ENDQP], wrap back to entry zero, or the entry pointed to by QWR[NEWQP] and continue execution.
13 WRTO	Wraparound location. Determines where the QSPI wraps to in wraparound mode. 0 Wrap to RAM entry zero. 1 Wrap to RAM entry pointed to by QWR[NEWQP].
12 CSIV	QSPI_CS inactive level. 0 QSPI chip select outputs return to zero when not driven from the value in the current command RAM entry during a transfer (that is, inactive state is 0, chip selects are active high). 1 QSPI chip select outputs return to one when not driven from the value in the current command RAM entry during a transfer (that is, inactive state is 1, chip selects are active low).
11–8 ENDQP	End of queue pointer. Points to the RAM entry that contains the last transfer description in the queue.
7–4 CPTQP	Completed queue entry pointer. Points to the RAM entry that contains the last command to have been completed. This field is read only.
3–0 NEWQP	Start of queue pointer. This 4-bit field points to the first entry in the RAM to be executed on initiating a transfer.

23.3.4 QSPI Interrupt Register (QIR)

Address: 0xFC05_C00C Access: User read/write

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	WCEFB	ABRTB	0	ABRTL	WCEFE	ABRTE	0	SPIFE	0	0	0	0	WCEF	ABRT	0	SPIF
W	0	0	0	0	0	0	0	0	0	0	0	0	w1c	w1c	0	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 23-6. QSPI Interrupt Register (QIR)

Table 23-6. QIR Field Descriptions

Field	Description
15 WCEFB	Write collision access error enable. A write collision occurs during a data transfer when the RAM entry containing the current command is written to by the CPU with the QDR. When this bit is asserted, the write access to QDR results in an access error.
14 ABRTB	Abort access error enable. An abort occurs when QDLYR[SPE] is cleared during a transfer. When set, an attempt to clear QDLYR[SPE] during a transfer results in an access error.
13	Reserved, should be cleared.
12 ABRTL	Abort lock-out. When set, QDLYR[SPE] cannot be cleared by writing to the QDLYR. QDLYR[SPE] is only cleared by the QSPI when a transfer completes.
11 WCEFE	Write collision (WCEF) interrupt enable. 0 Write collision interrupt disabled 1 Write collision interrupt enabled
10 ABRTE	Abort (ABRT) interrupt enable. 0 Abort interrupt disabled 1 Abort interrupt enabled
9	Reserved, should be cleared.
8 SPIFE	QSPI finished (SPIF) interrupt enable. 0 SPIF interrupt disabled 1 SPIF interrupt enabled
7–4	Reserved, should be cleared.
3 WCEF	Write collision error flag. Indicates that an attempt has been made to write to the RAM entry that is currently being executed. Writing a 1 to this bit (w1c) clears it and writing 0 has no effect.
2 ABRT	Abort flag. Indicates that QDLYR[SPE] has been cleared by the user writing to the QDLYR rather than by completion of the command queue by the QSPI. Writing a 1 to this bit (w1c) clears it and writing 0 has no effect.
1	Reserved, should be cleared.
0 SPIF	QSPI finished flag. Asserted when the QSPI has completed all the commands in the queue. Set on completion of the command pointed to by QWR[ENDQP], and on completion of the current command after assertion of QWR[HALT]. In wraparound mode, this bit is set every time the command pointed to by QWR[ENDQP] is completed. Writing a 1 to this bit (w1c) clears it and writing 0 has no effect.

The command and data RAM in the QSPI are indirectly accessible with QDR and QAR as 48 separate locations that comprise 16 words of transmit data, 16 words of receive data, and 16 bytes of commands.

A write to QDR causes data to be written to the RAM entry specified by QAR[ADDR]. This also causes the value in QAR to increment.

Correspondingly, a read at QDR returns the data in the RAM at the address specified by QAR[ADDR]. This also causes QAR to increment. A read access requires a single wait state.

NOTE

The QAR does not wrap after the last queue entry within each section of the RAM. The application software must handle address range errors.

23.3.5 QSPI Address Register (QAR)

The QAR is used to specify the location in the QSPI RAM that read and write operations affect. As shown in [Section 23.4.1, “QSPI RAM”](#), the transmit RAM is located at addresses 0x0 to 0xF, the receive RAM is located at 0x10 to 0x1F, and the command RAM is located at 0x20 to 0x2F. These addresses refer to the QSPI RAM space, not the device memory map.

Address: 0x0FC05_C010																Access: User read/write								
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	ADDR							
W																	ADDR							
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 23-7. QSPI Address Register (QAR)

23.3.6 QSPI Data Register (QDR)

The QDR, shown in [Figure 23-8](#), is used to access QSPI RAM indirectly. The CPU reads and writes all data from and to the QSPI RAM through this register.

Address: 0x0FC05_C014																Access: User read/write								
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	DATA							
W																	DATA							
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 23-8. QSPI Data Register (QDR)

23.3.7 Command RAM Registers (QCR0–QCR15)

The command RAM is accessed using the upper byte of the QDR; the QSPI cannot modify information in command RAM. There are 16 bytes in the command RAM. Each byte is divided into two fields. The chip select field enables external peripherals for transfer. The command field provides transfer operations.

NOTE

The command RAM is accessed only using the most significant byte of QDR and indirect addressing based on QAR[ADDR].

Address: QAR[ADDR]																Access: CPU write-only								
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	QSPI_CS							
W	CONT	BITSE	DT	DSCK	—	—	—	—	—	—	—	—	—	—	—	—	0	0	0	0	0	0	0	0
Reset	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—

Figure 23-9. Command RAM Registers (QCR0–QCR15)

Table 23-7. QCR0–QCR15 Field Descriptions

Field	Description
15 CONT	Continuous. 0 Chip selects return to inactive level defined by QWR[CSIV] when a single word transfer is complete. 1 Chip selects return to inactive level defined by QWR[CSIV] only after the transfer of the queue entries (max of 16 words). Note: In order to keep the chip selects asserted for transfers beyond 16 words, the QWR[CSIV] bit must be set to control the level that the chip selects return to after the first transfer.
14 BITSE	Bits per transfer enable. 0 Eight bits 1 Number of bits set in QMR[BITS]
13 DT	Delay after transfer enable. 0 Default reset value. 1 The QSPI provides a variable delay at the end of serial transfer to facilitate interfacing with peripherals that have a latency requirement. The delay between transfers is determined by QDLYR[DTL].
12 DSCK	Chip select to QSPI_CLK delay enable. 0 Chip select valid to QSPI_CLK transition is one-half QSPI_CLK period. 1 QDLYR[QCD] specifies the delay from QSPI_CS valid to QSPI_CLK.
11–8 QSPI_CS	Peripheral chip selects. Used to select an external device for serial data transfer. More than one chip select may be active at once, and more than one device can be connected to each chip select. Bits 11-8 map directly to the corresponding QSPI_CS n pins. If more than four chip selects are needed, then an external demultiplexor can be used with the QSPI_CS n pins. Note: Not all chip selects may be available on all device packages. See "Pinouts" for details on which chip selects are pinned-out.
7–0	Reserved, should be cleared.

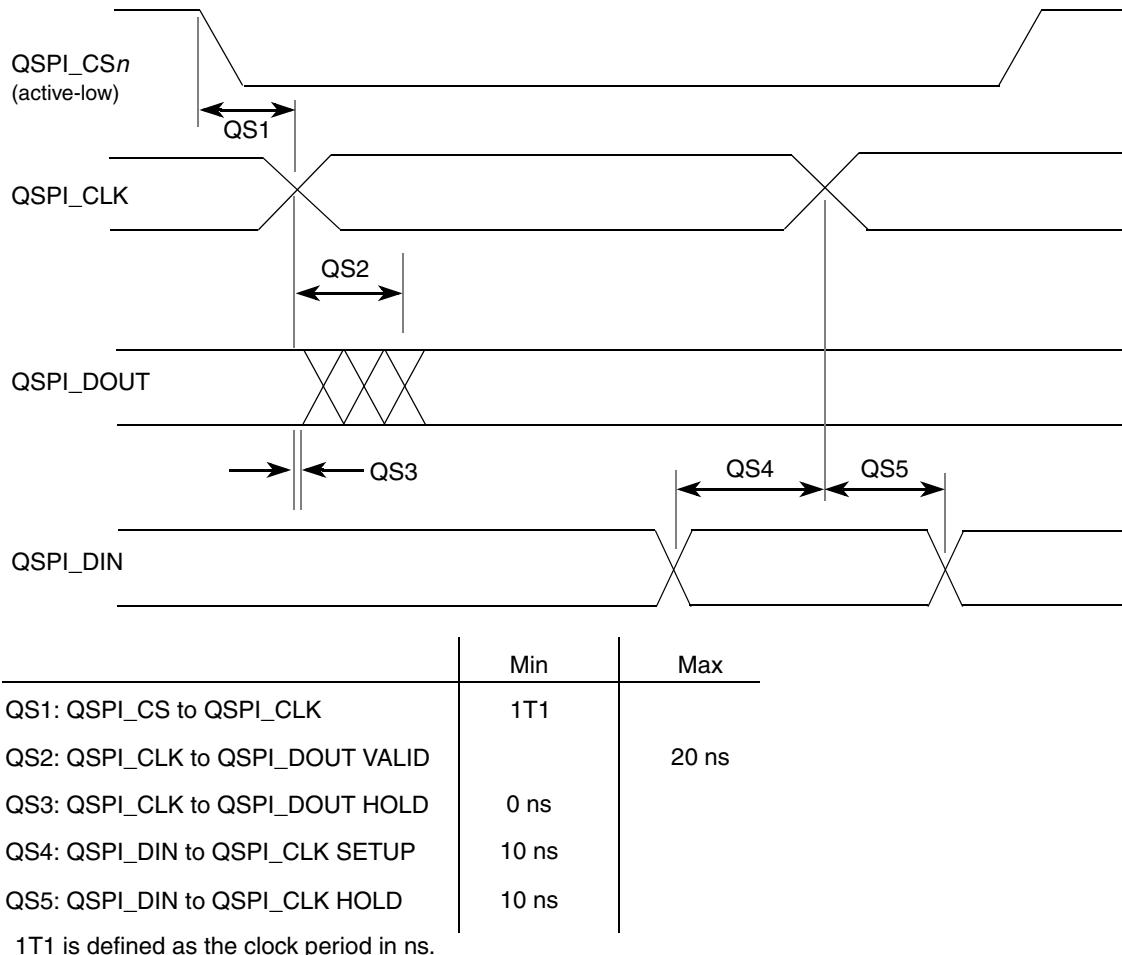


Figure 23-10. QSPI Timing

23.4 Functional Description

The QSPI uses a dedicated 80-byte block of static RAM accessible to both the module and CPU to perform queued operations. The RAM is divided into three segments:

- 16 command control bytes (command RAM)
- 32 transmit data bytes (transfer RAM)
- 32 receive data bytes (transfer RAM)

The RAM is organized so that 1 byte of command control data, 1 word of transmit data, and 1 word of receive data comprise 1 of the 16 queue entries (0x0–0xF).

NOTE

Throughout ColdFire documentation, the term ‘word’ is used to designate a 16-bit data unit. The only exceptions to this appear in discussions of serial communication modules such as QSPI that support variable-length data units. To simplify these discussions, the functional unit is referred to as a ‘word’ regardless of length.

The user initiates QSPI operation by loading a queue of commands in command RAM, writing transmit data into transmit RAM, and then enabling the QSPI data transfer. The QSPI executes the queued commands and sets the completion flag in the QSPI interrupt register (QIR[SPIF]) to signal their completion. As another option, QIR[SPIFE] can be enabled to generate an interrupt.

The QSPI uses four queue pointers. The user can access three of them through fields in QSPI wrap register (QWR):

- New queue pointer (QWR[NEWQP])—points to the first command in the queue
- Internal queue pointer—points to the command currently being executed
- Completed queue pointer (QWR[CPTQP])—points to the last command executed
- End queue pointer (QWR[ENDQP]) —points to the final command in the queue

The internal pointer is initialized to the same value as QWR[NEWQP]. During normal operation, the following sequence repeats:

1. The command pointed to by the internal pointer is executed.
2. The value in the internal pointer is copied into QWR[CPTQP].
3. The internal pointer is incremented.

Execution continues at the internal pointer address unless the QWR[NEWQP] value is changed. After each command is executed, QWR[ENDQP] and QWR[CPTQP] are compared. When a match occurs, QIR[SPIF] is set and the QSPI stops unless wraparound mode is enabled. Setting QWR[WREN] enables wraparound mode.

QWR[NEWQP] is cleared at reset. When the QSPI is enabled, execution begins at address 0x0 unless another value has been written into QWR[NEWQP]. QWR[ENDQP] is cleared at reset but is changed to show the last queue entry before the QSPI is enabled. QWR[NEWQP] and QWR[ENDQP] can be written at any time. When the QWR[NEWQP] value changes, the internal pointer value also changes unless a transfer is in progress, in which case the transfer completes normally. Leaving QWR[NEWQP] and QWR[ENDQP] set to 0x0 causes a single transfer to occur when the QSPI is enabled.

Data is transferred relative to QSPI_CLK, which can be generated in any one of four combinations of phase and polarity using QMR[CPHA,CPOL]. Data is transferred with the most significant bit (msb) first. The number of bits transferred defaults to 8, but can be set to any value between 8 and 16 by writing a value into the BITSE field of the command RAM (QCR[BITSE]).

23.4.1 QSPI RAM

The QSPI contains an 80-byte block of static RAM that can be accessed by both the user and the QSPI. This RAM does not appear in the device memory map, because it can only be accessed by the user indirectly through the QSPI address register (QAR) and the QSPI data register (QDR). The RAM is divided into three segments with 16 addresses each:

- Receive data RAM—the initial destination for all incoming data
- Transmit data RAM—a buffer for all out-bound data
- Command RAM—where commands are loaded

The transmit and command RAM are user write-only. The receive RAM is user read-only. [Figure 23-11](#) shows the RAM configuration. The RAM contents are undefined immediately after a reset.

The command and data RAM in the QSPI is indirectly accessible with QDR and QAR as 48 separate locations that comprise 16 words of transmit data, 16 words of receive data, and 16 bytes of commands.

A write to QDR causes data to be written to the RAM entry specified by QAR[ADDR] and causes the value in QAR to increment. Correspondingly, a read at QDR returns the data in the RAM at the address specified by QAR[ADDR]. This also causes QAR to increment. A read access requires a single wait state.

Relative Address	Register	Function
0x00	QTR0	Transmit RAM 16 bits wide
0x01	QTR1	
...	...	
0x0F	QTR15	
0x10	QRR0	Receive RAM 16 bits wide
0x11	QRR1	
...	...	
0x1F	QRR15	
0x20	QCR0	Command RAM 8 bits wide
0x21	QCR1	
...	...	
0x2F	QCR15	

Figure 23-11. QSPI RAM Model

23.4.1.1 Receive RAM

Data received by the QSPI is stored in the receive RAM segment located at 0x10 to 0x1F in the QSPI RAM space. The user reads this segment to retrieve data from the QSPI. Data words with less than 16 bits are stored in the least significant bits of the RAM. Unused bits in a receive queue entry are set to zero upon completion of the individual queue entry.

QWR[CPTQP] shows which queue entries have been executed. The user can query this field to determine which locations in receive RAM contain valid data.

23.4.1.2 Transmit RAM

Data to be transmitted by the QSPI is stored in the transmit RAM segment located at addresses 0x0 to 0xF. The user normally writes 1 word into this segment for each queue command to be executed. The user cannot read data in the transmit RAM.

Outbound data must be written to transmit RAM in a right-justified format. The unused bits are ignored. The QSPI copies the data to its data serializer (shift register) for transmission. The data is transmitted most significant bit first and remains in transmit RAM until overwritten by the user.

23.4.1.3 Command RAM

The CPU writes one byte of control information to this segment for each QSPI command to be executed. Command RAM, referred to as QCR0–15, is write-only memory from a user’s perspective.

Command RAM consists of 16 bytes, each divided into two fields. The peripheral chip select field controls the QSPI_CS signal levels for the transfer. The command control field provides transfer options.

A maximum of 16 commands can be in the queue. Queue execution proceeds from the address in QWR[NEWQP] through the address in QWR[ENDQP].

The QSPI executes a queue of commands defined by the control bits in each command RAM entry that sequence the following actions:

- Chip-select pins are activated.
- Data is transmitted from transmit RAM and received into the receive RAM.
- The synchronous transfer clock QSPI_CLK is generated.

Before any data transfers begin, control data must be written to the command RAM, and any out-bound data must be written to transmit RAM. Also, the queue pointers must be initialized to the first and last entries in the command queue.

Data transfer is synchronized with the internally generated QSPI_CLK, whose phase and polarity are controlled by QMR[CPHA] and QMR[CPOL]. These control bits determine which QSPI_CLK edge is used to drive outgoing data and to latch incoming data.

23.4.2 Baud Rate Selection

The maximum QSPI clock frequency is one-fourth the clock frequency of the internal bus clock ($f_{sys/2}$). Baud rate is selected by writing a value from 2–255 into QMR[BAUD]. The QSPI uses a prescaler to derive the QSPI_CLK rate from the internal bus clock divided by two.

A baud rate value of zero turns off the QSPI_CLK.

The desired QSPI_CLK baud rate is related to the internal bus clock and QMR[BAUD] by the following expression:

$$QMR[BAUD] = \frac{f_{sys/2}}{2 \times [\text{desired QSPI_CLK baud rate}]} \quad \text{Eqn. 23-1}$$

Table 23-8. QSPI_CLK Frequency as Function of Internal Bus Clock and Baud Rate

Internal Bus Clock = 83.33 MHz	
QMR [BAUD]	QSPI_CLK
2	20.75 MHz
4	10.375 MHz
8	5.188 MHz
16	2.594 MHz
32	1.297 MHz
255	162.7 kHz

23.4.3 Transfer Delays

The QSPI supports programmable delays for the QSPI_CS signals before and after a transfer. The time between QSPI_CS assertion and the leading QSPI_CLK edge, and the time between the end of one transfer and the beginning of the next, are both independently programmable.

The chip select to clock delay enable bit in command RAM, QCR[DSCK], enables the programmable delay period from QSPI_CS assertion until the leading edge of QSPI_CLK. QDLYR[QCD] determines the period of delay before the leading edge of QSPI_CLK. The following expression determines the actual delay before the QSPI_CLK leading edge:

$$\text{QSPI_CS-to-QSPI_CLK delay} = \frac{\text{QDLYR[QCD]}}{f_{\text{sys}/2}} \quad \text{Eqn. 23-2}$$

QDLYR[QCD] has a range of 1–127.

When QDLYR[QCD] or QCR[DSCK] equals zero, the standard delay of one-half the QSPI_CLK period is used.

The command RAM delay after transmit enable bit, QCR[DT], enables the programmable delay period from the negation of the QSPI_CS signals until the start of the next transfer. The delay after transfer can be used to provide a peripheral deselect interval. A delay can also be inserted between consecutive transfers to allow serial A/D converters to complete conversion. There are two transfer delay options: the user can choose to delay a standard period after serial transfer is complete or can specify a delay period. Writing a value to QDLYR[DTL] specifies a delay period. QCR[DT] determines whether the standard delay period (DT = 0) or the specified delay period (DT = 1) is used. The following expression is used to calculate the delay when DT = 1:

$$\text{Delay after transfer} = \frac{32 \times \text{QDLYR[DTL]}}{f_{\text{sys}/2}} \quad (\text{DT} = 1) \quad \text{Eqn. 23-3}$$

where QDLYR[DTL] has a range of 1–255. A zero value for DTL causes a delay-after-transfer value of $8192/f_{\text{sys}/2}$. Standard delay period (DT = 0) is calculated by the following:

$$\text{Standard delay after transfer} = \frac{17}{f_{\text{sys}/2}} \quad (\text{DT} = 0) \quad \text{Eqn. 23-4}$$

Adequate delay between transfers must be specified for long data streams because the QSPI module requires time to load a transmit RAM entry for transfer. Receiving devices need at least the standard delay between successive transfers. If the internal bus clock is operating at a slower rate, the delay between transfers must be increased proportionately.

23.4.4 Transfer Length

There are two transfer length options. The user can choose a default value of 8 bits or a programmed value of 8 to 16 bits. The programmed value must be written into QMR[BITS]. The command RAM bits per transfer enable field, QCR[BITSE], determines whether the default value (BITSE = 0) or the BITS[3–0] value (BITSE = 1) is used. QMR[BITS] indicates the required number of bits to be transferred, with the default value of 16 bits.

23.4.5 Data Transfer

The transfer operation is initiated by setting QDLYR[SPE]. Shortly after QDLYR[SPE] is set, the QSPI executes the command at the command RAM address pointed to by QWR[NEWQP]. Data at the pointer address in transmit RAM is loaded into the data serializer and transmitted. Data that is simultaneously received is stored at the pointer address in receive RAM.

When the proper number of bits has been transferred, the QSPI stores the working queue pointer value in QWR[CPTQP], increments the working queue pointer, and loads the next data for transfer from the transmit RAM. The command pointed to by the incremented working queue pointer is executed next unless a new value has been written to QWR[NEWQP]. If a new queue pointer value is written while a transfer is in progress, the current transfer is completed normally.

When the CONT bit in the command RAM is set, the QSPI_CS signals are asserted between transfers. When CONT is cleared, QSPI_CS_n are negated between transfers. Note, the QSPI_CS signals are not high impedance.

When the QSPI reaches the end of the queue, it asserts the SPIF flag, QIR[SPIF]. If QIR[SPIFE] is set, an interrupt request is generated when QIR[SPIF] is asserted. Then the QSPI clears QDLYR[SPE] and stops, unless wraparound mode is enabled.

Wraparound mode is enabled by setting QWR[WREN]. The queue can wrap to pointer address 0x0, or to the address specified by QWR[NEWQP], depending on the state of QWR[WRTO].

In wraparound mode, the QSPI cycles through the queue continuously, even while requesting interrupt service. QDLYR[SPE] is not cleared when the last command in the queue is executed. New receive data overwrites previously received data in the receive RAM. Each time the end of the queue is reached, QIR[SPIFE] is set. QIR[SPIF] is not automatically reset. If interrupt driven QSPI service is used, the service routine must clear QIR[SPIF] to abort the current request. Additional interrupt requests during servicing can be prevented by clearing QIR[SPIFE].

There are two recommended methods of exiting wraparound mode: clearing QWR[WREN] or setting QWR[HALT]. Exiting wraparound mode by clearing QDLYR[SPE] is not recommended because this may abort a serial transfer in progress. The QSPI sets SPIF, clears QDLYR[SPE], and stops the first time it

reaches the end of the queue after QWR[WREN] is cleared. After QWR[HALT] is set, the QSPI finishes the current transfer, then stops executing commands. After the QSPI stops, QDLYR[SPE] can be cleared.

23.5 Initialization/Application Information

The following steps are necessary to set up the QSPI 12-bit data transfers and a QSPI_CLK of 5.188 MHz. The QSPI RAM is set up for a queue of 16 transfers. All four QSPI_CS signals are used in this example.

1. Write the QMR with 0xB308 to set up 12-bit data words with the data shifted on the falling clock edge, and a QSPI_CLK frequency of 5.188 MHz (assuming a 83.33-MHz internal bus clock).
2. Write QDLYR with the desired delays.
3. Write QIR with 0xD00F to enable write collision, abort bus errors, and clear any interrupts.
4. Write QAR with 0x0020 to select the first command RAM entry.
5. Write QDR with 0x7E00, 0x7E00, 0x7E00, 0x7E00, 0x7D00, 0x7D00, 0x7D00, 0x7D00, 0x7B00, 0x7B00, 0x7B00, 0x7B00, 0x7700, 0x7700, 0x7700, and 0x7700 to set up four transfers for each chip select. The chip selects are active low in this example.
6. Write QAR with 0x0000 to select the first transmit RAM entry.
7. Write QDR with sixteen 12-bit words of data.
8. Write QWR with 0x0F00 to set up a queue beginning at entry 0 and ending at entry 15.
9. Set QDLYR[SPE] to enable the transfers.
10. Wait until the transfers are complete. QIR[SPIF] is set when the transfers are complete.
11. Write QAR with 0x0010 to select the first receive RAM entry.
12. Read QDR to get the received data for each transfer.
13. Repeat steps 5 through 13 to do another transfer.

Chapter 24

UART Modules

24.1 Introduction

This chapter describes the use of the three universal asynchronous receiver/transmitters (UARTs) and includes programming examples.

NOTE

The designation ‘*n*’ is used throughout this section to refer to registers or signals associated with one of the three identical UART modules: UART0, UART1, or UART2.

24.1.1 Overview

Each of the three independent UARTs can be clocked by the internal bus clock, eliminating the need for an external UART clock. As Figure 24-1 shows, each UART module interfaces directly to the CPU and consists of the following:

- Serial communication channel
- Programmable clock generation
- Interrupt control logic and DMA request logic
- Internal channel control logic

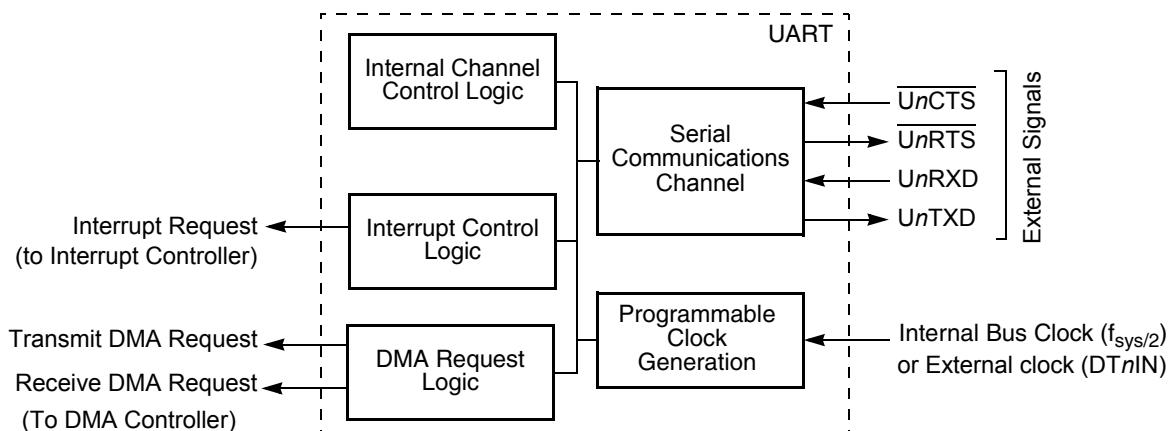


Figure 24-1. UART Block Diagram

NOTE

UART*n* can be clocked by the DT*n*IN pin. However, if the timers are used, then input capture mode is not available for that timer.

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 24.4.2.1, “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 24.4.2.2, “Receiver.”](#)

NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to [Chapter 13, “General Purpose I/O Module”](#)) prior to configuring the UART module.

24.1.2 Features

The device contains three independent UART modules with the following features:

- Each clocked by an external clock or by the internal bus clock (eliminating a 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 loop-back, or remote loop-back mode
- Automatic wake-up mode for multidrop applications
- Four maskable interrupt conditions
- All three 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

24.2 External Signal Description

Figure 24-1 shows both the external and internal signal groups.

An internal interrupt request signal is provided to notify the interrupt controller of an interrupt condition. The output is the logical NOR of unmasked UISR n bits. The interrupt level and priority are programmed in the interrupt controller. See [Chapter 14, “Interrupt Controller Module”](#) for more information.

Note that the UARTs can also be configured to automatically transfer data by using the DMA rather than interrupting the core. When there is data in the receiver FIFO or when the transmit holding register is empty, a DMA request can be issued. For more information on generating DMA requests, refer to [Section 24.4.6.1.2, “Setting up the UART to Request DMA Service,”](#) and [Chapter 16, “Enhanced Direct Memory Access \(eDMA\).”](#)

Table 24-1 briefly describes the UART module signals.

NOTE

The terms ‘assertion’ and ‘negation’ are used to avoid confusion between active-low and active-high signals. ‘Asserted’ indicates that a signal is active, independent of the voltage level; ‘negated’ indicates that a signal is inactive.

Table 24-1. UART Module Signals

Signal	Description
Transmitter Serial Data Output (U_{nTXD})	U_{nTXD} is held high (mark condition) when the transmitter is disabled, idle, or operating in the local loop-back mode. Data is shifted out on U_{nTXD} on the falling edge of the clock source, with the least significant bit (lsb) sent first.
Receiver Serial Data Input (U_{nRXD})	Data received on U_{nRXD} is sampled on the rising edge of the clock source, with the lsb received first.
Clear-to-Send (U_{nCTS})	This input can generate an interrupt on a change of state.
Request-to-Send (U_{nRTS})	This output can be programmed to be negated or asserted automatically by either the receiver or the transmitter. When connected to a transmitter’s U_{nCTS} , U_{nRTS} can control serial data flow.

Figure 24-2 shows a signal configuration for a UART/RS-232 interface.

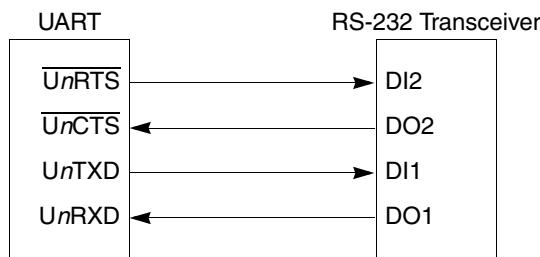


Figure 24-2. UART/RS-232 Interface

24.3 Memory Map/Register Definition

This section contains a detailed description of each register and its specific function. Flowcharts in Section 24.4.6, “Programming,” describe basic UART module programming. The operation of the UART module is controlled by writing control bytes into the appropriate registers. Table 24-2 is a memory map for UART module registers.

NOTE

UART registers are accessible only as bytes.

NOTE

Interrupt can mean either an interrupt request asserted to the CPU or a DMA request.

Table 24-2. UART Module Memory Map

Address	Register	Width (bit)	Access	Reset Value	Section/Page
UART0 UART1 UART2					
0xFC06_0000 0xFC06_4000 0xFC06_8000	UART Mode Registers ¹ (UMR1 n), (UMR2 n)	8	R/W	0x00	24.3.1/24-5 24.3.2/24-6
0xFC06_0004 0xFC06_4004 0xFC06_8004	UART Status Register (USR n) UART Clock Select Register ¹ (UCSR n)	8	R W	0x00	24.3.3/24-7 24.3.4/24-9
0xFC06_0008 0xFC06_4008 0xFC06_8008	UART Command Registers (UCR n)	8	W	0x00	24.3.5/24-9
0xFC06_000C 0xFC06_400C 0xFC06_800C	UART Receive Buffers (URB n) UART Transmit Buffers (UTB n)	8	R W	0xFF 0x00	24.3.6/24-11 24.3.7/24-12
0xFC06_0010 0xFC06_4010 0xFC06_8010	UART Input Port Change Register (UIPCR n) UART Auxiliary Control Register (UACR n)	8	R W	See Section 0x00	24.3.8/24-12 24.3.9/24-13
0xFC06_0014 0xFC06_4014 0xFC06_8014	UART Interrupt Status Register (UISR n) UART Interrupt Mask Register (UIMR n)	8	R W	0x00	24.3.10/24-13
0xFC06_0018 0xFC06_4018 0xFC06_8018	UART Baud Rate Generator Register (UBG1 n)	8	W ²	0x00	24.3.11/24-15
0xFC06_001C 0xFC06_401C 0xFC06_801C	UART Baud Rate Generator Register (UBG2 n)	8	W ²	0x00	24.3.11/24-15
0xFC06_0034 0xFC06_4034 0xFC06_8034	UART Input Port Register (UIP n)	8	R	0xFF	24.3.12/24-16

Table 24-2. UART Module Memory Map (continued)

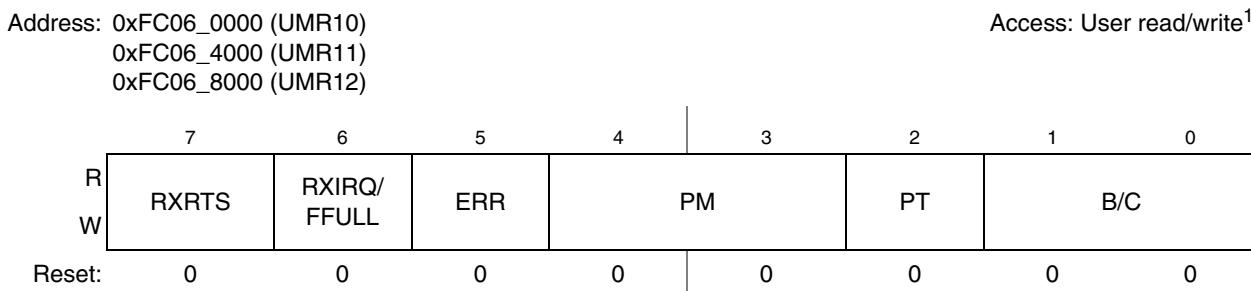
Address	Register	Width (bit)	Access	Reset Value	Section/Page
UART0 UART1 UART2					
0xFC06_0038 0xFC06_4038 0xFC06_8038	UART Output Port Bit Set Command Register (UOP1n)	8	W ²	0x00	24.3.13/24-16
0xFC06_003C 0xFC06_403C 0xFC06_803C	UART Output Port Bit Reset Command Register (UOP0n)	8	W ²	0x00	24.3.13/24-16

¹ UMR1n, UMR2n, and UCSRn should be changed only after the receiver/transmitter is issued a software reset command. That is, if operation is not disabled, undesirable results may occur.

² Reading this register results in undesired effects and possible incorrect transmission or reception of characters. Register contents may also be changed.

24.3.1 UART Mode Registers 1 (UMR1n)

The UMR1n registers control configuration. UMR1n 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 UMR1n is read or written, the pointer points to UMR2n.



¹ After UMR1n is read or written, the pointer points to UMR2n

Figure 24-3. UART Mode Registers 1 (UMR1n)**Table 24-3. UMR1n Field Descriptions**

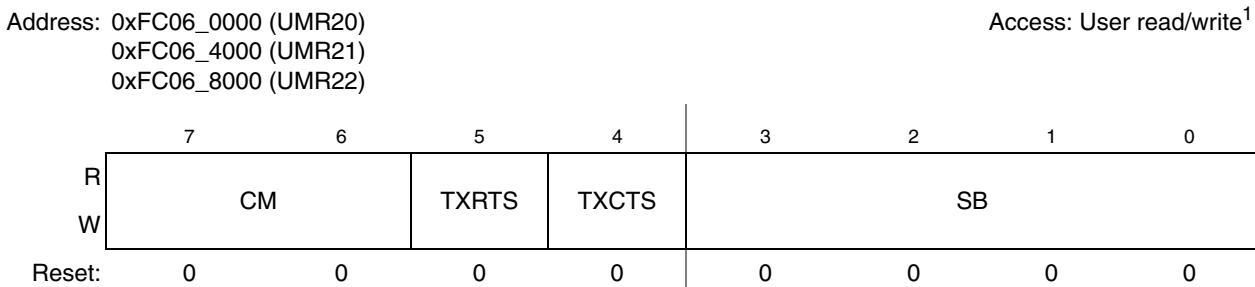
Field	Description
7 RXRTS	Receiver request-to-send. Allows the \overline{UnRTS} output to control the \overline{UnCTS} input of the transmitting device to prevent receiver overrun. If both the receiver and transmitter are incorrectly programmed for \overline{UnRTS} control, \overline{UnRTS} control is disabled for both. Transmitter RTS control is configured in UMR2n[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 that generates interrupt or DMA requests. 1 FFULL is the source that generates interrupt or DMA requests.

Table 24-3. UMR1 n Field Descriptions (continued)

Field	Description																				
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 Section 24.3.5, “UART Command Registers (UCRn) .																				
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.																				
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>PM</th> <th>Parity Mode</th> <th>Parity Type (PT= 0)</th> <th>Parity Type (PT= 1)</th> </tr> </thead> <tbody> <tr> <td>00</td> <td>With parity</td> <td>Even parity</td> <td>Odd parity</td> </tr> <tr> <td>01</td> <td>Force parity</td> <td>Low parity</td> <td>High parity</td> </tr> <tr> <td>10</td> <td>No parity</td> <td colspan="2">N/A</td> </tr> <tr> <td>11</td> <td>Multidrop mode</td> <td>Data character</td> <td>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																				

24.3.2 UART Mode Register 2 (UMR2 n)

The UMR2 n registers control UART module configuration. UMR2 n can be read or written when the mode register pointer points to it, which occurs after any access to UMR1 n . UMR2 n accesses do not update the pointer.

Address: 0xFC06_0000 (UMR20) 0xFC06_4000 (UMR21) 0xFC06_8000 (UMR22)	Access: User read/write ¹																								
 <p>Bit Map:</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <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>CM</td><td>TXRTS</td><td>TXCTS</td><td colspan="4">SB</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> </tr> </table> <p>Reset: 0 0 0 0 0 0 0 0</p>	7	6	5	4	3	2	1	0	R	CM	TXRTS	TXCTS	SB				W	0	0	0	0	0	0	0	
7	6	5	4	3	2	1	0																		
R	CM	TXRTS	TXCTS	SB																					
W	0	0	0	0	0	0	0																		

¹ After UMR1 n is read or written, the pointer points to UMR2 n

Figure 24-5. UART Mode Register 2 (UMR2 n)

Table 24-4. UMR2n Field Descriptions

Field	Description
7–6 CM	Channel mode. Selects a channel mode. Section 24.4.3, “Looping Modes,” describes individual modes. 00 Normal 01 Automatic echo 10 Local loop-back 11 Remote loop-back
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 both 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 the length of the 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, that is, 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

24.3.3 UART Status Registers (USRn)

The USR n registers, shown in [Figure 24-6](#), show the status of the transmitter, the receiver, and the FIFO.

Address: 0xFC06_0004 (UCSR0)
 0xFC06_4004 (UCSR1)
 0xFC06_8004 (UCSR2)

Access: User read-only

	7	6	5	4	3	2	1	0
R	RB	FE	PE	OE	TXEMP	TXRDY	FFULL	RXRDY
W								
Reset:	0	0	0	0	0	0	0	0

Figure 24-6. UART Status Register (USRn)

Table 24-5. USRn Field Descriptions

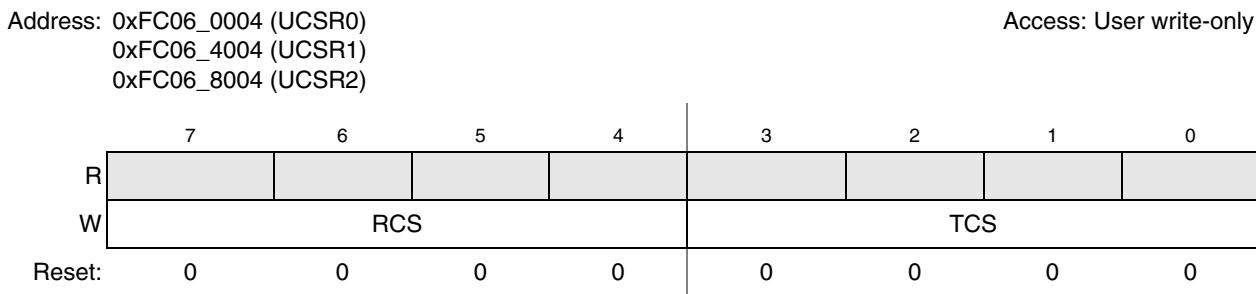
Field	Description
7 RB	Received break. The received break circuit detects breaks that originate in the middle of a received character. However, a break in the middle of a character must persist until the end of the next detected character time. 0 No break was received. 1 An all-zero character of the 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 is equal to two successive edges of the UART clock. RB is valid only when RXRDY = 1.
6 FE	Framing error. 0 No framing error occurred. 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 = 1.
5 PE	Parity error. Valid only if RXRDY = 1. 0 No parity error occurred. 1 If UMR1n[PM] = 0x (with parity or force parity), the corresponding character in the FIFO was received with incorrect parity. If UMR1n[PM] = 11 (multidrop), PE stores the received address or data (A/D) bit. PE is valid only when RXRDY = 1.
4 OE	Overrun error. Indicates whether an overrun occurs. 0 No overrun occurred. 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. OE is cleared by the RESET ERROR STATUS command in UCRn.
3 TEMP	Transmitter empty. 0 The transmit buffer is not empty. Either a character is being shifted out, or the transmitter is disabled. The transmitter is enabled/disabled by programming UCRn[TC]. 1 The transmitter has underrun (both 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.
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.

Table 24-5. USR n Field Descriptions (continued)

Field	Description
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.

24.3.4 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 24.4.1, “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.

**Figure 24-7. UART Clock Select Register (UCSR n)****Table 24-6. 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 DTIN divided by 16 1111 DTIN
3–0 TCS	Transmitter clock select. Selects the clock source for the transmitter. 1101 Prescaled internal bus clock ($f_{sys}/2$) 1110 DTIN divided by 16 1111 DTIN

24.3.5 UART Command Registers (UCR n)

The UCR n s, shown in [Figure 24-8](#), 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) 0xFC06_4008 (UCR1) 0xFC06_8008 (UCR2)								Access: User write-only
R								
W	0	0	MISC	0	0	TC	0	RC
Reset:	0	0	0	0	0	0	0	0

Figure 24-8. UART Command Register (UCR n)

Table 24-7 describes UCR n fields and commands. Examples in [Section 24.4.2, “Transmitter and Receiver Operating Modes,”](#) show how these commands are used.

Table 24-7. UCR n Field Descriptions

Field	Description		
7	Reserved, should 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, the break may be delayed up to one bit time. If the transmitter is active, the break starts when character transmission completes. The 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. The transmitter must be enabled for the command to be accepted. This command ignores the state of UnCTS.	
111	STOP BREAK	Causes UnTXD to go high (mark) within two bit times. Any characters in the transmit buffer are sent.	

Table 24-7. UCR n Field Descriptions (continued)

Field	Description		
3–2 TC	TC Field (This field selects a single command)		
	Command	Description	
	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	RC (This field selects a single command)		
	Command	Description	
	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 loop-back 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.

24.3.6 UART Receive Buffers (URB n)

The receive buffers (shown in [Figure 24-9](#)) 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 24-20](#)). RB contains the character in the receiver.

Address:	0xFC06_000C (URB0) 0xFC06_400C (URB1) 0xFC06_800C (URB2)	Access:	User read-only
7	6	5	4
R			RB
W			
Reset:	1	1	1
			1
			1
			1
			1
			1
			0

Figure 24-9. UART Receive Buffer (URBn)

24.3.7 UART Transmit Buffers (UTBn)

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 USR n [TXRDY] is set. A write to the transmit buffer clears USR n [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 (TXRDY = 0). If there is a valid character, the shift register loads it and sets USR n [TXRDY] again. Writes to the transmit buffer when the UART's TXRDY = 0 and the transmitter is disabled have no effect on the transmit buffer.

Figure 24-10 shows UTB n . TB contains the character in the transmit buffer.

Address:	0xFC06_000C (UTB0) 0xFC06_400C (UTB1) 0xFC06_800C (UTB2)	Access:	User write-only
7	6	5	4
R			
W			TB
Reset:	0	0	0
			0
			0
			0
			0
			0
			0
			0

Figure 24-10. UART Transmit Buffer (UTBn)

24.3.8 UART Input Port Change Registers (UIPCRn)

The UIPCRs, shown in Figure 24-11, hold the current state and the change-of-state for \overline{UnCTS} .

Address:	0xFC06_0010 (UIPCR0) 0xFC06_4010 (UIPCR1) 0xFC06_8010 (UIPCR2)	Access:	User read-only
7	6	5	4
R	0	0	0
W			COS
Reset:	0	0	0
			1
			1
			1
			CTS
			\overline{UnCTS}

Figure 24-11. UART Input Port Change Register (UIPCRn)

Table 24-8. UIPCR n Field Descriptions

Field	Description
7–5	Reserved
4 COS	Change of state (high-to-low or low-to-high transition). 0 No change-of-state since the CPU last read UIPCR n . Reading UIPCR n clears UISR n [COS]. 1 A change-of-state longer than 25–50 μ s occurred on the \overline{UnCTS} input. UACR n can be programmed to generate an interrupt to the CPU when a change of state is detected.
3–1	Reserved
0 CTS	Current state of clear-to-send. Starting two serial clock periods after reset, CTS reflects the state of \overline{UnCTS} . If \overline{UnCTS} is detected asserted at that time, COS is set, which initiates an interrupt if UACR n [IEC] is enabled. 0 The current state of the \overline{UnCTS} input is asserted. 1 The current state of the \overline{UnCTS} input is deasserted.

24.3.9 UART Auxiliary Control Register (UACR n)

The UACRs, shown in [Figure 24-9](#), control the input enable.

Address: 0xFC06_0010 (UACR0)
0xFC06_4010 (UACR1)
0xFC06_8010 (UACR2)

Access: User write-only

	7	6	5	4	3	2	1	0
R								
W	0	0	0	0	0	0	0	IEC
Reset:	0	0	0	0	0	0	0	0

Figure 24-12. UART Auxiliary Control Register (UACR n)**Table 24-9. UACR n Field Descriptions**

Field	Description
7–1	Reserved, should 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).

24.3.10 UART Interrupt Status/Mask Registers (UISR n /UIMR n)

The UISRs, shown in [Figure 24-13](#), 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, the internal interrupt output is asserted. If a UIMR n bit is cleared, the 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)	Access:	User read/write					
	0xFC06_4014 (UISR1)							
	0xFC06_8014 (UISR2)							
R (UISR_n)	7 COS	6 0	5 0	4 0	3 0	2 DB	1 FFULL/ RXRDY	0 TXRDY
W (UIMR_n)	COS	0	0	0	0	DB	FFULL/ RXRDY	TXRDY
Reset:	0	0	0	0	0	0	0	0

Figure 24-13. UART Interrupt Status/Mask Registers ($\text{UISR}_n/\text{UIMR}_n$)

Table 24-10. $\text{UISR}_n/\text{UIMR}_n$ Field Descriptions

Field	Description																											
7 COS	Change-of-state. 0 UIPCR n [COS] is not selected. 1 Change-of-state occurred on $\overline{\text{UnCTS}}$ and was programmed in UACR n [IEC] to cause an interrupt.																											
6–3	Reserved, should be cleared.																											
2 DB	Delta break. 0 No new break-change condition to report. Section 24.3.5, “UART Command Registers (UCRn) , describes the RESET BREAK-CHANGE INTERRUPT command. 1 The receiver detected the beginning or end of a received break.																											
1 FFULL/ RXRDY	Status of FIFO or receiver, depending on UMR1[FFULL/RXRDY] bit. Duplicate of USR n [FIFO] & USR n [RXRDY]																											
	<table border="1"> <thead> <tr> <th>UIMR_n [FFULL/RXRDY]</th> <th>UISR_n [FFULL/RXRDY]</th> <th colspan="2">UMR1n[FFULL/RXRDY]</th> </tr> <tr> <th></th> <th></th> <th>0 (RXRDY)</th> <th>1 (FIFO)</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>Receiver not ready</td> <td>FIFO not full</td> </tr> <tr> <td>1</td> <td>0</td> <td>Receiver not ready</td> <td>FIFO not full</td> </tr> <tr> <td>0</td> <td>1</td> <td>Receiver is ready, Do not interrupt</td> <td>FIFO is full, Do not interrupt</td> </tr> <tr> <td>1</td> <td>1</td> <td>Receiver is ready, interrupt</td> <td>FIFO is full, interrupt</td> </tr> </tbody> </table>				UIMR_n [FFULL/RXRDY]	UISR_n [FFULL/RXRDY]	UMR1 n [FFULL/RXRDY]				0 (RXRDY)	1 (FIFO)	0	0	Receiver not ready	FIFO not full	1	0	Receiver not ready	FIFO not full	0	1	Receiver is ready, Do not interrupt	FIFO is full, Do not interrupt	1	1	Receiver is ready, interrupt	FIFO is full, interrupt
UIMR_n [FFULL/RXRDY]	UISR_n [FFULL/RXRDY]	UMR1 n [FFULL/RXRDY]																										
		0 (RXRDY)	1 (FIFO)																									
0	0	Receiver not ready	FIFO not full																									
1	0	Receiver not ready	FIFO not full																									
0	1	Receiver is ready, Do not interrupt	FIFO is full, Do not interrupt																									
1	1	Receiver is ready, interrupt	FIFO is full, interrupt																									
0 TXRDY	Transmitter ready. This bit is the duplication of USR n [TXRDY]. 0 The transmitter holding register was loaded by the CPU or the transmitter is disabled. Characters loaded into the transmitter holding register when TXRDY = 0 are not sent. 1 The transmitter holding register is empty and ready to be loaded with a character.																											

24.3.11 UART Baud Rate Generator Registers (UBG1n/UBG2n)

The UBG1n registers hold the MSB, and the UBG2n registers hold the LSB of the preload value. UBG1n and UBG2n concatenate to provide a divider to the internal bus clock for transmitter/receiver operation, as described in Section 24.4.1.2.1, “Internal Bus Clock Baud Rates.”

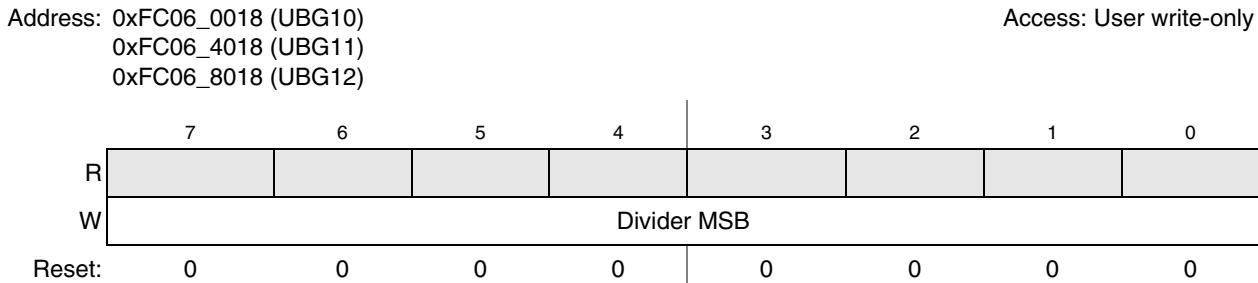


Figure 24-14. UART Baud Rate Generator Register (UBG1n)

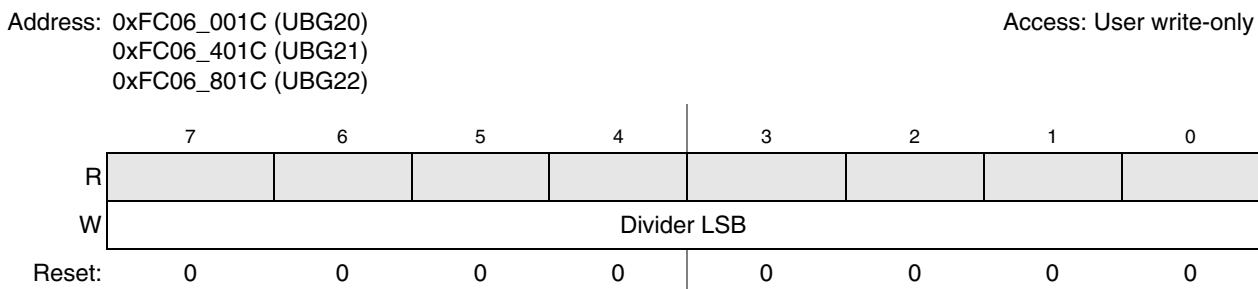


Figure 24-15. UART Baud Rate Generator Register (UBG2n)

NOTE

The minimum value that can be loaded on the concatenation of UBG1n with UBG2n is 0x0002. The UBG2n reset value of 0x00 is invalid and must be written to before the UART transmitter or receiver are enabled. Both UBG1n and UBG2n are write-only and cannot be read by the CPU.

24.3.12 UART Input Port Register (UIP n)

The UIP n registers, shown in [Figure 24-17](#), show the current state of the $\overline{\text{UnCTS}}$ input.

Address:	0xFC06_0034 (UIP0)	Access:	User read-only					
	0xFC06_4034 (UIP1)							
	0xFC06_8034 (UIP2)							
R	7 1	6 1	5 1	4 1	3 1	2 1	1 1	0 CTS
W								
Reset:	1	1	1	1	1	1	1	1

Figure 24-17. UART Input Port Register (UIP n)

Table 24-12. UIP n Field Descriptions

Field	Description
7–1	Reserved
0 CTS	Current state of clear-to-send. The $\overline{\text{UnCTS}}$ value is latched and reflects the state of the input pin when UIP n is read. Note: This bit has the same function and value as UIPCR n [RTS]. 0 The current state of the $\overline{\text{UnCTS}}$ input is logic 0. 1 The current state of the $\overline{\text{UnCTS}}$ input is logic 1.

24.3.13 UART Output Port Command Registers (UOP1 n /UOP0 n)

The $\overline{\text{UnRTS}}$ output can be asserted by writing a 1 to UOP1 n [RTS] and negated by writing a 1 to UOP0 n [RTS]. See [Figure 24-18](#).

Address:	0xFC06_0038 (UOP10)	Access:	User write-only					
	0xFC06_003C (UOP00)							
	0xFC06_4038 (UOP11)							
	0xFC06_403C (UOP01)							
	0xFC06_8038 (UOP12)							
	0xFC06_803C (UOP02)							
R	7 0	6 0	5 0	4 0	3 0	2 0	1 0	0 RTS
W								
Reset:	0	0	0	0	0	0	0	0

Figure 24-18. UART Output Port Command Registers (UOP1 n /UOP0 n)

Table 24-13. UOP1 n /UOP0 n Field Descriptions

Field	Description
7–1	Reserved, should be cleared.
0 RTS	Output port output. Controls assertion (UOP1)/negation (UOP0) of $\overline{\text{UnRTS}}$ output. 0 Not affected. 1 Asserts $\overline{\text{UnRTS}}$ in UOP1. Negates $\overline{\text{UnRTS}}$ in UOP0.

24.4 Functional Description

This section describes operation of the clock source generator, transmitter, and receiver.

24.4.1 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 clock generator might not produce standard baud rates if the internal bus clock is used, so the user must enable the 16-bit divider.

24.4.1.1 Programmable Divider

As Figure 24-19 shows, the UART_n transmitter and receiver can use the following clock sources:

- An external clock signal on the $\text{DT}_{n\text{IN}}$ pin. When divided by the 16-bit divider, $\text{DT}_{n\text{IN}}$ provides an asynchronous clock, which can be further divided by a 1 or 16 prescaler.
- The internal bus clock supplies an asynchronous clock source that is divided by 32 and then divided by the 16-bit value programmed in UBG1_n and UBG2_n . See Section 24.3.11, “UART Baud Rate Generator Registers ($\text{UBG1}_n/\text{UBG2}_n$).”

The choice of DTIN or internal bus clock is programmed in the UCSR .

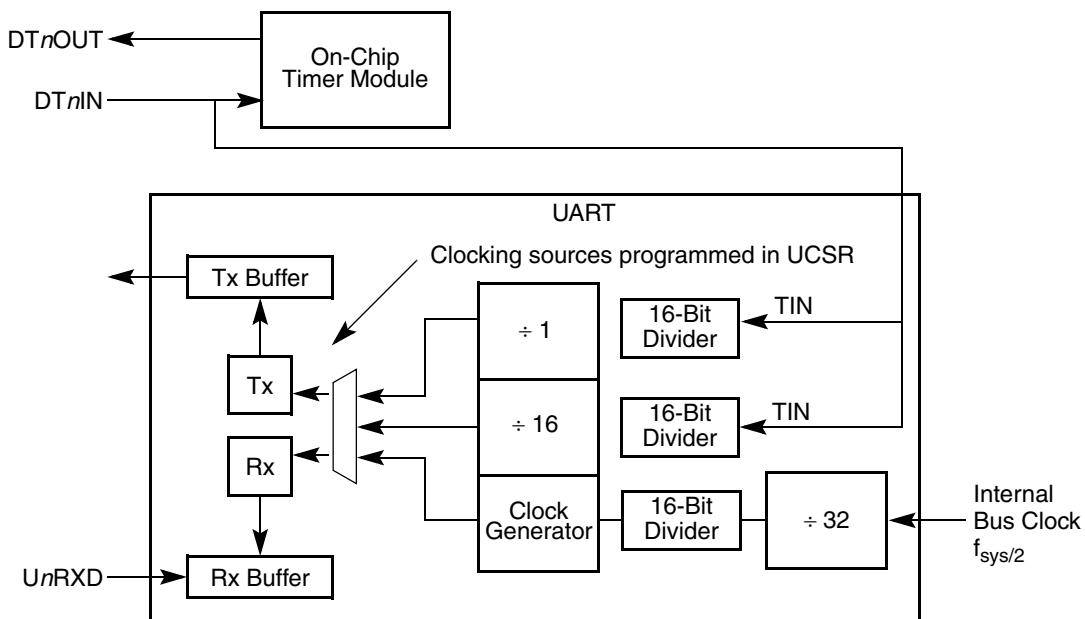


Figure 24-19. Clocking Source Diagram

NOTE

If $\text{DT}_{n\text{IN}}$ is a clocking source for either the timer or UART, that timer module cannot use $\text{DT}_{n\text{IN}}$ for timer input capture.

24.4.1.2 Calculating Baud Rates

The following sections describe how to calculate baud rates.

24.4.1.2.1 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 UBG1 n and UBG2 n registers. The baud-rate calculation is as follows:

$$\text{Baudrate} = \frac{f_{\text{sys}/2}}{[32 \times \text{divider}]} \quad \text{Eqn. 24-1}$$

Using a 83MHz internal bus clock and letting baud rate = 9600, then

Eqn. 24-2

$$\text{Divider} = \frac{83\text{MHz}}{[32 \times 9600]} = 270(\text{decimal}) = 0x010E(\text{hexadecimal}) \quad \text{Eqn. 24-3}$$

Therefore, UBG1 n = 0x01 and UBG2 n = 0x0E.

24.4.1.2.2 External Clock

An external source clock (DT n IN) passes through the 16-bit divider and then through a divide-by-1 or 16 prescaler. If f_{extc} is the external clock frequency, then the baud rate can be described with this equation:

$$\text{Baudrate} = \frac{f_{\text{extc}}}{(16 \text{ or } 1) \times (\text{16-bit divider})} \quad \text{Eqn. 24-4}$$

24.4.2 Transmitter and Receiver Operating Modes

Figure 24-20 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 24.3, “Memory Map/Register Definition.”](#)

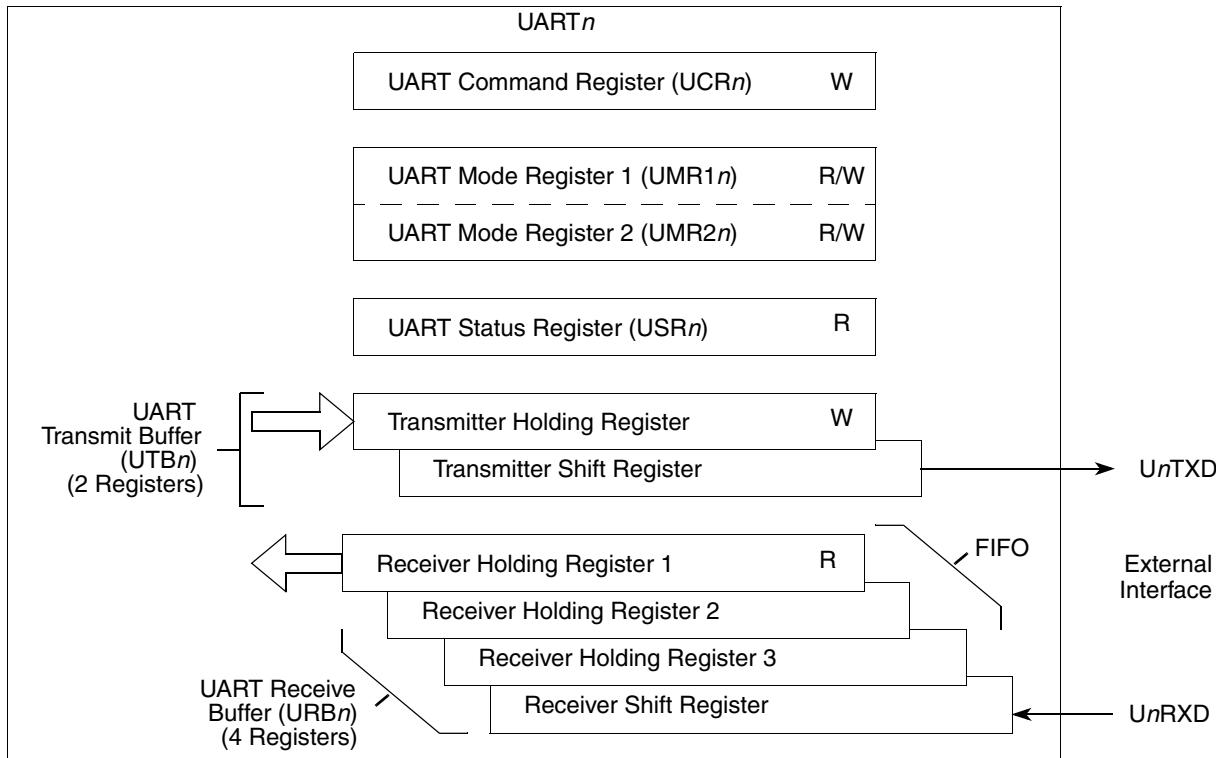


Figure 24-20. Transmitter and Receiver Functional Diagram

24.4.2.1 Transmitter

The transmitter is enabled through the **UART command register (UCR n)**. When it is ready to accept a character, the **UART** sets **USR n [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 (**USR n [TXEMP]**) is set. Transmission resumes and **TXEMP** is cleared when the CPU loads a new character into the **UART transmit buffer (UTB n)**. 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 24.3.5, “UART Command Registers \(UCR \$n\$ \)”](#)). The transmitter is reenabled through the **UCR n** to resume operation after a disable or software reset.

If the clear-to-send operation is enabled, **UnCTS** must be asserted for the character to be transmitted. If **UnCTS** is negated in the middle of a transmission, the character in the shift register is sent and **UnTXD** remains in mark state until **UnCTS** is reasserted. If the transmitter is forced to send a continuous low condition by issuing a SEND BREAK command, the transmitter ignores the state of **UnCTS**.

If the transmitter is programmed to automatically negate $\overline{\text{UnRTS}}$ when a message transmission completes, $\overline{\text{UnRTS}}$ must be asserted manually before a message is sent. In applications in which the transmitter is disabled after transmission is complete and $\overline{\text{UnRTS}}$ is appropriately programmed, $\overline{\text{UnRTS}}$ is negated one bit time after the character in the shift register is completely transmitted. The transmitter must be manually reenabled by reasserting $\overline{\text{UnRTS}}$ before the next message is to be sent.

Figure 24-21 shows the functional timing information for the transmitter.

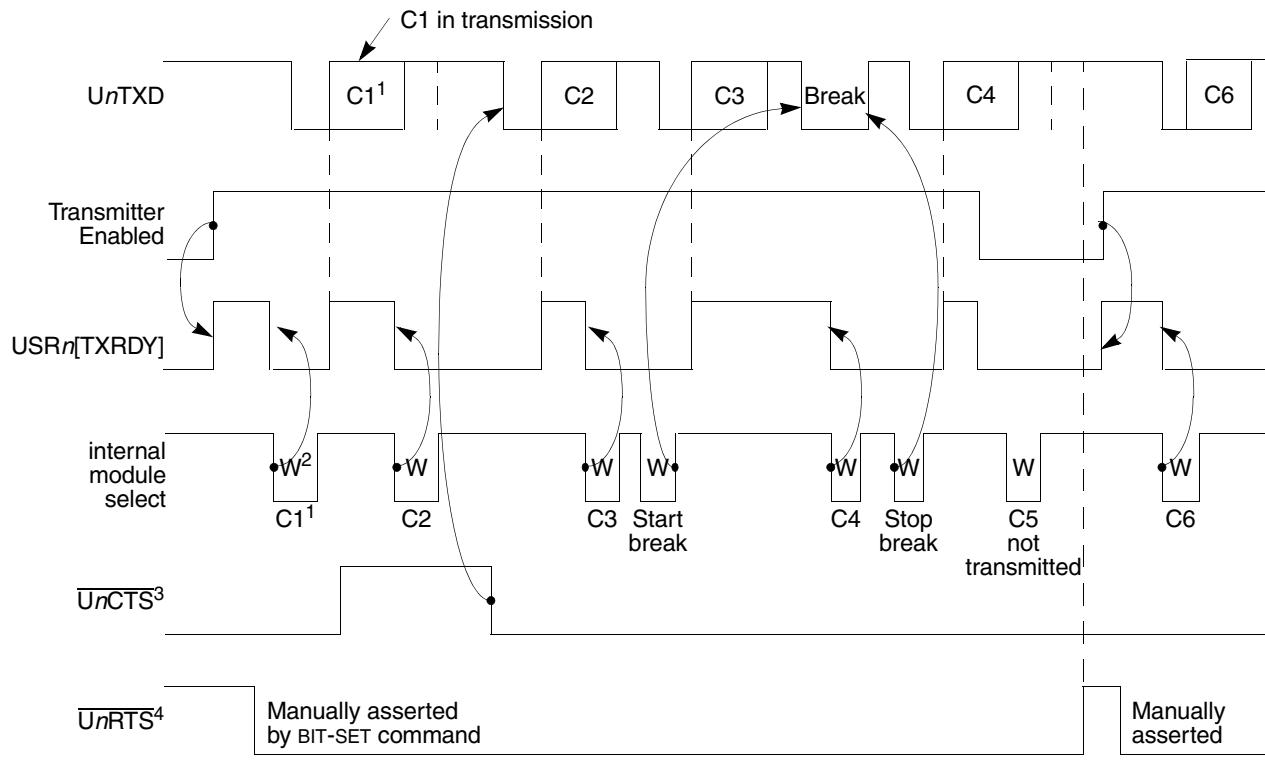


Figure 24-21. Transmitter Timing Diagram

24.4.2.2 Receiver

The receiver is enabled through its UCR_n , as described in Section 24.3.5, “UART Command Registers (UCR_n).”

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, the start bit is invalid and the search for the valid start bit begins again.

If UnRXD is still low, a valid start bit is assumed and 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

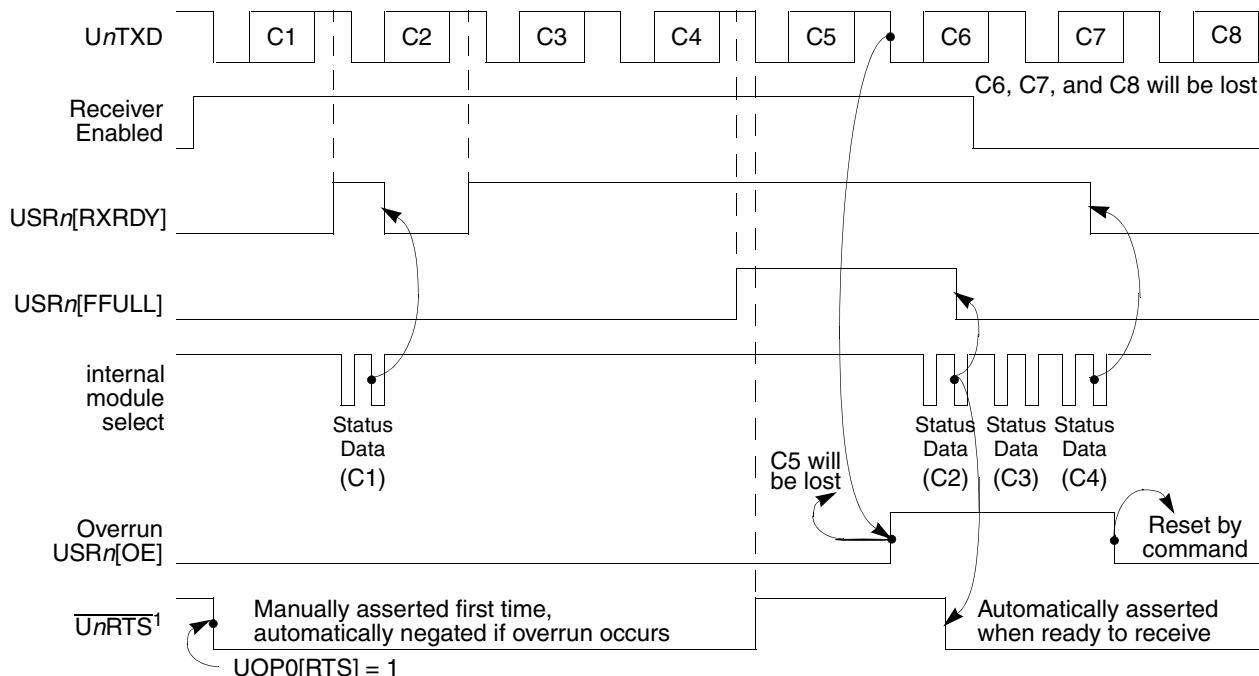
programmed clock source. The lsb is received first. The data is then transferred to a receiver holding register and $\text{USR}_n[\text{RXRDY}]$ is set. If the character is less than 8 bits, the most significant unused bits in the receiver holding register are cleared.

After the stop bit is detected, the 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, the 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 USR_n at the received character boundary. They are valid only if $\text{USR}_n[\text{RXRDY}]$ is set.

If a break condition is detected (UnRXD is low for the entire character including the stop bit), a character of all zeros is loaded into the receiver holding register and $\text{USR}_n[\text{RB},\text{RXRDY}]$ 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. If the break begins in the middle of a character, the receiver places the damaged character in the Rx FIFO and sets the corresponding USR_n error bits and $\text{USR}_n[\text{RXRDY}]$. Then, if the break lasts until the next character time, the receiver places an all-zero character into the Rx FIFO and sets $\text{USR}_n[\text{RB},\text{RXRDY}]$.

[Figure 24-22](#) shows receiver functional timing.



¹ $\text{UMR}_2[\text{TXRTS}] = 1$

Figure 24-22. Receiver Timing

24.4.2.3 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 \overline{UnRXD} (see [Figure 24-20](#)). Data is assembled in the receiver shift register and loaded into the top empty receiver holding register position of the FIFO. Thus, 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. Either the RXRDY or FFULL bit can be selected to cause an interrupt and either 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 $USRn$ 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 $USRn$ does not affect the FIFO. The FIFO is popped only when the receive buffer is read. The $USRn$ 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 $USRn[OE]$ 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, overrun errors can be prevented by connecting \overline{UnRTS} to the \overline{UnCTS} input of the transmitting device.

NOTE

The receiver can still read characters in the FIFO if the receiver is disabled. If the receiver is reset, the FIFO, \overline{UnRTS} control, all receiver status bits, and interrupts, and DMA requests are reset. No more characters are received until the receiver is reenabled.

24.4.3 Looping Modes

The UART can be configured to operate in various looping modes, as shown in [Figure 24-22](#). These modes are useful for local and remote system diagnostic functions. The modes are described in the following paragraphs and in [Section 24.3, “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.

24.4.3.1 Automatic Echo Mode

In automatic echo mode, shown in [Figure 24-23](#), 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.

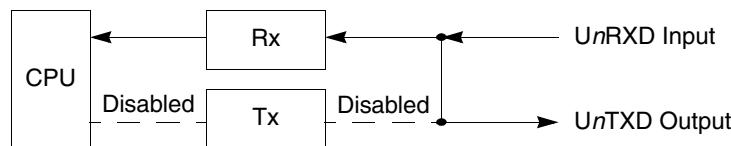


Figure 24-23. Automatic Echo

Because the transmitter is inactive, $USRn[TXEMP,TXRDY]$ are 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.

24.4.3.2 Local Loop-back Mode

[Figure 24-24](#) shows how $UnTXD$ and $UnRXD$ are internally connected in local loop-back 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.

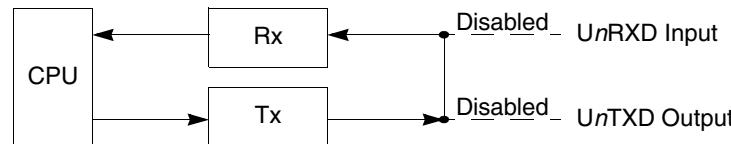


Figure 24-24. Local Loop-back

Features of this local loop-back mode are as follows:

- 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.

24.4.3.3 Remote Loop-back Mode

In remote loop-back mode, shown in [Figure 24-25](#), 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, the 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 the next valid start bit is detected.

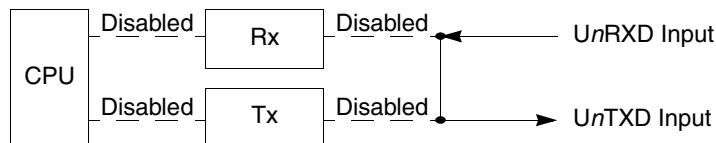


Figure 24-25. Remote Loop-back

24.4.4 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 USRn[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 24-26](#).

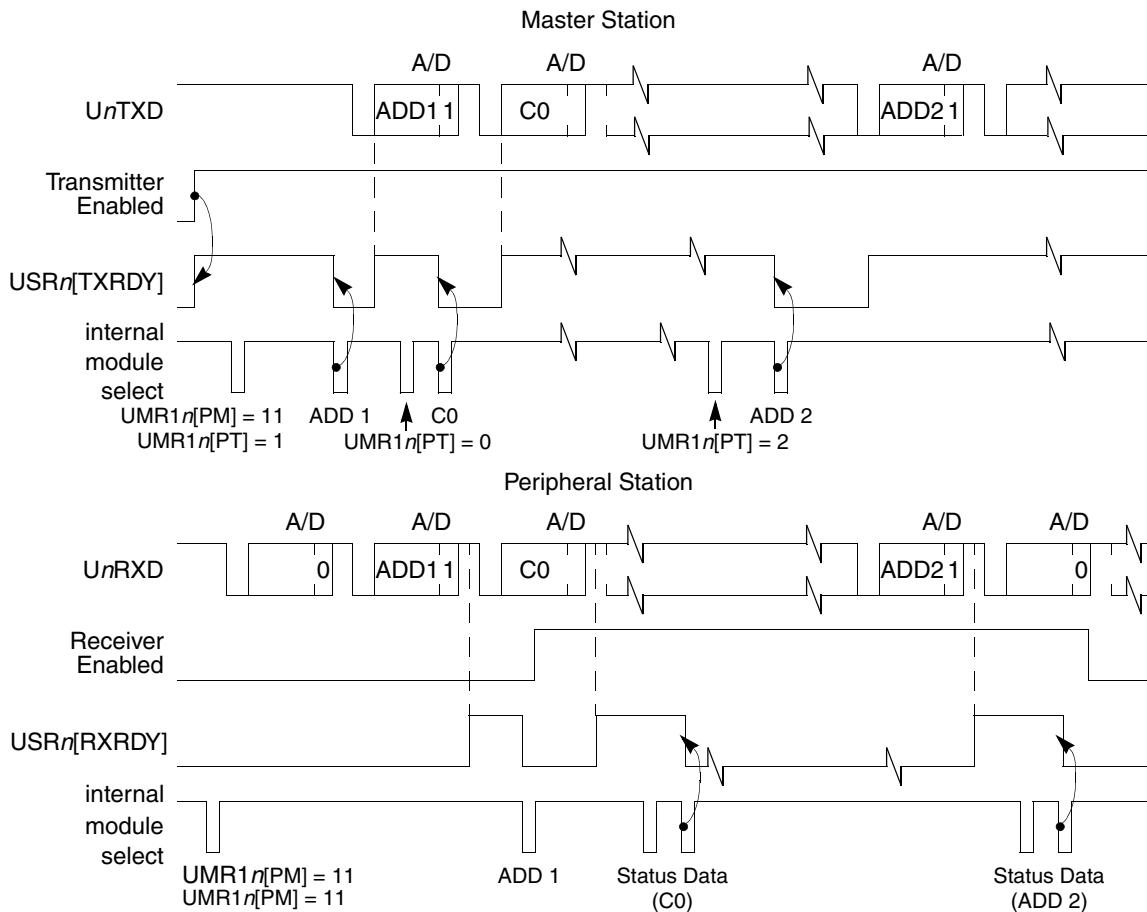


Figure 24-26. Multidrop Mode Timing Diagram

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 = 1 indicates an address character; A/D = 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 one (address tag). The character is discarded if the received A/D bit is zero (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, the data bits are loaded into the data portion of the FIFO while the A/D bit is loaded into the status portion of the FIFO normally used for a parity error (USRn[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 still contain 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.

24.4.5 Bus Operation

This section describes bus operation during read, write, and interrupt acknowledge cycles to the UART module.

24.4.5.1 Read Cycles

The UART module responds to reads with byte data. Reserved registers return zeros.

24.4.5.2 Write Cycles

The UART module accepts write data as bytes only. Write cycles to read-only or reserved registers complete normally without exception processing, but data is ignored.

24.4.6 Programming

The software flowchart, [Figure 24-27](#), consists of the following:

- UART module initialization—These routines consist of SINIT and CHCHK (See Sheet 1 p. 24-29 and Sheet 2 p. 24-30). 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 loop-back 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. 24-32 and Sheet 5 p. 24-33) 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. 24-32), 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.

24.4.6.1 Interrupt and DMA Request Initialization

24.4.6.1.1 Setting up the UART to Generate Core Interrupts

The list below gives the steps needed to properly initialize the UART to generate an interrupt request to the core. See [Section 14.2.9.1, “Interrupt Sources,”](#) for details on interrupt assignments for the UART modules.

1. Initialize the appropriate ICRx 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 are being used to generate interrupt requests, then verify that the corresponding UART DMA channels are not enabled.
5. Initialize interrupts in the UART, see [Table 24-14](#).

Table 24-14. UART Interrupts

Register	Bit	Interrupt
UMR1n	6	RxIRQ
UIMRn	7	Change of State (COS)
UIMRn	2	Delta Break
UIMRn	1	RxFIFO Full
UIMRn	0	TXRDY

24.4.6.1.2 Setting up the UART to Request DMA Service

The UART is capable of generating two different DMA request signals: transmit and receive.

The transmit DMA request signal is asserted when the TXRDY (transmitter ready) in the UART interrupt status register (UISRn[TXRDY]) is set. When the transmit DMA request signal is asserted, the DMA can initiate a data copy, reading the next character to be transmitted from memory and writing it into the UART transmit buffer (UTBn). This would allow the DMA channel to stream data from memory to the UART for transmission without processor intervention. Once 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 (UISRn[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 (URBn) and storing them in memory. This allows the DMA channel to stream data from the UART receive buffer into memory without processor intervention. Once 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 still supporting interrupt notification to the processor for CTS change-of-state and delta break error handling.

[Table 24-15](#) shows the DMA requests.

Table 24-15. UART DMA Requests

Register	Bit	DMA Request
UISR n	1	Receive DMA request
UISR n	0	Transmit DMA request

24.4.6.2 UART Module Initialization Sequence

Table 24-16 shows the UART module initialization sequence.

Table 24-16. UART Module Initialization Sequence

Register	Setting
UCR n	Reset the receiver and transmitter. Reset the mode pointer (MISC[2–0] = 0b001).
UIMR n	Enable the desired interrupt sources.
UACR n	Initialize the input enable control (IEC bit).
UCSR n	Select the receiver and transmitter clock. Use timer as source if required.
UMR1 n	If preferred, program operation of receiver ready-to-send (RXRTS bit). Select receiver-ready or FIFO-full notification (RXRDY/FFULL bit). Select character or block error mode (ERR bit). Select parity mode and type (PM and PT bits). Select number of bits per character (B/Cx bits).
UMR2 n	Select the mode of operation (CMx bits). If preferred, program operation of transmitter ready-to-send (TXRTS). If preferred, program operation of clear-to-send (TXCTS bit). Select stop-bit length (SBx bits).
UCR n	Enable transmitter and/or receiver.

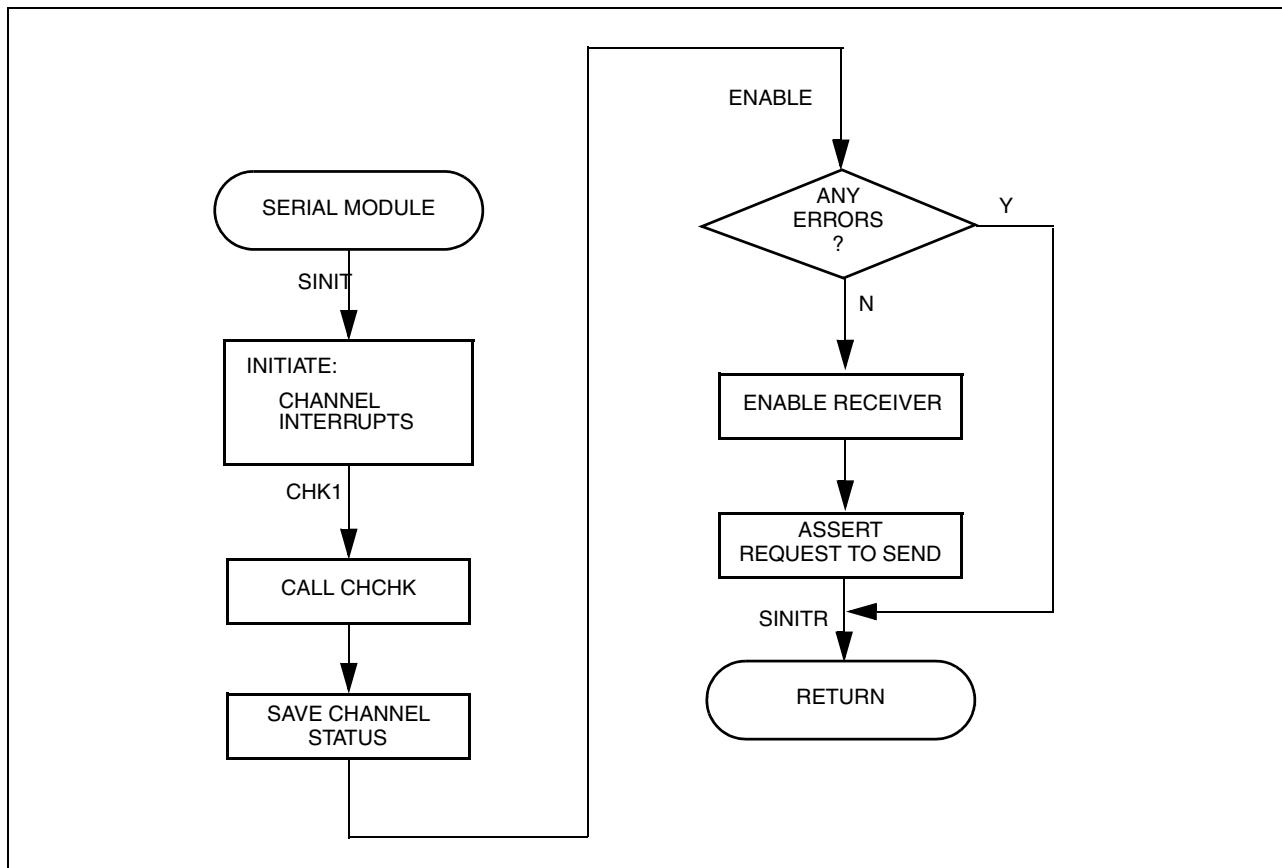


Figure 24-27. UART Mode Programming Flowchart (Sheet 1 of 5)

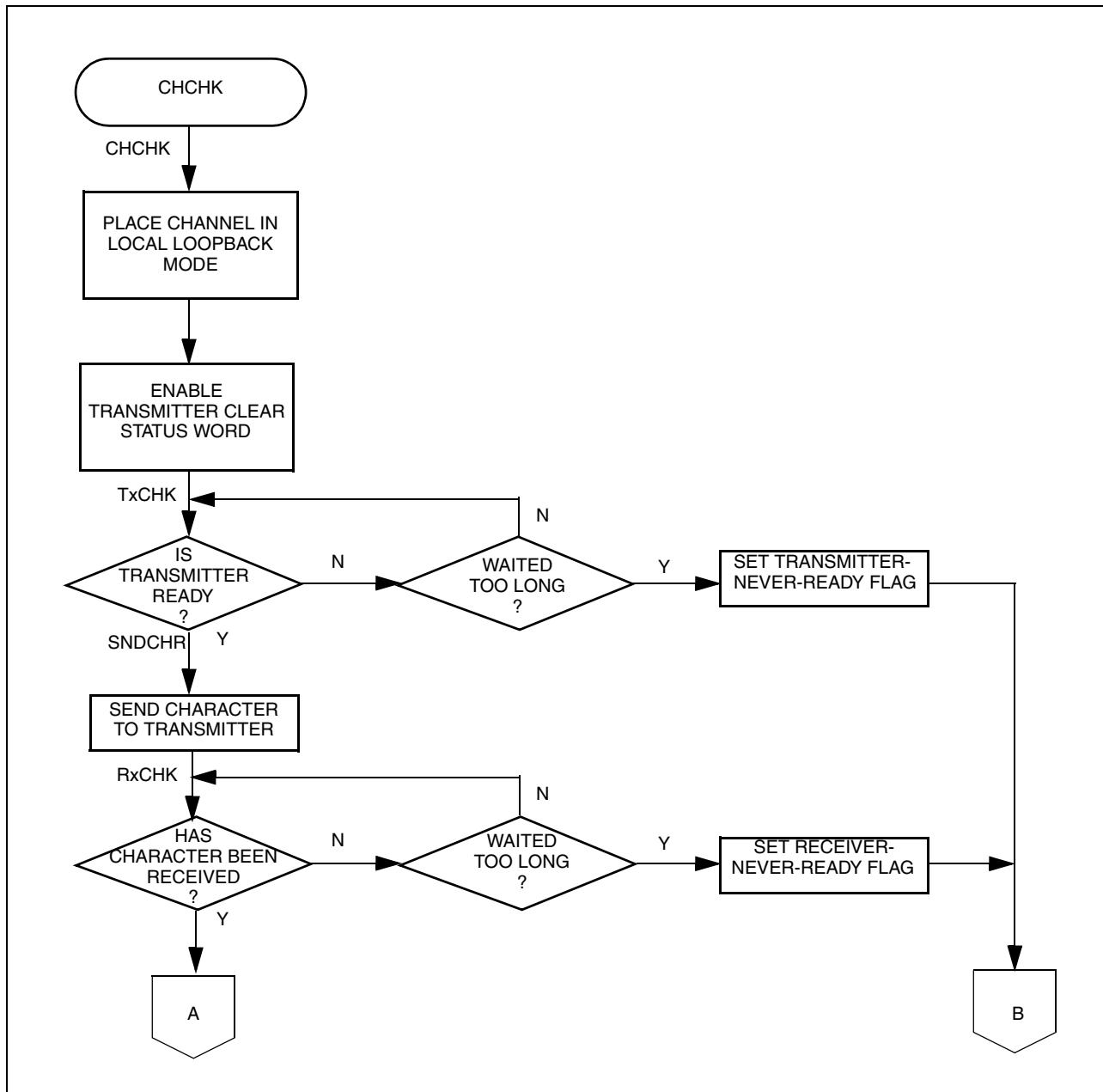


Figure 24-27. UART Mode Programming Flowchart (Sheet 2 of 5)

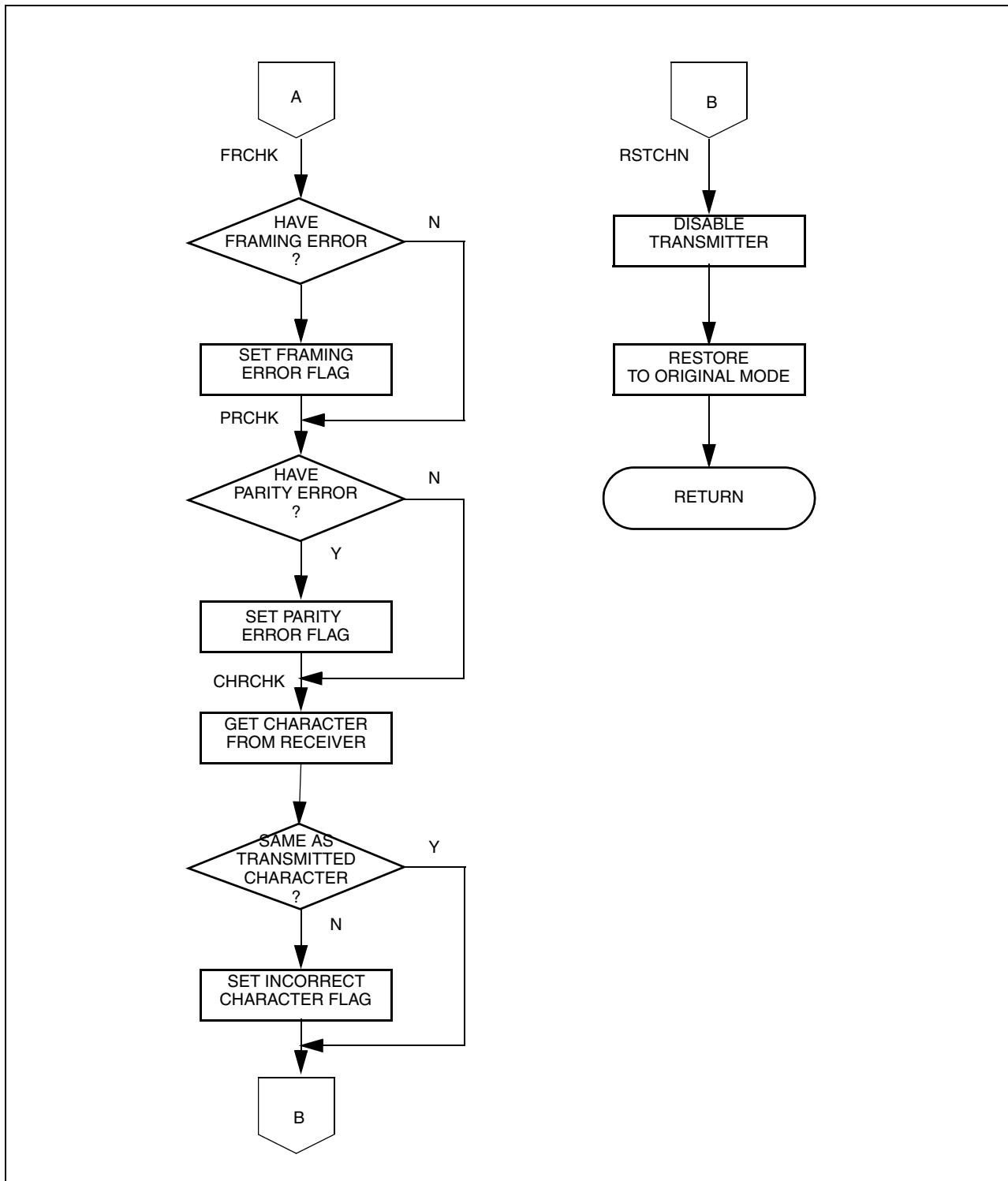


Figure 24-27. UART Mode Programming Flowchart (Sheet 3 of 5)

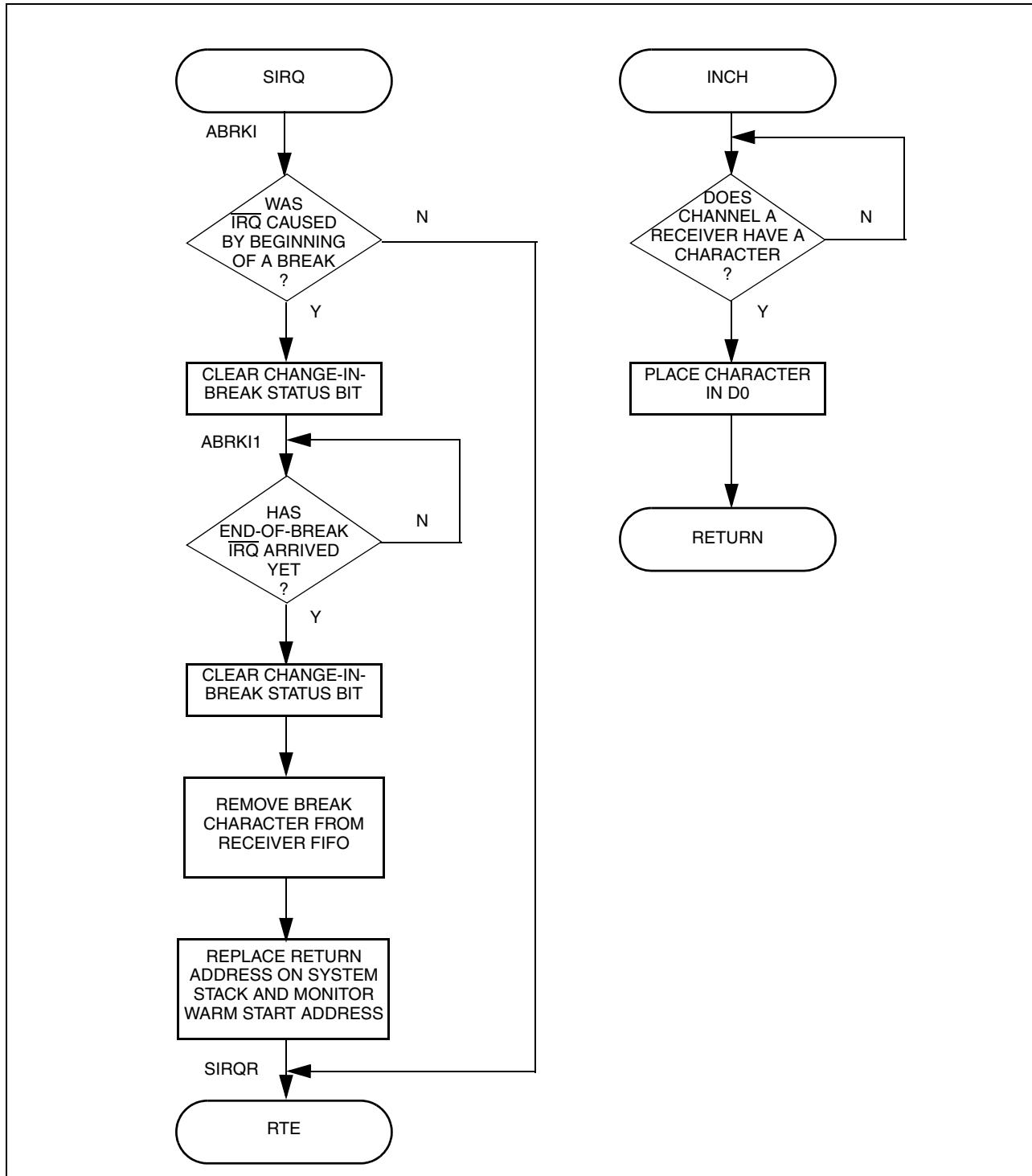


Figure 24-27. UART Mode Programming Flowchart (Sheet 4 of 5)

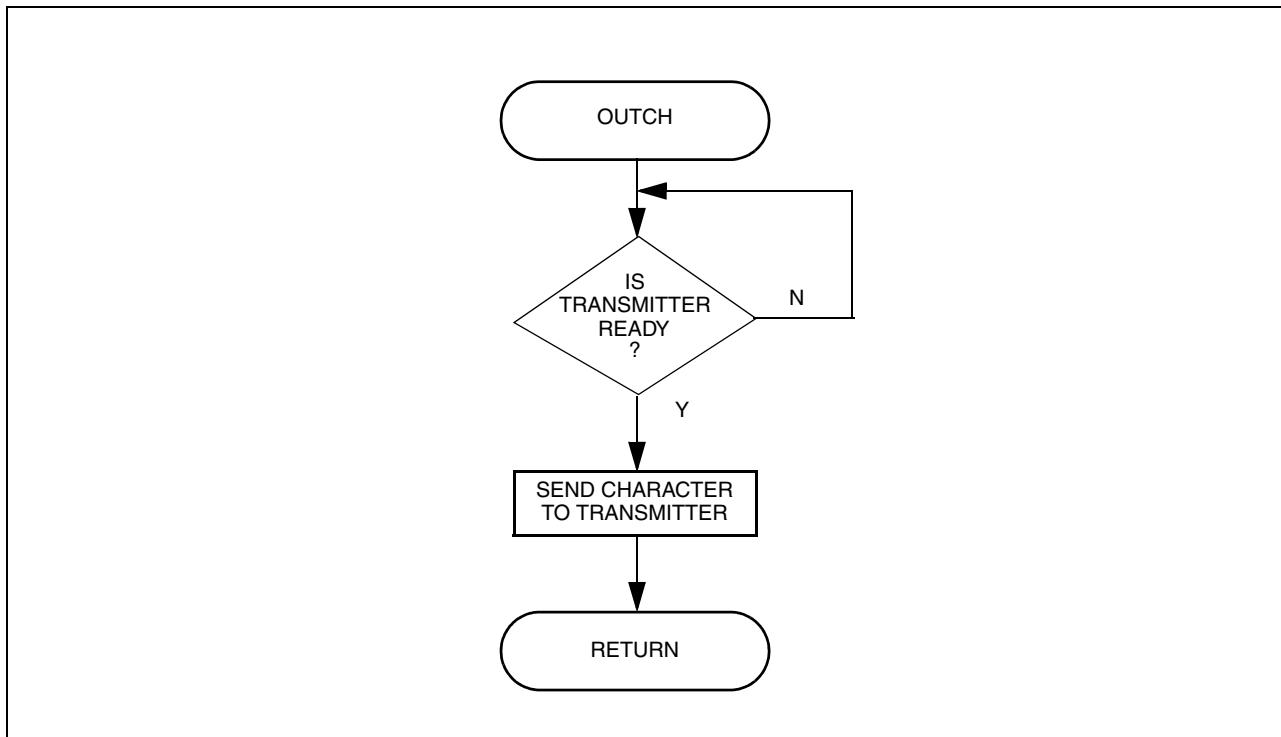


Figure 24-27. UART Mode Programming Flowchart (Sheet 5 of 5)

Chapter 25

I²C Interface

25.1 Introduction

This chapter describes the I²C module, clock synchronization, and I²C programming model registers. It also provides extensive programming examples.

25.2 Overview

I²C 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 I²C bus allows additional devices to be connected to the bus for expansion and system development.

The interface is designed to operate 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 that can be connected are limited by a maximum bus capacitance of 400 pF.

The I²C 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 I²C module is designed to be compatible with the Philips I²C bus protocol. For information on system configuration, protocol, and restrictions, see *The I²C Bus Specification, Version 2.1*.

NOTE

The GPIO module must be configured to enable the peripheral function of the appropriate pins (refer to [Chapter 13, “General Purpose I/O Module”](#)) prior to configuring the I²C module.

25.3 Features

The I²C module has the following key features:

- Compatibility with I²C bus standard version 2.1
- Support for 3.3-V tolerant devices
- Multiple-master operation

I²C Interface

- 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

Figure 25-1 is a block diagram of the I²C module.

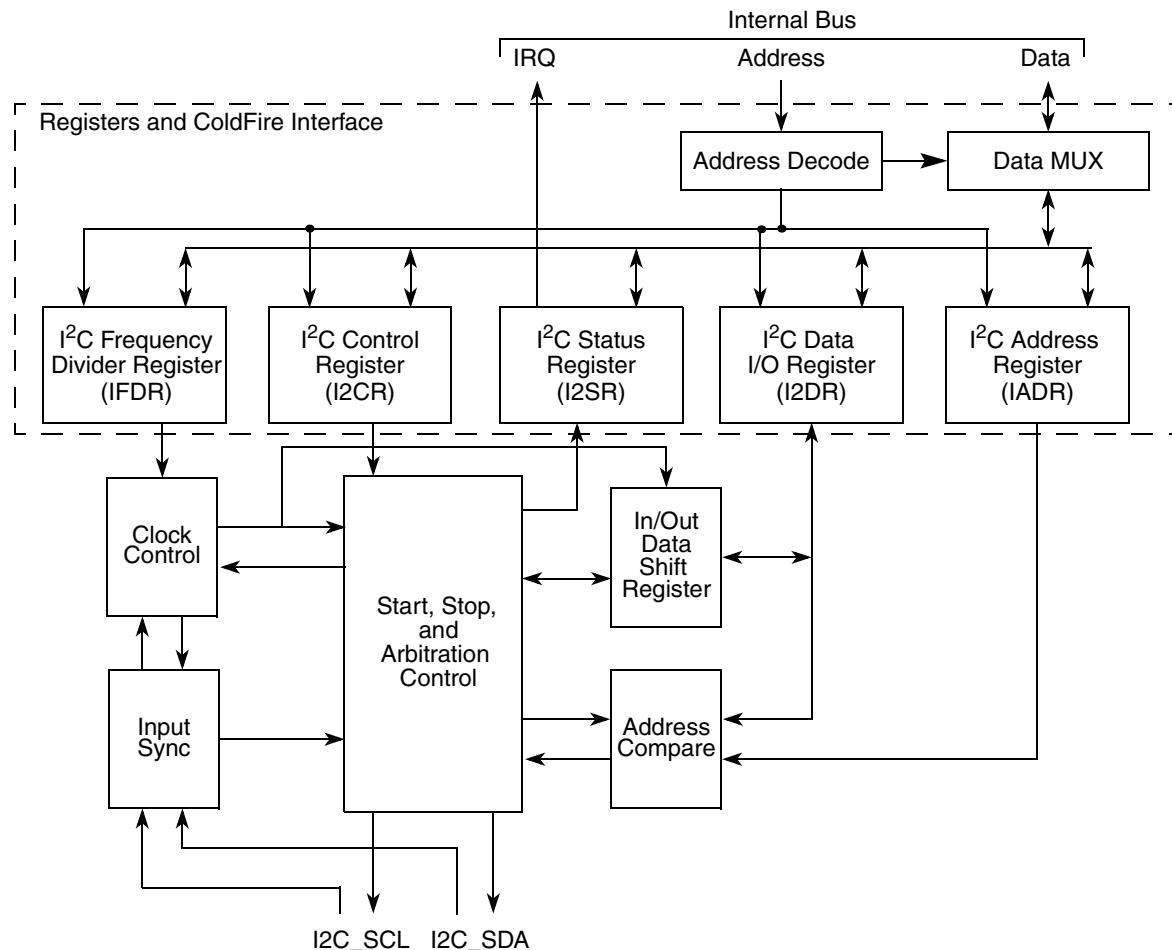


Figure 25-1. I²C Module Block Diagram

Figure 25-1 shows the I²C registers, which are described in Section 25.5, “Memory Map/Register Definition”:

- I²C address register (I2ADR)
- I²C frequency divider register (I2FDR)

- I²C control register (I2CR)
- I²C status register (I2SR)
- I²C data I/O register (I2DR)

25.4 I²C System Configuration

The I²C module uses a serial data line (I2C_SDA) and a serial clock line (I2C_SCL) for data transfer. For I²C 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 I²C default state is as a slave receiver. Thus, when not programmed to be a master or responding to a slave transmit address, the I²C module should return to the default slave receiver state. See [Section 25.6.1, “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.

25.4.1 START Signal

When no other device is bus master (both I2C_SCL and I2C_SDA lines are at logic high), a device can initiate communication by sending a START signal (see A in [Figure 25-2](#)). A START signal is defined as a high-to-low transition of I2C_SDA while I2C_SCL is high. This signal denotes the beginning of a data transfer (each data transfer can be several bytes long) and awakens all slaves.

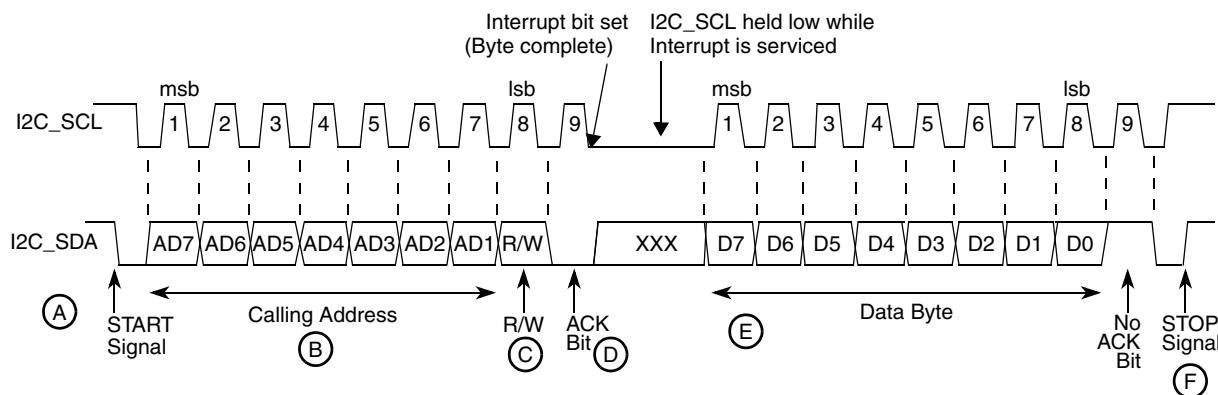


Figure 25-2. I²C Standard Communication Protocol

25.4.2 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 = write transfer, 1 = read transfer).

Each slave must have a unique address. An I²C 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 I₂C_SDA low at the ninth serial clock (D) to return an acknowledge bit.

25.4.3 Data Transfer

When successful slave addressing is achieved, the data transfer can proceed (see E in Figure 25-2) 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 I₂C_SCL is low and must be held stable while I₂C_SCL is high, as Figure 25-2 shows. I₂C_SCL is pulsed once for each data bit, with the msb being sent first. The receiving device must acknowledge each byte by pulling I₂C_SDA low at the ninth clock; therefore, a data byte transfer takes nine clock pulses. See Figure 25-3.

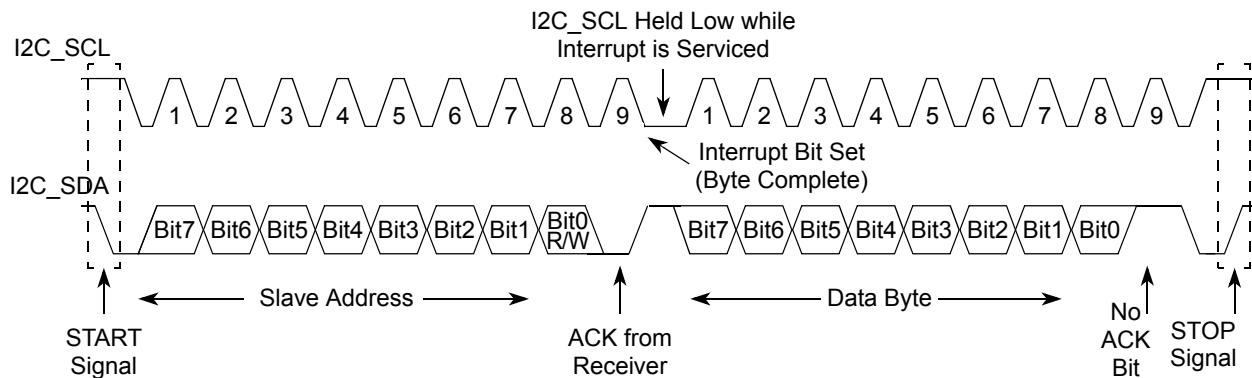


Figure 25-3. Data Transfer

25.4.4 Acknowledge

The transmitter releases the I₂C_SDA line high during the acknowledge clock pulse as shown in Figure 25-4. The receiver pulls down the I₂C_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 I₂C_SDA high. The master can then generate a STOP signal to abort the data transfer or generate a START signal (repeated start, shown in Figure 25-5 and discussed in Section 25.4.6, “Repeated START”) to start a new calling sequence.

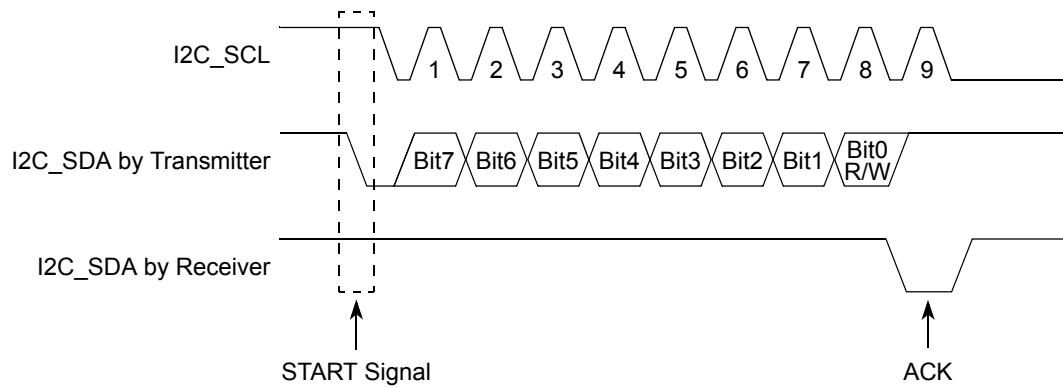


Figure 25-4. Acknowledgement by Receiver

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 I²C_SDA for the master to generate a STOP or START signal (Figure 25-4).

25.4.5 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 I²C_SDA while I²C_SCL is at logical high (see F in Figure 25-2). 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 25.4.6, “Repeated START.”

25.4.6 Repeated START

A repeated START signal is a START signal generated without first generating a STOP signal to terminate the communication. 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.

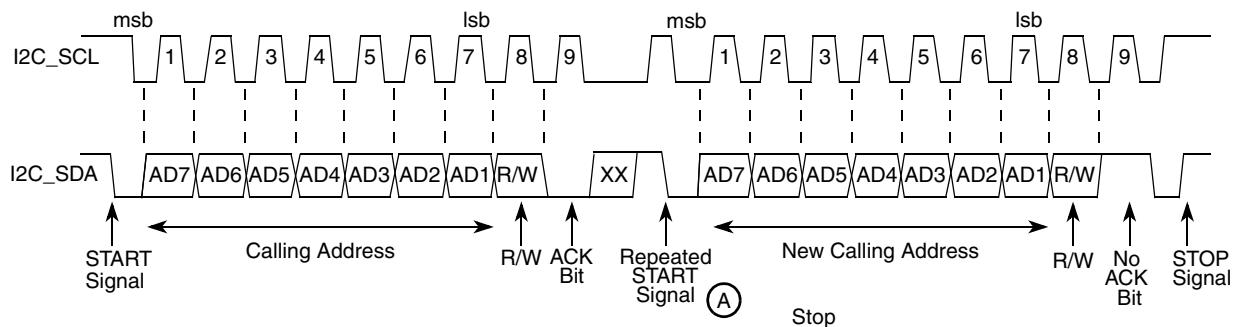


Figure 25-5. Repeated START

Various combinations of read/write formats are then possible:

- The first example in Figure 25-6 is the case of master-transmitter transmitting to slave-receiver. The transfer direction is not changed.

- The second example in [Figure 25-6](#) 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 25-6](#), the START condition and slave address are both 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 (I₂C_SDA low)
 \bar{A} = Not Acknowledge (I₂C_SDA high)
 Rept ST = Repeated Start

From Master to Slave 	From Slave to Master
--------------------------	--------------------------

Example 1:

ST	7bit Slave Address	0	A	Register Address	A	DATA	A/ \bar{A}	SP
----	--------------------	---	---	------------------	---	------	--------------	----

Example 2:

ST	7bit Slave Address	1	A	DATA	A	DATA	\bar{A}	SP
----	--------------------	---	---	------	---	------	-----------	----

Note: No acknowledge on the last byte

Example 3:

ST	7-bit Slave Address	1	A	DATA	\bar{A}	Rept ST	7-bit Slave Address	0	A	DATA	A	DATA	A/ \bar{A}	SP
Master Reads from Slave							Master Writes to Slave							

Figure 25-6. Data Transfer, Combined Format

25.4.7 Clock Synchronization and Arbitration

I²C is a true multi-master bus that allows more than one master to be 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 I₂C_SCL line, a high-to-low transition on the I₂C_SCL line affects all the devices connected on the bus. The devices start counting their low period and once a device's clock has gone low, it holds the I₂C_SCL line low until the clock high state is reached. However, the change of low to high in this device's clock may not change the state of the I₂C_SCL line if another device clock is still within its low period. Therefore, synchronized clock I₂C_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 25-8](#)). When all devices concerned have counted off their low period, the synchronized clock (I₂C_SCL) line is released and pulled high. At this point, the device clocks and the I₂C_SCL line are synchronized, and the devices start counting their high periods. The first device to complete its high period pulls the I₂C_SCL line low again.

The relative priority of the contending masters is determined by a data arbitration procedure. 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 I₂C_SDA output (see [Figure 25-7](#)). In this case the transition from master to slave mode does not generate a STOP condition. Meanwhile, hardware sets I₂SR[IAL] to indicate loss of arbitration.

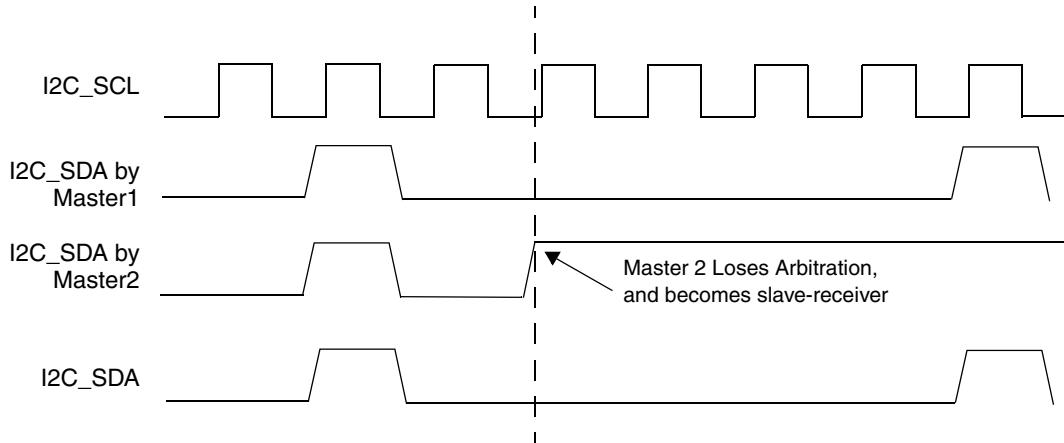


Figure 25-7. Arbitration Procedure

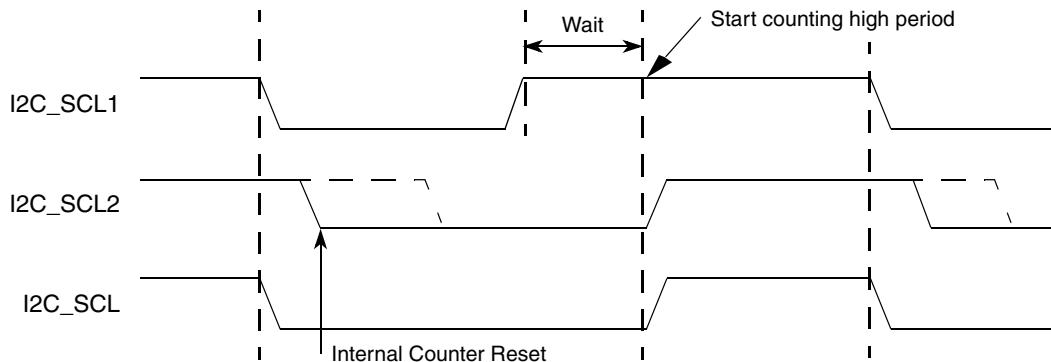


Figure 25-8. Clock Synchronization

25.4.8 Handshaking and Clock Stretching

The clock synchronization mechanism can be used as a handshake in data transfers. Slave devices can hold I₂C_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 I₂C_SCL.

Slaves may also slow down the transfer bit rate. After the master has driven I₂C_SCL low, the slave can drive I₂C_SCL low for the required period and then release it. If the slave I₂C_SCL low period is longer than the master I₂C_SCL low period, the resulting I₂C_SCL bus signal low period is stretched.

25.5 Memory Map/Register Definition

Table 25-1 lists the configuration registers used in the I²C interface.

Table 25-1. I²C Module Memory Map

Address	Register	Access	Reset Value	Section/Page
0xFC05_8000	I ² C Address Register (I2ADR)	R/W	0x00	25.5.1/25-8
0xFC05_8004	I ² C Frequency Divider Register (I2FDR)	R/W	0x00	25.5.2/25-8
0xFC05_8008	I ² C Control Register (I2CR)	R/W	0x00	25.5.3/25-9
0xFC05_800C	I ² C Status Register (I2SR)	R/W	0x81	25.5.4/25-10
0xFC05_8010	I ² C Data I/O Register (I2DR)	R/W	0x00	25.5.5/25-11

25.5.1 I²C Address Register (I2ADR)

The I2ADR holds the address the I²C responds to when addressed as a slave. Note that it is not the address sent on the bus during the address transfer when the module is performing a master transfer.

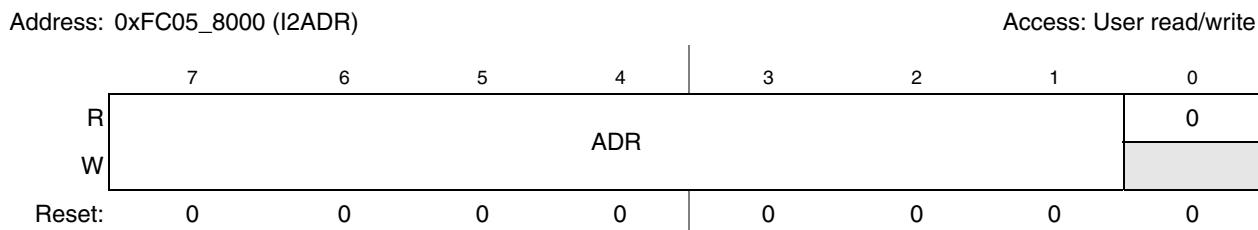


Figure 25-9. I²C Address Register (I2ADR)

Table 25-2. I2ADR Field Descriptions

Field	Description
7–1 ADR	Slave address. Contains the specific slave address to be used by the I ² C module. Slave mode is the default I ² C mode for an address match on the bus.
0	Reserved, should be cleared.

25.5.2 I²C Frequency Divider Register (I2FDR)

The I2FDR, shown in Figure 25-10, provides a programmable prescaler to configure the I²C clock for bit-rate selection.

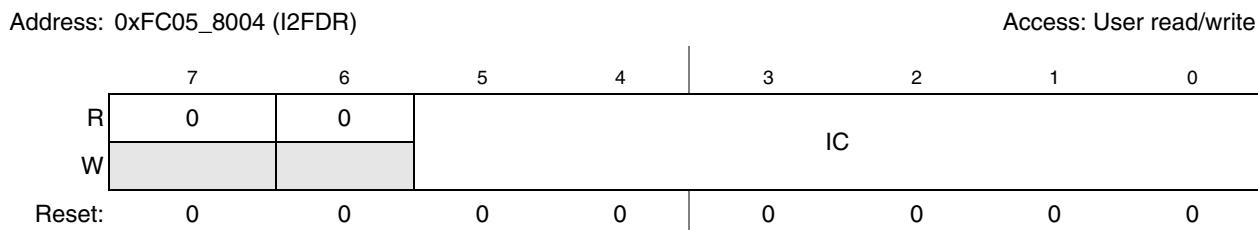


Figure 25-10. I²C Frequency Divider Register (I2FDR)

Table 25-3. I2FDR Field Descriptions

Field	Description																																																																																																																																									
7–6	Reserved, should be cleared.																																																																																																																																									
5–0 IC	I ² 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 I ² C_SCL and I ² C_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																																																																																																																																									

25.5.3 I²C Control Register (I2CR)

The I2CR is used to enable the I²C module and the I²C interrupt. It also contains bits that govern operation as a slave or a master.

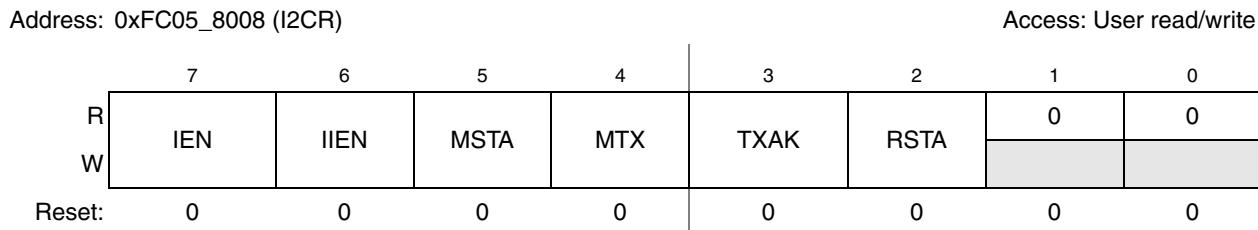
**Figure 25-11. I²C Control Register (I2CR)**

Table 25-4. I2CR Field Descriptions

Field	Description
7 IEN	I ² C enable. Controls the software reset of the entire I ² 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; so initiating a start cycle may corrupt the current bus cycle, ultimately causing either the current master or the I ² C module to lose arbitration, after which bus operation returns to normal. 0 The I ² C module is disabled, but registers can still be accessed. 1 The I ² C module is enabled. This bit must be set before any other I2CR bits have any effect.
6 IIEN	I ² C interrupt enable. 0 I ² C module interrupts are disabled, but currently pending interrupt condition is not cleared. 1 I ² C module interrupts are enabled. An I ² C interrupt occurs if I2SR[IIF] is also set.
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 should set MTX according to I2SR[SRW]. In master mode, MTX should 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 I2C_SDA during acknowledge cycles for both master and slave receivers. Note that writing TXAK applies only when the I ² 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 (that is, 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.
1–0	Reserved, should be cleared.

25.5.4 I²C Status Register (I2SR)

This I2SR contains bits that indicate transaction direction and status.

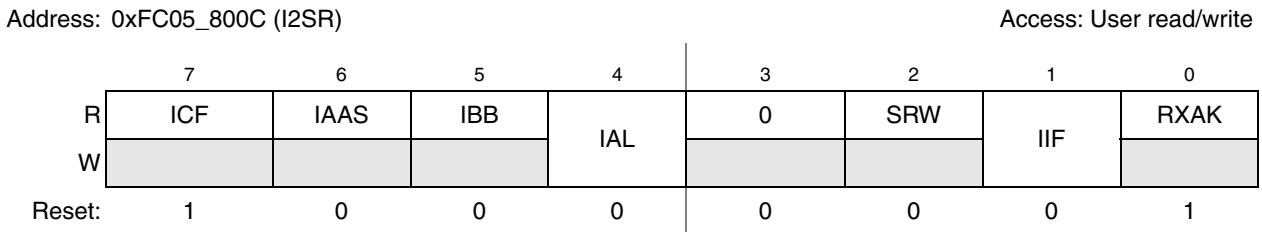
**Figure 25-12. I²C Status Register (I2SR)**

Table 25-5. I2SR Field Descriptions

Field	Description
7 ICF	I ² C Data transferring bit. While one byte of data is transferred, ICF is cleared. 0 Transfer in progress 1 Transfer complete. Set by the falling edge of the ninth clock of a byte transfer.
6 IAAS	I ² C addressed as a slave bit. The CPU is interrupted if I2CR[IIEN] is set. Next, the CPU must check SRW and set its TX/RX mode accordingly. Writing to I2CR clears this bit. 0 Not addressed. 1 Addressed as a slave. Set when its own address (IADR) matches the calling address.
5 IBB	I ² 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 ² 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"> • I2C_SDA sampled low when the master drives high during an address or data-transmit cycle. • I2C_SDA sampled low when the master drives high during the acknowledge bit of a data-receive cycle. • A start cycle is attempted when the bus is busy. • A repeated start cycle is requested in slave mode. • A stop condition is detected when the master did not request it.
3	Reserved, should 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 ² 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 ² C interrupt. Must be cleared by software by writing a zero in the interrupt routine. 0 No I ² C interrupt pending 1 An interrupt is pending, which causes a processor interrupt request (if IIEN = 1). Set when one of the following occurs: <ul style="list-style-type: none"> • Complete one byte transfer (set at the falling edge of the ninth clock) • Reception of a calling address that matches its own specific address in slave-receive mode • Arbitration lost
0 RXAK	Received acknowledge. The value of I2C_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.

25.5.5 I²C Data I/O Register (I2DR)

In master-receive mode, reading the I2DR allows a read to occur and for the next data byte to be received. In slave mode, the same function is available once the I²C has received its slave address.

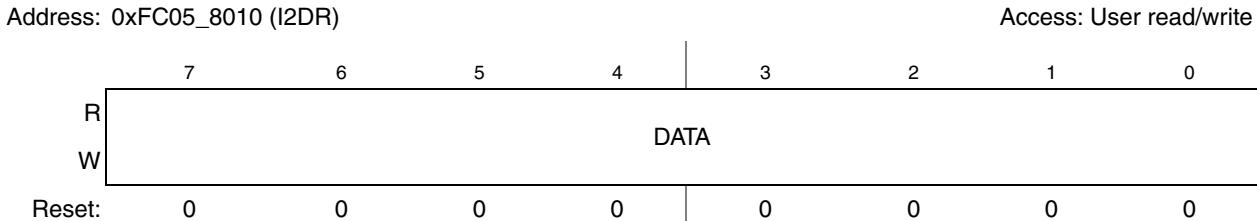
Figure 25-13. I²C Data I/O Register (I2DR)

Table 25-6. I2DR Field Description

Field	Description
7–0 DATA	I ² C data. In master transmit mode, when data is written to this register, 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. Note: In master transmit mode, the first byte of data written to I2DR following assertion of I2CR[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. Note: I2CR[MSTA] generates a start when a master does not already own the bus. I2CR[RSTA] generates a start (restart) without the master first issuing a stop (i.e., the master already owns the bus). In order to start the read of data, a dummy read to this register starts the read process from the slave. The next read of the I2DR register contains the actual data.

25.6 I²C Programming Examples

The following examples show programming for initialization, signaling START, post-transfer software response, signaling STOP, and generating a repeated START.

25.6.1 Initialization Sequence

Before the interface can transfer serial data, registers must be initialized:

1. Set I2FDR[IC] to obtain I2C_SCL frequency from the system bus clock. See [Section 25.5.2, “I2C Frequency Divider Register \(I2FDR\).”](#)
2. Update the I2ADR to define its slave address.
3. Set I2CR[IEN] to enable the I²C bus interface system.
4. Modify the I2CR to select or deselect master/slave mode, transmit/receive mode, and interrupt-enable or not.

NOTE

If I2SR[IBB] is set when the I²C 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 just power-cycled on.

```
I2CR = 0x0
I2CR = 0xA0
dummy read of I2DR
```

```
I2SR = 0x0
I2CR = 0x0
```

25.6.2 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, I2SR[IBB] must be tested to determine whether the serial bus is free. If the bus is free (IBB = 0), 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 I2C_SCL period, it may be necessary to wait until the I2C is busy after writing the calling address to the I2DR before proceeding with the following instructions.

The following example signals START and transmits the first byte of data (slave address):

```
CHFLAG MOVE.B I2SR,-(A0)          ;Check I2SR[MBB]
BTST.B #5, (A0)+

TXSTART BNE.S CHFLAG            ;If I2SR[MBB] = 1, wait until it is clear
MOVE.B I2CR,-(A0)              ;Set transmit mode
BSET.B #4, (A0)
MOVE.B (A0)+, I2CR
MOVE.B I2CR, -(A0)             ;Set master mode
BSET.B #5, (A0)                ;Generate START condition
MOVE.B (A0)+, I2CR
MOVE.B CALLING,-(A0)           ;Transmit the calling address, D0=R/W
MOVE.B (A0)+, I2DR
IFREE  MOVE.B I2SR,-(A0)          ;Check I2SR[MBB]
                                ;If it is clear, wait until it is set.
BTST.B #5, (A0)+
BEQ.S IFREE;
```

25.6.3 Post-Transfer Software Response

Sending or receiving a byte sets the I2SR[ICF], which indicates one byte communication is finished. I2SR[IIF] is also set. An interrupt is generated if the interrupt function is enabled during initialization by setting I2CR[IIEN]. Software must first clear I2SR[IIF] in the interrupt routine. I2SR[ICF] is cleared either by reading from I2DR in receive mode or by writing to I2DR in transmit mode.

Software can service the I2C 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; that is, the address is sent. If master receive mode is required, I2CR[MTX] should be toggled.

During slave-mode address cycles (I2SR[IAAS] = 1), I2SR[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 25-14).

```
I2SR    LEA.L I2SR,-(A7)      ;Load effective address
       BCLR.B #1,(A7)+    ;Clear the IIF flag
       MOVE.B I2CR,-(A7)    ;Push the address on stack,
       BTST.B #5,(A7)+    ;check the MSTA flag
       BEQ.S SLAVE        ;Branch if slave mode
       MOVE.B I2CR,-(A7)    ;Push the address on stack
       BTST.B #4,(A7)+    ;check the mode flag
       BEQ.S RECEIVE      ;Branch if in receive mode
       MOVE.B I2SR,-(A7)    ;Push the address on stack,
       BTST.B #0,(A7)+    ;check ACK from receiver
       BNE.B END          ;If no ACK, end of transmission
TRANSMIT MOVE.B DATABUF,-(A7)    ;Stack data byte
       MOVE.B (A7)+, I2DR   ;Transmit next byte of data
```

25.6.4 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.

```
MASTX    MOVE.B I2SR, -(A7)      ;If no ACK, branch to end
       BTST.B #0,(A7)+
       BNE.B END
       MOVE.B TXCNT,D0      ;Get value from the transmitting counter
       BEQ.S END            ;If no more data, branch to end
       MOVE.B DATABUF,-(A7)  ;Transmit next byte of data
       MOVE.B (A7)+, I2DR
       MOVE.B TXCNT,D0      ;Decrease the TXCNT
       SUBQ.L #1,D0
       MOVE.B D0,TXCNT
       BRA.S EMASTX;Exit
END      LEA.L I2CR,-(A7)      ;Generate a STOP condition
       BCLR.B #5,(A7)+
       EMASTX RTE           ;Return from interrupt
```

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[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.

```
MASR    MOVE.B RXCNT,D0      ;Decrease RXCNT
       SUBQ.L #1,D0
       MOVE.B D0,RXCNT
       BEQ.S ENMASR         ;Last byte to be read
       MOVE.B RXCNT,D1      ;Check second-to-last byte to be read
       EXTB.L D1
       SUBI.L #1,D1;
       BNE.S NXMAR          ;Not last one or second last
       LAMAR BSET.B #3,I2CR  ;Disable ACK
       BRA NXMAR
ENMASR  BCLR.B #5,I2CR      ;Last one, generate STOP signal
NXMAR   MOVE.B I2DR,RXBUF    ;Read data and store RTE
```

25.6.5 Generation of Repeated START

After the data transfer, if the master still wants the bus, it can signal another START followed by another slave address without signaling a STOP, as in the following example.

```
RESTART MOVE.B I2CR,-(A7)           ;Repeat START (RESTART)
          BSET.B #2, (A7)
          MOVE.B (A7)+, I2CR
          MOVE.B CALLING,-(A7)      ;Transmit the calling address, D0=R/W-
          MOVE.B CALLING,-(A7)
          MOVE.B (A7)+, I2DR
```

25.6.6 Slave Mode

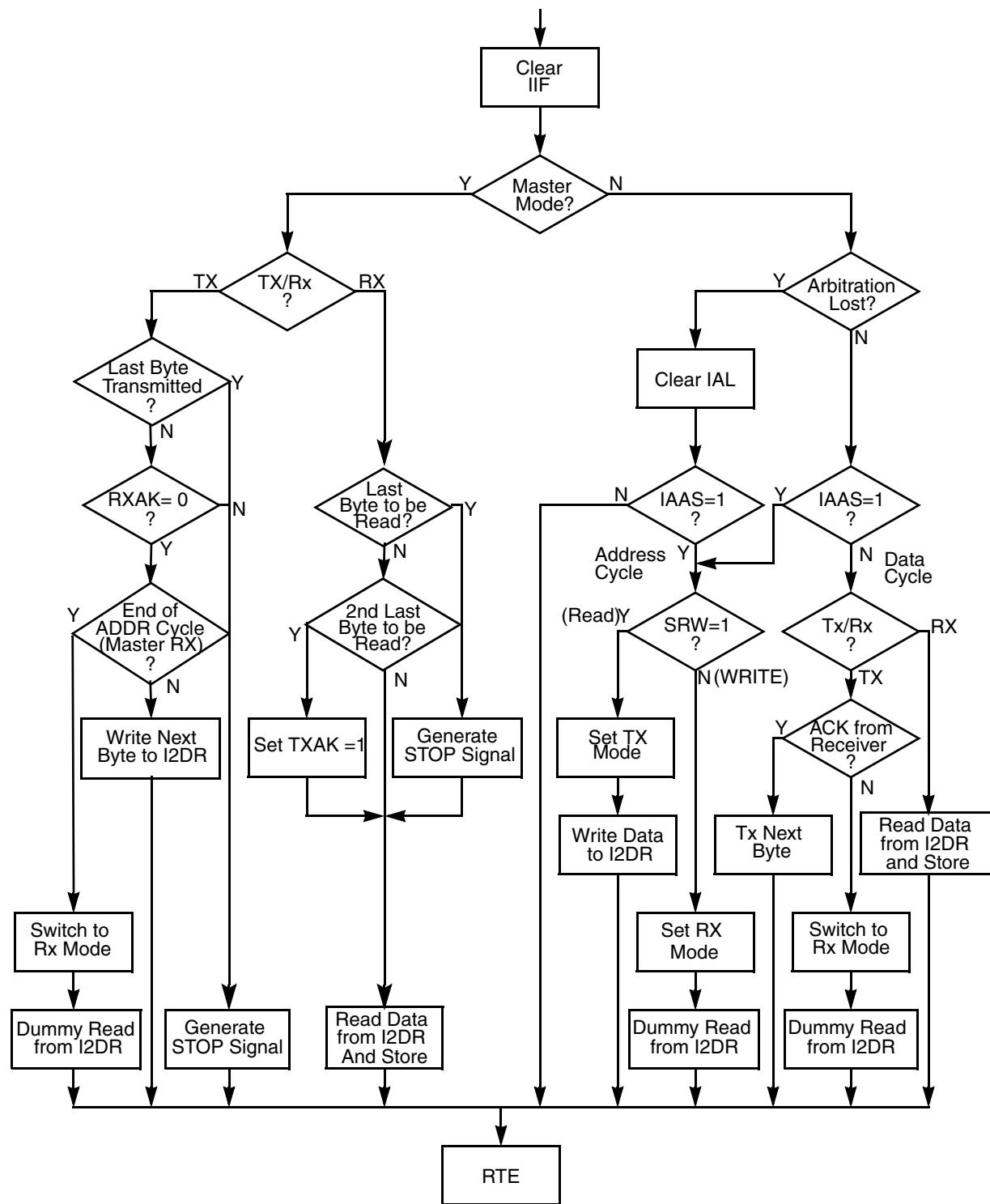
In the slave interrupt service routine, software should poll the I2SR[IAAS] bit to determine if the controller has received its slave address. If IAAS is set, software should set the transmit/receive mode select bit (I2CR[MTX]) according to the I2SR[SRW]. Writing to the I2CR clears the IAAS automatically. 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 will have IAAS cleared. A data transfer can now be initiated by writing information to I2DR for slave transmits, or read from I2DR in slave-receive mode. A dummy read of I2DR in slave/receive mode releases I2C_SCL, allowing the master to send data.

In the slave transmitter routine, I2SR[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 then releases I2C_SCL so that the master can generate a STOP signal.

25.6.7 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_SDA stops, but I2C_SCL is still generated 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[IAL] = 1 and I2CR[MSTA] = 0.

If a device that is not a master tries to transmit or execute a START, hardware will inhibit the transmission, clear MSTA without signaling a STOP, generate an interrupt to the CPU, and set IAL to indicate a failed attempt to engage the bus. When considering these cases, the slave service routine should first test IAL and software should clear it if it is set.

Figure 25-14. Flow-Chart of Typical I²C Interrupt Routine

Chapter 26

Debug Module

26.1 Introduction

This chapter describes the Revision B+ enhanced hardware debug module.

26.1.1 Overview

The debug module is shown in [Figure 26-1](#).

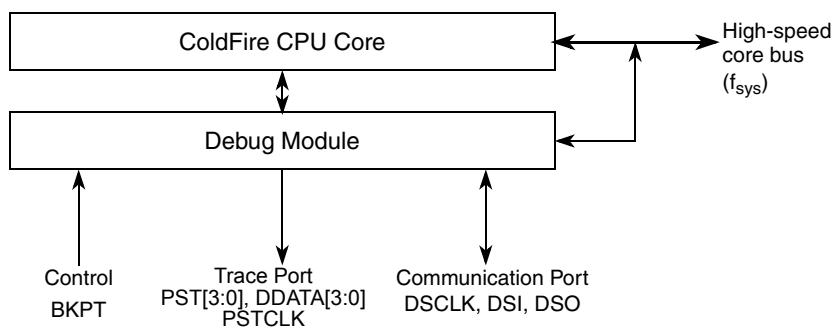


Figure 26-1. Processor/Debug Module Interface

NOTE

Some devices (MCF5207CABxxx and MCF5208CABxxx) contain only the ALLPST signal and do not have the PST[3:0] and DDATA[3:0] signals. ALLPST is a logical ‘AND’ of the PST[3:0] signals, and when asserted reflects that the core is halted.

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 26.3, “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 communication port. See [Section 26.5, “Background Debug Mode \(BDM\),”](#) and [Section 26.4, “Memory Map/Register Definition.”](#)
- Real-time debug support—BDM requires the processor to be halted, which many real-time embedded applications cannot do. External development systems can access memory because the

hardware supports concurrent operation of the processor and BDM-initiated commands. 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. See [Section 26.6, “Real-Time Debug Support.](#)

26.1.1.1 The New Debug Module Hardware (Rev. B+)

The revision B+ debug module features a small enhancement over revision B: the addition of three PC breakpoint registers (PCBR1–3). These new registers are mapped to DRc[3:0] addresses 0x18, 0x1A, and 0x1B. Additional PC breakpoints enable ROM/Flash software debugging. However, there are no masking registers associated with these new registers.

26.1.1.2 Enhancements over Revision A

The new debug hardware also contains the same enhancements that revision B has over revision A, while maintaining backwards compatibility with revision A. These enhancements are discussed below.

The revision B/B+ implementation has added registers that eliminate restrictions between BDM commands and the use of the hardware breakpoint logic. In some cases, the additional hardware is not program-visible; in other cases, there have been extensions to the debug module programming model.

The register containing the BDM memory address is not a program-visible resource. Rather, it is a register loaded automatically during the execution of a BDM command. In the Rev. B design, the execution of a BDM command does not affect the hardware breakpoint logic unless those registers are specifically accessed.

The other register added to the debug module programming model is the BDM address attribute register (BAAR). The BAAR is mapped to a DRc[3:0] address of 0x05. This 8-bit register is equivalent in format of the low-order byte of the AATR register (See [Section 26.4.3, “BDM Address Attribute \(BAAR\)”](#)). This register specifies the memory space attributes associated with all BDM memory-referencing commands.

Additionally, a bit was added to the CSR register (CSR[BKD]) that configures the debug module to assert or not assert an interrupt to the processor when the \overline{BKPT} signal is asserted. The level 1 and level 2 triggers are also configurable to trigger on either an AND or an OR condition. The revision A debug module only triggers on an AND condition.

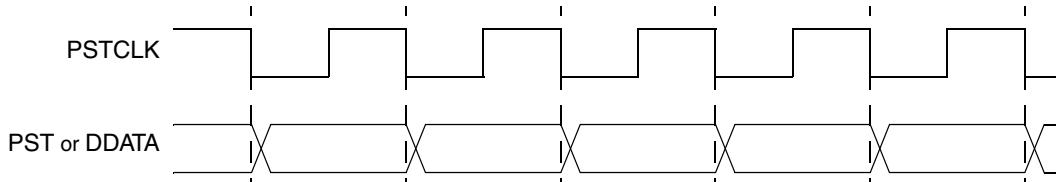
26.2 External Signal Description

[Table 26-1](#) describes debug module signals. All ColdFire debug signals are unidirectional and related to a rising edge of the processor’s clock signal. The standard 26-pin debug connector is shown in [Section 26.8, “Freescale-Recommended BDM Pinout.”](#)

Table 26-1. 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 PSTCLK edges.) Clocks the serial communication port to the debug module during packet transfers. Maximum frequency is 1/5 the processor status clock (PSTCLK) speed. 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.
Development Serial Output (DSO)	Provides serial output communication for debug module responses. DSO is registered internally.
Breakpoint ($\overline{\text{BKPT}}$)	Input used to request 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 signals (PST[3:0]) as the value 0xF. Also, in Rev B and B+ if the CSR[BKD] bit is set, the assertion of the $\overline{\text{BKPT}}$ signal will generate a debug interrupt exception to the processor.
PSTCLK	PSTCLK indicates when the development system should sample PST and DDATA values. See Figure 26-2 .
Debug Data (DDATA[3:0])	These output signals display the register breakpoint status as a default, or optionally, captured address and operand values. The capturing of data values is controlled by the setting of the CSR. Additionally, execution of the WDDATA instruction by the processor captures operands which are displayed on DDATA. These signals are updated each processor cycle.
Processor Status (PST[3:0])	These output signals report the processor status. Table 26-2 shows the encoding of these signals. These outputs indicate the current status of the processor pipeline and, as a result, are not related to the current bus transfer. The PST value is updated each processor cycle.
All Processor Status Outputs (ALLPST)	ALLPST is a logical ‘AND’ of the four PST signals. PST[3:0] and DDATA[3:0] are not available on the QFP devices (MCF5207CABxxx and MCF5208CABxxx). When asserted, reflects that the core is halted.

[Figure 26-2](#) shows PSTCLK timing with respect to PST and DDATA.

**Figure 26-2. PSTCLK Timing**

26.3 Real-Time Trace Support

Real-time trace, which defines the dynamic execution path, 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 port is partitioned into two 4-bit nibbles: one nibble allows the processor to transmit processor status, (PST), and the other allows operand data to be displayed (debug data, DDATA). The processor status may not be related to the current bus transfer.

External development systems can use PST 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). DDATA outputs can be configured to display the target address of such instructions in sequential nibble increments across multiple processor clock cycles, as described in [Section 26.3.1, “Begin Execution of Taken Branch \(PST = 0x5\).”](#) Two 32-bit storage elements form a FIFO buffer connecting the processor’s high-speed local bus to the external development system through PST[3:0] and DDATA[3:0]. The buffer captures branch target addresses and certain data values for eventual display on the DDATA port, one nibble at a time starting with the least significant bit (lsb).

Execution speeds affected only when both storage elements contain valid data to be dumped to the DDATA port. The core stalls until one FIFO entry is available.

[Table 26-2](#) shows the encoding of these signals.

Table 26-2. Processor Status Encoding

PST[3:0]		Definition
Hex	Binary	
0x0	0000	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 outputs with this encoding.
0x1	0001	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	0010	Reserved
0x3	0011	Entry into user-mode. Signaled after execution of the instruction that caused the ColdFire processor to enter user mode.
0x4	0100	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 DDATA port, independent of debug module configuration. When WDDATA is executed, a value of 0x4 is signaled on the PST port, followed by the appropriate marker, and then the data transfer on the DDATA port. Transfer length depends on the WDDATA operand size.
0x5	0101	Begin execution of taken branch. For some opcodes, a branch target address may be displayed on DDATA depending on the CSR settings. CSR also controls the number of address bytes displayed, indicated by the PST marker value preceding the DDATA nibble that begins the data output. See Section 26.3.1, “Begin Execution of Taken Branch (PST = 0x5).” Also indicates that the SYNC_PC command has been issued.
0x6	0110	Reserved
0x7	0111	Begin execution of return from exception (RTE) instruction.
0x8–0xB	1000–1011	Indicates the number of bytes to be displayed on the DDATA port on subsequent processor clock cycles. The value is driven onto the PST port one PSTCLK cycle before the data is displayed on DDATA. 0x8 Begin 1-byte transfer on DDATA. 0x9 Begin 2-byte transfer on DDATA. 0xA Begin 3-byte transfer on DDATA. 0xB Begin 4-byte transfer on DDATA.
0xC	1100	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, PST outputs are driven with 0xC until exception processing completes.

Table 26-2. Processor Status Encoding (continued)

PST[3:0]		Definition
Hex	Binary	
0xD	1101	Entry into emulator mode. Displayed during emulation mode (debug interrupt or optionally trace). Because this encoding defines a multiple-cycle mode, PST outputs are driven with 0xD until exception processing completes.
0xE	1110	Processor is stopped. Appears in multiple-cycle format when the processor executes a STOP instruction. The ColdFire processor remains stopped until an interrupt occurs, thus PST outputs display 0xE until the stopped mode is exited.
0xF	1111	Processor is halted. Because this encoding defines a multiple-cycle mode, the PST outputs display 0xF until the processor is restarted or reset. (see Section 26.5.1, “CPU Halt”)

26.3.1 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 DDATA depending on the CSR settings. CSR also controls the number of address bytes displayed, which is indicated by the PST marker value immediately preceding the DDATA nibble that begins the data output.

Bytes are displayed in least-to-most-significant order. The processor captures only those target addresses associated with taken branches which use a variant addressing mode; that is, 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 to output the following sequence of information on successive processor clock cycles:

1. Use PST (0x5) to identify that a taken branch was executed.
2. Using the PST pins, optionally signal the target address to be displayed sequentially on the DDATA pins. Encodings 0x9–0xB identify the number of bytes displayed.
3. The new target address is optionally available on subsequent cycles using the DDATA port. The number of bytes of the target address displayed on this port is configurable (2, 3, or 4 bytes).

Another example of a variant branch instruction would be a JMP (A0) instruction. [Figure 26-3](#) shows the PST and DDATA outputs that indicate a JMP (A0) execution (assuming the CSR was programmed to display the lower 2 bytes of an address).

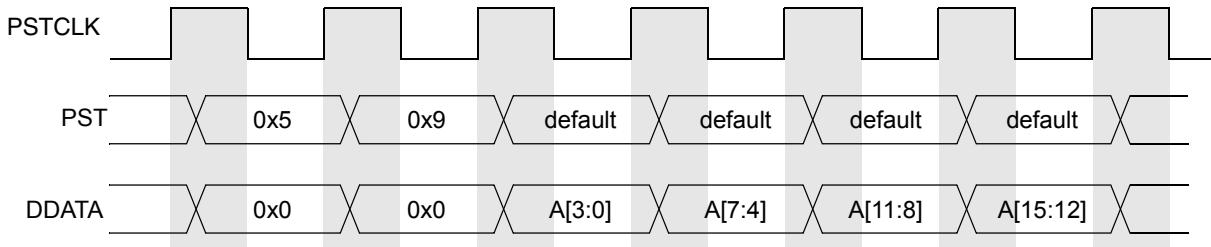


Figure 26-3. Example JMP Instruction Output on PST/DDATA

PST 0x5 indicates a taken branch and the marker value 0x9 indicates a 2-byte address. Thus, the subsequent 4 nibbles of DDATA display the lower 2 bytes of address register A0 in least-to-most-significant nibble order. The PST output after the JMP instruction completes depends on the target instruction. The PST can continue with the next instruction before the address has completely displayed on DDATA because of the DDATA FIFO. If the FIFO is full and the next instruction has captured values to display on DDATA, the pipeline stalls (PST = 0x0) until space is available in the FIFO.

26.4 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 contains 19 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). Thus, the breakpoint hardware in the debug module can be written by the external development system using the debug serial interface or by the operating system running on the processor core. Software is responsible for guaranteeing that accesses to these resources are serialized and logically consistent. Hardware provides a locking mechanism in the CSR to allow the 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, while DSCLK is quiescent.

These registers, shown in [Table 26-3](#), 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 26.5.3.3, “Command Set Descriptions.”](#) These commands contain a 5-bit field, DRc, that specifies the register, as shown in [Table 26-3](#).

Table 26-3. Debug Module Memory Map

DRc[4–0]	Register	Width (bits)	Access	Reset Value	Section/Page
0x00	Configuration/Status Register (CSR)	32	See Note	0x0090_0000	26.4.2/26-7
0x05	BDM Address Attribute Register (BAAR)	32 ¹	See Note	0x05	26.4.3/26-10
0x06	Address Attribute Trigger Register (AATR)	32 ¹	See Note	0x0005	26.4.4/26-10
0x07	Trigger Definition Register (TDR)	32	See Note	0x0000_0000	26.4.5/26-12
0x08	PC Breakpoint Register 0 (PBR0)	32	See Note	Undefined	26.4.6/26-14
0x09	PC Breakpoint Mask Register (PBMR)	32	See Note	Undefined	26.4.6/26-14

Table 26-3. Debug Module Memory Map (continued)

DRc[4–0]	Register	Width (bits)	Access	Reset Value	Section/Page
0x0C	Address High Breakpoint Register (ABHR)	32	See Note	Undefined	26.4.7/26-15
0x0D	Address Low Breakpoint Register (ABLR)	32	See Note	Undefined	26.4.7/26-15
0x0E	Data Breakpoint Register (DBR)	32	See Note	Undefined	26.4.8/26-16
0x0F	Data Breakpoint Mask Register (DBMR)	32	See Note	Undefined	26.4.8/26-16
0x18	PC Breakpoint Register 1 (PBR1)	32	See Note	Undefined	26.4.6/26-14
0x1A	PC Breakpoint Register 2 (PBR2)	32	See Note	Undefined	26.4.6/26-14
0x1B	PC Breakpoint Register 3 (PBR3)	32	See Note	Undefined	26.4.6/26-14

¹ Each debug register is accessed as a 32-bit register; reserved fields are not used (don't care).

NOTE

Debug control registers can be written by the external development system or the CPU through the WDEBUG instruction. CSR is write-only from the programming model. It can be read or written through the BDM port using the RDMREG and WDMREG commands.

26.4.1 Shared Debug Resources

The debug module implementation provides a common hardware structure for both BDM and breakpoint functionality. Certain hardware structures are used for both BDM and breakpoint purposes as shown in [Table 26-4](#).

Table 26-4. 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

Thus, 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.

26.4.2 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.

Debug Module

DRc[4:0]: 0x00 (CSR)												Access: Supervisor write-only					
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
W	BSTAT				FOF	TRG	HALT	BKPT	HRL					0	BKD	PCD	IPW
Reset	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	
R	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
W	MAP	TRC	EMU	DDC		UHE	BTB		0	NPL	IPI	SSM	0	0	0	0	
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Figure 26-4. Configuration/Status Register (CSR)

Table 26-5. CSR Field Descriptions

Field	Description
31–28 BSTAT	Breakpoint status. Provides read-only status information concerning hardware breakpoints. BSTAT is cleared by a TDR write or by a CSR read when either 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
27 FOF	Fault-on-fault. If FOF is set, a catastrophic halt occurred and forced entry into BDM. FOF is cleared whenever CSR is read.
26 TRG	Hardware breakpoint trigger. If TRG is set, a hardware breakpoint halted the processor core and forced entry into BDM. Reset, the debug GO command, or reading CSR will clear TRG.
25 HALT	Processor halt. If HALT is set, the processor executed a HALT and forced entry into BDM. Reset, the debug GO command, or reading CSR will clear HALT.
24 BKPT	Breakpoint assert. If BKPT is set, BKPT was asserted, forcing the processor into BDM. Reset, the debug GO command, or reading CSR will clear BKPT.
23–20 HRL	Hardware revision level. Indicates the level of debug module functionality. An emulator could use this information to identify the level of functionality supported. 1001 Revision B+ (This is the only valid value for this processor)
19	Reserved, should be cleared.
18 BKD	Breakpoint disable. Disables the normal BKPT input signal functionality, and allows the assertion of this pin to generate a debug interrupt. 0 Normal operation 1 BKPT is edge-sensitive; a high to low transition on BKPT generates 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 PSTDATA output signals, and forces these signals to remain quiescent

Table 26-5. CSR Field Descriptions (continued)

Field	Description
16 IPW	Inhibit processor writes. Setting IPW inhibits processor-initiated writes to the debug module's programming model registers. IPW can be modified only by commands from the external development system.
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 = 10, TM = 101 or 110.
14 TRC	Force emulation mode on trace exception. If TRC = 1, the processor enters emulator mode when a trace exception occurs. If TRC=0, the processor enters supervisor mode.
13 EMU	Force emulation mode. If EMU = 1, the processor begins executing in emulator mode. See Section 26.6.1.1, "Emulator Mode."
12–11 DDC	Debug data control. Controls operand data capture for DDATA, 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 cycles). See Table 26-2 . 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 DDATA 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 Section 26.3.1, "Begin Execution of Taken Branch (PST = 0x5)" .
7	Reserved, should be cleared.
6 NPL	Non-pipelined mode. Determines whether the core operates in pipelined or mode or not. 0 Pipelined mode 1 Nonpipelined 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. 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, the 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.
5 IPI	Ignore pending interrupts. 1 Core ignores any pending interrupt requests signalled while in single-instruction-step mode. 0 Core services any pending interrupt requests that were signalled while in single-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.
3–0	Reserved, should be cleared.

26.4.3 BDM Address Attribute (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 the Rev. A implementation, this register 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.

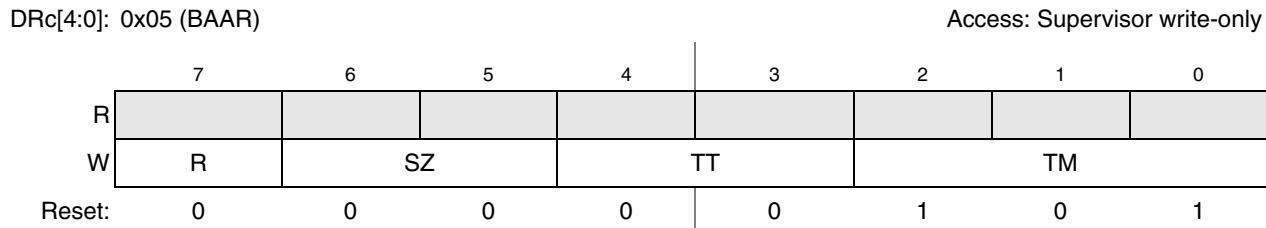


Figure 26-5. BDM Address Attribute Register (BAAR)

Table 26-6. BAAR Field Description

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, Section 26.4.4, “Address Attribute Trigger Register (AATR).”
2–0 TM	Transfer modifier. See the TM definition in the AATR description, Section 26.4.4, “Address Attribute Trigger Register (AATR).”

26.4.4 Address Attribute Trigger Register (AATR)

The AATR defines 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). AATR is accessible in supervisor mode as debug control register 0x06 using the WDEBUG instruction and through the BDM port using the WDMREG command.

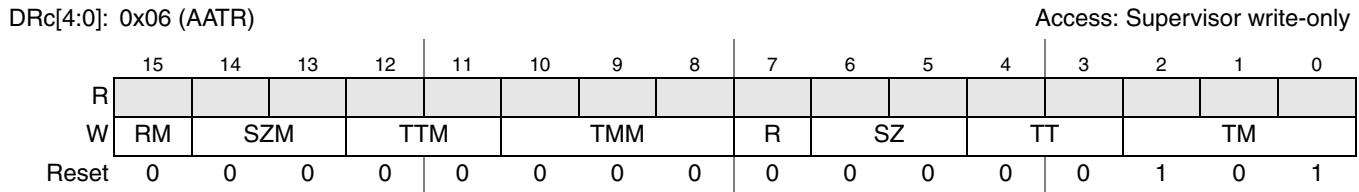


Figure 26-6. Address Attribute Trigger Register (AATR)

Table 26-7. AATR Field Descriptions

Field	Description																																							
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																																							
4–3 TT	Transfer type. Compared with the local bus transfer type signals. 00 Normal processor access 01 Reserved 10 Emulator mode access 11 Acknowledge/CPU space access These bits also define the TT encoding for BDM memory commands. In this case, the 01 encoding generates an external master or DMA access (for backward compatibility). These bits are used to decode the TM bits.																																							
2–0 TM	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).																																							
	<table border="1"> <thead> <tr> <th>TM</th> <th>TT=00 (normal mode)</th> <th>TT=10 (emulator mode)</th> <th>TT=11 (acknowledge/CPU space transfers)</th> </tr> </thead> <tbody> <tr> <td>000</td> <td>Explicit cache line push</td> <td>Reserved</td> <td>CPU space access</td> </tr> <tr> <td>001</td> <td>User data access</td> <td>Reserved</td> <td>Interrupt ack level 1</td> </tr> <tr> <td>010</td> <td>User code access</td> <td>Reserved</td> <td>Interrupt ack level 2</td> </tr> <tr> <td>011</td> <td>Reserved</td> <td>Reserved</td> <td>Interrupt ack level 3</td> </tr> <tr> <td>100</td> <td>Reserved</td> <td>Reserved</td> <td>Interrupt ack level 4</td> </tr> <tr> <td>101</td> <td>Supervisor data access</td> <td>Emulator mode access</td> <td>Interrupt ack level 5</td> </tr> <tr> <td>110</td> <td>Supervisor code access</td> <td>Emulator code Access</td> <td>Interrupt ack level 6</td> </tr> <tr> <td>111</td> <td>Reserved</td> <td>Reserved</td> <td>Interrupt ack level 7</td> </tr> </tbody> </table>				TM	TT=00 (normal mode)	TT=10 (emulator mode)	TT=11 (acknowledge/CPU space transfers)	000	Explicit cache line push	Reserved	CPU space access	001	User data access	Reserved	Interrupt ack level 1	010	User code access	Reserved	Interrupt ack level 2	011	Reserved	Reserved	Interrupt ack level 3	100	Reserved	Reserved	Interrupt ack level 4	101	Supervisor data access	Emulator mode access	Interrupt ack level 5	110	Supervisor code access	Emulator code Access	Interrupt ack level 6	111	Reserved	Reserved	Interrupt ack level 7
TM	TT=00 (normal mode)	TT=10 (emulator mode)	TT=11 (acknowledge/CPU space transfers)																																					
000	Explicit cache line push	Reserved	CPU space access																																					
001	User data access	Reserved	Interrupt ack level 1																																					
010	User code access	Reserved	Interrupt ack level 2																																					
011	Reserved	Reserved	Interrupt ack level 3																																					
100	Reserved	Reserved	Interrupt ack level 4																																					
101	Supervisor data access	Emulator mode access	Interrupt ack level 5																																					
110	Supervisor code access	Emulator code Access	Interrupt ack level 6																																					
111	Reserved	Reserved	Interrupt ack level 7																																					

26.4.5 Trigger Definition Register (TDR)

The TDR configures the operation of the hardware breakpoint logic that corresponds with the ABHR/ABLR/AATR, PBR/PBMR, and DBR/DBMR registers within the debug module. The TDR controls the actions taken under the defined conditions. Breakpoint logic may be configured as a one- or two-level trigger. TDR[31–16] define the second-level trigger and bits 15–0 define the first-level trigger.

NOTE

The debug module has no hardware interlocks, so to prevent spurious breakpoint triggers while the breakpoint registers are being loaded, disable TDR (by clearing TDR[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

Second Level Trigger																
R	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
W	TRC	EBL	EDLW	EDWL	EDWU	EDLL	EDLM	EDUM	EDUU	DI	EAI	EAR	EAL	EPC	PCI	
Reset	0	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	L2T	L1T	EBL	EDLW	EDWL	EDWU	EDLL	EDLM	EDUM	EDUU	DI	EAI	EAR	EAL	EPC	PCI
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 26-7. Trigger Definition Register (TDR)

Table 26-8. 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 DDATA. 00 Display on DDATA only 01 Processor halt 10 Debug interrupt 11 Reserved
15 L2T	Level-2 trigger. Determines the logic operation for the trigger between the PC_condition and the (Address_range & Data_condition) where the inclusion of a Data_condition is optional. The ColdFire debug architecture supports the creation of single or double-level triggers. 0 Level-2 trigger = PC_condition & Address_range & Data_condition 1 Level-2 trigger = PC_condition (Address_range & Data_condition) Note: Debug Rev A only had the 'AND' condition available for the triggers.

Table 26-8. TDR Field Descriptions (continued)

Field	Description																									
14 L1T	<p>Level-1 trigger. Determines the logic operation for the trigger between the PC_condition and the (Address_range & Data_condition) where the inclusion of a Data_condition is optional. The ColdFire debug architecture supports the creation of single or double-level triggers.</p> <p>0 Level-1 trigger = PC_condition & Address_range & Data_condition 1 Level-1 trigger = PC_condition (Address_range & Data_condition)</p> <p>Note: Debug Rev A only had the 'AND' condition available for the triggers.</p>																									
29 & 13 EBL	Enable breakpoint. Global enable for the breakpoint trigger. Setting TDR[EBL] enables a breakpoint trigger. Clearing it disables all breakpoints at that level.																									
28–22 & 12–6 EDx	<p>Setting an EDx bit enables the corresponding data breakpoint condition based on the size and placement on the processor's local data bus. Clearing all EDx bits disables data breakpoints.</p> <table border="1"> <thead> <tr> <th>Bits</th> <th>Field</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>28 & 12</td> <td>EDLW</td> <td>Data longword. Entire processor's local data bus.</td> </tr> <tr> <td>27 & 11</td> <td>EDWL</td> <td>Lower data word.</td> </tr> <tr> <td>26 & 10</td> <td>EDWU</td> <td>Upper data word.</td> </tr> <tr> <td>25 & 9</td> <td>EDLL</td> <td>Lower lower data byte. Low-order byte of the low-order word.</td> </tr> <tr> <td>24 & 8</td> <td>EDLM</td> <td>Lower middle data byte. High-order byte of the low-order word.</td> </tr> <tr> <td>23 & 7</td> <td>EDUM</td> <td>Upper middle data byte. Low-order byte of the high-order word.</td> </tr> <tr> <td>22 & 6</td> <td>EDUU</td> <td>Upper upper data byte. High-order byte of the high-order word.</td> </tr> </tbody> </table>		Bits	Field	Description	28 & 12	EDLW	Data longword. Entire processor's local data bus.	27 & 11	EDWL	Lower data word.	26 & 10	EDWU	Upper data word.	25 & 9	EDLL	Lower lower data byte. Low-order byte of the low-order word.	24 & 8	EDLM	Lower middle data byte. High-order byte of the low-order word.	23 & 7	EDUM	Upper middle data byte. Low-order byte of the high-order word.	22 & 6	EDUU	Upper upper data byte. High-order byte of the high-order word.
Bits	Field	Description																								
28 & 12	EDLW	Data longword. Entire processor's local data bus.																								
27 & 11	EDWL	Lower data word.																								
26 & 10	EDWU	Upper data word.																								
25 & 9	EDLL	Lower lower data byte. Low-order byte of the low-order word.																								
24 & 8	EDLM	Lower middle data byte. High-order byte of the low-order word.																								
23 & 7	EDUM	Upper middle data byte. Low-order byte of the high-order word.																								
22 & 6	EDUU	Upper upper data byte. High-order byte of the high-order word.																								
21 & 5 DI	Data breakpoint invert. Provides a way to invert 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.																									
20–18 & 4–2 EAx	<p>Enable address bits. Setting an EA bit enables the corresponding address breakpoint. Clearing all three bits disables the breakpoint.</p> <table border="1"> <thead> <tr> <th>Bits</th> <th>Field</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>20 & 4</td> <td>EAI</td> <td>Enable address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.</td> </tr> <tr> <td>19 & 3</td> <td>EAR</td> <td>Enable address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.</td> </tr> <tr> <td>18 & 2</td> <td>EAL</td> <td>Enable address breakpoint low. The breakpoint is based on the address in the ABLR.</td> </tr> </tbody> </table>		Bits	Field	Description	20 & 4	EAI	Enable address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.	19 & 3	EAR	Enable address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.	18 & 2	EAL	Enable address breakpoint low. The breakpoint is based on the address in the ABLR.												
Bits	Field	Description																								
20 & 4	EAI	Enable address breakpoint inverted. Breakpoint is based outside the range between ABLR and ABHR.																								
19 & 3	EAR	Enable address breakpoint range. The breakpoint is based on the inclusive range defined by ABLR and ABHR.																								
18 & 2	EAL	Enable address breakpoint low. The breakpoint is based on the address in the ABLR.																								
17 & 1 EPC	<p>Enable PC breakpoint.</p> <p>0 Disable PC breakpoint 1 Enable PC breakpoint</p>																									
16 & 0 PCI	Breakpoint invert. If set, this bit allows execution outside a given region as defined by PBR and PBMR to enable a trigger. If cleared, the PC breakpoint is defined within the region defined by PBR and PBMR.																									

26.4.6 Program Counter Breakpoint/Mask Registers (PBR0–3, PBMR)

The PBR n registers define an instruction address for use as part of the trigger. This register's contents are compared with the processor's program counter register when TDR is 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. [Figure 26-8](#) shows the PC breakpoint register. The 3 new PC breakpoint registers, PBR1–3, have no masking associated with them. These registers contents are compared with the processor's program counter register when TDR is configured appropriately.

The PC breakpoint register is accessible in supervisor mode using the WDEBUG instruction and through the BDM port using the RDMREG and WDMREG commands using values shown in [Section 26.5.3.3, “Command Set Descriptions.”](#)

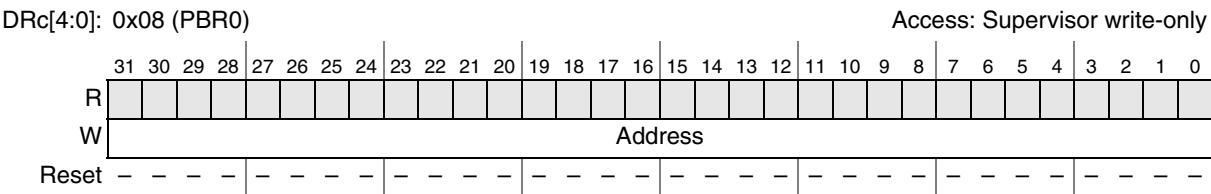


Figure 26-8. Program Counter Breakpoint Register 0 (PBR0)

Table 26-9. PBR0 Field Descriptions

Field	Description
31–1 Address	PC breakpoint address. The address to be compared with the PC as a breakpoint trigger.

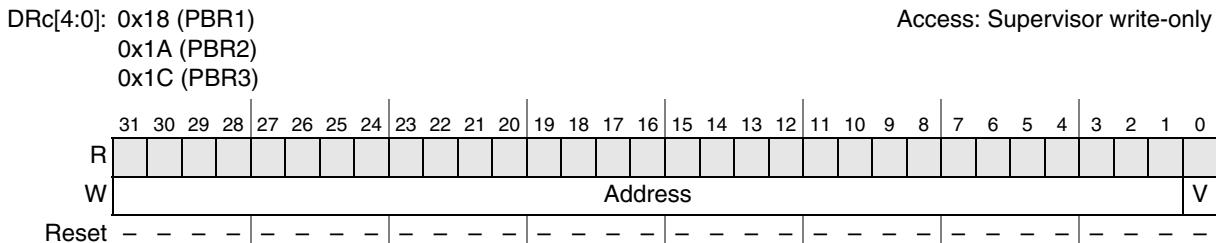


Figure 26-9. Program Counter Breakpoint Register n (PBR n)

Table 26-10. PBR n Field Descriptions

Field	Description
31–1 Address	PC breakpoint address. The 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 26-8](#) shows PBMR. PBMR is accessible in supervisor mode as debug control register 0x09 using the WDEBUG instruction and via the BDM port using the WDMREG command.

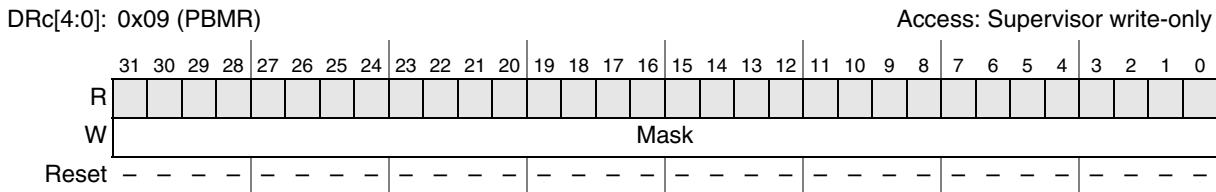


Figure 26-10. Program Counter Breakpoint Mask Register (PBMR)

Table 26-11. PBMR Field Descriptions

Field	Description
31–0 Mask	PC breakpoint mask. A zero in a bit position causes the corresponding PBR0 bit to be compared to the appropriate PC bit. Setting a PBMR bit causes the corresponding PBR0 bit to be ignored.

26.4.7 Address Breakpoint Registers (ABLR, ABHR)

The ABLR and ABHR, shown in [Figure 26-11](#), define regions in the processor's data address space that can be used 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:

1. Identical to the value in ABLR
2. Inside the range bound by ABLR and ABHR inclusive
3. Outside that same range

ABHR is accessible in supervisor mode as debug control register 0x0C using the WDEBUG instruction and via the BDM port using the RDMREG and WDMREG commands. ABLR is accessible in supervisor mode as debug control register 0x0D using the WDEBUG instruction and via the BDM port using the WDMREG command.

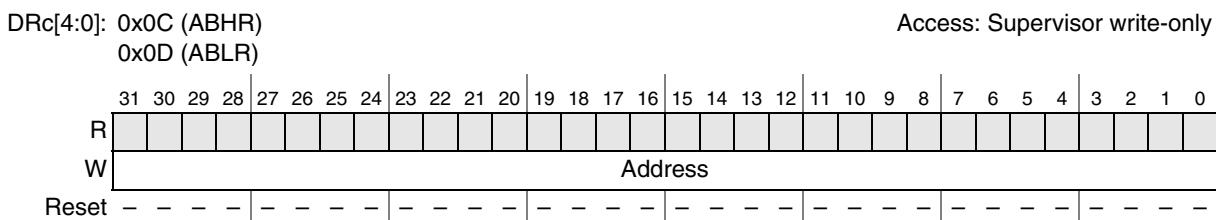


Figure 26-11. Address Breakpoint Registers (ABLR, ABHR)

Table 26-12. ABLR Field Description

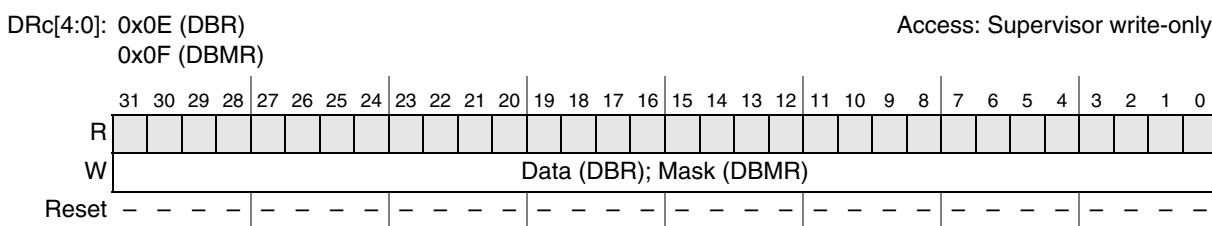
Field	Description
31–0 Address	Low address. Holds the 32-bit address marking the lower bound of the address breakpoint range. Breakpoints for specific addresses are programmed into ABLR.

Table 26-13. ABHR Field Description

Field	Description
31–0 Address	High address. Holds the 32-bit address marking the upper bound of the address breakpoint range.

26.4.8 Data Breakpoint/Mask Registers (DBR, DBMR)

The DBR specifies data patterns used as part of the trigger into debug mode. DBR bits are masked by setting the corresponding DBMR bits, as defined in TDR. DBR is accessible in supervisor mode as debug control register 0x0E, using the WDEBUG instruction and through the BDM port using the RDMREG and WDMREG commands. DBMR is accessible in supervisor mode as debug control register 0x0F, using the WDEBUG instruction and via the BDM port using the WDMREG command.

**Figure 26-12. Data Breakpoint & Mask Registers (DBR & DBMR)****Table 26-14. DBR 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.

Table 26-15. DBMR Field Descriptions

Field	Description
31–0 Mask	Data breakpoint mask. The 32-bit mask for the data breakpoint trigger. Clearing a DBR 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 DBR supports both aligned and misaligned references. Table 26-16 shows relationships between processor address, access size, and location within the 32-bit data bus.

Table 26-16. Access Size and Operand Data Location

A[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]

26.5 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 handled 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.

26.5.1 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 BKPT can be configured to generate a pending halt condition similar to the assertion of BKPT. This type of halt is always first made pending in the processor. Next, the processor 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 26.6.1, “Theory of Operation.”](#)
3. The execution of a HALT instruction immediately suspends execution. Attempting to execute HALT in user mode while CSR[UHE] = 0 generates a privilege violation exception. If CSR[UHE] = 1, 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 BKPT input is treated as a pseudo-interrupt; that is, the halt condition is postponed until the processor core samples for halts/interrupts. The processor samples for these conditions once during the execution of each instruction. If there is a pending halt condition at the sample time, the processor suspends execution and enters the halted state.

The assertion of BKPT should be considered in the following two special cases:

- After the system reset signal is negated, the processor waits for 16 processor clock cycles before beginning reset exception processing. If the BKPT input is asserted within eight cycles after RSTI

is negated, the processor enters the halt state, signaling halt status (0xF) on the PST 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 handles a special case of BKPT being 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, that is, the instruction following the STOP opcode.

The CSR[27–24] bits indicate the halt source, showing the highest priority source for multiple halt conditions.

26.5.2 BDM Serial Interface

When the CPU is halted and PST 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 26-1](#). 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 both 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 26-13](#), all state transitions are enabled on a rising edge of PSTCLK when DSCLK is high; that is, DSI is sampled and DSO is driven.

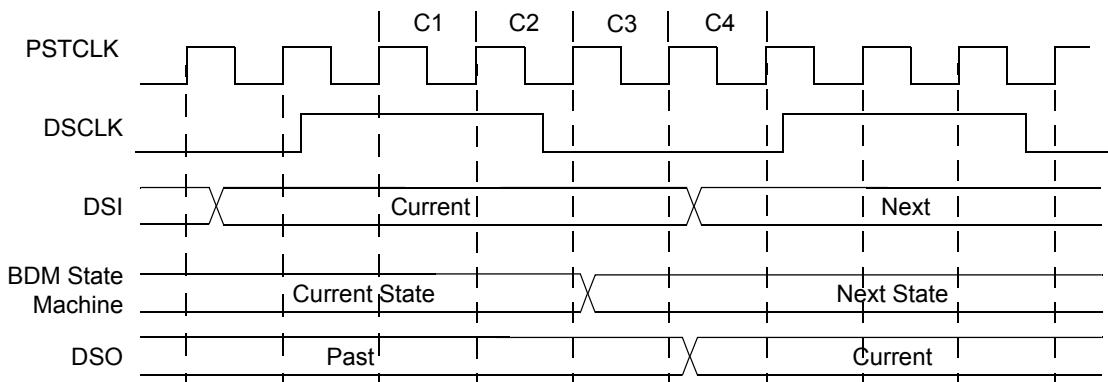


Figure 26-13. BDM Serial Interface Timing

DSCLK and DSI are synchronized inputs. DSCLK acts as a pseudo clock enable and is sampled on the rising edge of the processor clock as well as the DSI. DSO is delayed from the DSCLK-enabled CLK rising edge (registered after a BDM state machine state change). All events in the debug module's serial state

machine are based on the processor clock rising edge. DSCLK must also be sampled low (on a positive edge of CLK) between each bit exchange. The msb is transferred 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. C1–C4 are described as follows:

- C1—First synchronization cycle for DS_I (DSCLK is high).
- C2—Second synchronization cycle for DS_I (DSCLK is high).
- C3—BDM state machine changes state depending upon DS_I 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.

26.5.2.1 Receive Packet Format

The basic receive packet, [Figure 26-14](#), consists of 16 data bits and 1 status bit.

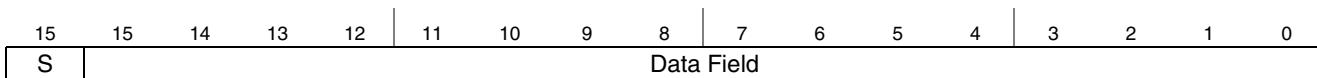


Figure 26-14. Receive BDM Packet

Table 26-17. Receive BDM Packet Field Description

Field	Description																				
16 S	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.																				
	<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.																				

26.5.2.2 Transmit Packet Format

The basic transmit packet, [Figure 26-15](#), consists of 16 data bits and 1 reserve bit.

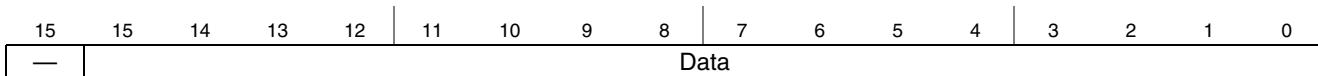


Figure 26-15. Transmit BDM Packet

Table 26-18. Transmit BDM Packet Field Description

Field	Description
16	Reserved, should be cleared.
15–0 Data	Data bits 15–0. Contains the data to be sent from the development system to the debug module.

26.5.3 BDM Command Set

Table 26-19 summarizes the BDM command set. Subsequent paragraphs 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.

Table 26-19. BDM Command Summary

Command	Mnemonic	Description	CPU State ¹	Section	Command (Hex)
Read A/D register	RAREG/ RDREG	Read the selected address or data register and return the results through the serial interface.	Halted	26.5.3.3.1	0x218 {A/D, Reg[2:0]}
Write A/D register	WAREG/ WDREG	Write the data operand to the specified address or data register.	Halted	26.5.3.3.2	0x208 {A/D, Reg[2:0]}
Read memory location	READ	Read the data at the memory location specified by the longword address.	Steal	26.5.3.3.3	0x1900—byte 0x1940—word 0x1980—lword ²
Write memory location	WRITE	Write the operand data to the memory location specified by the longword address.	Steal	26.5.3.3.4	0x1800—byte 0x1840—word 0x1880—lword ²
Dump memory block	DUMP	Used with READ to dump large blocks of memory. An initial READ is executed to set up the starting address of the block and to retrieve the first result. A DUMP command retrieves subsequent operands.	Steal	26.5.3.3.5	0x1D00—byte 0x1D40—word 0x1D80—lword ²
Fill memory block	FILL	Used with WRITE to fill large blocks of memory. An initial WRITE is executed to set up the starting address of the block and to supply the first operand. A FILL command writes subsequent operands.	Steal	26.5.3.3.6	0x1C00—byte 0x1C40—word 0x1C80—lword ²
Resume execution	GO	The pipeline is flushed and refilled before resuming instruction execution at the current PC.	Halted	26.5.3.3.7	0x0C00
No operation	NOP	Perform no operation; may be used as a null command.	Parallel	26.5.3.3.8	0x0000

Table 26-19. BDM Command Summary (continued)

Command	Mnemonic	Description	CPU State ¹	Section	Command (Hex)
Synchronize PC to PST/DDATA	SYNC_PC	Capture the current PC and display it on the PST/DDATA outputs	Parallel	26.5.3.3.9	0x0001
Read control register	RCREG	Read the system control register.	Halted	26.5.3.3.10	0x2980
Write control register	WCREG	Write the operand data to the system control register.	Halted	26.5.3.3.11	0x2880
Read debug module register	RDMREG	Read the debug module register.	Parallel	26.5.3.3.12	0x2D {0x4 ³ DRc[4:0]}
Write debug module register	WDMREG	Write the operand data to the debug module register.	Parallel	26.5.3.3.13	0x2C {0x4 ³ DRc[4:0]}

¹ 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.

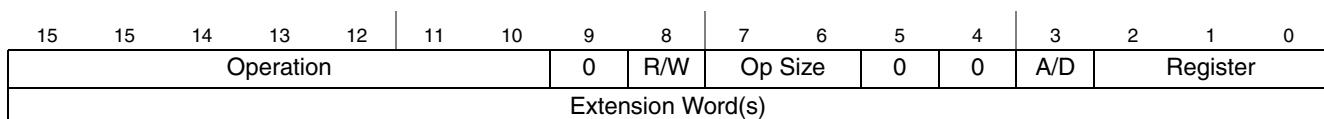
² Iword = longword

³ 0x4 is a three-bit field.

Unassigned command opcodes are reserved by Freescale. All unused command formats within any revision level perform a NOP and return the illegal command response.

26.5.3.1 ColdFire 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, as shown in [Figure 26-16](#).

**Figure 26-16. BDM Command Format****Table 26-20. BDM Field Descriptions**

Field	Description
15–10 Operation	Specifies the command. These values are listed in Table 26-19 .
9	Reserved, should 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.

Table 26-20. BDM Field Descriptions (continued)

Field	Description		
7–6 Op Size	Operand data size for sized operations. Addresses are expressed as 32-bit absolute values. Note that 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.		
	Operand Size	Bit Values	
	00	Byte	8 bits
	01	Word	16 bits
	10	Longword	32 bits
	11	Reserved	—
5–4	Reserved, should be cleared.		
3 A/D	Address/data. Determines whether the register field specifies a data or address register. 0 Indicates a data register. 1 Indicates an address register.		
2–0 Register	Contains the register number in commands that operate on processor registers.		

26.5.3.1.1 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 and 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.

26.5.3.2 Command Sequence Diagrams

The command sequence diagram in [Figure 26-17](#) 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.

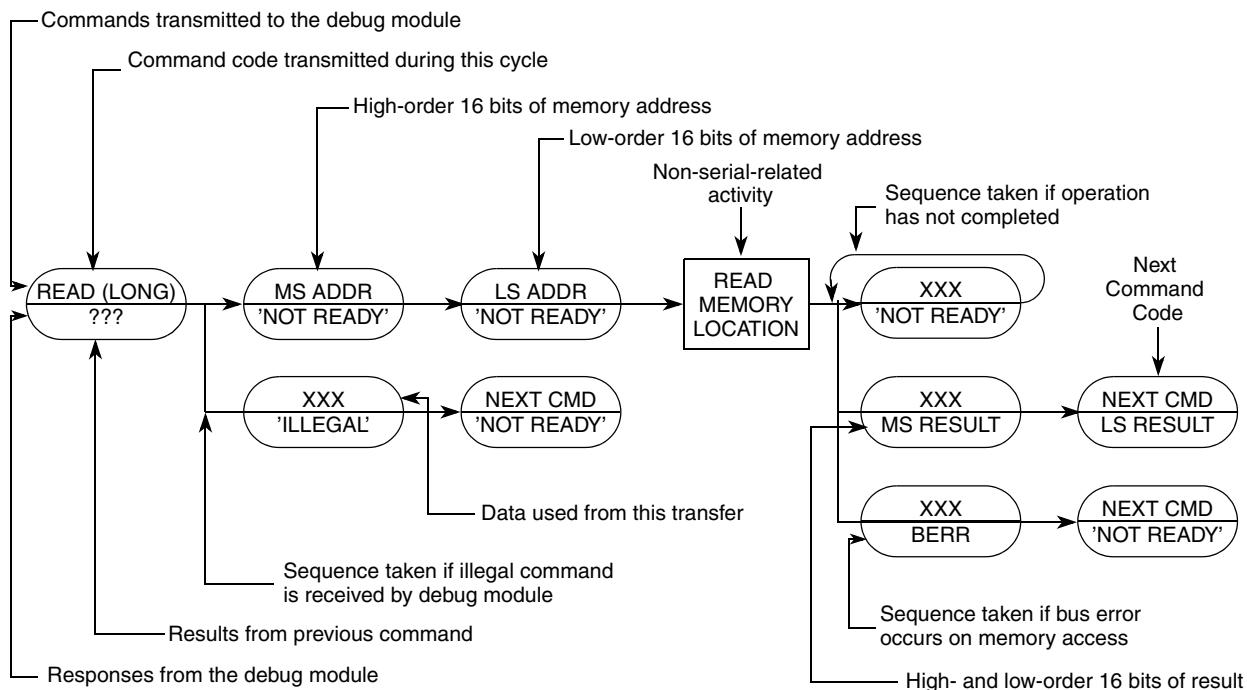


Figure 26-17. Command Sequence Diagram

The sequence is as follows:

- In cycle 1, the development system command is issued (READ in this example). The debug module responds with either 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 memory or register access is terminated with a bus error, the error status (S = 1, DATA = 0x0001) is returned instead of result data.

26.5.3.3 Command Set Descriptions

The following sections describe the commands summarized in [Table 26-19](#).

NOTE

The BDM status bit (S) is 0 for normally completed commands; S = 1 for illegal commands, not-ready responses, and transfers with bus-errors.

[Section 26.5.2, “BDM Serial Interface,”](#) describes the receive packet format.

Freescale reserves unassigned command opcodes for future expansion. Unused command formats in any revision level perform a NOP and return an illegal command response.

26.5.3.3.1 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:

Command	15 14 13 12 11 10 9 8	0x2	0x1	7 6 5 4	0x8	A/D	Register
Result				D[31:16]			

D[15:0]

Figure 26-18. RAREG/RDREG Command Format

Command Sequence:

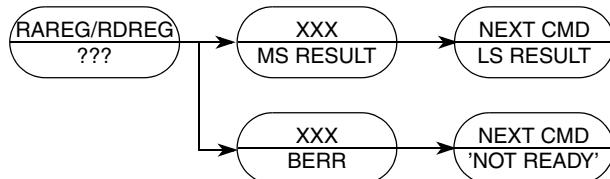


Figure 26-19. RAREG/RDREG Command Sequence

Operand Data: None

Result Data: The contents of the selected register are returned as a longword value, most-significant word first.

26.5.3.3.2 Write A/D Register (WAREG/WDREG)

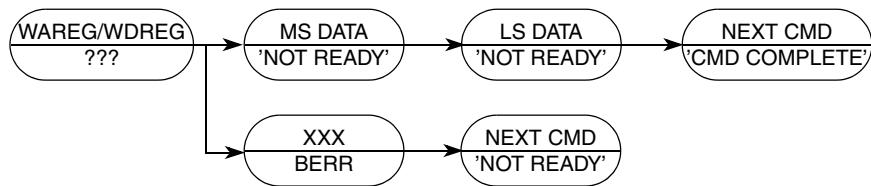
The operand longword data is written to the specified address or data register. A write alters all 32 register bits. A bus error response is returned if the CPU core is not halted.

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 26-20. WAREG/WDREG Command Format

Command Sequence:

**Figure 26-21. 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.

26.5.3.3.3 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 zeros 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]															
	Result	X	X	X	X	X	X	X	X	D[7:0]															
Word	Command	0x1				0x9				0x4				0x0											
		A[31:16]								A[15:0]															
	Result	D[15:0]																							
Longword	Command	0x1				0x9				0x8				0x0											
		A[31:16]								A[15:0]															
	Result	D[31:16]								D[15:0]															

Figure 26-22. READ Command/Result Formats

Command Sequence:

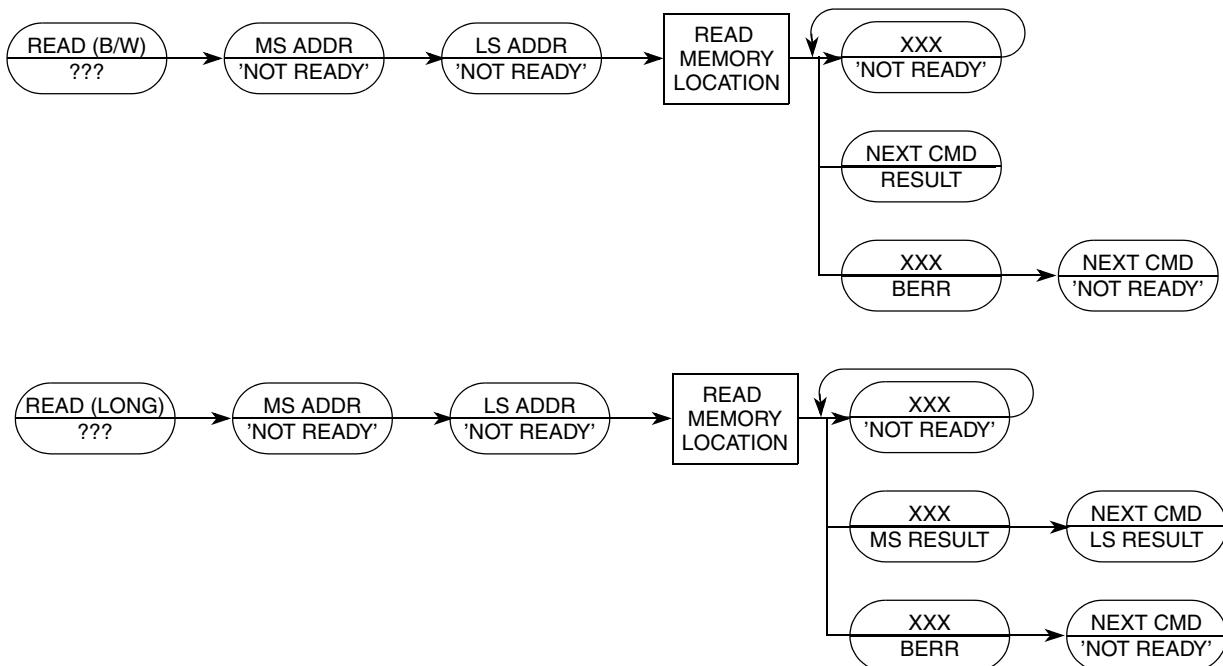


Figure 26-23. READ Command Sequence

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.

26.5.3.3.4 Write Memory Location (WRITE)

Write data to the memory location specified by the longword address. The address space is defined by BAAR[TT,TM]. Hardware forces low-order address bits to zeros 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 26-24. WRITE Command Format

Command Sequence:

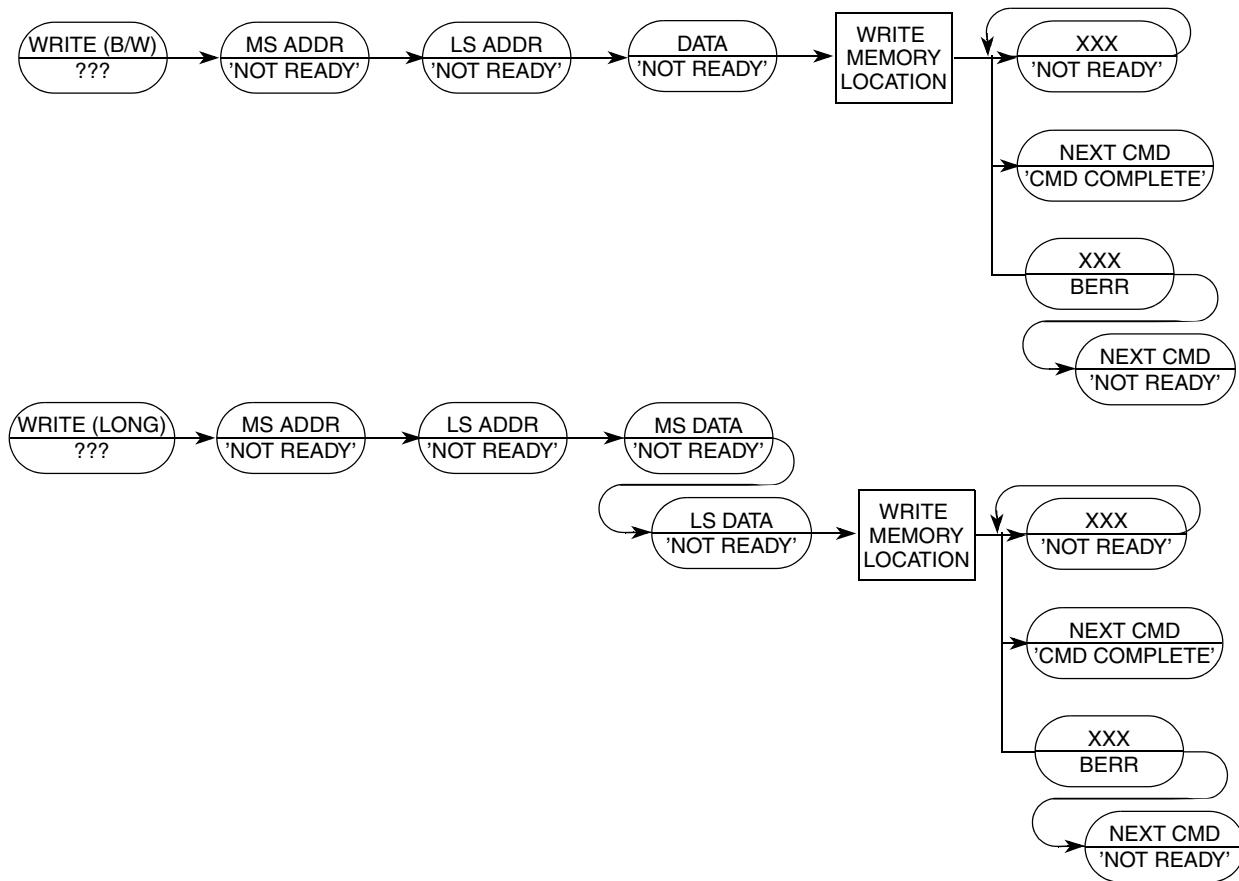


Figure 26-25. WRITE Command Sequence

Operand Data:

This two-operand instruction requires a longword absolute address that specifies a location to which the data operand is to be 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.

26.5.3.3.5 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 is incremented by the operand size (1, 2, or 4) and saved 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	12	11	8	7	4	3	0
	Result	X	X	X	X	X	X	X	D[7:0]
Word	Command	0x1				0xD		0x4	0x0
	Result					D[15:0]			
Longword	Command	0x1			0xD		0x8	0x0	
	Result				D[31:16]				D[15:0]

Figure 26-26. DUMP Command/Result Formats

Command Sequence:

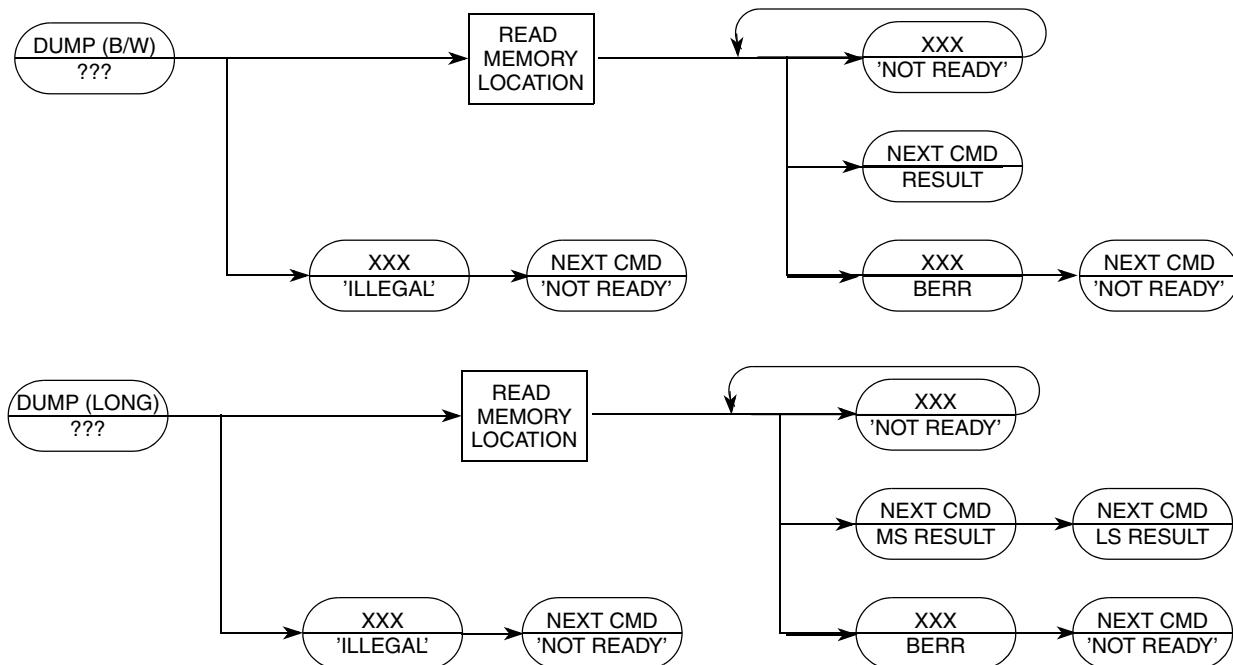


Figure 26-27. DUMP Command Sequence

Operand Data: None

Result Data:

Requested data is returned as either 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.

26.5.3.3.6 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 is incremented by the operand size (1, 2, or 4) and saved 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 26-28. FILL Command Format

Command Sequence:

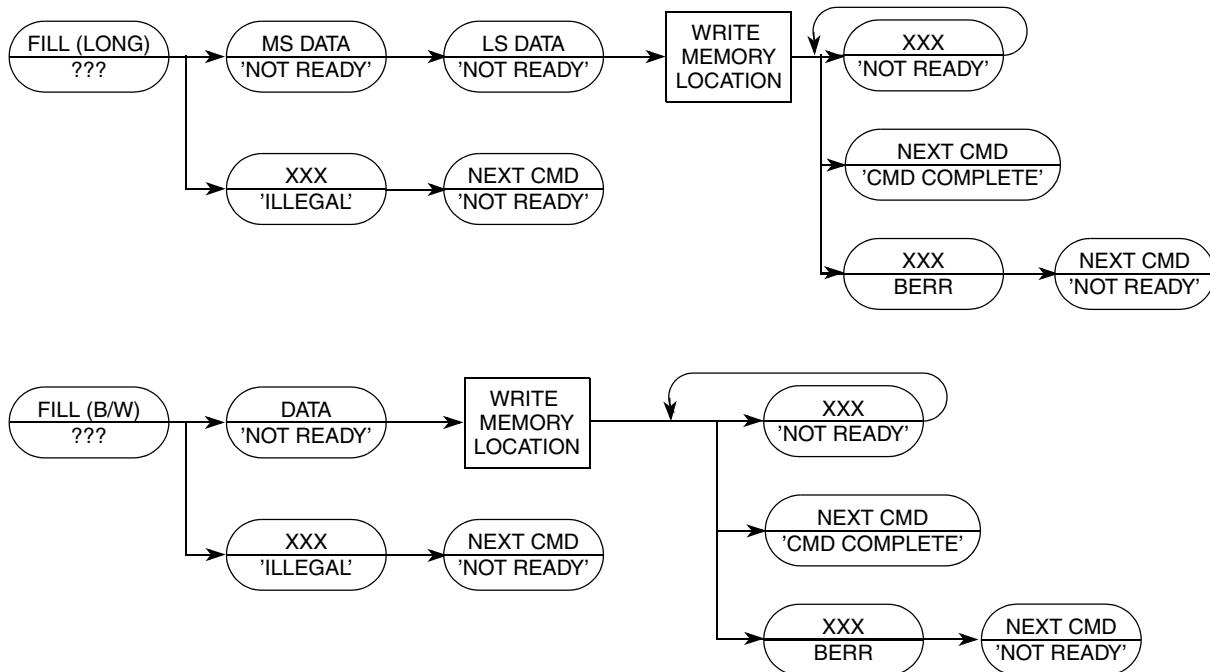


Figure 26-29. FILL Command Sequence

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.

26.5.3.3.7 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 is issued and the CPU is not halted, the command is ignored.

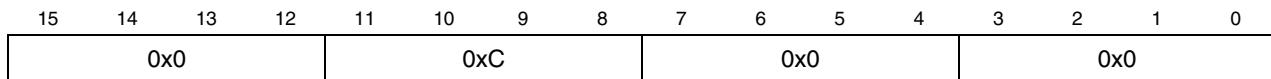


Figure 26-30. Go Command Format

Command Sequence:

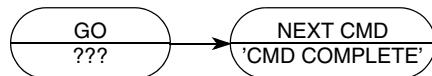


Figure 26-31. Go Command Sequence

Operand Data: None

Result Data: The command-complete response (0xFFFF) is returned during the next shift operation.

26.5.3.8 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 26-32. NOP Command Format

Command Sequence:

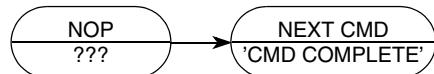


Figure 26-33. NOP Command Sequence

Operand Data: None

Result Data: The command-complete response, 0xFFFF (with S cleared), is returned during the next shift operation.

26.5.3.9 Synchronize PC to the PST/DDATA Lines (SYNC_PC)

Capture the current PC (program counter) and display it on the PST/DDATA outputs. After the debug module receives the command, it sends a signal to the ColdFire core that the current PC must be displayed. The core 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 PST and DDATA values is defined below:

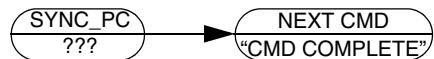
1. Debug signals a SYNC_PC command is pending.
2. CPU completes the current instruction.
3. CPU forces an instruction fetch to the next PC, generates a PST = \$5 value indicating a taken branch and signals DDATA. DDATA captures the instruction address corresponding to the PC. DDATA generates a PST marker (\$9-\$B) as defined by CSR[BTB] and displays the captured PC address.

This 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.

Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
					0x0		0x0		0x0		0x0		0x1		

Command Sequence:



Operand Data: None

Result Data: The command complete response, \$FFFF (with the status bit cleared), is returned during the next shift operation.

26.5.3.3.10 Read Control Register (RCREG)

Reads the selected control register and returns 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 as that used by the MOVEC instruction.

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				Rc																															
Result	D[31:16]																																			
	D[15:0]																																			

Figure 26-34. RCREG Command/Result Formats

Rc encoding:

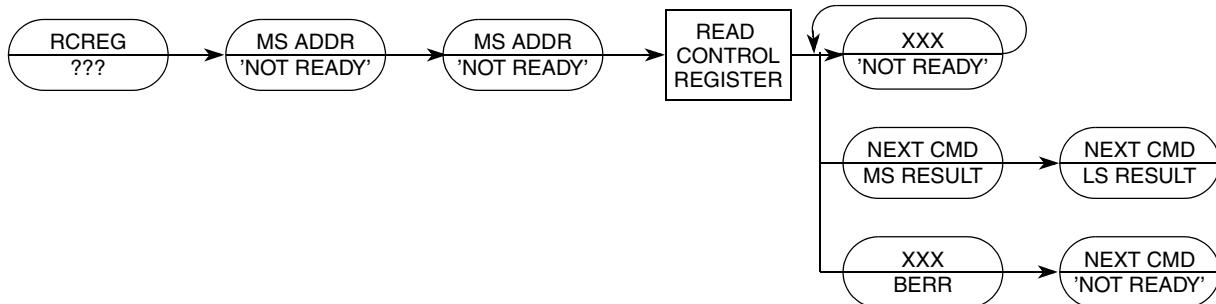
Table 26-21. Control Register Map

Rc	Register Definition
0x002	Cache Control Register (CACR)
0x004	Access Control Register (ACR0)
0x005	Access Control Register (ACR1)
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)

Table 26-21. Control Register Map (continued)

Rc	Register Definition
0x80A	MAC Accumulator 2 (ACC2)
0x80B	MAC Accumulator 3 (ACC3)
0x80E	Status Register (SR)
0x80F	Program Register (PC)
0xC05	RAM Base Address Register 1 (RAMBAR1)

Command Sequence:

**Figure 26-35. 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.

BDM Accesses of the Stack Pointer Registers (A7: SSP, USP)

The V2 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 and the other register is named simply the “other_A7”. Thus, 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 the A7 and other_A7 registers directly. It is the responsibility of the external development system to determine the mapping of the two hardware registers (A7, other_A7) to the two program-visible definitions (supervisor and user stack pointers), based on the supervisor bit of the status register.

BDM Accesses of the EMAC Registers

The presence of rounding logic in the output datapath of the EMAC requires that special care be taken during any BDM-initiated reads and writes of its programming model. In particular, it is necessary that any result rounding modes be disabled during the read/write process so the exact bit-wise contents of the EMAC registers are preserved.

As an example, any BDM read of an accumulator register (ACCn) must be preceded by two commands accessing the MAC status register. Specifically, the following BDM sequence is required:

```
BdmReadACCn (
    rcreg    macsr;          // read macsr contents & save
    wcreg    #0,macsr;       // disable all rounding modes
    rcreg    accn;           // read the desired accumulator
    wcreg    #saved_data,macsr;// restore the original macsr
)
```

Likewise, the following BDM sequence is needed to write an accumulator register:

```
BdmWriteACCn (
    rcreg    macsr;          // read macsr contents & save
    wcreg    #0,macsr;       // disable all rounding modes
    wcreg    #data,accn;      // write the desired accumulator
    wcreg    #saved_data,macsr// restore the original macsr
)
```

Additionally, it is required that writes to the accumulator extension registers be performed after the corresponding accumulators are updated. This is needed since a write to any accumulator alters the contents of the corresponding extension register.

For more information on saving and restoring the complete EMAC programming model, see [Section 4.4.1.1.2, “Saving and Restoring the EMAC Programming Model.”](#)

26.5.3.11 Write Control Register (WCREG)

The operand (longword) data is written to the specified control register. The write alters all 32 register bits.

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 26-36. WCREG Command/Result Formats

Command Sequence:

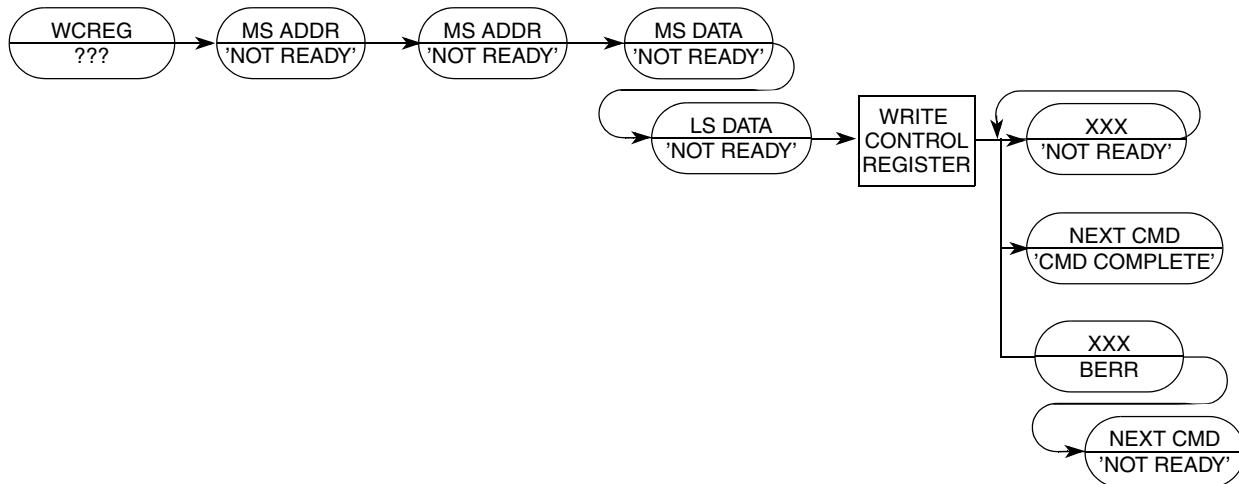


Figure 26-37. WCREG Command Sequence

Operand Data:

This instruction requires two longword operands. The first selects the register to which the operand data is to be written; 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.

26.5.3.3.12 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). Note that this read of the CSR clears CSR[FOF, TRG, HALT, BKPT]; as well as the trigger status bits (CSR[BSTAT]) if either a level-2 breakpoint has been triggered or a level-1 breakpoint has been triggered and no level-2 breakpoint has been enabled.

Command/Result Formats:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Command		0x2			0xD			100							DRc	
Result								D[31:16]								
								D[15:0]								

Figure 26-38. RDMREG Command/Result Formats

Table 26-22 shows the definition of DRc encoding.

Table 26-22. Definition of DRc Encoding—Read

DRc[4:0]	Debug Register Definition	Mnemonic	Initial State	Page
0x00	Configuration/Status	CSR	0x0	p. 26-7
0x01–0x1F	Reserved	—	—	—

Command Sequence:

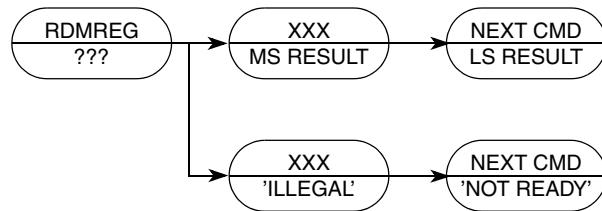


Figure 26-39. RDMREG Command Sequence

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.

26.5.3.3.13 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:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x2				0xC				100						DRc	
D[31:16]															
D[15:0]															

Figure 26-40. WDMREG BDM Command Format

Table 26-3 shows the definition of the DRc write encoding.

Command Sequence:

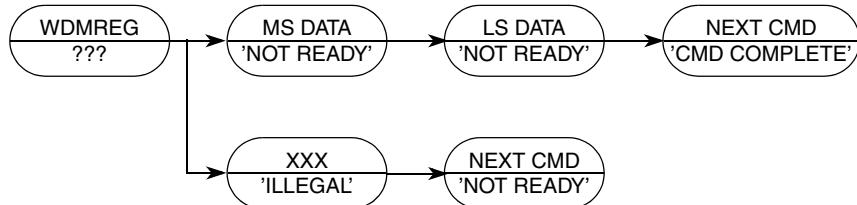


Figure 26-41. WDMREG Command Sequence

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.

26.6 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 small intrusions into the 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 either 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.

26.6.1 Theory of Operation

Breakpoint hardware can be configured to respond to triggers in several ways. The desired response is programmed into TDR. As shown in [Table 26-23](#), when a breakpoint is triggered, an indication (CSR[BSTAT]) is provided on the DDATA output port when it is not displaying captured processor status, operands, or branch addresses.

Table 26-23. DDATA[3:0]/CSR[BSTAT] Breakpoint Response

DDATA[3:0] ¹	CSR[BSTAT] ¹	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

¹ Encodings not shown are reserved for future use.

The breakpoint status is also posted in the CSR. Note that CSR[BSTAT] is cleared by a CSR read when either 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 TDR.

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 is executed. 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] = 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 the configuration TDR[TRC] = 10 the 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. 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, 12, from the vector table. (Refer to the *ColdFire Programmer's Reference Manual*).

Execution continues at the instruction address in the vector corresponding to the breakpoint triggered. All interrupts are ignored while the processor is in emulator mode. 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.

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.

If a hardware breakpoint such as a PC trigger is left unmodified by the debug interrupt service routine, another debug interrupt is generated after the completion of the RTE instruction. Therefore the debug interrupt service routine should clear the respective TDR setting.

In the Rev. A implementation, if a hardware breakpoint (e.g., a PC trigger) is left unmodified by the debug interrupt service routine, another debug interrupt is generated after the RTE instruction completes execution. In the Rev. B design, the hardware has been modified to inhibit the generation of another debug interrupt during the first instruction after the RTE exits emulator mode. This behavior is consistent with the existing logic involving trace mode, where the execution of the first instruction occurs before another trace exception is generated. This Rev. B enhancement disables all hardware breakpoints until the first instruction after the RTE has completed execution, regardless of the programmed trigger response.

26.6.1.1 Emulator Mode

Emulator mode is used to facilitate 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 26.5.1, "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:

- All interrupts are ignored, including level-7 interrupts.
- If CSR[MAP] = 1, all caching of memory and the SRAM module are disabled. All memory accesses are forced into a specially mapped address space signaled by TT = 0x2, TM = 0x5 or 0x6. This includes stack frame writes and the 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).

26.6.2 Concurrent BDM and Processor Operation

The debug module supports concurrent operation of both the processor and most BDM commands. BDM commands may be executed while the processor is running, except those 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.

Breakpoint registers must be carefully configured in a development system if the processor is executing. The debug module contains no hardware interlocks, so TDR should be disabled while breakpoint registers are loaded, after which TDR 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).

26.7 Processor Status, DDATA Definition

This section specifies the ColdFire processor and debug module's generation of the processor status (PST) and debug data (DDATA) output on an instruction basis. In general, the PST/DDATA output for an instruction is defined as follows:

$$\text{PST} = 0x1, \{\text{PST} = [0x89B], \text{DDATA} = \text{operand}\}$$

where the {...} 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 DDATA 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 DDATA output {2, 3, or 4 bytes} using a PST value of {0x9, 0xA, or 0xB}.

26.7.1 User Instruction Set

Table 26-24 shows the PST/DDATA 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. The ‘DD’ nomenclature refers to the DDATA outputs.

Table 26-24. PST/DDATA Specification for User-Mode Instructions

Instruction	Operand Syntax	PST/DDATA
add.l	<ea>y,Rx	PST = 0x1, {PST = 0xB, DD = source operand}
add.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
addi.l	#imm,Dx	PST = 0x1
addq.l	#imm,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
addx.l	Dy,Dx	PST = 0x1
and.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
and.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
andi.l	#imm,Dx	PST = 0x1
asl.l	{Dy,#imm},Dx	PST = 0x1
asr.l	{Dy,#imm},Dx	PST = 0x1
bitrev.l	Dx	PST = 0x1
byterev.l	Dx	PST = 0x1
bcc.{b,w}		if taken, then PST = 0x5, else PST = 0x1
bchg	#imm,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bchg	Dy,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bclr	#imm,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bclr	Dy,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bra.{b,w}		PST = 0x5
bset	#imm,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bset	Dy,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
bsr.{b,w}		PST = 0x5, {PST = 0xB, DD = destination operand}
btst	#imm,<ea>x	PST = 0x1, {PST = 0x8, DD = source operand}
btst	Dy,<ea>x	PST = 0x1, {PST = 0x8, DD = source operand}
clr.b	<ea>x	PST = 0x1, {PST = 0x8, DD = destination operand}
clr.l	<ea>x	PST = 0x1, {PST = 0xB, DD = destination operand}
clr.w	<ea>x	PST = 0x1, {PST = 0x9, DD = destination operand}
cmp.l	<ea>y,Rx	PST = 0x1, {PST = 0xB, DD = source operand}
cmpl.l	#imm,Dx	PST = 0x1

Table 26-24. PST/DDATA Specification for User-Mode Instructions (continued)

Instruction	Operand Syntax	PST/DDATA
divs.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
divs.w	<ea>y,Dx	PST = 0x1, {PST = 0x9, DD = source operand}
divu.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
divu.w	<ea>y,Dx	PST = 0x1, {PST = 0x9, DD = source operand}
eor.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
eori.l	#imm,Dx	PST = 0x1
ext.l	Dx	PST = 0x1
ext.w	Dx	PST = 0x1
extb.l	Dx	PST = 0x1
ff1.l	Dx	PST = 0x1
jmp	<ea>x	PST = 0x5, {PST = [0x9AB], DD = target address} ¹
jsr	<ea>x	PST = 0x5, {PST = [0x9AB], DD = target address}, {PST = 0xB, DD = destination operand} ¹
lea	<ea>y,Ax	PST = 0x1
link.w	Ay,#imm	PST = 0x1, {PST = 0xB, DD = destination operand}
lsl.l	{Dy,#imm},Dx	PST = 0x1
lsr.l	{Dy,#imm},Dx	PST = 0x1
move.b	<ea>y,<ea>x	PST = 0x1, {PST = 0x8, DD = source}, {PST = 0x8, DD = destination}
move.l	<ea>y,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
move.w	<ea>y,<ea>x	PST = 0x1, {PST = 0x9, DD = source}, {PST = 0x9, DD = destination}
move.w	CCR,Dx	PST = 0x1
move.w	{Dy,#imm},CCR	PST = 0x1
movem.l	#list,<ea>x	PST = 0x1, {PST = 0xB, DD = destination},... ²
movem.l	<ea>y,#list	PST = 0x1, {PST = 0xB, DD = source},... ²
moveq	#imm,Dx	PST = 0x1
muls.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
muls.w	<ea>y,Dx	PST = 0x1, {PST = 0x9, DD = source operand}
mulu.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
mulu.w	<ea>y,Dx	PST = 0x1, {PST = 0x9, DD = source operand}
neg.l	Dx	PST = 0x1
negx.l	Dx	PST = 0x1
nop		PST = 0x1
not.l	Dx	PST = 0x1

Table 26-24. PST/DDATA Specification for User-Mode Instructions (continued)

Instruction	Operand Syntax	PST/DDATA
or.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = source operand}
or.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
ori.l	#imm,Dx	PST = 0x1
pea	<ea>y	PST = 0x1, {PST = 0xB, DD = destination operand}
pulse		PST = 0x4
rems.l	<ea>y,Dx:Dw	PST = 0x1, {PST = 0xB, DD = source operand}
remu.l	<ea>y,Dx:Dw	PST = 0x1, {PST = 0xB, DD = source operand}
rts		PST = 0x1, {PST = 0xB, DD = source operand}, PST = 0x5, {PST = [0x9AB], DD = target address}
scc	Dx	PST = 0x1
sub.l	<ea>y,Rx	PST = 0x1, {PST = 0xB, DD = source operand}
sub.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
subi.l	#imm,Dx	PST = 0x1
subq.l	#imm,<ea>x	PST = 0x1, {PST = 0xB, DD = source}, {PST = 0xB, DD = destination}
subx.l	Dy,Dx	PST = 0x1
swap	Dx	PST = 0x1
trap	#imm	PST = 0x1 ³
trapf		PST = 0x1
tst.b	<ea>x	PST = 0x1, {PST = 0x8, DD = source operand}
tst.l	<ea>x	PST = 0x1, {PST = 0xB, DD = source operand}
tst.w	<ea>x	PST = 0x1, {PST = 0x9, DD = source operand}
unlk	Ax	PST = 0x1, {PST = 0xB, DD = destination operand}
wddata.b	<ea>y	PST = 0x4, {PST = 0x8, DD = source operand}
wddata.l	<ea>y	PST = 0x4, {PST = 0xB, DD = source operand}
wddata.w	<ea>y	PST = 0x4, {PST = 0x9, DD = source operand}

- ¹ 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).
- ² 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.
- ³ During normal exception processing, the PST output is driven to a 0xC 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.

Debug Module

```
Exception ProcessingPST = 0xC,{PST = 0xB,DD = destination},// stack frame
    {PST = 0xB,DD = destination},// stack frame
    {PST = 0xB,DD = source},// vector read
    PST = 0x5,{PST = [0x9AB],DD = target}// handler PC
```

The PST/DDATA specification for the reset exception is shown below:

```
Exception ProcessingPST = 0xC,
    PST = 0x5,{PST = [0x9AB],DD = target}// handler PC
```

The initial references at address 0 and 4 are never captured nor displayed since these accesses are treated as instruction fetches.

For all types of exception processing, the PST = 0xC value is driven at all times, unless the PST output is needed for one of the optional marker values or for the taken branch indicator (0x5).

[Table 26-25](#) shows the PST/DDATA specification for multiply-accumulate instructions.

Table 26-25. PST/DDATA Specification for MAC Instructions

Instruction	Operand Syntax	PST/DDATA
mac.l	Ry,Rx,Accx	PST = 0x1
mac.l	RyRx,<ea>,Rw,Accx	PST = 0x1, {PST = 0xB, DD = source operand}
mac.w	Ry,Rx,Accx	PST = 0x1
mac.w	Ry,Rx,<ea>,Rw,Accx	PST = 0x1, {PST = 0xB, DD = source operand}
move.l	<ea>y,Accx	PST = 0x1
move.l	Accy,Accx	PST = 0x1
move.l	<ea>y,MACR	PST = 0x1
move.l	<ea>y,MASK	PST = 0x1
move.l	<ea>y,Accext01	PST = 0x1
move.l	<ea>y,Accext23	PST = 0x1
move.l	Accy,Rx	PST = 0x1
move.l	MACSR,CCR	PST = 0x1
move.l	MACSR,Rx	PST = 0x1
move.l	MASK,Rx	PST = 0x1
move.l	Accext01,Rx	PST = 0x1
move.l	Accext23,Rx	PST = 0x1
msac.l	Ry,Rx,Accx	PST = 0x1
msac.l	Ry,Rx,<ea>,Rw,Accx	PST = 0x1, {PST = 0xB, DD = source operand}
msac.w	Ry,Rx,Accx	PST = 0x1
msac.w	Ry,Rx,<ea>,Rw,Accx	PST = 0x1, {PST = 0xB, DD = source operand}

26.7.2 Supervisor Instruction Set

The supervisor instruction set has complete access to the user mode instructions plus the opcodes shown below. The PST/DDATA specification for these opcodes is shown in [Table 26-26](#).

Table 26-26. PST/DDATA Specification for Supervisor-Mode Instructions

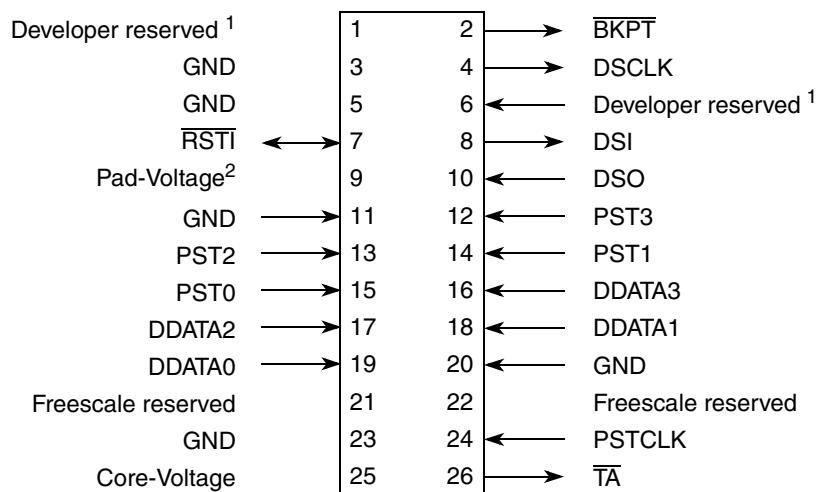
Instruction	Operand Syntax	PST/DDATA
cpushl		PST = 0x1
halt		PST = 0x1, PST = 0xF
move.w	SR,Dx	PST = 0x1
move.w	{Dy,#imm},SR	PST = 0x1, {PST = 0x3}
movec	Ry,Rc	PST = 0x1
rte		PST = 0x7, {PST = 0xB, DD = source operand}, {PST = 3},{ PST =0xB, DD =source operand}, PST = 0x5, {[PST = 0x9AB], DD = target address}
stldsr.w	#imm	PST = 0x1, {PST = 0xA, DD = destination operand, PST = 0x3}
stop	#imm	PST = 0x1, PST = 0xE
wdebug	<ea>y	PST = 0x1, {PST = 0xB, DD = source, PST = 0xB, DD = source}

The move-to-SR, STLDSR, and RTE instructions include an optional PST = 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 (PST = 0xE) and the halted state (PST = 0xF) display this status throughout the entire time the ColdFire processor is in the given mode.

26.8 Freescale-Recommended BDM Pinout

The ColdFire BDM connector, [Figure 26-42](#), is a 26-pin Berg connector arranged 2×13.



¹Pins reserved for BDM developer use.

²Supplied by target

Figure 26-42. Recommended BDM Connector

Chapter 27

IEEE 1149.1 Test Access Port (JTAG)

27.1 Introduction

The joint test action group or JTAG is a dedicated user-accessible test logic, that complies with the IEEE 1149.1 standard for boundary-scan testability, to help 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, $\overline{\text{TRST}}$.

27.1.1 Block Diagram

Figure 27-1 shows the block diagram of the JTAG module.

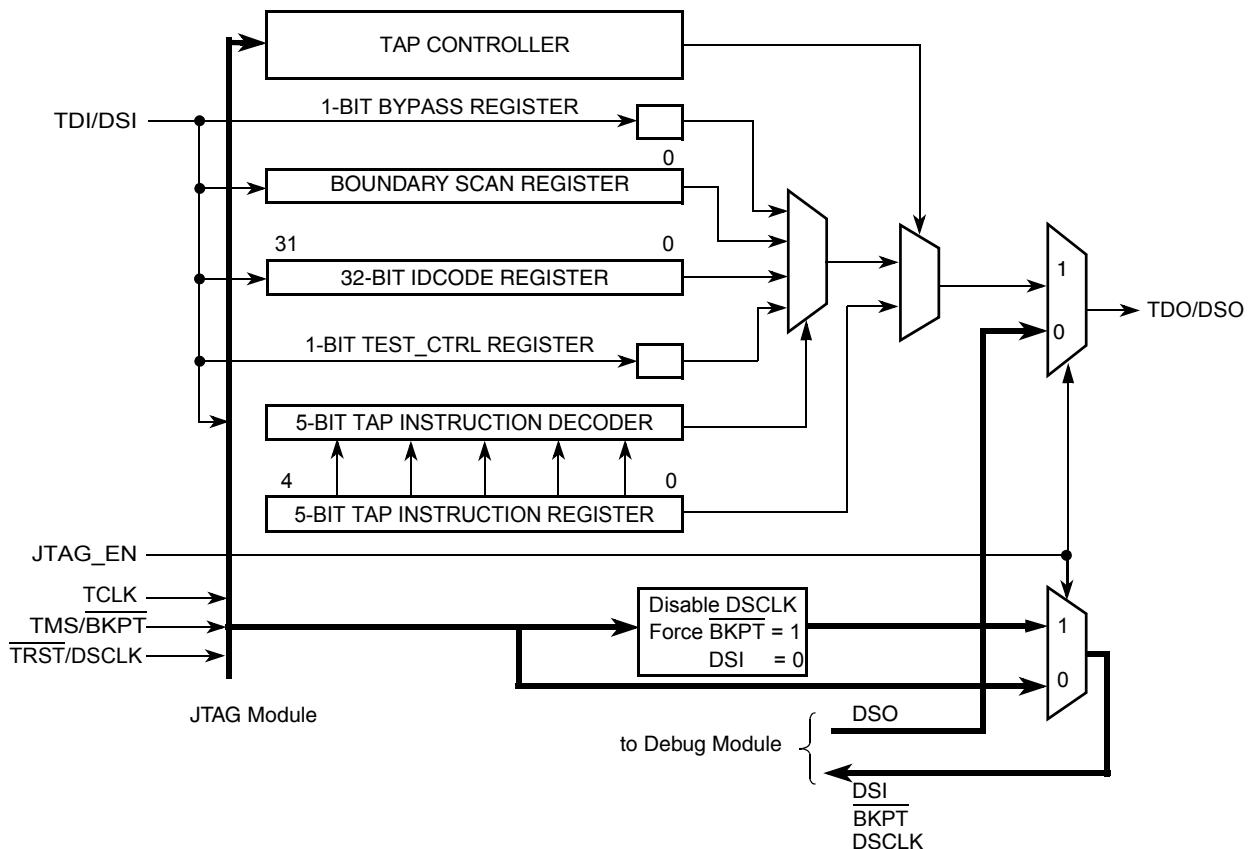


Figure 27-1. JTAG Block Diagram

27.1.2 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 shift out the result
- Selects between JTAG TAP controller and background debug module (BDM) using a dedicated JTAG_EN pin

27.1.3 Modes of Operation

The JTAG_EN pin can select between the following modes of operation:

- JTAG mode (JTAG_EN = 1)
- BDM - background debug mode (For more information, refer to [Section 26.5, “Background Debug Mode \(BDM\)”](#)) (JTAG_EN = 0)

27.2 External Signal Description

The JTAG module has five input and one output external signals, as described in [Table 27-1](#).

Table 27-1. 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	—

27.2.1 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 27-2](#) summarizes the pin function selected depending upon JTAG_EN logic state.

Table 27-2. Pin Function Selected

	JTAG_EN = 0	JTAG_EN = 1	Pin Name
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 27-3](#), in order to disable the corresponding module.

Table 27-3. Signal State to the Disable Module

	JTAG_EN = 0	JTAG_EN = 1
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.

27.2.2 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.

27.2.3 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.

27.2.4 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.

27.2.5 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, the data input on DSI is sampled and DSO changes state.

27.2.6 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 is actively driven in the shift-IR and shift-DR controller states.

The DSO pin provides serial output data in BDM mode.

27.3 Memory Map/Register Definition

The JTAG module registers are not memory mapped and are only accessible through the TDO/DSO pin.

27.3.1 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 27.4.3, “JTAG Instructions”](#) for a list of possible instruction codes.

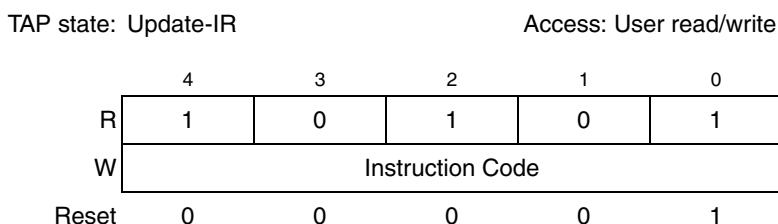


Figure 27-2. 5-Bit Instruction Register (IR)

27.3.2 IDCODE Register

The IDCODE is a read-only register; its value is chip dependent. For more information, see [Section 27.4.3.1, “IDCODE Instruction.”](#)

IR[4:0]: 0_0001 (IDCODE)

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	PRN				DC				PIN				JEDEC								ID											
W	0	0	0	0	0	1	1	1	0	1	Device Dependent	0	0	0	0	0	0	1	1	1	0	1	1	0	1	0	1	0	1	0	1	
Reset	0	0	0	0	0	1	1	1	0	1	Device Dependent	0	0	0	0	0	0	1	1	1	0	1	1	0	1	0	1	0	1	0	1	

Figure 27-3. IDCODE Register

Table 27-4. 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. 0x044 MCF5208 0x045 MCF5207
11–1 JEDEC	Joint Electron Device Engineering Council ID bits. Indicate the reduced JEDEC ID for Freescale.
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.

27.3.3 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.

27.3.4 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.

IR[4:0]: 0_0110 Access: User read-only

	0
R	DSE
W	

Reset 0

Figure 27-4. 1-Bit TEST_CTRL Register

27.3.5 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.

27.4 Functional Description

27.4.1 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.

27.4.2 TAP Controller

The TAP controller is a state machine that changes state based on the sequence of logical values on the TMS pin. [Figure 27-5](#) 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 27-5](#) 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.

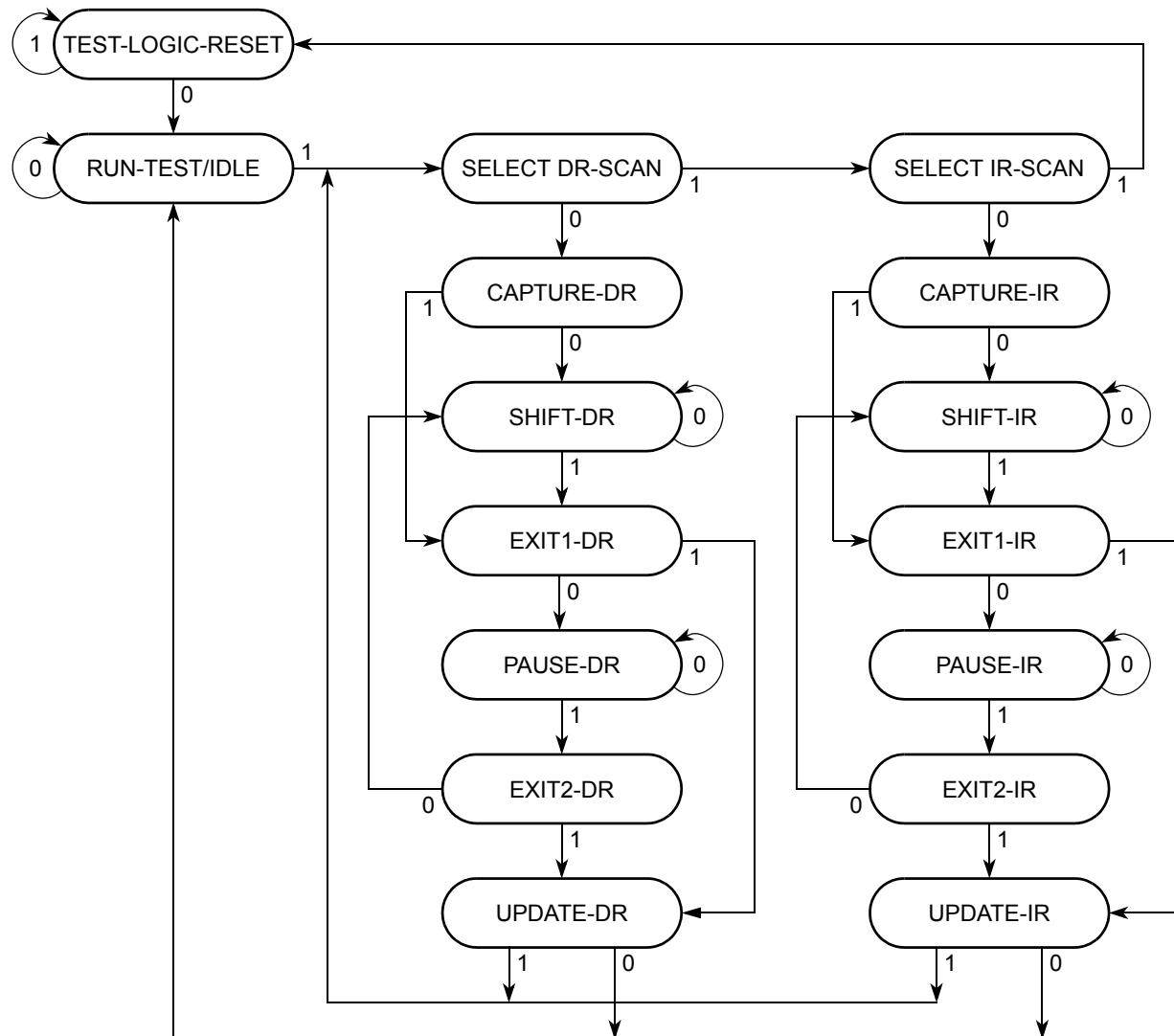


Figure 27-5. TAP Controller State Machine Flow

27.4.3 JTAG Instructions

Table 27-5 describes public and private instructions.

Table 27-5. 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 27-5. 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 ¹	Decoded to select bypass register

¹ Freescale reserves the right to change the decoding of the unused opcodes in the future.

27.4.3.1 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.

27.4.3.2 SAMPLE/PRELOAD Instruction

The SAMPLE/PRELOAD instruction has two functions:

- SAMPLE - See [Section 27.4.3.3, “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.

27.4.3.3 SAMPLE Instruction

The SAMPLE instruction obtains a sample of the system data and control signals present at the MCU input pins and just 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. Both 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.

27.4.3.4 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.

27.4.3.5 ENABLE_TEST_CTRL Instruction

The ENABLE_TEST_CTRL instruction selects a -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.

27.4.3.6 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.

27.4.3.7 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.

27.4.3.8 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.

27.5 Initialization/Application Information

27.5.1 Restrictions

The test logic is a static logic design, and TCLK can be stopped in either a high or low state without loss of data. However, the system clock is not synchronized to TCLK internally. Any mixed operation using both the test logic and the 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 either 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_{DD}.
- 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 either connected to EV_{DD} or left unconnected.

27.5.2 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, since 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.