# MCF5407 ColdFire® Integrated Microprocessor User's Manual

MCF5407UM/D
Rev. 0.1, 11/2001

Digital DNA from Motorola

68K ColdFire MICROPROCESSORS

MOTOROLA

ColdFire is a registered trademark and DigitalDNA is a trademark of Motorola, Inc.
$I^2C$ is a registered trademark of Philips Semiconductors

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola 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 which may be provided in Motorola 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. Motorola does not convey any license under its patent rights nor the rights of others. Motorola 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 Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola 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 Motorola was negligent regarding the design or manufacture of the part. Motorola and Ⓜ are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

**How to reach us:**

**USA/EUROPE/Locations Not Listed:** Motorola Literature Distribution; P.O. Box 5405, Denver, Colorado 80217. 1–303–675–2140 or 1–800–441–2447

**JAPAN:** Motorola Japan Ltd.; SPS, Technical Information Center, 3–20–1, Minami–Azabu. Minato–ku, Tokyo 106–8573 Japan. 81–3–3440–3569

**ASIA/PACIFIC:** Motorola Semiconductors H.K. Ltd.; Silicon Harbour Centre, 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong. 852–26668334

**Technical Information Center:** 1–800–521–6274

**HOME PAGE:** http://www.motorola.com/semiconductors

**Document Comments**: FAX (512) 895-2638, Attn: TECD Applications Engineering

© Motorola Inc., 2001. All rights reserved.

Overview 1
Part I: MCF5407 Processor Core Part I
ColdFire Core 2
Hardware Multiply/Accumulate (MAC) Unit 3
Local Memory 4
Debug Support 5
Part II: System Integration Module (SIM) Part II
SIM Overview 6
Phase-Locked Loop (PLL) 7
$I^2C$ Module 8
Interrupt Controller 9
Chip-Select Module 10
Synchronous/Asynchronous DRAM Controller Module 11
Part III: Peripheral Module Part III
DMA Controller Module 12
Timer Module 13
UART Modules 14
Parallel Port (General-Purpose I/O) 15
Part IV: Hardware Interface Part IV
Mechanical Data 16
Signal Descriptions 17
Bus Operation 18
IEEE 1149.1 Test Access Port (JTAG) 19
Electrical Specifications 20
Appendix A: Migration A
Appendix B: Memory Map B
Glossary of Terms and Abbreviations GLO
Index IND

| | |
|---|---|
| 1 | Overview |
| Part I | Part I: MCF5407 Processor Core |
| 2 | ColdFire Core |
| 3 | Hardware Multiply/Accumulate (MAC) Unit |
| 4 | Local Memory |
| 5 | Debug Support |
| Part II | Part II: System Integration Module (SIM) |
| 6 | SIM Overview |
| 7 | Phase-Locked Loop (PLL) |
| 8 | $I^2C$ Module |
| 9 | Interrupt Controller |
| 10 | Chip-Select Module |
| 11 | Synchronous/Asynchronous DRAM Controller Module |
| Part III | Part III: Peripheral Module |
| 12 | DMA Controller Module |
| 13 | Timer Module |
| 14 | UART Modules |
| 15 | Parallel Port (General-Purpose I/O) |
| Part IV | Part IV: Hardware Interface |
| 16 | Mechanical Data |
| 17 | Signal Descriptions |
| 18 | Bus Operation |
| 19 | IEEE 1149.1 Test Access Port (JTAG) |
| 20 | Electrical Specifications |
| A | Appendix A: Migration |
| B | Appendix B: Memory Map |
| GLO | Glossary of Terms and Abbreviations |
| IND | Index |

# CONTENTS


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

**Chapter 1**
**Overview**

| | | |
|---|---|---|
| 1.1 | Features | 1-1 |
| 1.2 | MCF5407 Features | 1-4 |
| 1.2.1 | Process | 1-7 |
| 1.3 | ColdFire Module Description | 1-7 |
| 1.3.1 | ColdFire Core | 1-7 |
| 1.3.1.1 | Instruction Fetch Pipeline (IFP) | 1-7 |
| 1.3.1.2 | Operand Execution Pipeline (OEP) | 1-8 |
| 1.3.1.3 | MAC Module | 1-8 |
| 1.3.1.4 | Integer Divide Module | 1-8 |
| 1.3.2 | Harvard Architecture | 1-8 |
| 1.3.2.1 | 16-Kbyte Instruction Cache/8-Kbyte Data Cache | 1-9 |
| 1.3.2.2 | Internal 2-Kbyte SRAMs | 1-9 |
| 1.3.3 | DRAM Controller | 1-9 |
| 1.3.4 | DMA Controller | 1-9 |
| 1.3.5 | UART Modules | 1-10 |
| 1.3.6 | Timer Module | 1-11 |
| 1.3.7 | $I^2C$ Module | 1-11 |
| 1.3.8 | System Interface | 1-11 |
| 1.3.8.1 | External Bus Interface | 1-11 |
| 1.3.8.2 | Chip Selects | 1-11 |
| 1.3.8.3 | 16-Bit Parallel Port Interface | 1-12 |
| 1.3.8.4 | Interrupt Controller | 1-12 |
| 1.3.8.5 | JTAG | 1-12 |
| 1.3.9 | System Debug Interface | 1-12 |
| 1.3.10 | PLL Module | 1-13 |
| 1.4 | Programming Model, Addressing Modes, and Instruction Set | 1-13 |
| 1.4.1 | Programming Model | 1-15 |
| 1.4.2 | User Registers | 1-15 |

# CONTENTS


| Paragraph Number | Title | Page Number |
|---|---|---|
| 1.4.3 | Supervisor Registers | 1-16 |
| 1.4.4 | Instruction Set | 1-16 |

**Part I**
**MCF5407 Processor Core**

**Chapter 2**
**ColdFire Core**

| | | |
|---|---|---|
| 2.1 | Features and Enhancements | 2-1 |
| 2.1.1 | Clock-Multiplied Microprocessor Core | 2-2 |
| 2.1.2 | Enhanced Pipelines | 2-2 |
| 2.1.2.1 | Instruction Fetch Pipeline (IFP) | 2-4 |
| 2.1.2.1.1 | Branch Acceleration | 2-4 |
| 2.1.2.2 | Operand Execution Pipeline (OEP) | 2-4 |
| 2.1.2.2.1 | Illegal Opcode Handling | 2-5 |
| 2.1.2.2.2 | Hardware Multiply/Accumulate (MAC) Unit | 2-5 |
| 2.1.2.2.3 | Hardware Divide Unit | 2-6 |
| 2.1.2.3 | Harvard Memory Architecture | 2-6 |
| 2.1.3 | Debug Module Enhancements | 2-6 |
| 2.2 | Programming Model | 2-7 |
| 2.2.1 | User Programming Model | 2-8 |
| 2.2.1.1 | Data Registers (D0–D7) | 2-8 |
| 2.2.1.2 | Address Registers (A0–A6) | 2-9 |
| 2.2.1.3 | Stack Pointer (A7, SP) | 2-9 |
| 2.2.1.4 | Program Counter (PC) | 2-9 |
| 2.2.1.5 | Condition Code Register (CCR) | 2-9 |
| 2.2.1.6 | MAC Programming Model | 2-10 |
| 2.2.2 | Supervisor Programming Model | 2-10 |
| 2.2.2.1 | Status Register (SR) | 2-11 |
| 2.2.2.2 | Vector Base Register (VBR) | 2-12 |
| 2.2.2.3 | Cache Control Register (CACR) | 2-12 |
| 2.2.2.4 | Access Control Registers (ACR0–ACR3) | 2-12 |
| 2.2.2.5 | RAM Base Address Registers (RAMBAR0 and RAMBAR1) | 2-12 |
| 2.2.2.6 | Module Base Address Register (MBAR) | 2-12 |
| 2.3 | Integer Data Formats | 2-13 |
| 2.4 | Organization of Data in Registers | 2-13 |
| 2.4.1 | Organization of Integer Data Formats in Registers | 2-13 |
| 2.4.2 | Organization of Integer Data Formats in Memory | 2-14 |
| 2.5 | Addressing Mode Summary | 2-15 |
| 2.6 | Instruction Set Summary | 2-15 |
| 2.6.1 | Additions to the Instruction Set Architecture | 2-18 |

# CONTENTS


| Paragraph Number | Title | Page Number |
|---|---|---|
| 2.6.2 | Instruction Set Summary | 2-19 |
| 2.7 | Execution Timings | 2-23 |
| 2.7.1 | MOVE Instruction Execution Timing | 2-25 |
| 2.7.2 | Execution Timings—One-Operand Instructions | 2-26 |
| 2.7.3 | Execution Timings—Two-Operand Instructions | 2-27 |
| 2.7.4 | Miscellaneous Instruction Execution Times | 2-29 |
| 2.7.5 | Branch Instruction Execution Times | 2-30 |
| 2.8 | Exception Processing Overview | 2-31 |
| 2.8.1 | Exception Stack Frame Definition | 2-32 |
| 2.8.2 | Processor Exceptions | 2-34 |
| 2.9 | ColdFire Instruction Set Architecture Enhancements | 2-36 |


## Chapter 3
## Hardware Multiply/Accumulate (MAC) Unit


| | | |
|---|---|---|
| 3.1 | Overview | 3-1 |
| 3.1.0.1 | MAC Programming Model | 3-2 |
| 3.1.0.2 | General Operation | 3-3 |
| 3.1.0.3 | MAC Instruction Set Summary | 3-4 |
| 3.1.0.4 | Data Representation | 3-4 |
| 3.2 | MAC Instruction Execution Timings | 3-5 |


## Chapter 4
## Local Memory


| | | |
|---|---|---|
| 4.1 | Interactions between Local Memory Modules | 4-1 |
| 4.2 | SRAM Overview | 4-1 |
| 4.3 | SRAM Operation | 4-2 |
| 4.4 | SRAM Programming Model | 4-3 |
| 4.4.1 | SRAM Base Address Registers (RAMBAR0/RAMBAR1) | 4-3 |
| 4.5 | SRAM Initialization | 4-4 |
| 4.5.1 | SRAM Initialization Code | 4-5 |
| 4.6 | Power Management | 4-6 |
| 4.7 | Cache Overview | 4-6 |
| 4.8 | Cache Organization | 4-8 |
| 4.8.1 | Cache Line States: Invalid, Valid-Unmodified, and Valid-Modified | 4-8 |
| 4.8.2 | The Cache at Start-Up | 4-9 |
| 4.9 | Cache Operation | 4-11 |
| 4.9.1 | Caching Modes | 4-13 |
| 4.9.1.1 | Cacheable Accesses | 4-14 |
| 4.9.1.2 | Write-Through Mode (Data Cache Only) | 4-14 |
| 4.9.1.3 | Copyback Mode (Data Cache Only) | 4-14 |

# CONTENTS


| Paragraph Number | Title | Page Number |
|---|---|---|
| 4.9.2 | Cache-Inhibited Accesses | 4-14 |
| 4.9.3 | Cache Protocol | 4-15 |
| 4.9.3.1 | Read Miss | 4-16 |
| 4.9.3.2 | Write Miss (Data Cache Only) | 4-16 |
| 4.9.3.3 | Read Hit | 4-16 |
| 4.9.3.4 | Write Hit (Data Cache Only) | 4-17 |
| 4.9.4 | Cache Coherency (Data Cache Only) | 4-17 |
| 4.9.5 | Memory Accesses for Cache Maintenance | 4-17 |
| 4.9.5.1 | Cache Filling | 4-17 |
| 4.9.5.2 | Cache Pushes | 4-18 |
| 4.9.5.2.1 | Push and Store Buffers | 4-18 |
| 4.9.5.2.2 | Push and Store Buffer Bus Operation | 4-18 |
| 4.9.6 | Cache Locking | 4-19 |
| 4.10 | Cache Registers | 4-21 |
| 4.10.1 | Cache Control Register (CACR) | 4-21 |
| 4.10.2 | Access Control Registers (ACR0–ACR3) | 4-23 |
| 4.11 | Cache Management | 4-24 |
| 4.12 | Cache Operation Summary | 4-27 |
| 4.12.1 | Instruction Cache State Transitions | 4-27 |
| 4.12.2 | Data Cache State Transitions | 4-28 |
| 4.13 | Cache Initialization Code | 4-32 |


## Chapter 5
## Debug Support


| | | |
|---|---|---|
| 5.1 | Overview | 5-1 |
| 5.2 | Signal Descriptions | 5-2 |
| 5.2.1 | Processor Status/Debug Data (PSTDDATA[7:0]) | 5-3 |
| 5.3 | Real-Time Trace Support | 5-4 |
| 5.3.1 | Begin Execution of Taken Branch (PST = 0x5) | 5-6 |
| 5.3.2 | Processor Stopped or Breakpoint State Change (PST = 0xE) | 5-7 |
| 5.3.3 | Processor Halted (PST = 0xF) | 5-7 |
| 5.4 | Programming Model | 5-8 |
| 5.4.1 | Address Attribute Trigger Registers (AATR, AATR1) | 5-10 |
| 5.4.2 | Address Breakpoint Registers (ABLR/ABLR1, ABHR/ABHR1) | 5-12 |
| 5.4.3 | BDM Address Attribute Register (BAAR) | 5-12 |
| 5.4.4 | Configuration/Status Register (CSR) | 5-13 |
| 5.4.5 | Data Breakpoint/Mask Registers (DBR/DBR1, DBMR/DBMR1) | 5-15 |
| 5.4.6 | Program Counter Breakpoint/Mask Registers (PBR, PBR1, PBR2, PBR3, PBMR) | 5-16 |
| 5.4.7 | Trigger Definition Register (TDR) | 5-18 |
| 5.4.8 | Extended Trigger Definition Register (XTDR) | 5-19 |

# CONTENTS


| Paragraph Number | Title | Page Number |
|---|---|---|
| 5.4.9 | Resulting Set of Possible Trigger Combinations | 5-21 |
| 5.5 | Background Debug Mode (BDM) | 5-22 |
| 5.5.1 | CPU Halt | 5-22 |
| 5.5.2 | BDM Serial Interface | 5-24 |
| 5.5.2.1 | Receive Packet Format | 5-25 |
| 5.5.2.2 | Transmit Packet Format | 5-26 |
| 5.5.3 | BDM Command Set | 5-26 |
| 5.5.3.1 | ColdFire BDM Command Format | 5-27 |
| 5.5.3.1.1 | Extension Words as Required | 5-28 |
| 5.5.3.2 | Command Sequence Diagrams | 5-28 |
| 5.5.3.3 | Command Set Descriptions | 5-30 |
| 5.5.3.3.1 | Read A/D Register (RAREG/RDREG) | 5-30 |
| 5.5.3.3.2 | Write A/D Register (WAREG/WDREG) | 5-31 |
| 5.5.3.3.3 | Read Memory Location (READ) | 5-32 |
| 5.5.3.3.4 | Write Memory Location (WRITE) | 5-33 |
| 5.5.3.3.5 | Dump Memory Block (DUMP) | 5-35 |
| 5.5.3.3.6 | Fill Memory Block (FILL) | 5-37 |
| 5.5.3.3.7 | Resume Execution (GO) | 5-39 |
| 5.5.3.3.8 | No Operation (NOP) | 5-40 |
| 5.5.3.3.9 | Synchronize PC to the PSTDDATA Lines (SYNC_PC) | 5-41 |
| 5.5.3.3.10 | Read Control Register (RCREG) | 5-42 |
| 5.5.3.3.11 | Write Control Register (WCREG) | 5-43 |
| 5.5.3.3.12 | Read Debug Module Register (RDMREG) | 5-44 |
| 5.5.3.3.13 | Write Debug Module Register (WDMREG) | 5-45 |
| 5.6 | Real-Time Debug Support | 5-45 |
| 5.6.1 | Theory of Operation | 5-46 |
| 5.6.1.1 | Emulator Mode | 5-48 |
| 5.6.2 | Concurrent BDM and Processor Operation | 5-48 |
| 5.7 | Motorola-Recommended BDM Pinout | 5-49 |
| 5.8 | Debug C Definition of PSTDDATA Outputs | 5-49 |
| 5.8.1 | User Instruction Set | 5-50 |
| 5.8.2 | Supervisor Instruction Set | 5-53 |


## Part II
## System Integration Module (SIM)

### Chapter 6
### SIM Overview


| | | |
|---|---|---|
| 6.1 | Features | 6-1 |
| 6.2 | Programming Model | 6-3 |
| 6.2.1 | SIM Register Memory Map | 6-3 |

# CONTENTS


| Paragraph Number | Title | Page Number |
|---|---|---|
| 6.2.2 | Module Base Address Register (MBAR) | 6-4 |
| 6.2.3 | Reset Status Register (RSR) | 6-5 |
| 6.2.4 | Software Watchdog Timer | 6-6 |
| 6.2.5 | System Protection Control Register (SYPCR) | 6-8 |
| 6.2.6 | Software Watchdog Interrupt Vector Register (SWIVR) | 6-9 |
| 6.2.7 | Software Watchdog Service Register (SWSR) | 6-9 |
| 6.2.8 | PLL Clock Control for CPU STOP Instruction | 6-10 |
| 6.2.9 | Pin Assignment Register (PAR) | 6-10 |
| 6.2.10 | Bus Arbitration Control | 6-11 |
| 6.2.10.1 | Default Bus Master Park Register (MPARK) | 6-11 |
| 6.2.10.1.1 | Arbitration for Internally Generated Transfers (MPARK[PARK]) | 6-12 |
| 6.2.10.1.2 | Arbitration between Internal and External Masters for Accessing Internal Resources | 6-14 |


## Chapter 7
## Phase-Locked Loop (PLL)


| | | |
|---|---|---|
| 7.1 | Overview | 7-1 |
| 7.1.1 | PLL:PCLK Ratios | 7-2 |
| 7.2 | PLL Operation | 7-2 |
| 7.2.1 | Reset/Initialization | 7-2 |
| 7.2.2 | Normal Mode | 7-2 |
| 7.2.3 | Reduced-Power Mode | 7-3 |
| 7.2.4 | PLL Control Register (PLLCR) | 7-3 |
| 7.3 | PLL Port List | 7-4 |
| 7.4 | Timing Relationships | 7-4 |
| 7.4.1 | PCLK, PSTCLK, and BCLKO | 7-4 |
| 7.4.2 | $\overline{\text{RSTI}}$ Timing | 7-5 |
| 7.5 | PLL Power Supply Filter Circuit | 7-6 |


## Chapter 8
## $I^2C$ Module


| | | |
|---|---|---|
| 8.1 | Overview | 8-1 |
| 8.2 | Interface Features | 8-1 |
| 8.3 | $I^2C$ System Configuration | 8-3 |
| 8.4 | $I^2C$ Protocol | 8-3 |
| 8.4.1 | Arbitration Procedure | 8-4 |
| 8.4.2 | Clock Synchronization | 8-5 |
| 8.4.3 | Handshaking | 8-5 |
| 8.4.4 | Clock Stretching | 8-5 |
| 8.5 | Programming Model | 8-6 |

# CONTENTS


| Paragraph Number | Title | Page Number |
|---|---|---|
| 8.5.1 | $I^2C$ Address Register (IADR) | 8-6 |
| 8.5.2 | $I^2C$ Frequency Divider Register (IFDR) | 8-6 |
| 8.5.3 | $I^2C$ Control Register (I2CR) | 8-7 |
| 8.5.4 | $I^2C$ Status Register (I2SR) | 8-8 |
| 8.5.5 | $I^2C$ Data I/O Register (I2DR) | 8-9 |
| 8.6 | $I^2C$ Programming Examples | 8-10 |
| 8.6.1 | Initialization Sequence | 8-10 |
| 8.6.2 | Generation of START | 8-10 |
| 8.6.3 | Post-Transfer Software Response | 8-11 |
| 8.6.4 | Generation of STOP | 8-12 |
| 8.6.5 | Generation of Repeated START | 8-12 |
| 8.6.6 | Slave Mode | 8-13 |
| 8.6.7 | Arbitration Lost | 8-13 |


## Chapter 9
## Interrupt Controller


| | | |
|---|---|---|
| 9.1 | Overview | 9-1 |
| 9.2 | Interrupt Controller Registers | 9-2 |
| 9.2.1 | Interrupt Control Registers (ICR0–ICR9) | 9-3 |
| 9.2.2 | Autovector Register (AVR) | 9-5 |
| 9.2.3 | Interrupt Pending and Mask Registers (IPR and IMR) | 9-6 |
| 9.2.4 | Interrupt Port Assignment Register (IRQPAR) | 9-7 |


## Chapter 10
## Chip-Select Module


| | | |
|---|---|---|
| 10.1 | Overview | 10-1 |
| 10.2 | Chip-Select Module Signals | 10-1 |
| 10.3 | Chip-Select Operation | 10-2 |
| 10.3.1 | General Chip-Select Operation | 10-3 |
| 10.3.1.1 | 8-, 16-, and 32-Bit Port Sizing | 10-4 |
| 10.3.1.2 | Global Chip-Select Operation | 10-4 |
| 10.4 | Chip-Select Registers | 10-5 |
| 10.4.1 | Chip-Select Module Registers | 10-6 |
| 10.4.1.1 | Chip-Select Address Registers (CSAR0–CSAR7) | 10-6 |
| 10.4.1.2 | Chip-Select Mask Registers (CSMR0–CSMR7) | 10-7 |
| 10.4.1.3 | Chip-Select Control Registers (CSCR0–CSCR7) | 10-8 |
| 10.4.1.4 | Code Example | 10-9 |


## Chapter 11
## Synchronous/Asynchronous DRAM Controller Module

# CONTENTS


| Paragraph Number | Title | Page Number |
|---|---|---|
| 11.1 | Overview | 11-1 |
| 11.1.1 | Definitions | 11-2 |
| 11.1.2 | Block Diagram and Major Components | 11-2 |
| 11.2 | DRAM Controller Operation | 11-3 |
| 11.2.1 | DRAM Controller Registers | 11-3 |
| 11.3 | Asynchronous Operation | 11-4 |
| 11.3.1 | DRAM Controller Signals in Asynchronous Mode | 11-4 |
| 11.3.2 | Asynchronous Register Set | 11-4 |
| 11.3.2.1 | DRAM Control Register (DCR) in Asynchronous Mode | 11-4 |
| 11.3.2.2 | DRAM Address and Control Registers (DACR0/DACR1) | 11-5 |
| 11.3.2.3 | DRAM Controller Mask Registers (DMR0/DMR1) | 11-7 |
| 11.3.3 | General Asynchronous Operation Guidelines | 11-8 |
| 11.3.3.1 | Non-Page-Mode Operation | 11-11 |
| 11.3.3.2 | Burst Page-Mode Operation | 11-12 |
| 11.3.3.3 | Continuous Page Mode | 11-13 |
| 11.3.3.4 | Extended Data Out (EDO) Operation | 11-15 |
| 11.3.3.5 | Refresh Operation | 11-16 |
| 11.4 | Synchronous Operation | 11-16 |
| 11.4.1 | DRAM Controller Signals in Synchronous Mode | 11-17 |
| 11.4.2 | Using Edge Select (EDGESEL) | 11-18 |
| 11.4.3 | Synchronous Register Set | 11-19 |
| 11.4.3.1 | DRAM Control Register (DCR) in Synchronous Mode | 11-19 |
| 11.4.3.2 | DRAM Address and Control Registers (DACR0/DACR1) in Synchronous Mode | 11-20 |
| 11.4.3.3 | DRAM Controller Mask Registers (DMR0/DMR1) | 11-22 |
| 11.4.4 | General Synchronous Operation Guidelines | 11-23 |
| 11.4.4.1 | Address Multiplexing | 11-23 |
| 11.4.4.2 | Interfacing Example | 11-27 |
| 11.4.4.3 | Burst Page Mode | 11-27 |
| 11.4.4.4 | Continuous Page Mode | 11-29 |
| 11.4.4.5 | Auto-Refresh Operation | 11-31 |
| 11.4.4.6 | Self-Refresh Operation | 11-32 |
| 11.4.5 | Initialization Sequence | 11-32 |
| 11.4.5.1 | Mode Register Settings | 11-33 |
| 11.5 | SDRAM Example | 11-34 |
| 11.5.1 | SDRAM Interface Configuration | 11-34 |
| 11.5.2 | DCR Initialization | 11-35 |
| 11.5.3 | DACR Initialization | 11-35 |
| 11.5.4 | DMR Initialization | 11-37 |
| 11.5.5 | Mode Register Initialization | 11-38 |
| 11.5.6 | Initialization Code | 11-39 |

# CONTENTS


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

**Part III**
**Peripheral Module**

**Chapter 12**
**DMA Controller Module**

| | | |
|---|---|---|
| 12.1 | Overview | 12-1 |
| 12.1.1 | DMA Module Features | 12-2 |
| 12.2 | DMA Signal Description | 12-2 |
| 12.3 | DMA Transfer Overview | 12-4 |
| 12.4 | DMA Controller Module Programming Model | 12-5 |
| 12.4.1 | Source Address Registers (SAR0–SAR3) | 12-7 |
| 12.4.2 | Destination Address Registers (DAR0–DAR3) | 12-7 |
| 12.4.3 | Byte Count Registers (BCR0–BCR3) | 12-7 |
| 12.4.4 | DMA Control Registers (DCR0–DCR3) | 12-8 |
| 12.4.5 | DMA Status Registers (DSR0–DSR3) | 12-10 |
| 12.4.6 | DMA Interrupt Vector Registers (DIVR0–DIVR3) | 12-11 |
| 12.5 | DMA Controller Module Functional Description | 12-11 |
| 12.5.1 | Transfer Requests (Cycle-Steal and Continuous Modes) | 12-12 |
| 12.5.2 | Data Transfer Modes | 12-12 |
| 12.5.2.1 | Dual-Address Transfers | 12-12 |
| 12.5.2.2 | Single-Address Transfers | 12-13 |
| 12.5.3 | Channel Initialization and Startup | 12-13 |
| 12.5.3.1 | Channel Prioritization | 12-13 |
| 12.5.3.2 | Programming the DMA Controller Module | 12-13 |
| 12.5.4 | Data Transfer | 12-14 |
| 12.5.4.1 | External Request and Acknowledge Operation | 12-14 |
| 12.5.4.2 | Auto-Alignment | 12-17 |
| 12.5.4.3 | Bandwidth Control | 12-18 |
| 12.5.5 | Termination | 12-18 |

**Chapter 13**
**Timer Module**

| | | |
|---|---|---|
| 13.1 | Overview | 13-1 |
| 13.1.1 | Key Features | 13-2 |
| 13.2 | General-Purpose Timer Units | 13-2 |
| 13.3 | General-Purpose Timer Programming Model | 13-2 |
| 13.3.1 | Timer Mode Registers (TMR0/TMR1) | 13-3 |
| 13.3.2 | Timer Reference Registers (TRR0/TRR1) | 13-4 |
| 13.3.3 | Timer Capture Registers (TCR0/TCR1) | 13-4 |
| 13.3.4 | Timer Counters (TCN0/TCN1) | 13-5 |

# CONTENTS


| Paragraph Number | Title | Page Number |
|---|---|---|
| 13.3.5 | Timer Event Registers (TER0/TER1) | 13-5 |
| 13.4 | Code Example | 13-6 |
| 13.5 | Calculating Time-Out Values | 13-7 |


## Chapter 14
## UART Modules


| | | |
|---|---|---|
| 14.1 | Overview | 14-1 |
| 14.2 | Serial Module Overview | 14-2 |
| 14.3 | Register Descriptions | 14-3 |
| 14.3.1 | UART Mode Registers 1 (UMR1n) | 14-5 |
| 14.3.2 | UART Mode Register 2 (UMR2n) | 14-7 |
| 14.3.3 | Rx FIFO Threshold Register (RXLVL) | 14-8 |
| 14.3.4 | Modem Control Register (MODCTL) | 14-9 |
| 14.3.5 | Tx FIFO Threshold Register (TXLVL) | 14-10 |
| 14.3.6 | UART Status Registers (USRn) | 14-10 |
| 14.3.7 | UART Clock-Select Registers (UCSRn) | 14-12 |
| 14.3.8 | Receive Samples Available Register (RSMP) | 14-12 |
| 14.3.9 | Transmit Space Available Register (TSPC) | 14-13 |
| 14.3.10 | UART Command Registers (UCRn) | 14-13 |
| 14.3.11 | UART Receiver Buffers (URBn) | 14-15 |
| 14.3.12 | UART Transmitter Buffers (UTBn) | 14-16 |
| 14.3.13 | UART Input Port Change Registers (UIPCRn) | 14-17 |
| 14.3.14 | UART Auxiliary Control Register (UACRn) | 14-17 |
| 14.3.15 | UART Interrupt Status/Mask Registers (UISRn/UIMRn) | 14-18 |
| 14.3.16 | UART Divider Upper/Lower Registers (UDUn/UDLn) | 14-19 |
| 14.3.17 | UART Interrupt Vector Register (UIVRn) | 14-20 |
| 14.3.18 | UART Input Port Register (UIPn) | 14-20 |
| 14.3.19 | UART Output Port Data Registers (UOP1n/UOP0n) | 14-21 |
| 14.4 | UART Module Signal Definitions | 14-21 |
| 14.5 | Operation | 14-23 |
| 14.5.1 | Transmitter/Receiver Clock Source | 14-23 |
| 14.5.1.1 | Programmable Divider | 14-24 |
| 14.5.1.2 | Calculating Baud Rates | 14-24 |
| 14.5.1.2.1 | CLKIN Baud Rates | 14-24 |
| 14.5.1.2.2 | External Clock | 14-25 |
| 14.5.2 | Transmitter and Receiver Operating Modes | 14-25 |
| 14.5.2.1 | Transmitting in UART Mode | 14-26 |
| 14.5.2.2 | Transmitter in Modem Mode (UART1) | 14-27 |
| 14.5.2.2.1 | AC ‘97 Low-Power Mode | 14-29 |
| 14.5.2.3 | Receiver | 14-29 |
| 14.5.2.4 | UART1 in UART Mode | 14-31 |

# CONTENTS


| Paragraph Number | Title | Page Number |
|---|---|---|
| 14.5.2.4.1 | Receiver in Modem Mode (UART1) | 14-31 |
| 14.5.2.5 | FIFO Stack in UART0 | 14-32 |
| 14.5.2.6 | FIFOs in UART1 | 14-33 |
| 14.5.3 | Looping Modes | 14-34 |
| 14.5.3.1 | Automatic Echo Mode | 14-34 |
| 14.5.3.2 | Local Loop-Back Mode | 14-34 |
| 14.5.3.3 | Remote Loop-Back Mode | 14-35 |
| 14.5.4 | Multidrop Mode | 14-35 |
| 14.5.5 | Bus Operation | 14-37 |
| 14.5.5.1 | Read Cycles | 14-37 |
| 14.5.5.2 | Write Cycles | 14-37 |
| 14.5.5.3 | Interrupt Acknowledge Cycles | 14-37 |
| 14.5.6 | Programming | 14-37 |
| 14.5.6.1 | UART Module Initialization Sequence | 14-38 |

**Chapter 15**
**Parallel Port (General-Purpose I/O)**

| | | |
|---|---|---|
| 15.1 | Parallel Port Operation | 15-1 |
| 15.1.1 | Pin Assignment Register (PAR) | 15-1 |
| 15.1.2 | Port A Data Direction Register (PADDR) | 15-2 |
| 15.1.3 | Port A Data Register (PADAT) | 15-2 |
| 15.1.4 | Code Example | 15-4 |

**Part IV**
**Hardware Interface**

**Chapter 16**
**Mechanical Data**

| | | |
|---|---|---|
| 16.1 | Package | 16-1 |
| 16.2 | Pinout | 16-1 |
| 16.3 | Mechanical Diagram | 16-8 |
| 16.4 | Case Drawing | 16-9 |

**Chapter 17**
**Signal Descriptions**

| | | |
|---|---|---|
| 17.1 | Overview | 17-1 |
| 17.2 | MCF5407 Bus Signals | 17-7 |
| 17.2.1 | Address Bus | 17-7 |

# CONTENTS


| Paragraph Number | Title | Page Number |
|---|---|---|
| 17.2.1.1 | Address Bus (A[23:0]) | 17-7 |
| 17.2.1.2 | Address Bus (A[31:24]/PP[15:8]) | 17-7 |
| 17.2.2 | Data Bus (D[31:0]) | 17-8 |
| 17.2.3 | Read/Write (R/$\overline{W}$) | 17-8 |
| 17.2.4 | Size (SIZ[1:0]) | 17-8 |
| 17.2.5 | Transfer Start ($\overline{TS}$) | 17-9 |
| 17.2.6 | Address Strobe (AS) | 17-9 |
| 17.2.7 | Transfer Acknowledge ($\overline{TA}$) | 17-9 |
| 17.2.8 | Transfer In Progress ($\overline{TIP}$/PP7) | 17-10 |
| 17.2.9 | Transfer Type (TT[1:0]/PP[1:0]) | 17-10 |
| 17.2.10 | Transfer Modifier (TM[2:0]/PP[4:2]/DACK[1:0]) | 17-10 |
| 17.3 | Interrupt Control Signals | 17-12 |
| 17.3.1 | Interrupt Request ($\overline{IRQ1}$/$\overline{IRQ2}$, $\overline{IRQ3}$/$\overline{IRQ6}$, $\overline{IRQ5}$/$\overline{IRQ4}$, and $\overline{IRQ7}$) | 17-12 |
| 17.4 | Bus Arbitration Signals | 17-12 |
| 17.4.1 | Bus Request ($\overline{BR}$) | 17-12 |
| 17.4.2 | Bus Grant ($\overline{BG}$) | 17-12 |
| 17.4.3 | Bus Driven (BD) | 17-13 |
| 17.5 | Clock and Reset Signals | 17-13 |
| 17.5.1 | Reset In ($\overline{RSTI}$) | 17-13 |
| 17.5.2 | Clock Input (CLKIN) | 17-13 |
| 17.5.3 | Bus Clock Output (BCLKO) | 17-13 |
| 17.5.4 | Reset Out (RSTO) | 17-13 |
| 17.5.5 | Data/Configuration Pins (D[7:0]) | 17-14 |
| 17.5.5.1 | D[7:5,3]—Boot Chip-Select ($\overline{CS0}$) Configuration | 17-14 |
| 17.5.5.2 | D7—Auto Acknowledge Configuration (AA_CONFIG) | 17-14 |
| 17.5.5.3 | D[6:5]—Port Size Configuration (PS_CONFIG[1:0]) | 17-14 |
| 17.5.5.4 | D3—Byte-Enable Configuration (BE_CONFIG) | 17-15 |
| 17.5.6 | D4—Address Configuration (ADDR_CONFIG) | 17-15 |
| 17.5.6.1 | D[2:0]—Divide Control (DIVIDE[2:0]) | 17-15 |
| 17.6 | Chip-Select Module Signals | 17-15 |
| 17.6.1 | Chip-Select (CS[7:0]) | 17-15 |
| 17.6.2 | Byte Enables/Byte Write Enables (BE[3:0]/BWE[3:0]) | 17-16 |
| 17.6.3 | Output Enable (OE) | 17-16 |
| 17.7 | DRAM Controller Signals | 17-16 |
| 17.7.1 | Row Address Strobes (RAS[1:0]) | 17-16 |
| 17.7.2 | Column Address Strobes (CAS[3:0]) | 17-16 |
| 17.7.3 | DRAM Write (DRAMW) | 17-16 |
| 17.7.4 | Synchronous DRAM Column Address Strobe (SCAS) | 17-17 |
| 17.7.5 | Synchronous DRAM Row Address Strobe (SRAS) | 17-17 |
| 17.7.6 | Synchronous DRAM Clock Enable (SCKE) | 17-17 |
| 17.7.7 | Synchronous Edge Select (EDGESEL) | 17-17 |
| 17.8 | DMA Controller Module Signals | 17-17 |
| 17.8.1 | DMA Request (DREQ[1:0]/PP[6:5]) | 17-17 |

# CONTENTS


| Paragraph Number | Title | Page Number |
|---|---|---|
| 17.8.2 | Transfer Modifier/DMA Acknowledge (TM[2:0]/DACK[1:0]) | 17-18 |
| 17.9 | Serial Module Signals | 17-18 |
| 17.9.1 | Transmitter Serial Data Output (TxD) | 17-18 |
| 17.9.2 | Receiver Serial Data Input (RxD) | 17-19 |
| 17.9.3 | Clear to Send (CTS) | 17-19 |
| 17.9.4 | Request to Send (RTS) | 17-19 |
| 17.10 | Timer Module Signals | 17-19 |
| 17.10.1 | Timer Inputs (TIN[1:0]) | 17-19 |
| 17.10.2 | Timer Outputs (TOUT1, TOUT0) | 17-19 |
| 17.11 | Parallel I/O Port (PP[15:0]) | 17-19 |
| 17.12 | $I^2C$ Module Signals | 17-20 |
| 17.12.1 | $I^2C$ Serial Clock (SCL) | 17-20 |
| 17.12.2 | $I^2C$ Serial Data (SDA) | 17-20 |
| 17.13 | Debug and Test Signals | 17-20 |
| 17.13.1 | Test Mode (MTMOD[3:0]) | 17-20 |
| 17.13.2 | High Impedance ($\overline{HIZ}$) | 17-20 |
| 17.13.3 | Processor Clock Output (PSTCLK) | 17-20 |
| 17.13.4 | Processor Status Debug Data (PSTDDATA[7:0]) | 17-21 |
| 17.14 | Debug Module/JTAG Signals | 17-21 |
| 17.14.1 | Test Reset/Development Serial Clock ($\overline{TRST}$/DSCLK) | 17-21 |
| 17.14.2 | Test Mode Select/Breakpoint (TMS/$\overline{BKPT}$) | 17-21 |
| 17.14.3 | Test Data Input/Development Serial Input (TDI/DSI) | 17-22 |
| 17.14.4 | Test Data Output/Development Serial Output (TDO/DSO) | 17-22 |
| 17.14.5 | Test Clock (TCK) | 17-22 |


## Chapter 18
## Bus Operation


| | | |
|---|---|---|
| 18.1 | Features | 18-1 |
| 18.2 | Bus and Control Signals | 18-1 |
| 18.3 | Bus Characteristics | 18-2 |
| 18.4 | Data Transfer Operation | 18-2 |
| 18.4.1 | Bus Cycle Execution | 18-4 |
| 18.4.2 | Data Transfer Cycle States | 18-5 |
| 18.4.3 | Read Cycle | 18-7 |
| 18.4.4 | Write Cycle | 18-8 |
| 18.4.5 | Fast-Termination Cycles | 18-9 |
| 18.4.6 | Back-to-Back Bus Cycles | 18-10 |
| 18.4.7 | Burst Cycles | 18-11 |
| 18.4.7.1 | Line Transfers | 18-12 |
| 18.4.7.2 | Line Read Bus Cycles | 18-12 |
| 18.4.7.3 | Line Write Bus Cycles | 18-14 |

# CONTENTS


| Paragraph Number | Title | Page Number |
|---|---|---|
| 18.4.7.4 | Transfers Using Mixed Port Sizes | 18-15 |
| 18.5 | Misaligned Operands | 18-16 |
| 18.6 | Bus Errors | 18-17 |
| 18.7 | Interrupt Exceptions | 18-17 |
| 18.7.1 | Level 7 Interrupts | 18-18 |
| 18.7.2 | Interrupt-Acknowledge Cycle | 18-19 |
| 18.8 | Bus Arbitration | 18-20 |
| 18.8.1 | Bus Arbitration Signals | 18-21 |
| 18.9 | General Operation of External Master Transfers | 18-21 |
| 18.9.1 | Two-Device Bus Arbitration Protocol (Two-Wire Mode) | 18-25 |
| 18.9.2 | Multiple External Bus Device Arbitration Protocol (Three-Wire Mode) | 18-29 |
| 18.10 | Reset Operation | 18-33 |
| 18.10.1 | Master Reset | 18-34 |
| 18.10.2 | Software Watchdog Reset | 18-35 |


## Chapter 19
## IEEE 1149.1 Test Access Port (JTAG)


| | | |
|---|---|---|
| 19.1 | Overview | 19-1 |
| 19.2 | JTAG Signal Descriptions | 19-2 |
| 19.3 | TAP Controller | 19-3 |
| 19.4 | JTAG Register Descriptions | 19-4 |
| 19.4.1 | JTAG Instruction Shift Register | 19-5 |
| 19.4.2 | IDCODE Register | 19-6 |
| 19.4.3 | JTAG Boundary-Scan Register | 19-7 |
| 19.4.4 | JTAG Bypass Register | 19-10 |
| 19.5 | Restrictions | 19-10 |
| 19.6 | Disabling IEEE Standard 1149.1 Operation | 19-10 |
| 19.7 | Obtaining the IEEE Standard 1149.1 | 19-11 |


## Chapter 20
## Electrical Specifications


| | | |
|---|---|---|
| 20.1 | General Parameters | 20-1 |
| 20.1.1 | Supply Voltage Sequencing and Separation Cautions | 20-3 |
| 20.2 | Clock Timing Specifications | 20-4 |
| 20.3 | Input/Output AC Timing Specifications | 20-6 |
| 20.4 | Reset Timing Specifications | 20-15 |
| 20.5 | Debug AC Timing Specifications | 20-16 |
| 20.6 | Timer Module AC Timing Specifications | 20-17 |
| 20.7 | $I^2C$ Input/Output Timing Specifications | 20-18 |
| 20.8 | UART Module AC Timing Specifications | 20-19 |

# CONTENTS


| Paragraph Number | Title | Page Number |
|---|---|---|
| 20.9 | Parallel Port (General-Purpose I/O) Timing Specifications | 20-22 |
| 20.10 | DMA Timing Specifications | 20-23 |
| 20.11 | IEEE 1149.1 (JTAG) AC Timing Specifications | 20-24 |

**Appendix A**
**Migrating from the ColdFire MCF5307 to the MCF5407**

| | | |
|---|---|---|
| A.1 | Overview | A-1 |
| A.2 | Instruction Set Additions | A-2 |
| A.3 | Enhanced Memories | A-3 |
| A.4 | On-Chip DMA Modifications | A-4 |
| A.5 | UART Enhancements | A-5 |
| A.6 | Timing Differences | A-6 |
| A.6.1 | Phase-Locked Loop (PLL) | A-6 |
| A.6.2 | Timing Relationships | A-7 |
| A.7 | Reset Initialization Modifications | A-8 |
| A.8 | Revision C Debug | A-10 |
| A.8.1 | Debug Interrupts and Interrupt Requests in Emulator Mode | A-10 |
| A.8.2 | On-Chip Breakpoint Registers | A-12 |
| A.8.2.1 | Write Debug Module Register (wdmreg) | A-12 |
| A.8.3 | Debug Programming Model | A-14 |
| A.8.3.1 | Address Breakpoint 1 Registers (ABLR1, ABHR1) | A-14 |
| A.8.3.2 | Address Attribute Breakpoint Register 1 (AATR1) | A-14 |
| A.8.3.3 | Program Counter Breakpoint Registers 1–3 (PBR1–PBR3) | A-14 |
| A.8.3.4 | Data Breakpoint Register 1 (DBR1, DBMR1) | A-15 |
| A.8.3.5 | Extended Trigger Definition Register (XTDR) | A-15 |
| A.8.4 | Debug Interrupt Exception Vectors | A-15 |
| A.8.5 | Processor Status and Debug Data Output Signals | A-16 |
| A.8.6 | Debug C Summary | A-17 |
| A.9 | Voltage Input Changes | A-17 |
| A.10 | PLL Power Supply Filter Circuit | A-18 |
| A.11 | Pin-Assignment Compatibility | A-18 |

**Appendix B**
**List of Memory Maps**

# CONTENTS


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

# ILLUSTRATIONS


| Figure Number | Title | Page Number |
|---|---|---|
| 1-1 | MCF5407 Block Diagram | 1-2 |
| 1-2 | UART Module Block Diagram | 1-10 |
| 1-3 | PLL Module | 1-13 |
| 1-4 | ColdFire MCF5407 Programming Model | 1-15 |
| 2-1 | ColdFire Enhanced Pipeline | 2-3 |
| 2-2 | ColdFire Multiply-Accumulate Functionality Diagram | 2-5 |
| 2-3 | ColdFire Programming Model | 2-8 |
| 2-4 | Condition Code Register (CCR) | 2-9 |
| 2-5 | Status Register (SR) | 2-11 |
| 2-6 | Vector Base Register (VBR) | 2-12 |
| 2-7 | Organization of Integer Data Formats in Data Registers | 2-13 |
| 2-8 | Organization of Integer Data Formats in Address Registers | 2-14 |
| 2-9 | Memory Operand Addressing | 2-14 |
| 2-1 | Exception Stack Frame Form | 2-33 |
| 3-1 | ColdFire MAC Multiplication and Accumulation | 3-2 |
| 3-2 | MAC Programming Model | 3-2 |
| 4-1 | SRAM Base Address Registers (RAMBARn) | 4-3 |
| 4-2 | Data Cache Organization | 4-7 |
| 4-3 | Data Cache Organization and Line Format | 4-8 |
| 4-4 | Data Cache—A: at Reset, B: after Invalidation, C and D: Loading Pattern | 4-10 |
| 4-5 | Data Caching Operation | 4-11 |
| 4-6 | Write-Miss in Copyback Mode | 4-16 |
| 4-7 | Data Cache Locking | 4-20 |
| 4-8 | Cache Control Register (CACR) | 4-21 |
| 4-9 | Access Control Register Format (ACRn) | 4-24 |
| 4-10 | An Format (Data Cache) | 4-25 |
| 4-11 | An Format (Instruction Cache) | 4-25 |
| 4-12 | Instruction Cache Line State Diagram | 4-27 |
| 4-13 | Data Cache Line State Diagram—Copyback Mode | 4-28 |
| 4-14 | Data Cache Line State Diagram—Write-Through Mode | 4-29 |
| 5-1 | Processor/Debug Module Interface | 5-1 |
| 5-2 | PSTCLK Timing | 5-3 |
| 5-3 | PSTDDATA: Single-Cycle Instruction Timing | 5-3 |
| 5-4 | Example JMP Instruction Output on PSTDDATA | 5-6 |
| 5-5 | Debug Programming Model | 5-9 |
| 5-6 | Address Attribute Trigger Registers (AATR, AATR1) | 5-11 |

# ILLUSTRATIONS


| Figure Number | Title | Page Number |
|---|---|---|
| 5-7 | Address Breakpoint Registers (ABLR, ABHR, ABLR1, ABHR1) | 5-12 |
| 5-8 | BDM Address Attribute Register (BAAR) | 5-13 |
| 5-9 | Configuration/Status Register (CSR) | 5-13 |
| 5-10 | Data Breakpoint/Mask Registers (DBR/DBR1 and DBMR/DBMR1) | 5-16 |
| 5-11 | Program Counter Breakpoint Registers (PBR, PBR1, PBR2, PBR3) | 5-17 |
| 5-12 | Program Counter Breakpoint Mask Register (PBMR) | 5-17 |
| 5-13 | Trigger Definition Register (TDR) | 5-18 |
| 5-14 | Extended Trigger Definition Register (XTDR) | 5-20 |
| 5-15 | BDM Serial Interface Timing | 5-24 |
| 5-16 | Receive BDM Packet | 5-25 |
| 5-17 | Transmit BDM Packet | 5-26 |
| 5-18 | BDM Command Format | 5-27 |
| 5-19 | Command Sequence Diagram | 5-29 |
| 5-21 | RAREG/RDREG Command Sequence | 5-30 |
| 5-20 | RAREG/RDREG Command Format | 5-30 |
| 5-23 | WAREG/WDREG Command Sequence | 5-31 |
| 5-22 | WAREG/WDREG Command Format | 5-31 |
| 5-25 | READ Command Sequence | 5-32 |
| 5-24 | read Command/Result Formats | 5-32 |
| 5-26 | WRITE Command Format | 5-33 |
| 5-27 | WRITE Command Sequence | 5-34 |
| 5-28 | DUMP Command/Result Formats | 5-35 |
| 5-29 | DUMP Command Sequence | 5-36 |
| 5-30 | FILL Command Format | 5-37 |
| 5-31 | FILL Command Sequence | 5-38 |
| 5-33 | GO Command Sequence | 5-39 |
| 5-32 | GO Command Format | 5-39 |
| 5-35 | NOP Command Sequence | 5-40 |
| 5-34 | NOP Command Format | 5-40 |
| 5-37 | SYNC_PC Command Sequence | 5-41 |
| 5-36 | SYNC_PC Command Format | 5-41 |
| 5-39 | RCREG Command Sequence | 5-42 |
| 5-38 | RCREG Command/Result Formats | 5-42 |
| 5-41 | WCREG Command Sequence | 5-43 |
| 5-40 | WCREG Command/Result Formats | 5-43 |
| 5-43 | RDMREG Command Sequence | 5-44 |
| 5-42 | RDMREG bdm Command/Result Formats | 5-44 |
| 5-45 | WDMREG Command Sequence | 5-45 |
| 5-44 | WDMREG BDM Command Format | 5-45 |
| 5-46 | Recommended BDM Connector | 5-49 |
| 6-1 | SIM Block Diagram | 6-1 |
| 6-2 | Module Base Address Register (MBAR) | 6-4 |
| 6-3 | Reset Status Register (RSR) | 6-5 |

# ILLUSTRATIONS


| Figure Number | Title | Page Number |
|---|---|---|
| 6-4 | MCF5407 Embedded System Recovery from Unterminated Access | 6-7 |
| 6-5 | System Protection Control Register (SYPCR) | 6-8 |
| 6-6 | Software Watchdog Interrupt Vector Register (SWIVR) | 6-9 |
| 6-7 | Software Watchdog Service Register (SWSR) | 6-9 |
| 6-8 | Pin Assignment Register (PAR) | 6-10 |
| 6-9 | Default Bus Master Register (MPARK) | 6-11 |
| 6-10 | Round Robin Arbitration (PARK = 00) | 6-12 |
| 6-11 | Park on Master Core Priority (PARK = 01) | 6-13 |
| 6-12 | Park on DMA Module Priority (PARK = 10) | 6-13 |
| 6-13 | Park on Current Master Priority (PARK = 01) | 6-14 |
| 7-1 | PLL Module Block Diagram | 7-1 |
| 7-2 | PLL Control Register (PLLCR) | 7-3 |
| 7-3 | CLKIN, PCLK, PSTCLK, and BCLKO Timing | 7-5 |
| 7-4 | Reset and Initialization Timing | 7-6 |
| 7-5 | PLL Power Supply Filter Circuit | 7-6 |
| 8-1 | $I^2C$ Module Block Diagram | 8-2 |
| 8-2 | $I^2C$ Standard Communication Protocol | 8-3 |
| 8-3 | Repeated START | 8-4 |
| 8-4 | Synchronized Clock SCL | 8-5 |
| 8-5 | $I^2C$ Address Register (IADR) | 8-6 |
| 8-6 | $I^2C$ Frequency Divider Register (IFDR) | 8-7 |
| 8-7 | $I^2C$ Control Register (I2CR) | 8-8 |
| 8-8 | $I^2CR$ Status Register (I2SR) | 8-9 |
| 8-9 | $I^2C$ Data I/O Register (I2DR) | 8-10 |
| 8-10 | Flow-Chart of Typical I2C Interrupt Routine | 8-14 |
| 9-1 | Interrupt Controller Block Diagram | 9-1 |
| 9-2 | Interrupt Control Registers (ICR0–ICR9) | 9-3 |
| 9-3 | Autovector Register (AVR) | 9-5 |
| 9-4 | Interrupt Pending Register (IPR) and Interrupt Mask Register (IMR) | 9-7 |
| 9-5 | Interrupt Port Assignment Register (IRQPAR) | 9-7 |
| 10-1 | Connections for External Memory Port Sizes | 10-4 |
| 10-2 | Chip Select Address Registers (CSAR0–CSAR7) | 10-6 |
| 10-3 | Chip Select Mask Registers (CSMRn) | 10-7 |
| 10-4 | Chip-Select Control Registers (CSCR0–CSCR7) | 10-8 |
| 11-1 | Asynchronous/Synchronous DRAM Controller Block Diagram | 11-2 |
| 11-2 | DRAM Control Register (DCR) (Asynchronous Mode) | 11-5 |
| 11-3 | DRAM Address and Control Registers (DACR0/DACR1) | 11-6 |
| 11-4 | DRAM Controller Mask Registers (DMR0 and DMR1) | 11-7 |
| 11-5 | Basic Non-Page-Mode Operation RCD = 0, RNCN = 1 (4-4-4-4) | 11-11 |
| 11-6 | Basic Non-Page-Mode Operation RCD = 1, RNCN = 0 (5-5-5-5) | 11-12 |
| 11-7 | Burst Page-Mode Read Operation (4-3-3-3) | 11-13 |
| 11-8 | Burst Page-Mode Write Operation (4-3-3-3) | 11-13 |
| 11-9 | Continuous Page-Mode Operation | 11-14 |

# ILLUSTRATIONS


| Figure Number | Title | Page Number |
|---|---|---|
| 11-10 | Write Hit in Continuous Page Mode | 11-15 |
| 11-11 | EDO Read Operation (3-2-2-2) | 11-15 |
| 11-12 | DRAM Access Delayed by Refresh | 11-16 |
| 11-13 | MCF5407 SDRAM Interface | 11-18 |
| 11-14 | Using EDGESEL to Change Signal Timing | 11-19 |
| 11-15 | DRAM Control Register (DCR) (Synchronous Mode) | 11-19 |
| 11-16 | DACR0 and DACR1 Registers (Synchronous Mode) | 11-20 |
| 11-17 | DRAM Controller Mask Registers (DMR0 and DMR1) | 11-22 |
| 11-18 | Burst Read SDRAM Access | 11-28 |
| 11-19 | Burst Write SDRAM Access | 11-29 |
| 11-20 | Synchronous, Continuous Page-Mode Access—Consecutive Reads | 11-30 |
| 11-21 | Synchronous, Continuous Page-Mode Access—Read after Write | 11-31 |
| 11-22 | Auto-Refresh Operation | 11-32 |
| 11-23 | Self-Refresh Operation | 11-32 |
| 11-24 | Mode Register Set (mrs) Command | 11-34 |
| 11-25 | Initialization Values for DCR | 11-35 |
| 11-26 | SDRAM Configuration | 11-36 |
| 11-27 | DACR Register Configuration | 11-36 |
| 11-28 | DMR0 Register | 11-37 |
| 11-29 | Mode Register Mapping to MCF5407 A[31:0] | 11-38 |
| 12-1 | DMA Signal Diagram | 12-1 |
| 12-2 | MCF5307/MCF5407 TM[2:0] Pin Remapping | 12-4 |
| 12-3 | Dual-Address Transfer | 12-4 |
| 12-4 | Single-Address Transfers | 12-5 |
| 12-6 | Destination Address Registers (DARn) | 12-7 |
| 12-5 | Source Address Registers (SARn) | 12-7 |
| 12-7 | Byte Count Registers (BCRn) | 12-8 |
| 12-8 | DMA Control Registers (DCRn) | 12-8 |
| 12-9 | DMA Status Registers (DSRn) | 12-10 |
| 12-10 | DMA Interrupt Vector Registers (DIVRn) | 12-11 |
| 12-11 | DREQ Timing Constraints, Dual-Address DMA Transfer | 12-15 |
| 12-12 | Dual-Address, Peripheral-to-SDRAM, Lower-Priority DMA Transfer | 12-16 |
| 12-13 | Single-Address DMA Transfer | 12-17 |
| 13-1 | Timer Block Diagram | 13-1 |
| 13-2 | Timer Mode Registers (TMR0/TMR1) | 13-3 |
| 13-3 | Timer Reference Registers (TRR0/TRR1) | 13-4 |
| 13-4 | Timer Capture Register (TCR0/TCR1) | 13-5 |
| 13-5 | Timer Counters (TCN0/TCN1) | 13-5 |
| 13-6 | Timer Event Registers (TER0/TER1) | 13-5 |
| 14-1 | Simplified Block Diagram | 14-1 |
| 14-2 | UART Mode Registers 1 (UMR1n) | 14-6 |
| 14-3 | UART Mode Register 2 (UMR2n) | 14-7 |
| 14-4 | Rx FIFO Threshold Register (RXLVL) | 14-8 |

# ILLUSTRATIONS


| Figure Number | Title | Page Number |
|---|---|---|
| 14-5 | Modem Control Register (MODCTL) | 14-9 |
| 14-6 | Tx FIFO Threshold Register (TXLVL) | 14-10 |
| 14-7 | UART Status Register (USRn) | 14-10 |
| 14-8 | UART Clock-Select Register (UCSRn) | 14-12 |
| 14-9 | Receive Samples Available Register (RSMP) | 14-13 |
| 14-10 | Tx Space Available Register (TSPC) | 14-13 |
| 14-11 | UART Command Register (UCRn) | 14-14 |
| 14-12 | UART Receiver Buffer for UART0 (URB0) | 14-16 |
| 14-13 | UART Receiver Buffer for UART1 (URB1) | 14-16 |
| 14-14 | UART Transmitter Buffer for UART0 (UTB0) | 14-16 |
| 14-15 | UART Transmitter Buffer for UART1 (UTB1) | 14-17 |
| 14-16 | UART Input Port Change Register (UIPCRn) | 14-17 |
| 14-17 | UART Auxiliary Control Register (UACRn) | 14-18 |
| 14-18 | UART Interrupt Status/Mask Registers (UISRn/UIMRn) | 14-18 |
| 14-19 | UART Divider Upper Register (UDUn) | 14-19 |
| 14-20 | UART Divider Lower Register (UDLn) | 14-19 |
| 14-21 | UART Interrupt Vector Register (UIVRn) | 14-20 |
| 14-22 | UART Input Port Register (UIPn) | 14-20 |
| 14-24 | UART Block Diagram Showing External and Internal Interface Signals | 14-21 |
| 14-23 | UART Output Port Data 1 Register (UOP1/UOP0) | 14-21 |
| 14-25 | UART/RS-232 Interface | 14-23 |
| 14-26 | UART1/CODEC Interface | 14-23 |
| 14-27 | UART1/AC ’97 Interface | 14-23 |
| 14-28 | Clocking Source Diagram | 14-24 |
| 14-29 | Transmitter and Receiver Functional Diagram | 14-25 |
| 14-30 | Transmitter Timing Diagram | 14-27 |
| 14-31 | 16-Bit CODEC Interface Timing (lsb First) | 14-27 |
| 14-32 | 8-Bit CODEC Interface Timing (msb First) | 14-28 |
| 14-33 | AC ‘97 Interface Timing | 14-28 |
| 14-34 | Receiver Timing | 14-30 |
| 14-35 | Automatic Echo | 14-34 |
| 14-36 | Local Loop-Back | 14-34 |
| 14-37 | Remote Loop-Back | 14-35 |
| 14-38 | Multidrop Mode Timing Diagram | 14-36 |
| 14-39 | UART Mode Programming Flowchart | 14-39 |
| 15-1 | Parallel Port Pin Assignment Register (PAR) | 15-1 |
| 15-2 | Port A Data Direction Register (PADDR) | 15-2 |
| 15-3 | Port A Data Register (PADAT) | 15-3 |
| 16-1 | Mechanical Diagram | 16-9 |
| 16-2 | MCF5407 Case Drawing (General View) | 16-10 |
| 16-3 | Case Drawing (Details) | 16-11 |
| 17-1 | MCF5407 Block Diagram with Signal Interfaces | 17-2 |
| 17-2 | MCF5307 to MCF5407 TM[2:0] Pin Remapping | 17-18 |

# ILLUSTRATIONS


| Figure Number | Title | Page Number |
|---|---|---|
| 18-1 | Signal Relationship to CLKIN for Non-DRAM Access | 18-2 |
| 18-2 | Connections for External Memory Port Sizes | 18-4 |
| 18-3 | Chip-Select Module Output Timing Diagram | 18-4 |
| 18-4 | Data Transfer State Transition Diagram | 18-6 |
| 18-5 | Read Cycle Flowchart | 18-7 |
| 18-6 | Basic Read Bus Cycle | 18-8 |
| 18-7 | Write Cycle Flowchart | 18-9 |
| 18-8 | Basic Write Bus Cycle | 18-9 |
| 18-9 | Read Cycle with Fast Termination | 18-10 |
| 18-10 | Write Cycle with Fast Termination | 18-10 |
| 18-11 | Back-to-Back Bus Cycles | 18-11 |
| 18-12 | Line Read Burst (2-1-1-1), External Termination | 18-12 |
| 18-13 | Line Read Burst (2-1-1-1), Internal Termination | 18-13 |
| 18-14 | Line Read Burst (3-2-2-2), External Termination | 18-13 |
| 18-15 | Line Read Burst-Inhibited, Fast, External Termination | 18-14 |
| 18-16 | Line Write Burst (2-1-1-1), Internal/External Termination | 18-14 |
| 18-17 | Line Write Burst (3-2-2-2) with One Wait State, Internal Termination | 18-15 |
| 18-18 | Line Write Burst-Inhibited, Internal Termination | 18-15 |
| 18-19 | Longword Read from an 8-Bit Port, External Termination | 18-16 |
| 18-20 | Longword Read from an 8-Bit Port, Internal Termination | 18-16 |
| 18-21 | Example of a Misaligned Longword Transfer (32-Bit Port) | 18-17 |
| 18-22 | Example of a Misaligned Word Transfer (32-Bit Port) | 18-17 |
| 18-23 | Interrupt-Acknowledge Cycle Flowchart | 18-20 |
| 18-24 | Basic No-Wait-State External Master Access | 18-22 |
| 18-25 | External Master Burst Line Access to 32-Bit Port | 18-24 |
| 18-26 | MCF5407 Two-Wire Mode Bus Arbitration Interface | 18-25 |
| 18-27 | Two-Wire Bus Arbitration with Bus Request Asserted | 18-26 |
| 18-28 | Two-Wire Implicit and Explicit Bus Mastership | 18-27 |
| 18-29 | MCF5407 Two-Wire Bus Arbitration Protocol State Diagram | 18-28 |
| 18-30 | Three-Wire Implicit and Explicit Bus Mastership | 18-30 |
| 18-31 | Three-Wire Bus Arbitration | 18-31 |
| 18-32 | Three-Wire Bus Arbitration Protocol State Diagram | 18-32 |
| 18-33 | Master Reset Timing | 18-34 |
| 18-34 | Software Watchdog Reset Timing | 18-35 |
| 19-1 | JTAG Test Logic Block Diagram | 19-2 |
| 19-2 | JTAG TAP Controller State Machine | 19-4 |
| 19-3 | IDCODE Register | 19-6 |
| 19-4 | Disabling JTAG in JTAG Mode | 19-11 |
| 19-5 | Disabling JTAG in Debug Mode | 19-11 |
| 20-1 | Supply Voltage Sequencing and Separation Cautions | 20-3 |
| 20-2 | Example Circuit to Control Supply Sequencing | 20-4 |
| 20-3 | CLKIN-to-Core Clock Frequency Ranges | 20-4 |
| 20-4 | Clock Timing | 20-5 |

# ILLUSTRATIONS


| Figure Number | Title | Page Number |
|---|---|---|
| 20-5 | PSTCLK Timing | 20-6 |
| 20-6 | AC Timings—Normal Read and Write Bus Cycles | 20-8 |
| 20-7 | SDRAM Read Cycle with EDGESEL Tied to Buffered CLKIN | 20-9 |
| 20-8 | SDRAM Write Cycle with EDGESEL Tied to Buffered CLKIN | 20-10 |
| 20-9 | SDRAM Read Cycle with EDGESEL Tied High | 20-11 |
| 20-10 | SDRAM Write Cycle with EDGESEL Tied High | 20-12 |
| 20-11 | SDRAM Read Cycle with EDGESEL Tied Low | 20-13 |
| 20-12 | SDRAM Write Cycle with EDGESEL Tied Low | 20-14 |
| 20-13 | AC Output Timing—High Impedance | 20-14 |
| 20-14 | Reset Timing | 20-15 |
| 20-15 | Real-Time Trace AC Timing | 20-16 |
| 20-16 | BDM Serial Port AC Timing | 20-16 |
| 20-17 | Timer Module AC Timing | 20-17 |
| 20-18 | $I^2C$ Input/Output Timings | 20-19 |
| 20-19 | UART0 and UART1 Module AC Timing—UART Mode | 20-20 |
| 20-20 | UART1 in 8- and 16-bit CODEC Mode | 20-21 |
| 20-21 | UART1 in AC ‘97 Mode | 20-21 |
| 20-22 | General-Purpose I/O Timing | 20-22 |
| 20-23 | DMA Timing | 20-23 |
| 20-24 | IEEE 1149.1 (JTAG) AC Timing | 20-25 |
| A-1 | MCF5307 to MCF5407 TM[2:0] Pin Remapping | A-5 |
| A-2 | Simplified Block Diagram | A-6 |
| A-3 | PLL Module | A-7 |
| A-4 | Exception Stack Frame Form | A-11 |
| A-5 | Write Debug Module Register Command (WDMREG) | A-12 |
| A-6 | WDMREG Command Sequence | A-13 |
| A-7 | PLL Power Supply Filter Circuit | A-18 |

# ILLUSTRATIONS

| Figure Number | Title | Page Number |
|---|---|---|

# TABLES


| Table Number | Title | Page Number |
|---|---|---|
| 1-1 | User-Level Registers | 1-15 |
| 1-2 | Supervisor-Level Registers | 1-16 |
| 2-1 | CCR Field Descriptions | 2-10 |
| 2-2 | MOVEC Register Map | 2-11 |
| 2-3 | Status Field Descriptions | 2-11 |
| 2-4 | Integer Data Formats | 2-13 |
| 2-5 | ColdFire Effective Addressing Modes | 2-15 |
| 2-6 | Notational Conventions | 2-16 |
| 2-7 | ColdFire ISA_B Extension Summary | 2-19 |
| 2-8 | User-Level Instruction Set Summary | 2-19 |
| 2-9 | Supervisor-Level Instruction Set Summary | 2-23 |
| 2-10 | Misaligned Operand References | 2-24 |
| 2-11 | Move Byte and Word Execution Times | 2-25 |
| 2-12 | Move Long Execution Times | 2-25 |
| 2-13 | Miscellaneous Move Execution Times | 2-26 |
| 2-14 | One-Operand Instruction Execution Times | 2-27 |
| 2-15 | Two-Operand Instruction Execution Times | 2-27 |
| 2-16 | Miscellaneous Instruction Execution Times | 2-29 |
| 2-17 | Branch Instruction Execution Times | 2-30 |
| 2-18 | Bcc Instruction Execution Times | 2-30 |
| 2-19 | Exception Vector Assignments | 2-32 |
| 2-20 | Format Field Encoding | 2-33 |
| 2-21 | Fault Status Encodings | 2-33 |
| 2-22 | MCF5407 Exceptions | 2-34 |
| 3-1 | MAC Instruction Summary | 3-4 |
| 3-2 | Two-Operand MAC Instruction Execution Times | 3-5 |
| 3-3 | MAC Move Instruction Execution Times | 3-6 |
| 4-1 | RAMBARn Field Description | 4-3 |
| 4-2 | Examples of Typical RAMBAR Settings | 4-6 |
| 4-3 | Valid and Modified Bit Settings | 4-8 |
| 4-4 | CACR Field Descriptions | 4-21 |
| 4-5 | ACRn Field Descriptions | 4-24 |
| 4-6 | Instruction Cache Line State Transitions | 4-27 |
| 4-7 | Data Cache Line State Transitions | 4-29 |
| 4-8 | Data Cache Line State Transitions (Current State Invalid) | 4-30 |
| 4-9 | Data Cache Line State Transitions (Current State Valid) | 4-31 |

# TABLES


| Table Number | Title | Page Number |
|---|---|---|
| 4-10 | Data Cache Line State Transitions (Current State Modified) | 4-31 |
| 5-1 | Debug Module Signals | 5-2 |
| 5-2 | PSTDDATA: Sequential Execution of Single-Cycle Instructions | 5-3 |
| 5-3 | PSTDDATA: Data Operand Captured | 5-4 |
| 5-4 | Processor Status Encoding | 5-5 |
| 5-5 | 0xE Status Posting | 5-7 |
| 5-6 | BDM/Breakpoint Registers | 5-9 |
| 5-7 | AATR and AATR1 Field Descriptions | 5-11 |
| 5-8 | ABLR and ABLR1 Field Description | 5-12 |
| 5-9 | ABHR and ABHR1 Field Description | 5-12 |
| 5-10 | BAAR Field Descriptions | 5-13 |
| 5-11 | CSR Field Descriptions | 5-14 |
| 5-12 | DBRn Field Descriptions | 5-16 |
| 5-13 | DBMRn Field Descriptions | 5-16 |
| 5-14 | Access Size and Operand Data Location | 5-16 |
| 5-15 | PBR, PBR1, PBR2, PBR3 Field Descriptions | 5-17 |
| 5-16 | PBMR Field Descriptions | 5-17 |
| 5-17 | TDR Field Descriptions | 5-19 |
| 5-18 | XTDR Field Descriptions | 5-20 |
| 5-19 | Receive BDM Packet Field Description | 5-25 |
| 5-20 | Transmit BDM Packet Field Description | 5-26 |
| 5-21 | BDM Command Summary | 5-26 |
| 5-22 | BDM Field Descriptions | 5-27 |
| 5-23 | Control Register Map | 5-42 |
| 5-24 | Definition of DRc Encoding—Read | 5-44 |
| 5-25 | PSTDDATA Nibble/CSR[BSTAT] Breakpoint Response | 5-46 |
| 5-26 | Exception Vector Assignments | 5-47 |
| 5-27 | PSTDDATA Specification for User-Mode Instructions | 5-50 |
| 5-28 | PSTDDATA Specification for Supervisor-Mode Instructions | 5-54 |
| 6-1 | SIM Registers | 6-3 |
| 6-2 | MBAR Field Descriptions | 6-5 |
| 6-3 | RSR Field Descriptions | 6-6 |
| 6-4 | SYPCR Field Descriptions | 6-8 |
| 6-5 | PLLIPL Settings | 6-10 |
| 6-6 | MPARK Field Descriptions | 6-11 |
| 7-1 | Divide Ratio Encodings | 7-2 |
| 7-2 | PLLCR Field Descriptions | 7-3 |
| 7-3 | PLL Module Input SIgnals | 7-4 |
| 7-4 | PLL Module Output Signals | 7-4 |
| 8-1 | $I^2C$ Interface Memory Map | 8-6 |
| 8-2 | $I^2C$ Address Register Field Descriptions | 8-6 |
| 8-3 | IFDR Field Descriptions | 8-7 |
| 8-4 | $I^2CR$ Field Descriptions | 8-8 |

# TABLES


| Table Number | Title | Page Number |
|---|---|---|
| 8-5 | I2SR Field Descriptions | 8-9 |
| 9-1 | Interrupt Controller Registers | 9-2 |
| 9-2 | Interrupt Control Registers | 9-2 |
| 9-3 | ICRn Field Descriptions | 9-3 |
| 9-4 | Interrupt Priority Scheme | 9-4 |
| 9-5 | AVR Field Descriptions | 9-6 |
| 9-6 | Autovector Register Bit Assignments | 9-6 |
| 9-7 | IPR and IMR Field Descriptions | 9-7 |
| 9-8 | IRQPAR Field Descriptions | 9-8 |
| 10-1 | Chip-Select Module Signals | 10-1 |
| 10-2 | Byte Enables/Byte Write Enable Signal Settings | 10-2 |
| 10-3 | Accesses by Matches in CSCRs and DACRs | 10-3 |
| 10-4 | D7/AA, Automatic Acknowledge of Boot CS0 | 10-4 |
| 10-5 | D[6:5]/PS[1:0], Port Size of Boot CS0 | 10-5 |
| 10-6 | D3/BE_CONFIG0, $\overline{BE}$[3:0] Boot Configuration | 10-5 |
| 10-7 | Chip-Select Registers | 10-5 |
| 10-8 | CSARn Field Description | 10-7 |
| 10-9 | CSMRn Field Descriptions | 10-7 |
| 10-10 | CSCRn Field Descriptions | 10-8 |
| 11-1 | DRAM Controller Registers | 11-3 |
| 11-2 | SDRAM Signal Summary | 11-4 |
| 11-3 | DCR Field Descriptions (Asynchronous Mode) | 11-5 |
| 11-4 | DACR0/DACR1 Field Description | 11-6 |
| 11-5 | DMR0/DMR1 Field Descriptions | 11-7 |
| 11-6 | Generic Address Multiplexing Scheme | 11-8 |
| 11-7 | DRAM Addressing for Byte-Wide Memories | 11-10 |
| 11-8 | DRAM Addressing for 16-Bit Wide Memories | 11-10 |
| 11-9 | DRAM Addressing for 32-Bit Wide Memories | 11-11 |
| 11-10 | SDRAM Commands | 11-17 |
| 11-11 | Synchronous DRAM Signal Connections | 11-17 |
| 11-12 | DCR Field Descriptions (Synchronous Mode) | 11-19 |
| 11-13 | DACR0/DACR1 Field Descriptions (Synchronous Mode) | 11-21 |
| 11-14 | DMR0/DMR1 Field Descriptions | 11-23 |
| 11-15 | MCF5407 to SDRAM Interface (8-Bit Port, 9-Column Address Lines) | 11-24 |
| 11-16 | MCF5407 to SDRAM Interface (8-Bit Port,10-Column Address Lines) | 11-24 |
| 11-17 | MCF5407 to SDRAM Interface (8-Bit Port,11-Column Address Lines) | 11-24 |
| 11-18 | MCF5407 to SDRAM Interface (8-Bit Port,12-Column Address Lines) | 11-24 |
| 11-19 | MCF5407 to SDRAM Interface (8-Bit Port,13-Column Address Lines) | 11-25 |
| 11-20 | MCF5407 to SDRAM Interface (16-Bit Port, 8-Column Address Lines) | 11-25 |
| 11-21 | MCF5407 to SDRAM Interface (16-Bit Port, 9-Column Address Lines) | 11-25 |
| 11-22 | MCF5407 to SDRAM Interface (16-Bit Port, 10-Column Address Lines) | 11-25 |
| 11-23 | MCF5407 to SDRAM Interface (16-Bit Port, 11-Column Address Lines) | 11-25 |
| 11-24 | MCF5407 to SDRAM Interface (16-Bit Port, 12-Column Address Lines) | 11-26 |

# TABLES


| Table Number | Title | Page Number |
|---|---|---|
| 11-25 | MCF5407 to SDRAM Interface (16-Bit Port, 13-Column-Address Lines) | 11-26 |
| 11-26 | MCF5407 to SDRAM Interface (32-Bit Port, 8-Column Address Lines) | 11-26 |
| 11-27 | MCF5407 to SDRAM Interface (32-Bit Port, 9-Column Address Lines) | 11-26 |
| 11-28 | MCF5407 to SDRAM Interface (32-Bit Port, 10-Column Address Lines) | 11-26 |
| 11-29 | MCF5407 to SDRAM Interface (32-Bit Port, 11-Column Address Lines) | 11-27 |
| 11-30 | MCF5407 to SDRAM Interface (32-Bit Port, 12-Column Address Lines) | 11-27 |
| 11-31 | SDRAM Hardware Connections | 11-27 |
| 11-32 | SDRAM Example Specifications | 11-34 |
| 11-33 | SDRAM Hardware Connections | 11-35 |
| 11-34 | DCR Initialization Values | 11-35 |
| 11-35 | DACR Initialization Values | 11-36 |
| 11-36 | DMR0 Initialization Values | 11-37 |
| 11-37 | Mode Register Initialization | 11-38 |
| 12-1 | DMA Signals | 12-2 |
| 12-2 | MCF5407 Signal Configurations for PP[4:2]/TM[2:0]/$\overline{\text{DACK}}$[1:0] | 12-3 |
| 12-3 | Memory Map for DMA Controller Module Registers | 12-6 |
| 12-4 | DCRn Field Descriptions | 12-8 |
| 12-5 | DSRn Field Descriptions | 12-10 |
| 13-1 | General-Purpose Timer Module Memory Map | 13-3 |
| 13-2 | TMRn Field Descriptions | 13-4 |
| 13-3 | TERn Field Descriptions | 13-6 |
| 13-4 | Time-Out Values (in Seconds)—TRR[REF] = 0xFFFF(162-MHz Processor Clock) | 13-7 |
| 14-1 | UART Module Programming Model | 14-4 |
| 14-2 | UMR1n Field Descriptions | 14-6 |
| 14-3 | UMR2n Field Descriptions | 14-7 |
| 14-4 | RXLVL Field Descriptions | 14-8 |
| 14-5 | Modem Control Register (MODCTL) Field Descriptions | 14-9 |
| 14-6 | TXLVL Field Descriptions | 14-10 |
| 14-7 | USRn Field Descriptions | 14-11 |
| 14-8 | UCSRn Field Descriptions | 14-12 |
| 14-9 | RSMP Field Descriptions | 14-13 |
| 14-10 | TSPC Field Descriptions | 14-13 |
| 14-11 | UCRn Field Descriptions | 14-14 |
| 14-12 | UIPCRn Field Descriptions | 14-17 |
| 14-13 | UACRn Field Descriptions | 14-18 |
| 14-14 | UISRn/UIMRn Field Descriptions | 14-19 |
| 14-15 | UIVRn Field Descriptions | 14-20 |
| 14-16 | UIPn Field Descriptions | 14-20 |
| 14-17 | UOP1/UOP0 Field Descriptions | 14-21 |
| 14-18 | UART Module Signals | 14-22 |
| 14-19 | UART Module Initialization Sequence | 14-38 |
| 15-1 | Parallel Port Pin Descriptions | 15-2 |
| 15-2 | PADDR Field Description | 15-2 |

# TABLES


| Table Number | Title | Page Number |
|---|---|---|
| 15-3 | Relationship between PADAT Register and Parallel Port Pin (PP) | 15-3 |
| 16-1 | Pins 1–52 (Left, Top-to-Bottom) | 16-1 |
| 16-2 | Pins 53–104 (Bottom, Left-to-Right) | 16-3 |
| 16-3 | Pins 105–156 (Right, Bottom-to-Top) | 16-5 |
| 16-4 | Pins 157–208 (Top, Right-to-Left) | 16-6 |
| 16-5 | Dimensions | 16-11 |
| 17-1 | MCF5407 Signal Index | 17-3 |
| 17-2 | MCF5407 Alphabetical Signal Index | 17-5 |
| 17-3 | Data Pin Configuration | 17-8 |
| 17-4 | Bus Cycle Size Encoding | 17-9 |
| 17-5 | Bus Cycle Transfer Type Encoding | 17-10 |
| 17-6 | TM[2:0] Encodings for TT = 00 (Normal Access) | 17-10 |
| 17-7 | TM2 Encoding for DMA as Master (TT = 01) | 17-11 |
| 17-8 | TM[1:0] Encoding for DMA as Master (TT = 01) | 17-11 |
| 17-9 | TM[2:0] Encodings for TT = 10 (Emulator Access) | 17-11 |
| 17-10 | TM[2:0] Encodings for TT = 11 (Interrupt Level) | 17-12 |
| 17-11 | Data Pin Configuration | 17-14 |
| 17-12 | D7 Selection of CS0 Automatic Acknowledge | 17-14 |
| 17-13 | D6 and D5 Selection of CS0 Port Size | 17-14 |
| 17-14 | D3/BE_CONFIG, BE[3:0] Boot Configuration | 17-15 |
| 17-15 | D4/ADDR_CONFIG, Address Pin Assignment | 17-15 |
| 18-1 | ColdFire Bus Signal Summary | 18-1 |
| 18-2 | Bus Cycle Size Encoding | 18-3 |
| 18-3 | Accesses by Matches in CSCRs and DACRs | 18-5 |
| 18-4 | Bus Cycle States | 18-6 |
| 18-5 | Allowable Line Access Patterns | 18-12 |
| 18-6 | MCF5407 Arbitration Protocol States | 18-20 |
| 18-7 | ColdFire Bus Arbitration Signal Summary | 18-21 |
| 18-8 | Cycles for Basic No-Wait-State External Master Access | 18-23 |
| 18-9 | Cycles for External Master Burst Line Access to 32-Bit Port | 18-24 |
| 18-10 | MCF5407 Two-Wire Bus Arbitration Protocol Transition Conditions | 18-28 |
| 18-11 | Three-Wire Bus Arbitration Protocol Transition Conditions | 18-32 |
| 18-12 | Data Pin Configuration | 18-35 |
| 19-1 | JTAG Pin Descriptions | 19-3 |
| 19-2 | JTAG Instructions | 19-5 |
| 19-3 | IDCODE Bit Assignments | 19-6 |
| 19-4 | Boundary-Scan Bit Definitions | 19-7 |
| 20-1 | Absolute Maximum Ratings | 20-1 |
| 20-2 | Operating Temperatures | 20-1 |
| 20-3 | DC Electrical Specifications | 20-2 |
| 20-4 | Divide Ratio Encodings | 20-4 |
| 20-5 | Clock Timing Specification | 20-5 |
| 20-6 | Input AC Timing Specification | 20-6 |

# TABLES


| Table Number | Title | Page Number |
|---|---|---|
| 20-7 | Output AC Timing Specification | 20-6 |
| 20-8 | Reset Timing Specification | 20-15 |
| 20-9 | Debug AC Timing Specification | 20-16 |
| 20-10 | Timer Module AC Timing Specification | 20-17 |
| 20-11 | $I^2C$ Input Timing Specifications between SCL and SDA | 20-18 |
| 20-12 | $I^2C$ Output Timing Specifications between SCL and SDA | 20-18 |
| 20-13 | UART Module AC Timing Specifications | 20-19 |
| 20-14 | General-Purpose I/O Port AC Timing Specifications | 20-22 |
| 20-15 | DMA AC Timing Specifications | 20-23 |
| 20-16 | IEEE 1149.1 (JTAG) AC Timing Specifications | 20-24 |
| A-1 | Differences between MCF5307 and MCF5407 | A-1 |
| A-2 | MOVEC CPU Space Register Map | A-4 |
| A-3 | TM[2:1] Encoding for MCF5307 Internal DMA as Master (TT = 01) | A-4 |
| A-4 | TM0 Encoding for MCF5307 Internal DMA as Master (TT = 01) | A-5 |
| A-5 | Divide Ratio Encodings | A-7 |
| A-6 | D[7:0] Multiplexing | A-8 |
| A-7 | D7/AA, Automatic Acknowledge of Boot CS0 | A-9 |
| A-8 | D[6:5]/PS[1:0], Port Size of Boot CS0 | A-9 |
| A-9 | D4/ADDR_CONFIG, Address Pin Assignment | A-9 |
| A-10 | D3/BE_CONFIG, BE[3:0] Boot Configuration | A-9 |
| A-11 | Definition of DRc Encoding—Write | A-13 |
| A-12 | Debug C Exception Vector Assignments | A-16 |
| A-13 | Version 4 Debug C Processor Status Encodings | A-17 |
| B-1 | SIM Registers | B-1 |
| B-2 | Interrupt Controller Registers | B-1 |
| B-3 | Chip-Select Registers | B-2 |
| B-4 | DRAM Controller Registers | B-3 |
| B-5 | General-Purpose Timer Registers | B-4 |
| B-6 | UART0 Control Registers | B-4 |
| B-7 | UART1 Control Registers | B-6 |
| B-8 | Parallel Port Memory Map | B-7 |
| B-9 | $I^2C$ Interface Memory Map | B-8 |
| B-10 | DMA Controller Registers | B-8 |

# About This Book

The primary objective of this user's manual is to define the functionality of the MCF5407 processors for use by software and hardware developers.

The information in this book is subject to change without notice, as described in the disclaimers on the title page of this book. As with any technical documentation, it is the readers' responsibility to be sure they are 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.motorola.com/coldfire.

## Audience

This manual is intended for system software and hardware developers and applications programmers who want to develop products for the MCF5407. 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 a brief description of the major sections of this manual:

- Chapter 1, "Overview," includes general descriptions of the modules and features incorporated in the MCF5407, focussing in particular on new features defined by the Version 4 (V4) programming model, such as the Harvard memory architecture implementation, new instructions, and new registers.
- Part I is intended for system designers who need to understand the operation of the MCF5407 ColdFire core and its multiply/accumulate (MAC) execution unit. It describes the programming and exception models, Harvard memory implementation, and debug module.
  - Chapter 2, "ColdFire Core," provides an overview of the microprocessor core of the MCF5407. The chapter begins with a description of enhancements from the V3 ColdFire core, and then fully describes the V4 programming model as it is implemented on the MCF5407. It also includes a full description of exception handling, data formats, an instruction set summary, and a table of instruction timings.

— Chapter 3, “Hardware Multiply/Accumulate (MAC) Unit,” describes the MCF5407 multiply/accumulate unit, which executes integer multiply, multiply-accumulate, and miscellaneous register instructions. The MAC is integrated into the operand execution pipeline (OEP).

— Chapter 4, “Local Memory.” This chapter describes the MCF5407 implementation of the ColdFire V4 local memory specification. It consists of the two following major sections.

– Section 4.2, “SRAM Overview,” describes the MCF5407 on-chip static RAM (SRAM) implementation. It covers general operations, configuration, and initialization. It also provides information and examples showing how to minimize power consumption when using the SRAM.

– Section 4.7, “Cache Overview,” describes the MCF5407 cache implementation, including organization, configuration, and coherency. It describes cache operations and how the cache interacts with other memory structures.

— Chapter 5, “Debug Support,” describes the Revision C enhanced hardware debug support in the MCF5407. This revision of the ColdFire debug architecture encompasses earlier revisions.

- Part II, “System Integration Module (SIM),” describes the system integration module, which provides overall control of the bus and serves as the interface between the ColdFire core processor complex and internal peripheral devices. It includes a general description of the SIM and individual chapters that describe components of the SIM, such as the phase-lock loop (PLL) timing source, interrupt controller for peripherals, configuration and operation of chip selects, and the SDRAM controller.

— Chapter 6, “SIM Overview,” describes the SIM programming model, bus arbitration, and system-protection functions for the MCF5407.

— Chapter 7, “Phase-Locked Loop (PLL),” describes configuration and operation of the PLL module. It describes in detail the registers and signals that support the PLL implementation.

— Chapter 8, “I2C Module,” describes the MCF5407 $I^2C$ module, including $I^2C$ protocol, clock synchronization, and the registers in the $I^2C$ programing model. It also provides extensive programming examples.

— Chapter 9, “Interrupt Controller,” describes operation of the interrupt controller portion of the SIM. Includes descriptions of the registers in the interrupt controller memory map and the interrupt priority scheme.

— Chapter 10, “Chip-Select Module,” describes the MCF5407 chip-select implementation, including the operation and programming model, which includes the chip-select address, mask, and control registers.

— Chapter 11, “Synchronous/Asynchronous DRAM Controller Module,” describes configuration and operation of the synchronous/asynchronous DRAM

controller component of the SIM. It begins with a general description and brief glossary, and includes a description of signals involved in DRAM operations. The remainder of the chapter is divided between descriptions of asynchronous and synchronous operations.

- Part III, “Peripheral Module,” describes the operation and configuration of the MCF5407 DMA, timer, UART, and parallel port modules, and describes how they interface with the system integration unit, described in Part II.
  - Chapter 12, “DMA Controller Module,” provides an overview of the DMA controller 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, showing timing diagrams for various operations.
  - Chapter 13, “Timer Module,” describes configuration and operation of the two general-purpose timer modules, timer 0 and timer 1. It includes programming examples.
  - Chapter 14, “UART Modules,” describes the use of the universal asynchronous/synchronous receiver/transmitters (UARTs) implemented on the MCF5407 and includes programming examples. Particular attention is given to the UART1 implementation of a synchronous interface that provides a controller for an 8- or 16-bit CODEC interface and an audio CODEC ‘97 (AC ’97) digital interface.
  - Chapter 15, “Parallel Port (General-Purpose I/O),” describes the operation and programming model of the parallel port pin assignment, direction-control, and data registers. It includes a code example for setting up the parallel port.
- Part IV, “Hardware Interface,” provides a pinout and both electrical and functional descriptions of the MCF5407 signals. It also describes how these signals interact to support the variety of bus operations shown in timing diagrams.
  - Chapter 16, “Mechanical Data,” provides a functional pin listing and package diagram for the MCF5407.
  - Chapter 17, “Signal Descriptions,” provides an alphabetical listing of MCF5407 signals. This chapter describes the MCF5407 signals. In particular, it shows which are inputs or outputs, how they are multiplexed, which signals require pull-up resistors, and the state of each signal at reset.
  - Chapter 18, “Bus Operation,” describes data transfers, error conditions, bus arbitration, and reset operations. It describes transfers initiated by the MCF5407 and by an external bus master, and includes detailed timing diagrams showing the interaction of signals in supported bus operations. Note that Chapter 11, “Synchronous/Asynchronous DRAM Controller Module,” describes DRAM cycles.
  - Chapter 19, “IEEE 1149.1 Test Access Port (JTAG),” describes configuration and operation of the MCF5407 JTAG test implementation. It describes the use of JTAG instructions and how to disable JTAG functionality.

— Chapter 20, “Electrical Specifications,” describes AC and DC electrical specifications and thermal characteristics for the MCF5407. Because additional speeds may have become available since the publication of this book, consult Motorola’s ColdFire web page, http://www.motorola.com/coldfire, to confirm that this is the latest information.

This manual includes the following two appendixes:

- Appendix A, “Migrating from the ColdFire MCF5307 to the MCF5407,” highlights the differences between the MCF5307B and MCF5407. Users of the MCF5307 and MCF5307A should use this document in conjunction with the *MCF5307 User's Manual Mask Set Addendum*. For additional information, see the *MCF5407 Integrated ColdFire Microprocessor Product Brief*.
- Appendix B, “List of Memory Maps,” lists the entire address-map for MCF5407 memory-mapped registers.

This manual also includes a glossary and an index.

# 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.

## General Information

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

## ColdFire Documentation

The ColdFire documentation is available from the sources listed on the back cover of this manual. Document order numbers are included in parentheses for ease in ordering.

- *ColdFire Programmers Reference Manual, R1.0* (MCF5200PRM/AD)
- 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*. These include the following:
  - *ColdFire MCF5102 User’s Manual* (MCF5102UM/AD)
  - *ColdFire MCF5202 User’s Manual* (MCF5202UM/AD)
  - *ColdFire MCF5204 User’s Manual* (MCF5204UM/AD)
  - *ColdFire MCF5206 User’s Manual* (MCF5206EUM/AD)
  - *ColdFire MCF5206E User’s Manual* (MCF5206EUM/AD)
  - *ColdFire MCF5307 User’s Manual* (MCF5307UM/AD)
- *ColdFire Programmers Reference Manual, R1.0* (MCF5200PRM/AD)

- *Using Microprocessors and Microcomputers: The Motorola Family,* William C. Wray, Ross Bannatyne, Joseph D. Greenfield

Additional literature on ColdFire implementations is being released as new processors become available. For a current list of ColdFire documentation, refer to the World Wide Web at http://www.motorola.com/ColdFire/.

## Conventions

This document uses the following notational conventions:

| | |
|---|---|
| MNEMONICS | In text, instruction mnemonics are shown in uppercase. |
| `mnemonics` | In code and tables, instruction mnemonics are shown in lowercase. |
| *italics* | Italics indicate variable command parameters.<br>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 |
| 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 |

## Acronyms and Abbreviations

Table i lists acronyms and abbreviations used in this document.

**Table i. Acronyms and Abbreviated Terms**

| Term | Meaning |
|---|---|
| ADC | Analog-to-digital conversion |
| ALU | Arithmetic logic unit |
| AVEC | Autovector |

## Table i. Acronyms and Abbreviated Terms (Continued)

| Term | Meaning |
|---|---|
| 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 |
| EDO | Extended data output (DRAM) |
| FIFO | First-in, first-out |
| GPIO | General-purpose I/O |
| $I^2C$ | 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 |
| LIFO | Last-in, first-out |
| LRU | Least recently used |
| LSB | Least-significant byte |
| lsb | Least-significant bit |
| MAC | Multiple accumulate unit |
| 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 |
| PLL | Phase-locked loop |
| PLRU | Pseudo least recently used |

**Table i. Acronyms and Abbreviated Terms (Continued)**

| Term | Meaning |
|---|---|
| POR | Power-on reset |
| PQFP | Plastic quad flat pack |
| RISC | Reduced instruction set computing |
| Rx | Receive |
| SIM | System integration module |
| SOF | Start of frame |
| TAP | Test access port |
| TTL | Transistor-to-transistor logic |
| Tx | Transmit |
| UART | Universal asynchronous/synchronous receiver transmitter |

# Terminology and Notational Conventions

Table ii shows notational conventions used throughout this document.

**Table ii 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) |

## Table ii Notational Conventions (Continued)

| Instruction | Operand Syntax |
|---|---|
| **Register Names** | |
| ACC | MAC accumulator register |
| CCR | Condition code register (lower byte of SR) |
| MACSR | MAC status register |
| MASK | MAC mask register |
| PC | Program counter |
| SR | Status register |
| **Port Name** | |
| PSTDDATA | Processor status/debug data port |
| **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) |
| **Operations** | |
| + | Arithmetic addition or postincrement indicator |
| – | Arithmetic subtraction or predecrement indicator |
| x | Arithmetic multiplication |

## Table ii Notational Conventions (Continued)

| Instruction | Operand Syntax |
|---|---|
| / | 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 |
| **Condition Code Register Bit Names** | |
| C | Carry |
| N | Negative |
| V | Overflow |
| X | Extend |
| Z | Zero |

# Chapter 1
# Overview

This chapter is an overview of the MCF5407 ColdFire® processor. It includes general descriptions of the modules and features incorporated in the MCF5407, focusing in particular on new features defined by the Version 4 (V4) programming model, such as the Harvard memory architecture implementation, new instructions, and new registers.

## 1.1 Features

The MCF5407 integrated microprocessor combines a V4 ColdFire processor core with the following components, as shown in Figure 1-1:

- Harvard architecture memory system with 16-Kbyte instruction cache and 8-Kbyte data cache
- Two, 2-Kbyte on-chip SRAMs
- Integer/fractional multiply-accumulate (MAC) unit
- Divide unit
- System debug interface
- DRAM controller for synchronous and asynchronous DRAM
- Four-channel DMA controller
- Two general-purpose timers
- Two UARTs, one that supports synchronous operations
- $I^2C$™ interface
- Parallel I/O interface
- System integration module (SIM)

Designed for embedded control applications, the MCF5407 delivers 233 Dhrystone MIPS at 162 MHz while minimizing system costs.

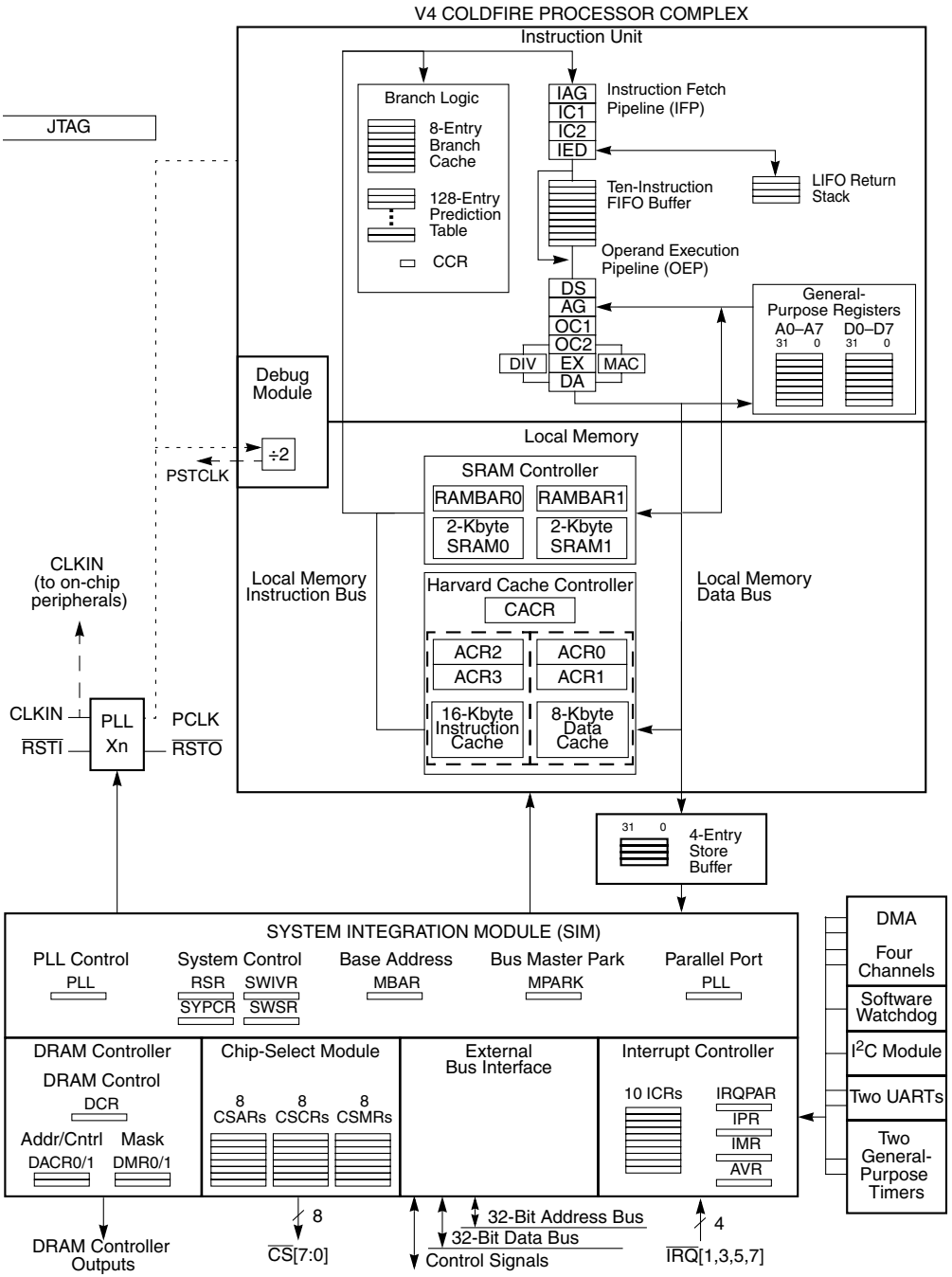



**Figure 1-1. MCF5407 Block Diagram**

Features common to many embedded applications, such as DMAs, various DRAM controller interfaces, and on-chip memories, are integrated using advanced process technologies.

The MCF5407 extends the legacy of Motorola's 68K family by providing a compatible path for 68K and ColdFire customers in which development tools and customer code can be leveraged. In fact, customers moving from 68K to ColdFire can use code translation and emulation tools that facilitate modifying 68K assembly code to the ColdFire architecture. The package, pinout, and integration mix of the MCF5407 create an especially simple upgrade for current MCF5307 designs with over triple the system performance.

Based on the concept of variable-length RISC technology, the ColdFire family combines the architectural simplicity of conventional 32-bit RISC with a memory-saving, variable-length instruction set. In defining the ColdFire architecture for embedded processing applications, a 68K-code compatible core combines performance advantages of a RISC architecture with the optimum code density of a streamlined, variable-length M68000 instruction set.

By using a variable-length instruction set architecture, embedded system designers using ColdFire RISC processors enjoy significant advantages over conventional fixed-length RISC architectures. The denser binary code for ColdFire processors consumes less memory than many fixed-length instruction set RISC processors available. This improved code density means more efficient system memory use for a given application and allows use of slower, less costly memory to help achieve a target performance level.

The MCF5407 is the first standard product to implement the Version 4 ColdFire microprocessor core. The V4 microarchitecture implements a number of advanced techniques, including a Harvard memory architecture, branch cache acceleration logic, and limited superscalar support (dual-instruction issue), which contribute to the 233 Dhrystone MIPS performance level. Increasing the internal speed of the core also allows higher performance while providing the system designer with an easy-to-use lower speed system interface. The processor complex frequency is an integer multiple, 3 to 6 times, of the external bus frequency. The core clock can be stopped to support a low-power mode.

Serial communication channels are provided by an $I^2C$ interface module and two programmable full-duplex UARTs, one of which provides synchronous communications for soft-modem applications. Four channels of DMA allow for fast data transfer using a programmable burst mode independent of processor execution. The two 16-bit general-purpose multimode timers provide separate input and output signals. For system protection, the processor includes a programmable 16-bit software watchdog timer. In addition, common system functions such as chip selects, interrupt control, bus arbitration, and an IEEE 1149.1 JTAG module are included.A sophisticated debug interface supports background-debug mode plus real-time trace and debug with an expanded set of on-chip breakpoint registers. This interface is present in all ColdFire standard products and allows common emulator support across the entire family of microprocessors.

## 1.2 MCF5407 Features

The following list summarizes MCF5407 features:

- ColdFire processor core
  - Variable-length RISC, clock-multiplied Version 4 microprocessor core
  - Implementation of Revision B of the ColdFire instruction set architecture (ISA), which leverages the 68K programming model
  - Two independent decoupled pipelines: four-stage instruction fetch pipeline (IFP) and five-stage operand execution pipeline (OEP)
  - Ten-instruction FIFO buffer provides decoupling between the pipelines
  - Limited superscalar design achieves performance levels close to dual-issue performance
  - Programmable two-level branch acceleration mechanism with an 8-entry branch cache plus a 128-entry prediction table for increased performance
  - 32-bit internal address bus supporting 4 Gbytes of linear address space
  - 32-bit data bus
  - 16 user-accessible, 32-bit-wide, general-purpose registers
  - Supervisor/user modes for system protection
  - Vector base register to relocate exception-vector table
  - Optimized for high-level language constructs
- Multiply and accumulate unit (MAC)
  - High-speed, complex arithmetic processing for DSP applications
  - Tightly coupled to the OEP
  - Three-stage execute pipeline with one clock issue rate for 16 x 16 operations
  - 16 x 16 and 32 x 32 multiplies support, all with 32-bit accumulate
  - Signed or unsigned integer support, plus signed fractional operands
- Hardware integer divide unit
  - Unsigned and signed integer divide support
  - Tightly coupled to the OEP
  - 32/16 and 32/32 operation support producing quotient and/or remainder results
- 16-Kbyte instruction cache, 8-Kbyte data cache
  - Four-way set-associative organization
  - Operates at higher processor core frequency
  - Provides pipelined, single-cycle access to critical code and data
  - Data cache supports write-through and copyback modes
  - Four-entry, 32-bit store buffer to improve performance of operand writes

- Two, 2-Kbyte SRAMs
  - Programmable location anywhere within 4-Gbyte linear address space
  - Higher core-frequency operation
  - Pipelined, single-cycle access to critical code or data
  - Each block mappable to either the instruction or data operand bus
- DMA controller
  - Four fully programmable channels: two support external requests and external acknowledges
  - Dual-address and single-address transfer support with 8-, 16-, and 32-bit data capability
  - Source/destination address pointers that can increment or remain constant
  - 24-bit transfer counter per channel
  - Operand packing and unpacking supported
  - Auto-alignment transfers supported for efficient block movement
  - Bursting and cycle steal support
  - Two-bus-clock internal access
  - Automatic DMA transfers from on-chip UARTs using internal interrupts
- DRAM controller
  - Synchronous DRAM (SDRAM), extended-data-out (EDO) DRAM, and fast page mode support
  - Up to 512 Mbytes of DRAM
  - Programmable timer provides CAS-before-RAS refresh for asynchronous DRAMs
  - Support for two separate memory blocks
- Two UARTs
  - One UART offers synchronous mode with expanded buffers for soft modem support
  - Full-duplex operation
  - Programmable clock
  - Modem control signals available ($\overline{\text{CTS}}$, $\overline{\text{RTS}}$)
  - Processor-interrupt capability
- Dual 16-bit general-purpose multiple-mode timers
  - 8-bit prescaler
  - Timer input and output pins
  - Processor-interrupt capability
  - Up to 18.5-nS resolution at 54 MHz

- $I^2C$ module
  - Interchip bus interface for EEPROMs, LCD controllers, A/D converters, and keypads
  - Fully compatible with industry-standard $I^2C$ bus
  - Master or slave modes support multiple masters
  - Automatic interrupt generation with programmable level
- System interface module (SIM)
  - Chip selects provide direct interface to 8-, 16-, and 32-bit SRAM, ROM, FLASH, and memory-mapped I/O devices
  - Eight fully programmable chip selects, each with a base address register
  - Programmable wait states and port sizes per chip select
  - User-programmable processor clock/input clock frequency ratio
  - Programmable interrupt controller
  - Low interrupt latency
  - Four external interrupt request inputs
  - Programmable autovector generator
  - Software watchdog timer
- 16-bit general-purpose I/O interface
- IEEE 1149.1 test (JTAG) module
- System debug support
  - Real-time trace for determining dynamic execution path while in emulator mode
  - Background debug mode (BDM) for debug features while halted
  - Real-time debug support, including 13 user-visible hardware breakpoint registers supporting 8 separate breakpoints
  - Supports servicing of critical, real-time interrupt requests while the BDM is in emulator mode
  - Supports comprehensive emulator functions through trace and breakpoint logic
- On-chip PLL
  - Accepts various clock input (CLKIN) frequencies between 25 and 54 MHz
  - Supports core frequencies between 100 and 162 MHz
  - Supports low-power mode
- Product offerings
  - 233 Dhrystone MIPS at 162 MHz
  - Implemented in 0.22 $\mu$, quad-layer-metal process technology with 1.8-V operation (3.3-V compliant I/O pads)
  - 208-pin plastic QFP package
  - 0°–70° C operating temperature

### 1.2.1 Process

The MCF5407 is manufactured in a 0.22-$\mu$ CMOS process with quad-layer-metal routing technology. This process combines the high performance and low power needed for embedded system applications. Inputs are 3.3-V tolerant; outputs are CMOS or open-drain CMOS with outputs operating from VDD + 0.5 V to GND - 0.5 V, with guaranteed TTL-level specifications.

## 1.3 ColdFire Module Description

The following sections provide overviews of the various modules incorporated in the MCF5407.

### 1.3.1 ColdFire Core

The Version 4 ColdFire core consists of two independent and decoupled pipelines to maximize performance—the instruction fetch pipeline (IFP) and the operand execution pipeline (OEP).

#### 1.3.1.1 Instruction Fetch Pipeline (IFP)

The four-stage instruction fetch pipeline (IFP) is designed to prefetch instructions for the operand execution pipeline (OEP). Because the fetch and execution pipelines are decoupled by a ten-instruction FIFO buffer, the fetch mechanism can prefetch instructions in advance of their use by the OEP, thereby minimizing the time stalled waiting for instructions. To maximize the performance of conditional branch instructions, the Version 4 IFP implements a sophisticated two-level acceleration mechanism.

The first level is an 8-entry, direct-mapped branch cache with a 2-bit prediction state (strongly/weakly, taken/not-taken) for each entry. The branch cache implements instruction folding techniques. These allow conditional branch instructions that are predicted correctly as taken to execute in zero cycles.

For those conditional branches with no information in the branch cache, a second-level, direct-mapped prediction table containing 128 entries is accessed. Again, each entry uses the same 2-bit prediction state definition as the branch cache. This branch prediction state is then used to predict the direction of prefetched conditional branch instructions.

Other change-of-flow instructions, including unconditional branches, jumps, and subroutine calls, use a similar mechanism where the IFP calculates the target address. The performance of subroutine return instructions is improved through the use of a four-entry, LIFO return stack.

In all cases, these mechanisms allow the IFP to redirect the fetch stream down the path predicted to be taken well in advance of the actual instruction execution. The result is significantly improved performance.

#### 1.3.1.2 Operand Execution Pipeline (OEP)

The prefetched instruction stream is gated from the FIFO buffer into the five-stage OEP. The OEP consists of two, traditional two-stage RISC compute engines with a register file access feeding an arithmetic/logic unit (ALU). The compute engine located at the top of the OEP is typically used for operand memory address calculations (the address ALU), while the compute engine located at the bottom of the pipeline is used for instruction execution (the execution ALU). The resulting structure provides 3.9 Gbytes/S data operand bandwidth at 162 MHz to the two compute engines and supports single-cycle execution speeds for most instructions, including all load, store, and most embedded-load operations. In response to users and developers' comments, the V4 design supports execution of the ColdFire Revision B instruction set, which adds a small number of new instructions to improve performance and code density.

The OEP also implements two advanced performance features. It dynamically determines the appropriate location of instruction execution (either in the address ALU or the execution ALU) based on the pipeline state. The address compute engine, in conjunction with register renaming resources, can be used to execute a number of heavily-used opcodes and forward the results to subsequent instructions without any pipeline stalls. Additionally, the OEP implements instruction folding techniques involving MOVE instructions so that two instructions can be issued in a single machine cycle. The resulting microarchitecture approaches the performance of a full superscalar implementation, but at a much lower silicon cost.

#### 1.3.1.3 MAC Module

The MAC unit provides signal processing capabilities for the MCF5407 in a variety of applications including digital audio and servo control. Integrated as an execution unit in the processor's OEP, the MAC unit implements a three-stage arithmetic pipeline optimized for 16 x 16 multiplies. Both 16- and 32-bit input operands are supported by this design in addition to a full set of extensions for signed and unsigned integers, plus signed, fixed-point fractional input operands.

#### 1.3.1.4 Integer Divide Module

Integrated into the OEP, the divide module performs operations using signed and unsigned integers. The module supports word and longword divides producing quotients and/or remainders.

### 1.3.2 Harvard Architecture

A Harvard memory architecture implements separate instruction and data buses to the processor-local memories, removing conflicts between instruction fetches and operand accesses.

#### 1.3.2.1 16-Kbyte Instruction Cache/8-Kbyte Data Cache

The MCF5407 Harvard architecture includes a 16-Kbyte instruction cache and an 8-Kbyte data cache. These four-way, set-associative caches provide pipelined, single-cycle access on cached instructions and operands.

As with all ColdFire caches, the cache controllers implement a non-lockup, streaming design. The use of processor-local memories decouples performance from external memory speeds and increases available bandwidth for external devices or the on-chip 4-channel DMA.

Both caches implement line-fill buffers to optimize 16-byte line burst accesses. Additionally, the data cache supports copyback, write-through, or cache-inhibited modes. A 4-entry, 32-bit buffer is used for cache line push operations and can be configured for deferred write buffering in write-through or cache-inhibited modes.

The INTOUCH instruction can be used to prefetch instructions that, when used with the cache locking feature, cannot be displaced from the instruction cache by instruction cache misses. This function may be desirable in systems where deterministic real-time performance is critical.

#### 1.3.2.2 Internal 2-Kbyte SRAMs

Two 2-Kbyte on-chip SRAM modules are also connected to the Harvard memory architecture and provide pipelined, single-cycle access to memory regions mapped to these devices. Each memory can be independently mapped to any 0-modulo-2K location in the 4-Gbyte address space and can be configured either for instruction or data accesses. Time-critical functions can be mapped onto the instruction memory bus, while the system stack or heavily-referenced data operands can be mapped onto the data bus.

### 1.3.3 DRAM Controller

The MCF5407 DRAM controller provides a direct interface for up to two blocks of DRAM. The controller supports 8-, 16-, or 32-bit memory widths and can easily interface to PC-100 DIMMs. A unique addressing scheme allows for increases in system memory size without rerouting address lines and rewiring boards. The controller operates in normal mode or in page mode and supports SDRAMs and EDO DRAMs.

### 1.3.4 DMA Controller

The MCF5407 provides four fully programmable DMA channels for quick data transfer. Dual- and single-address modes support bursting and cycle steal. Data transfers are 32 bits long with packing and unpacking supported along with an auto-alignment option for efficient block transfers. Automatic block transfers from on-chip serial UARTs are also supported through the DMA channels.

## 1.3.5 UART Modules

The MCF5407 contains two UARTs, which function independently. One UART has been enhanced to provide synchronous operation and a CODEC interface for soft modem support. Either UART can be clocked by the system bus clock, eliminating the need for an external crystal. Each UART module interfaces directly to the CPU, as shown in Figure 1-2.

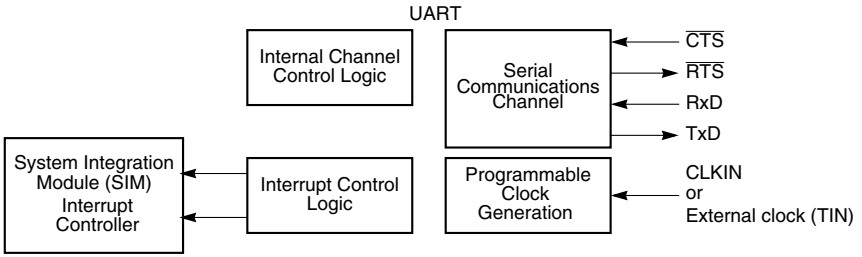



**Figure 1-2. UART Module Block Diagram**

Each UART module consists of the following major functional areas:

- Serial communication channel
- 16-bit divider for clock generation
- Internal channel control logic
- Interrupt control logic

UART1 is enhanced to provide a CODEC interface for soft modem support. UART1 can be programmed to function like UART0 or in one of following modem modes:

- An 8-bit CODEC interface
- A 16-bit CODEC interface
- An audio CODEC ’97 (AC97) digital interface controller

Each UART contains an programmable clock-rate generator. Data formats can be 5, 6, 7, or 8 bits with even, odd, or no parity, and up to 2 stop bits in 1/16 increments. The UARTs include the following transmit and receive FIFO buffers:

- UART0 has a 4-byte FIFO receive buffer and a 2-byte FIFO transmit buffer.
- In UART1, the Tx and Rx FIFOs can hold the following:
  - — 32 1-byte samples when programmed as a UART or as an 8-bit CODEC interface
  - — 16 2-byte samples when programmed as a 16-bit CODEC interface
  - — 16 20-bit samples when programmed as a Digital AC ’97 Controller

The UART modules also provide several error-detection and maskable-interrupt capabilities. Modem support includes request-to-send ($\overline{\text{RTS}}$) and clear-to-send ($\overline{\text{CTS}}$) lines.

CLKIN provides the time base through a programmable prescaler. The UART time scale can also be sourced from a timer input. Full-duplex, auto-echo loopback, local loopback,

and remote loopback modes allow testing of UART connections. The programmable UARTs can interrupt the CPU on various normal or error-condition events.

### 1.3.6 Timer Module

The timer module includes two general-purpose timers, each of which contains a free-running 16-bit timer for use in any of three modes. One mode captures the timer value with an external event. Another mode triggers an external signal or interrupts the CPU when the timer reaches a set value, while a third mode counts external events.

The timer unit has an 8-bit prescaler that allows programming of the clock input frequency, which is derived from the system bus cycle or an external clock input pin (TIN). The programmable timer-output pin generates either an active-low pulse or toggles the output.

### 1.3.7 $I^2C$ Module

The $I^2C$ interface is a two-wire, bidirectional serial bus used for quick data exchanges between devices. The $I^2C$ minimizes the interconnection between devices in the end system and is best suited for applications that need occasional bursts of rapid communication over short distances among several devices. The $I^2C$ can operate in master, slave, or multiple-master modes.

### 1.3.8 System Interface

The MCF5407 processor provides a direct interface to 8-, 16-, and 32-bit FLASH, SRAM, ROM, and peripheral devices through the use of fully programmable chip selects and write enables. Support for burst ROMs is also included. Through the on-chip PLL, users can input a slower clock (25 to 54 MHz) that is internally multiplied to create the faster processor clock (100 to 162 MHz).

#### 1.3.8.1 External Bus Interface

The bus interface controller transfers information between the ColdFire core or DMA and memory, peripherals, or other devices on the external bus. The external bus interface provides up to 32 bits of address bus space, a 32-bit data bus, and all associated control signals. This interface implements an extended synchronous protocol that supports bursting operations.

Simple two-wire request/acknowledge bus arbitration between the MCF5407 processor and another bus master, such as an external DMA device, is glueless with arbitration logic internal to the MCF5407 processor. Multiple-master arbitration is also available with some simple external arbitration logic.

#### 1.3.8.2 Chip Selects

Eight fully programmable chip select outputs support the use of external memory and peripheral circuits with user-defined wait-state insertion. These signals interface to 8-, 16-,

or 32-bit ports. The base address, access permissions, and internal bus transfer terminations are programmable with configuration registers for each chip select. $\overline{CS0}$ also provides global chip select functionality of boot ROM upon reset for initializing the MCF5407.

#### 1.3.8.3 16-Bit Parallel Port Interface

A 16-bit general-purpose programmable parallel port serves as either an input or an output on a pin-by-pin basis.

#### 1.3.8.4 Interrupt Controller

The interrupt controller provides user-programmable control of ten internal peripheral interrupts and implements four external fixed interrupt-request pins. Each internal interrupt can be programmed to any one of seven interrupt levels and four priority levels within each of these levels. Additionally, the external interrupt request pins can be mapped to levels 1, 3, 5, and 7 or levels 2, 4, 6, and 7. Autovector capability is available for both internal and external interrupts.

#### 1.3.8.5 JTAG

To help with system diagnostics and manufacturing testing, the MCF5407 processor includes dedicated user-accessible test logic that complies with the IEEE 1149.1a standard for boundary-scan testability, often referred to as the Joint Test Action Group, or JTAG. For more information, refer to the IEEE 1149.1a standard.

### 1.3.9 System Debug Interface

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 real-time trace and debug information. This allows the processor and system to be debugged at full speed without the need for costly in-circuit emulators. The debug unit in the MCF5407 is a compatible upgrade to the MCF52xx and MCF53xx debug modules with added breakpoint registers and support for I/O interrupt request servicing while in emulator mode.

The on-chip breakpoint resources include a total of 13 programmable registers—two sets of address registers (each with two 32-bit registers), two sets of data registers (each with a 32-bit data register plus a 32-bit data mask register), one 32-bit PC register plus a 32-bit PC mask register and three additional 32-bit PC registers. 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.

The MCF5407's new interrupt servicing options during emulator mode allow real-time critical interrupt service routines to be serviced while processing a debug interrupt event, thereby ensuring that the system continues to operate even during debugging.

To support program trace, the Version 4 debug module has combined the processor status and debug data outputs into a single 8-bit bus (PSTDDATA[7:0]). This bus 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.

### 1.3.10 PLL Module

The MCF5407 PLL module is shown in Figure 1-3.

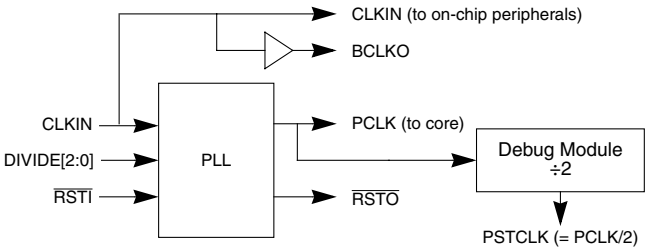



**Figure 1-3. PLL Module**

The PLL module's three modes of operation are described as follows.

- Reset mode—When $\overline{\text{RSTI}}$ is asserted, the PLL enters reset mode. At reset, the PLL asserts $\overline{\text{RSTO}}$ from the MCF5407. The core:bus frequency ratio and other MCF5407 configuration information are sampled during reset.
- Normal mode—In normal mode, the input frequency programmed at reset is clock-multiplied to provide the processor clock (PCLK).
- Reduced-power mode—In reduced-power mode, the PCLK is disabled by executing a sequence that includes programming a control bit in the system configuration register (SCR) and then executing the STOP instruction. Register contents are retained in reduced-power mode, so the system can be reenabled quickly when an unmasked interrupt or reset is detected.

## 1.4 Programming Model, Addressing Modes, and Instruction Set

The ColdFire programming model has two privilege levels—supervisor and user. The S bit in the status register (SR) indicates the privilege level. The processor identifies a logical address that differentiates between supervisor and user modes by accessing either the supervisor or user address space.

- User mode—When the processor is in user mode (SR[S] = 0), only a subset of registers can be accessed, and privileged instructions cannot be executed. Typically, most application processing occurs in user mode. User mode is usually entered by executing a return from exception instruction (RTE, assuming the value of SR[S] saved on the stack is 0) or a MOVE, SR instruction (assuming SR[S] is 0).
- Supervisor mode—This mode protects system resources from uncontrolled access by users. In supervisor mode, complete access is provided to all registers and the entire ColdFire instruction set. Typically, system programmers use the supervisor programming model to implement operating system functions and provide I/O control. The supervisor programming model provides access to the same registers as the user model, plus additional registers for configuring on-chip system resources, as described in Section 1.4.3, “Supervisor Registers.”

  Exceptions (including interrupts) are handled in supervisor mode.

## 1.4.1 Programming Model

Figure 1-4 shows the MCF5407 programming model.

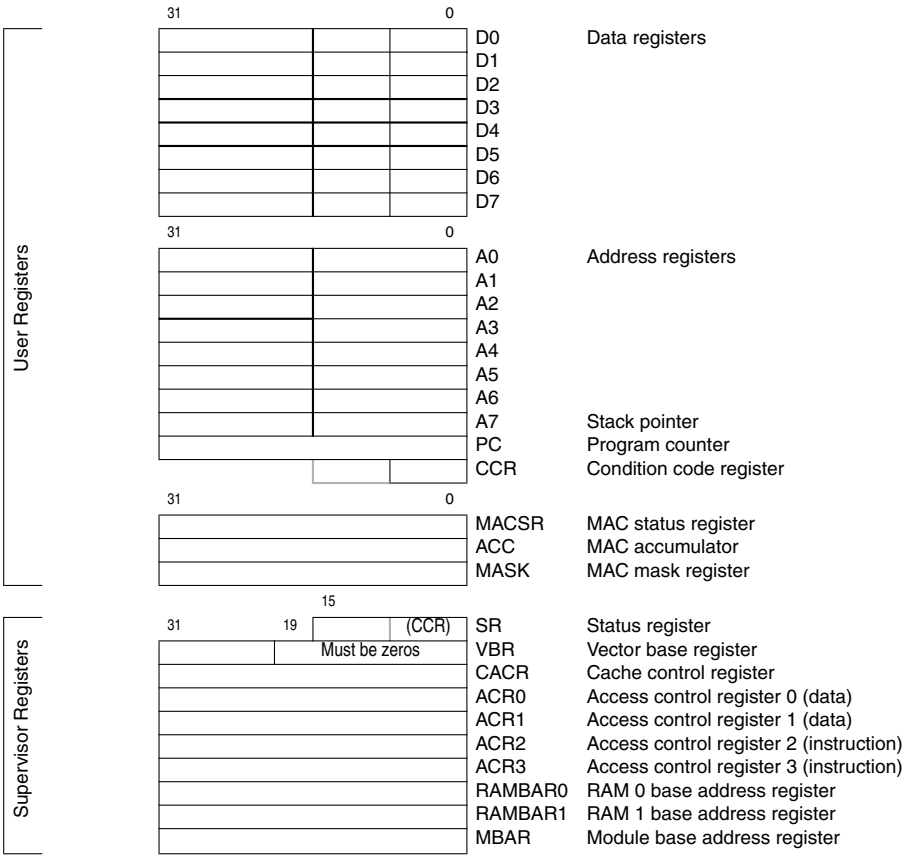



**Figure 1-4. ColdFire MCF5407 Programming Model**

## 1.4.2 User Registers

The user programming model is shown in Figure 1-4 and summarized in Table 1-1.

**Table 1-1. User-Level Registers**

| Register | Description |
|---|---|
| Data registers (D0–D7) | These 32-bit registers are for bit, byte, word, and longword operands. They can also be used as index registers. |
| Address registers (A0–A7) | These 32-bit registers serve as software stack pointers, index registers, or base address registers. The base address registers can be used for word and longword operations. A7 functions as a hardware stack pointer during stacking for subroutine calls and exception handling. |

**Table 1-1. User-Level Registers (Continued)**

| Register | Description |
|---|---|
| Program counter (PC) | Contains the address of the instruction currently being executed by the MCF5407 processor |
| Condition code register (CCR) | The CCR is the lower byte of the SR. It contains indicator flags that reflect the result of a previous operation and are used for conditional instruction execution. |
| MAC status register (MACSR) | Defines the operating configuration of the MAC unit and contains indicator flags from the results of MAC instructions. |
| Accumulator (ACC) | General-purpose register used to accumulate the results of MAC operations |
| Mask register (MASK) | General-purpose register provides an optional address mask for MAC instructions that fetch operands from memory. It is useful in the implementation of circular queues in operand memory. |

## 1.4.3 Supervisor Registers

Table 1-2 summarizes the MCF5407 supervisor-level registers.

**Table 1-2. Supervisor-Level Registers**

| Register | Description |
|---|---|
| Status register (SR) | The upper byte of the SR provides interrupt information in addition to a variety of mode indicators signaling the operating state of the ColdFire processor. The lower byte of the SR is the CCR, as shown in Figure 1-4. |
| Vector base register (VBR) | Defines the upper 12 bits of the base address of the exception vector table used during exception processing. The low-order 20 bits are forced to zero, locating the vector table on 0-modulo-1 Mbyte address. |
| Cache configuration register (CACR) | Defines the operating modes of the Version 4 cache memories. Control fields configuring the instruction, data, and branch cache are provided by this register, along with the default attributes for the 4-Gbyte address space. |
| Access control registers (ACR0/1, ACR2/3) | Define address ranges and attributes associated with various memory regions within the 4-Gbyte address space. Each ACR defines the location of a given memory region and assigns attributes such as write-protection and cache mode (copyback, write-through, cacheability). ACR0 and ACR1 support data memory; ACR2 and ACR3 support instruction memory. Additionally, CACR fields assign default attributes to the instruction and data memory spaces. |
| RAM base address registers (RAMBAR0, RAMBAR1) | Provide the logical base address for the two 2-Kbyte SRAM modules and define attributes and access types allowed for the corresponding SRAM. |
| Module base address register (MBAR) | Defines the logical base address for the memory-mapped space containing the control registers for the on-chip peripherals. |

## 1.4.4 Instruction Set

The Version 4 ColdFire core implements Revision B of the instruction set, which adds opcodes to enhance support for byte- and word-sized operands and position-independent code. The ColdFire instruction set supports high-level languages and is optimized for those instructions most commonly generated by compilers in embedded applications. Table 2-8 provides an alphabetized listing of the ColdFire instruction set opcodes, supported

operation sizes, and assembler syntax. For two-operand instructions, the first operand is generally the source operand and the second is the destination.

Because the ColdFire architecture provides an upgrade path for 68K customers, its instruction set supports most of the common 68K opcodes. A majority of the instructions are binary compatible or optimized 68K opcodes. This feature, when coupled with the code conversion tools from third-party developers, generally minimizes software porting issues for customers with 68K applications.

The following list summarizes new and enhanced instructions of Revision B ISA:

- New instructions:
  - INTOUCH loads blocks of instructions to be locked in the instruction cache.
  - MOV3Q.L moves 3-bit immediate data to destination location.
  - MVS.{B,W} sign-extends the source operand and moves it to destination register.
  - MVZ.{B,W} zero-fills the source operand and moves it to destination register.
  - SATS.L updates bit 31 of destination register depending on CCR overflow bit.
  - TAS.B tests and set byte operand being addressed.
- Enhancements to existing Revision A instructions:
  - Longword support for branch instructions (Bcc, BRA, BSR)
  - Byte and word support for compare instructions (CMP, CMPI)
  - Byte and longword support for MOVE.x where the source is of type #<data> and the destination is of type d16(Ax); that is, move.b #<data>, d16(Ax)

# Part I
# MCF5407 Processor Core

## Intended Audience

Part I is intended for system designers who need a general understanding of the functionality supported by the MCF5407. It also describes the operation of the MCF5407 ColdFire core and its multiply/accumulate (MAC) execution unit. It describes the programming and exception models, Harvard memory implementation, and debug module.

## Contents

- Chapter 2, “ColdFire Core,” provides an overview of the microprocessor core of the MCF5407. The chapter begins with a description of enhancements from the V3 ColdFire core, and then fully describes the V4 programming model as it is implemented on the MCF5407. It also includes a full description of exception handling, data formats, an instruction set summary, and a table of instruction timings.
- Chapter 3, “Hardware Multiply/Accumulate (MAC) Unit,” describes the MCF5407 multiply/accumulate unit, which executes integer multiply, multiply-accumulate, and miscellaneous register instructions. The MAC is integrated into the operand execution pipeline (OEP).
- Chapter 4, “Local Memory.” This chapter describes the MCF5407 implementation of the ColdFire V4 local memory specification. It consists of the two following major sections.
  - Section 4.2, “SRAM Overview,” describes the MCF5407 on-chip static RAM (SRAM) implementation. It covers general operations, configuration, and initialization. It also provides information and examples showing how to minimize power consumption when using the SRAM.
  - Section 4.7, “Cache Overview,” describes the MCF5407 cache implementation, including organization, configuration, and coherency. It describes cache operations and how the cache interacts with other memory structures.

- Chapter 5, “Debug Support,” describes the Revision C enhanced hardware debug support in the MCF5407. This revision of the ColdFire debug architecture encompasses earlier revisions.

## Suggested Reading

The following literature may be helpful with respect to the topics in Part I:

- *ColdFire Programmers Reference Manual, R1.0* (MCF5200PRM/AD)
- *Using Microprocessors and Microcomputers: The Motorola Family,* William C. Wray, Ross Bannatyne, Joseph D. Greenfield

## Acronyms and Abbreviations

Table I-i contains acronyms and abbreviations are used in Part I.

**Table I-i. 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 |
| EDO | Extended data output (DRAM) |
| FIFO | First-in, first-out |
| GPIO | General-purpose I/O |
| $I^2C$ | 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 |
| LIFO | Last-in, first-out |

## Table I-i. Acronyms and Abbreviated Terms (Continued)

| Term | Meaning |
|---|---|
| LRU | Least recently used |
| LSB | Least-significant byte |
| lsb | Least-significant bit |
| MAC | Multiple accumulate unit |
| 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 |
| PLL | Phase-locked loop |
| PLRU | Pseudo least recently used |
| POR | Power-on reset |
| PQFP | Plastic quad flat pack |
| RISC | Reduced instruction set computing |
| Rx | Receive |
| SIM | System integration module |
| SOF | Start of frame |
| TAP | Test access port |
| TTL | Transistor-to-transistor logic |
| Tx | Transmit |
| UART | Universal asynchronous/synchronous receiver transmitter |

# Chapter 2
# ColdFire Core

This chapter provides an overview of the microprocessor core of the MCF5407. The chapter begins with a description of enhancements from the Version 3 (V3) ColdFire core, and then fully describes the V4 programming model as it is implemented on the MCF5407. It also includes a full description of exception handling, data formats, an instruction set summary, and a table of instruction timings.

## 2.1 Features and Enhancements

The MCF5407 is the first standard product to contain a Version 4 ColdFire microprocessor core. To create this next-generation, high-performance core, many advanced microarchitectural techniques were implemented. Most notable are a Harvard memory architecture, branch cache acceleration logic, and limited superscalar dual-instruction issue capabilities, which together provide 233 (Dhrystone 2.1) MIPS performance at 162 MHz.

The MCF5407 core design emphasizes performance and backward compatibility, and represents the next step on the ColdFire performance roadmap.

The following list summarizes MCF5407 features:

- Variable-length RISC, clock-multiplied Version 4 microprocessor core
- Revision B of the ColdFire instruction set architecture provides new instructions to improve performance and code density
- Two independent, decoupled pipelines—four-stage instruction fetch pipeline (IFP) and five-stage operand execution pipeline (OEP) for increased performance of conditional branch instructions
- Ten-instruction FIFO buffer provides decoupling between the pipelines
- Limited superscalar design approaches dual-issue performance
- Sophisticated two-level branch acceleration mechanism with a branch cache plus a prediction table for increased performance of conditional Bcc instructions
- 32-bit internal address bus supporting 4 Gbytes of linear address space
- 32-bit data bus
- 16 user-accessible, 32-bit-wide, general-purpose registers
- Supervisor/user modes for system protection

- Vector base register to relocate exception-vector table
- Optimized for high-level language constructs

### 2.1.1 Clock-Multiplied Microprocessor Core

The MCF5407 incorporates a clock-multiplying phase-locked loop (PLL). Increasing the internal speed of the core also allows higher performance while providing the system designer with an easy-to-use lower speed system interface.

The frequency of the processor complex is an integer multiple of the external bus speed. Chapter 20, “Electrical Specifications,” lists the supported clock ratios.

The processor, instruction and data caches, integrated SRAMs, and misalignment module operate at the higher speed clock (PCLK); other system integrated modules operate at the speed of the input clock (CLKIN). When combined with the enhanced pipeline structure of the Version 4 ColdFire core, the processor and its local memories provide a high level of performance for today’s demanding embedded applications.

PCLK can be disabled to minimize dissipation when a low-power mode is entered. This is described in Section 7.2.3, “Reduced-Power Mode.”

### 2.1.2 Enhanced Pipelines

The IFP prefetches instructions. The OEP decodes instructions, fetches required operands, then executes the specified function. The two independent, decoupled pipeline structures maximize performance while minimizing core size. Pipeline stages are shown in Figure 2-1 and are summarized as follows:

- Four-stage IFP (plus optional instruction buffer stage)
  - Instruction address generation (IAG) calculates the next prefetch address.
  - Instruction fetch cycle 1 (IC1) initiates prefetch on the processor’s local instruction bus.
  - Instruction fetch cycle 2 (IC2) completes prefetch on the processor’s instruction local bus.
  - Instruction early decode (IED) generates time-critical decode signals needed for the OEP.
  - Instruction buffer (IB) optional stage uses FIFO queue to minimize effects of fetch latency.
- Five-stage OEP with two optional processor bus write cycles.
  - Decode stage (DS/secDS) decodes and selects for two sequential instructions.
  - Operand address generation (OAG) generates the address for the data operand.
  - Operand fetch cycle 1 and 2 (OC1 and OC2) fetch data operands.
  - Execute (EX) performs prescribed operations on previously fetched data operands.

— Write data available (DA) makes data available for operand write operations only.
— Store data (ST) updates memory element for operand write operations only.

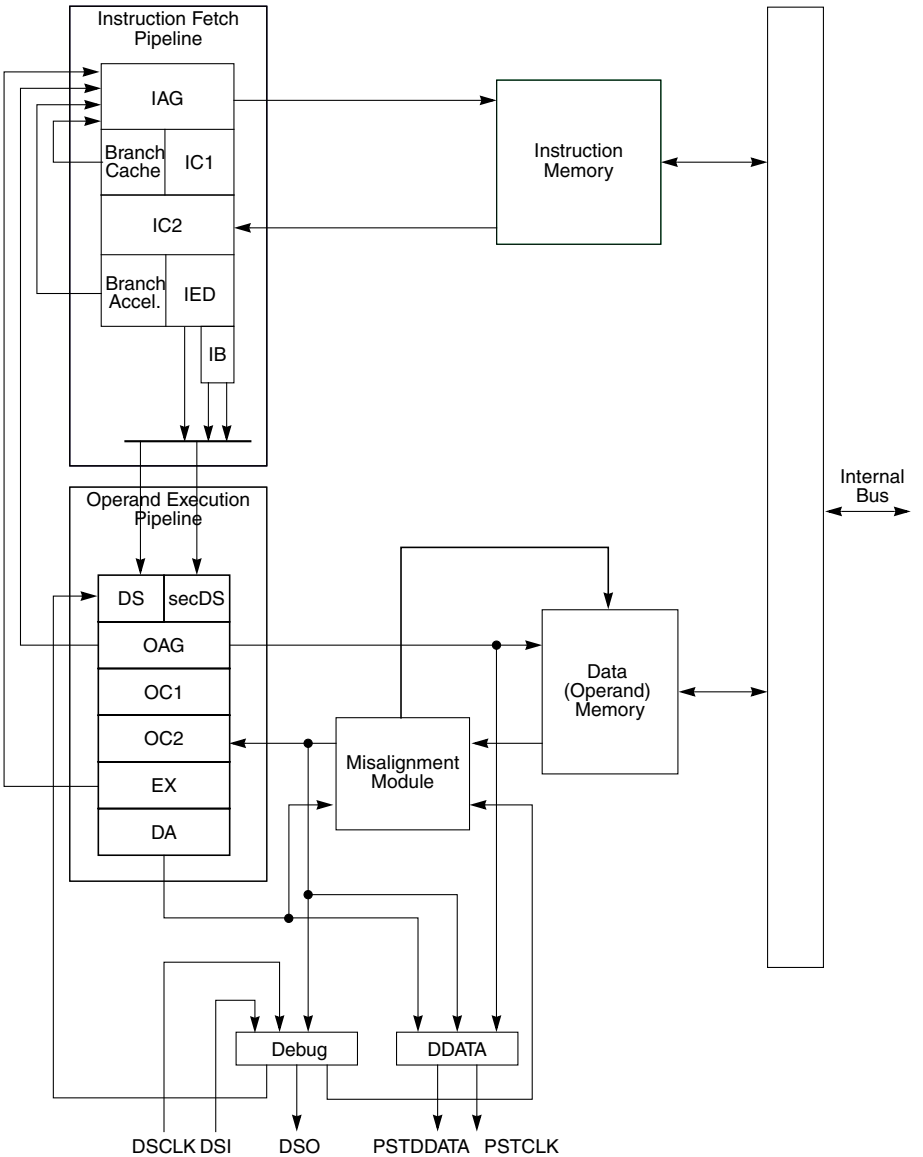



**Figure 2-1. ColdFire Enhanced Pipeline**

### 2.1.2.1 Instruction Fetch Pipeline (IFP)

Because the fetch and execution pipelines are decoupled by a ten-instruction FIFO buffer, the IFP can prefetch instructions before the OEP needs them, minimizing stalls.

#### 2.1.2.1.1 Branch Acceleration

To maximize the performance of conditional branch instructions, the IFP implements a sophisticated two-level acceleration mechanism. The first level is an 8-entry, direct-mapped branch cache with 2 bits for indicating four prediction states (strongly/weakly taken/not-taken) for each entry. The branch cache also provides the association between instruction addresses and the corresponding target address. In the event of a branch cache hit, if the branch is predicted as taken, the branch cache sources the target address from the IC1 stage back into the IAG to redirect the prefetch stream to the new location.

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

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

### 2.1.2.2 Operand Execution Pipeline (OEP)

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

The compute engine at the top of the OEP (the address ALU) is used typically for operand address calculations; the execution ALU at the bottom is used for instruction execution. The resulting structure provides 4 Gbytes/S operand bandwidth (at 162 MHz) to the two compute engines and supports single-cycle execution speeds for most instructions, including all load and store operations and most embedded-load operations. The V4 OEP supports the ColdFire Revision B instruction set, which adds a few new instructions to improve performance and code density.

The OEP also implements the following advanced performance features:

- Stalls are minimized by dynamically basing the choice between the address ALU or execution ALU for instruction execution on the pipeline state.
- The address ALU and register renaming resources together can execute heavily used opcodes and forward results to subsequent instructions with no pipeline stalls.

- Instruction folding involving MOVE instructions allows two instructions to be issued in one cycle. The resulting microarchitecture approaches full superscalar performance at a much lower silicon cost.

#### 2.1.2.2.1 Illegal Opcode Handling

To aid in conversion from M68000 code, every 16-bit operation word is decoded to ensure that each instruction is valid. If the processor attempts execution of an illegal or unsupported instruction, an illegal instruction exception (vector 4) is taken.

#### 2.1.2.2.2 Hardware Multiply/Accumulate (MAC) Unit

The MAC is an optional unit in Version 4 that provides hardware support for a limited set of digital signal processing (DSP) operations used in embedded code, while supporting the integer multiply instructions in the ColdFire microprocessor family. The MAC features a three-stage execution pipeline, optimized for 16 x 16 multiplies. It is tightly coupled to the OEP, which can issue a 16 x 16 multiply with a 32-bit accumulation plus fetch a 32-bit operand in a single cycle. A 32 x 32 multiply with a 32-bit accumulation requires three cycles before the next instruction can be issued.

Figure 2-2 shows basic functionality of the MAC. A full set of instructions are provided for signed and unsigned integers plus signed, fixed-point fractional input operands.

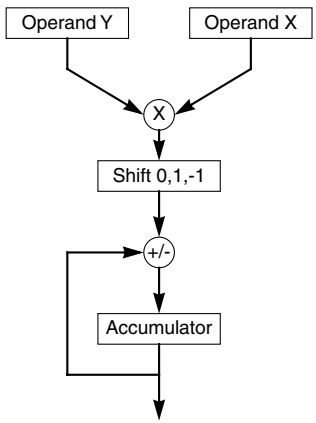



**Figure 2-2. ColdFire Multiply-Accumulate Functionality Diagram**

The MAC provides functionality in the following three related areas, which are described in detail in Chapter 3, “Hardware Multiply/Accumulate (MAC) Unit.”

- Signed and unsigned integer multiplies
- Multiply-accumulate operations with signed and unsigned fractional operands
- Miscellaneous register operations

#### 2.1.2.2.3 Hardware Divide Unit

The hardware divide unit performs the following integer division operations:

- 32-bit operand/16-bit operand producing a 16-bit quotient and a 16-bit remainder
- 32-bit operand/32-bit operand producing a 32-bit quotient
- 32-bit operand/32-bit operand producing a 32-bit remainder

### 2.1.2.3 Harvard Memory Architecture

A Harvard memory architecture supports the increased bandwidth requirements of the V4 processor pipelines by providing separate configuration, access control, and protection resources for data (operand) and instruction memory. The MCF5407 has separate instruction and data buses to processor-local memories, eliminating conflicts between instruction fetches and operand accesses.

## 2.1.3 Debug Module Enhancements

The ColdFire processor core debug interface supports system integration in conjunction with low-cost development tools. Real-time trace and debug information can be accessed through a standard interface, which allows the processor and system to be debugged at full speed without costly in-circuit emulators. The MCF5407 debug unit is a compatible upgrade to MCF52xx and MCF53xx debug modules with added breakpoint registers and support for I/O interrupt request servicing while in emulator mode.

On-chip breakpoint resources include the following:

- Configuration/status register (CSR)
- Background debug mode (BDM) address attributes register (BAAR)
- Bus attributes and mask registers (AATR and AATR1)
- Breakpoint registers. These can be used to define triggers combining address, data, and PC conditions in single- or dual-level definitions. They include the following:
  - Four PC breakpoint registers (PBR, PBR1, PBR2, and PBR3)
  - PC breakpoint mask register (PBMR)
  - Two pairs of data operand address breakpoint registers (ABHR/ABLR and ABLR1/ABHR1)
  - Data breakpoint registers (DBR and DBR1)
  - Data breakpoint mask registers (DBMR and DBMR1)
- Trigger event registers. These can be programmed to generate a processor halt or initiate a debug interrupt exception. They include the following:
  - Trigger definition register (TDR)
  - Extended trigger definition register (XTDR)

These registers can be accessed through the dedicated debug serial communication channel, or from the processor's supervisor programming model, using the WDEBUG instruction.

The MCF5407's new interrupt servicing options during emulator mode allow real-time critical interrupt service routines to be serviced while processing a debug interrupt event, thereby ensuring that the system continues to operate even during debugging.

To support program trace, the Version 4 debug module combines the processor status and debug data outputs into a single 8-bit bus, PSTDDATA[7:0]. This bus along with 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 enhancements of the Revision C debug specification are fully backward-compatible with the A and B revisions. For more information, see Chapter 5, "Debug Support."

## 2.2 Programming Model

The MCF5407 programming model consists of three instruction and register groups—user, MAC (also user-mode), and supervisor, shown in Figure 2-2. User mode programs are restricted to user and MAC instructions and programming models. Supervisor-mode system software can reference all user-mode and MAC instructions and registers and additional supervisor instructions and control registers. The user or supervisor programming model is selected based on SR[S]. The following sections describe the registers in the user, MAC, and supervisor programming models.

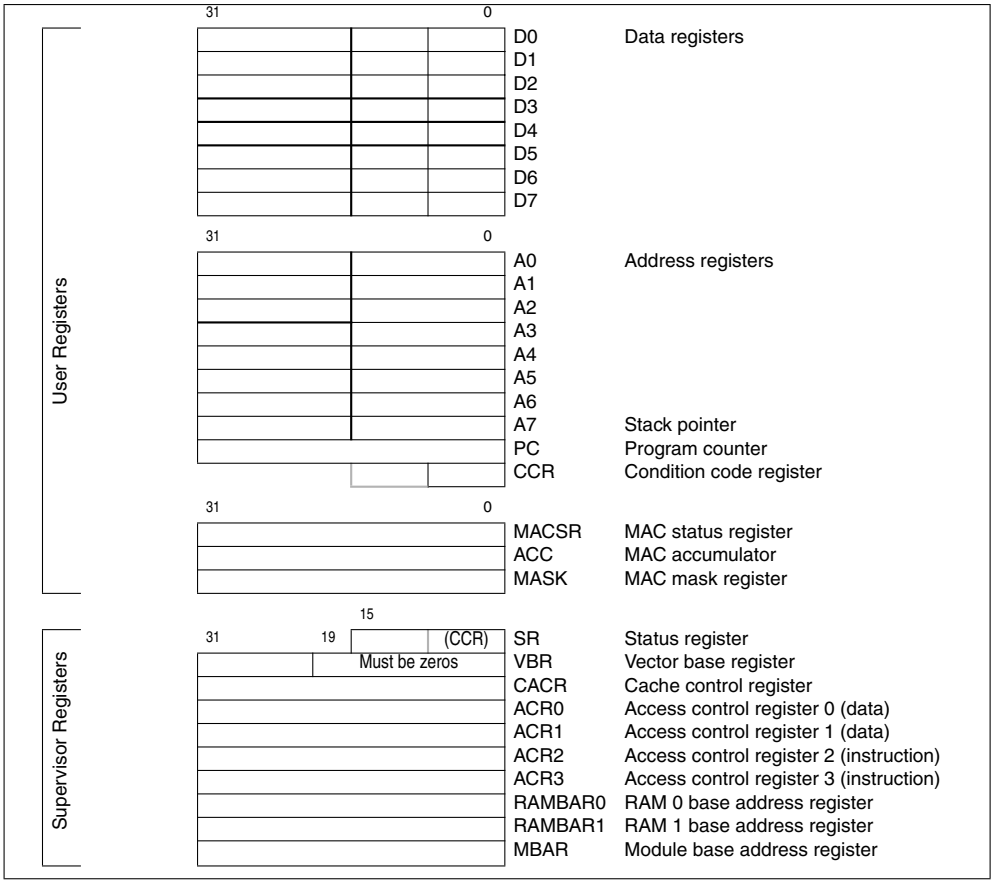



**Figure 2-3. ColdFire Programming Model**

## 2.2.1 User Programming Model

As Figure 2-3 shows, the user programming model consists of the following registers:

- 16 general-purpose 32-bit registers, D0–D7 and A0–A7
- 32-bit program counter
- 8-bit condition code register

### 2.2.1.1 Data Registers (D0–D7)

Registers D0–D7 are used as data registers for bit, byte (8-bit), word (16-bit), and longword (32-bit) operations. They may also be used as index registers.

### 2.2.1.2 Address Registers (A0–A6)

The address registers (A0–A6) can be used as software stack pointers, index registers, or base address registers and may be used for word and longword operations.

### 2.2.1.3 Stack Pointer (A7, SP)

The processor core supports a single hardware stack pointer (A7) used during stacking for subroutine calls, returns, and exception handling. The stack pointer is implicitly referenced by certain operations and can be explicitly referenced by any instruction specifying an address register. The initial value of A7 is loaded from the reset exception vector, address 0x0000. The same register is used for user and supervisor modes, and may be used for word and longword operations.

A subroutine call saves the program counter (PC) on the stack and the return restores the PC from the stack. The PC and the status register (SR) are saved on the stack during exception and interrupt processing. The return from exception instruction restores SR and PC values from the stack.

### 2.2.1.4 Program Counter (PC)

The PC holds the address of the executing instruction. For sequential instructions, the processor automatically increments PC. When program flow changes, the PC is updated with the target instruction. For some instructions, the PC specifies the base address for PC-relative operand addressing modes.

### 2.2.1.5 Condition Code Register (CCR)

The CCR, Figure 2-4, occupies SR[7–0], as shown in Figure 2-3. CCR[4–0] are indicator flags based on results generated by arithmetic operations.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field | — | | | X | N | Z | V | C |
| Reset | Undefined | | | | | | | |
| R/W | R | | | R/W | R/W | R/W | R/W | R/W |

**Figure 2-4. Condition Code Register (CCR)**

Table 2-1 describes the CCR fields.

**Table 2-1. CCR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–5 | — | Reserved, should be cleared. |
| 4 | X | Extend condition code bit. Assigned the value of the carry bit for arithmetic operations; otherwise not affected or set to a specified result. Also used as an input operand for multiple-precision arithmetic. |
| 3 | N | Negative condition code bit. Set if the msb 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 data operand msb occurs for an addition or if a borrow occurs in a subtraction; otherwise cleared. |

#### 2.2.1.6 MAC Programming Model

Figure 2-3 shows the registers in the MAC portion of the user programming model. These registers are described as follows:

- Accumulator (ACC)—This 32-bit, read/write, general-purpose register is used to accumulate the results of MAC operations.
- Mask register (MASK)—This 16-bit general-purpose register provides an optional address mask for MAC instructions that fetch operands from memory. It is useful in the implementation of circular queues in operand memory.
- MAC status register (MACSR)—This 8-bit register defines configuration of the MAC unit and contains indicator flags affected by MAC instructions. Unless noted otherwise, MACSR indicator flag settings are based on the final result, that is, the result of the final operation involving the product and accumulator.

### 2.2.2 Supervisor Programming Model

The MCF5407 supervisor programming model is shown in Figure 2-3. Typically, system programmers use the supervisor programming model to implement operating system functions and provide memory and I/O control. The supervisor programming model provides access to the user registers and additional supervisor registers, which include the upper byte of the status register (SR), the vector base register (VBR), and registers for configuring attributes of the address space connected to the Version 4 processor core. Most supervisor-level registers are accessed by using the MOVEC instruction with the control register definitions in Table 2-2.

**Table 2-2. MOVEC Register Map**

| Rc[11–0] | Register Definition |
|---|---|
| 0x002 | Cache control register (CACR) |
| 0x004 | Access control register 0 (ACR0) |
| 0x005 | Access control register 1 (ACR1) |
| 0x006 | Access control register 2 (ACR2) |
| 0x007 | Access control register 3 (ACR3) |
| 0x801 | Vector base register (VBR) |
| 0xC04 | RAM base address register 0 (RAMBAR0) |
| 0xC05 | RAM base address register 1 (RAMBAR1) |
| 0xC0F | Module base address register (MBAR) |

### 2.2.2.1 Status Register (SR)

The SR stores the processor status, the interrupt priority mask, and other control bits. Supervisor software can read or write the entire SR; user software can read or write only SR[7–0], described in Section 2.2.1.5, “Condition Code Register (CCR).” The control bits indicate processor states—trace mode (T), supervisor or user mode (S), and master or interrupt state (M). SR is set to 0x27*xx* after reset.

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | System byte | | | | | | | | Condition code register (CCR) | | | | | | | |
| Field | T | — | S | M | — | I | | | — | | | X | N | Z | V | C |
| Reset | 0 | 0 | 1 | 0 | 0 | 111 | | | 000 | | | — | — | — | — | — |
| R/W | R/W | R | R/W | R/W | R | R/W | | | R | | | R/W | R/W | R/W | R/W | R/W |

**Figure 2-5. Status Register (SR)**

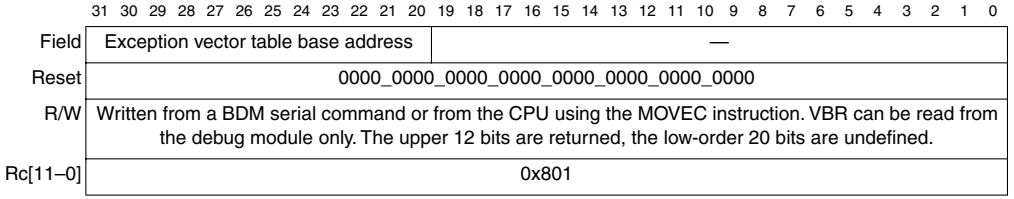
Table 2-3 describes SR fields.

**Table 2-3. Status Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 15 | T | Trace enable. When T is set, the processor performs a trace exception after every instruction. |
| 13 | S | Supervisor/user state. Indicates whether the processor is in supervisor or user mode<br>0 User mode<br>1 Supervisor mode |
| 12 | M | Master/interrupt state. Cleared by an interrupt exception. It can be set by software during execution of the RTE or move to SR instructions so the OS can emulate an interrupt stack pointer. |
| 10–8 | I | Interrupt priority mask. Defines the current interrupt priority. Interrupt requests are inhibited for all priority levels less than or equal to the current priority, except the edge-sensitive level-7 request, which cannot be masked. |
| 7–0 | CCR | Condition code register. See Table 2-1. |

### 2.2.2.2 Vector Base Register (VBR)

The VBR holds the base address of the exception vector table in memory. The displacement of an exception vector is added to the value in this register to access the vector table. VBR[19–0] are not implemented and are assumed to be zero, forcing the vector table to be aligned on a 0-modulo-1-Mbyte boundary.



| | 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 |
|---|---|---|
| Field | Exception vector table base address | — |
| Reset | 0000_0000_0000_0000_0000_0000_0000_0000 | |
| R/W | Written from a BDM serial command or from the CPU using the MOVEC instruction. VBR can be read from the debug module only. The upper 12 bits are returned, the low-order 20 bits are undefined. | |
| Rc[11–0] | 0x801 | |

**Figure 2-6. Vector Base Register (VBR)**

### 2.2.2.3 Cache Control Register (CACR)

The CACR controls operation of both the instruction and data cache memory. It includes bits for enabling, freezing, and invalidating cache contents. It also includes bits for defining the default cache mode and write-protect fields. See Section 4.10.1, “Cache Control Register (CACR).”

### 2.2.2.4 Access Control Registers (ACR0–ACR3)

The access control registers (ACR0–ACR3) define attributes for four user-defined memory regions. ACR0 and ACR1 control data memory space and ACR2 and ACR3 control instruction memory space. Attributes include definition of cache mode, write protect and buffer write enables. See Section 4.10.2, “Access Control Registers (ACR0–ACR3).”

### 2.2.2.5 RAM Base Address Registers (RAMBAR0 and RAMBAR1)

The RAMBAR registers determine the base address location of the internal SRAM modules and indicate the types of references mapped to each. Each RAMBAR includes a base address, write-protect bit, address space mask bits, and an enable. The RAM base address must be aligned on a 0-module-2-Kbyte boundary. See Section 4.4.1, “SRAM Base Address Registers (RAMBAR0/RAMBAR1).”

### 2.2.2.6 Module Base Address Register (MBAR)

The module base address register (MBAR) defines the logical base address for the memory-mapped space containing the control registers for the on-chip peripherals. See Section 6.2.2, “Module Base Address Register (MBAR).”

## 2.3 Integer Data Formats

Table 2-4 lists the integer operand data formats. Integer operands can reside in registers, memory, or instructions. The operand size for each instruction is either explicitly encoded in the instruction or implicitly defined by the instruction operation.

**Table 2-4. Integer Data Formats**

| Operand Data Format | Size |
|---|---|
| Bit | 1 bit |
| Byte integer | 8 bits |
| Word integer | 16 bits |
| Longword integer | 32 bits |

## 2.4 Organization of Data in Registers

The following sections describe data organization within the data, address, and control registers.

### 2.4.1 Organization of Integer Data Formats in Registers

Figure 2-7 shows the integer format for data registers. Each integer data register is 32 bits wide. Byte and word operands occupy the lower 8- and 16-bit portions of integer data registers, respectively. Longword operands occupy the entire 32 bits of integer data registers. A data register that is either a source or destination operand only uses or changes the appropriate lower 8 or 16 bits in byte or word operations, respectively. The remaining high-order portion does not change. The least significant bit (lsb) of all integer sizes is zero, the most-significant bit (msb) of a longword integer is 31, the msb of a word integer is 15, and the msb of a byte integer is 7.

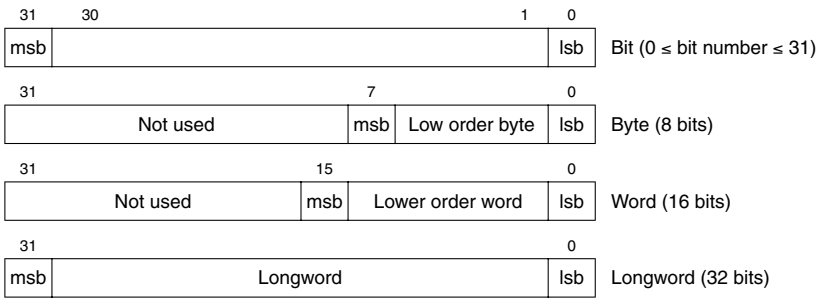



**Figure 2-7. Organization of Integer Data Formats in Data Registers**

The instruction set encodings do not allow the use of address registers for byte-sized operands. When an address register is a source operand, either the low-order word or the entire longword operand is used, depending on the operation size. Word-length source

operands are sign-extended to 32 bits and then used in the operation with anaddress register destination. When an address register is a destination, the entire register is affected, regardless of the operation size. Figure 2-8 shows integer formats for address registers.

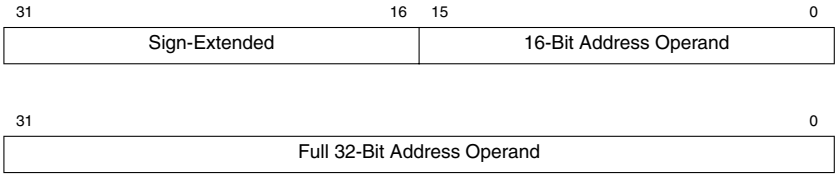



**Figure 2-8. Organization of Integer Data Formats in Address Registers**

The size of control registers varies according to function. Some have undefined bits reserved for future definition by Motorola. Those particular bits read as zeros and must be written as zeros for future compatibility.

All operations to the SR and CCR are word-size operations. For all CCR operations, the upper byte is read as all zeros and is ignored when written, regardless of privilege mode.

## 2.4.2 Organization of Integer Data Formats in Memory

All ColdFire processors use a big-endian addressing scheme. The byte-addressable organization of memory allows lower addresses to correspond to higher order bytes. The address N of a longword data item corresponds to the address of the high-order word. The lower order word is located at address N + 2. The address N of a word data item corresponds to the address of the high-order byte. The lower order byte is located at address N + 1. This organization is shown in Figure 2-9.

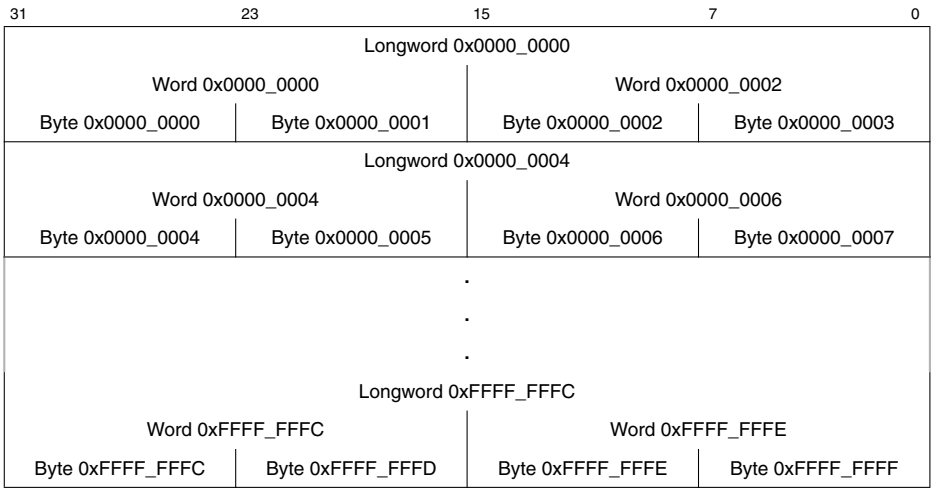



**Figure 2-9. Memory Operand Addressing**

## 2.5 Addressing Mode Summary

Addressing modes are categorized by how they are used. Data addressing modes refer to data operands. Memory addressing modes refer to memory operands. Alterable addressing modes refer to alterable (writable) data operands. Control addressing modes refer to memory operands without an associated size.

These categories sometimes combine to form more restrictive categories. Two combined classifications are alterable memory (both alterable and memory) and data alterable (both alterable and data). Twelve of the most commonly used effective addressing modes from the M68000 Family are available on ColdFire microprocessors. Table 2-5 summarizes these modes and their categories.

**Table 2-5. ColdFire Effective Addressing Modes**

| Addressing Modes | Syntax | Mode Field | Reg. Field | Category | | | |
|---|---|---|---|---|---|---|---|
| | | | | Data | Memory | Control | Alterable |
| Register direct<br>Data<br>Address | <br>Dn<br>An | <br>000<br>001 | <br>reg. no.<br>reg. no. | <br>X<br>— | <br>—<br>— | <br>—<br>— | <br>X<br>X |
| Register indirect<br>Address<br>Address with Postincrement<br>Address with Predecrement<br>Address with Displacement | <br>(An)<br>(An)+<br>–(An)<br>($d_{16}$, An) | <br>010<br>011<br>100<br>101 | <br>reg. no.<br>reg. no.<br>reg. no.<br>reg. no. | <br>X<br>X<br>X<br>X | <br>X<br>X<br>X<br>X | <br>X<br>—<br>—<br>X | <br>X<br>X<br>X<br>X |
| Address register indirect with scaled index 8-bit displacement | ($d_8$, An, Xi*SF) | 110 | reg. no. | X | X | X | X |
| Program counter indirect with displacement | ($d_{16}$, PC) | 111 | 010 | X | X | X | — |
| Program counter indirect with scaled index 8-bit displacement | ($d_8$, PC, Xi*SF) | 111 | 011 | X | X | X | — |
| Absolute data addressing<br>Short<br>Long | <br>(xxx).W<br>(xxx).L | <br>111<br>111 | <br>000<br>001 | <br>X<br>X | <br>X<br>X | <br>X<br>X | <br>—<br>— |
| Immediate | #<xxx> | 111 | 100 | X | X | — | — |

## 2.6 Instruction Set Summary

The ColdFire instruction set is a simplified version of the M68000 instruction set. The removed instructions include BCD, bit field, logical rotate, decrement and branch, and integer multiply with a 64-bit result. Nine new MAC instructions have been added.

Table 2-6 lists notational conventions used throughout this manual.

## Table 2-6. 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) |
| **Register Names** | |
| ACC | MAC accumulator register |
| CCR | Condition code register (lower byte of SR) |
| MACSR | MAC status register |
| MASK | MAC mask register |
| PC | Program counter |
| SR | Status register |
| **Port Name** | |
| PSTDDATA | Processor status/debug data port |
| **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 |

## Table 2-6. Notational Conventions (Continued)

| Instruction | Operand Syntax |
|---|---|
| <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) |
| **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 the condition is true, the operations in the then clause are performed. If the condition is false and the optional else clause is present, the operations in the else clause are performed. If the condition is false and the else clause 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 |

**Table 2-6. Notational Conventions (Continued)**

| Instruction | Operand Syntax |
|---|---|
| **Condition Code Register Bit Names** | |
| C | Carry |
| N | Negative |
| V | Overflow |
| X | Extend |
| Z | Zero |

## 2.6.1 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.

The original ISA definition minimized the support for instructions referencing byte and word operands. Full support for the MOVE.B and MOVEC.W instructions was provided, but **clr** (clear) and **tst** (test) are the only other opcodes supporting these data types. Based on input from compiler writers and system users, a set of instruction enhancements was proposed that address the two following areas:

- Enhanced support for byte and word-sized operands through new move operations
- Enhanced support for position-independent code

The following list summarizes new and enhanced instructions of Revision_B ISA:

- New instructions:
  - INTOUCH loads blocks of instructions to be locked in the instruction cache
  - MOV3Q.L moves 3-bit immediate data to the destination location
  - MVS.{B,W} sign-extends the source operand and moves it to the destination register
  - MVZ.{B,W} zero-fills the source operand and moves it to the destination register
  - SATS.L updates the destination register depending on CCR[V] and bit 31 of the register
  - TAS.B performs an indivisible read-modify-write cycle to test and set the addressed memory byte.

- Enhancements to existing Revision_A instructions:
  - Longword support for branch instructions (Bcc, BRA, BSR)
  - Byte and word support for compare instructions (CMP, CMPI)
  - Word support for the compare address register instruction (CMPA)
  - Byte and longword support for MOVE.x ,where the source is immediate data and the destination is specified by d16(Ax); that is, MOVE.{B,W} #<data>, d16(Ax)

Table 2-7 shows the syntax for the new and enhanced instructions, which are fully described in Section 2.9, “ColdFire Instruction Set Architecture Enhancements.”

**Table 2-7. ColdFire ISA_B Extension Summary**

| Instruction | Mnemonic[1] | Source | Destination | 68K |
|---|---|---|---|---|
| Branch Always | bra.l | | <label> | Yes |
| Branch Conditionally | bcc.l | | <label> | Yes |
| Branch to Subroutine | bsr.l | | <label> | Yes |
| Compare | cmp.{b,w,l} | <ea>y | Dx | Yes |
| Compare Address | cmpa.w | <ea>y | Ax | Yes |
| Compare Immediate | cmpi.{b,w} | #<data> | Dx | Yes |
| Instruction Fetch Touch | intouch | <Ay> | | |
| Move 3-Bit Data Quick | mov3q.l | #<data> | <ea>x | |
| Move Data Source to Destination | move.{b,w} | #<data> | d16(Ax) | Yes |
| Move with Sign Extend | mvs.{b,w} | <ea>y | Dx | |
| Move with Zero-Fill | mvz.{b,w} | <ea>y | Dx | |
| Signed Saturate | sats.l | | Dx | |
| Test and Set an Operand | tas.b | | <ea>x | Yes |

1 Operand sizes in this column reflect only newly supported operand sizes for existing instructions (Bcc, BRA, BSR, CMP, CMPA, CMPI, and MOVE)

Some proposed opcodes were formerly present in the M68000 family, while other opcodes are new functions.

## 2.6.2 Instruction Set Summary

Table 2-8 lists implemented user-level instructions by opcode.

**Table 2-8. User-Level Instruction Set Summary**

| Instruction | Operand Syntax | Operand Size | Operation |
|---|---|---|---|
| ADD | Dy,<ea>x<br><ea>y,Dx | .L<br>.L | Source + destination → destination |
| ADDA | <ea>y,Ax | .L | Source + destination → destination |

## Table 2-8. User-Level Instruction Set Summary (Continued)

| Instruction | Operand Syntax | Operand Size | Operation |
|---|---|---|---|
| ADDI | #<data>,Dx | .L | Immediate data + destination → destination |
| ADDQ | #<data>,<ea>x | .L | Immediate data + destination → destination |
| ADDX | Dy,Dx | .L | Source + destination + X → destination |
| AND | Dy,<ea>x<br><ea>y,Dx | .L<br>.L | Source & destination → destination |
| ANDI | #<data>,Dx | .L | Immediate data & destination → destination |
| ASL | Dy,Dx<br>#<data>,Dx | .L<br>.L | X/C ← (Dx << Dy) ← 0<br>X/C ← (Dx << #<data>) ← 0 |
| ASR | Dy,Dx<br>#<data>,Dx | .L<br>.L | MSB → (Dx >> Dy) → X/C<br>MSB → (Dx >> #<data>) → X/C |
| Bcc | <label> | .B,.W,.L | If condition true, then PC + 2 + $d_n$ → PC |
| BCHG | Dy,<ea>x<br>#<data>,<ea-1>x | .B,.L<br>.B,.L | ~(<bit number> of destination) → Z,<br>Bit of destination |
| BCLR | Dy,<ea>x<br>#<data>,<ea-1>x | .B,.L<br>.B,.L | ~(<bit number> of destination) → Z;<br>0 → bit of destination |
| BRA | <label> | .B,.W,.L | PC + 2 + $d_n$ → PC |
| BSET | Dy,<ea>x<br>#<data>,<ea-1>x | .B,.L<br>.B,.L | ~(<bit number> of destination) → Z;<br>1→ bit of destination |
| BSR | <label> | .B,.W,.L | SP – 4 → SP; next sequential PC→ (SP); PC + 2 + $d_n$ → PC |
| BTST | Dy,<ea>x<br>#<data>,<ea-1>x | .B,.L<br>.B,.L | ~(<bit number> of destination) → Z |
| CLR | <ea>y,Dx | .B,.W,.L | 0 → destination |
| CMP | <ea>y,Ax | .B,.W,.L | Destination – source |
| CMPA | <ea>y,Dx | .W,.L | Destination – source |
| CMPI | <ea>y,Dx | .B,.W,.L | Destination – immediate data |
| DIVS | <ea-1>y,Dx<br><ea>y,Dx | .W<br>.L | Dx /<ea>y → Dx {16-bit remainder; 16-bit quotient}<br>Dx /<ea>y → Dx {32-bit quotient}<br>Signed operation |
| DIVU | <ea-1>y,Dx<br>Dy,<ea>x | .W<br>.L | Dx /<ea>y → Dx {16-bit remainder; 16-bit quotient}<br>Dx /<ea>y → Dx {32-bit quotient}<br>Unsigned operation |
| EOR | Dy,<ea>x | .L | Source ^ destination → destination |
| EORI | #<data>,Dx | .L | Immediate data ^ destination → destination |
| EXT | #<data>,Dx | .B →.W<br>.W →.L | Sign-extended destination → destination |
| EXTB | Dx | .B →.L | Sign-extended destination → destination |
| HALT[1] | None | Unsized | Enter halted state |
| JMP | <ea-3>y | Unsized | Address of <ea> → PC |
| JSR | <ea-3>y | Unsized | SP – 4 → SP; next sequential PC → (SP); <ea> → PC |
| LEA | <ea-3>y,Ax | .L | <ea> → Ax |

## Table 2-8. User-Level Instruction Set Summary (Continued)

| Instruction | Operand Syntax | Operand Size | Operation |
|---|---|---|---|
| LINK | Ax,#<d16> | .W | SP – 4 → SP; Ax → (SP); SP → Ax; SP + d16 → SP |
| LSL | Dy,Dx<br>#<data>,Dx | .L<br>.L | X/C ← (Dx << Dy) ← 0<br>X/C ← (Dx << #<data>) ← 0 |
| LSR | Dy,Dx<br>#<data>,Dx | .L<br>.L | 0 → (Dx >> Dy) → X/C<br>0 → (Dx >> #<data>) → X/C |
| MAC | Ry,RxSF | .L + (.W × .W) → .L<br>.L + (.L × .L) → .L | ACC + (Ry × Rx){<< 1 \| >> 1} → ACC<br>ACC + (Ry × Rx){<< 1 \| >> 1} → ACC; (<ea>y{&MASK}) → Rw |
| MACL | Ry,RxSF,<ea-1>y,Rw | .L + (.W × .W) → .L, .L<br>.L + (.L × .L) → .L, .L | ACC + (Ry × Rx){<< 1 \| >> 1} → ACC<br>ACC + (Ry × Rx){<< 1 \| >> 1} → ACC; (<ea-1>y{&MASK}) → Rw |
| MOV3Q | #<data>,<ea>x | .L | 3-bit immediate→destination |
| MOVE | <ea>y,<ea>x | .B,.W,.L | <ea>y → <ea>x |
| MOVE from MAC | MASK,Rx<br>ACC,Rx<br>MACSR,Rx | .L | Rm → Rx |
| | MACSR,CCR | .L | MACSR → CCR |
| MOVE to MAC | Ry,ACC<br>Ry,MACSR<br>Ry,MASK | .L | Ry → Rm |
| | #<data>,ACC<br>#<data>,MACSR<br>#<data>,MASK | .L | #<data> → Rm |
| MOVE from CCR | CCR,Dx | .W | CCR → Dx |
| MOVE to CCR | Dy,CCR<br>#<data>,CCR | .B | Dy → CCR<br>#<data> → CCR |
| MOVEA | <ea>y,Ax | .W,.L → .L | Source → destination |
| MOVEM | #<list>,<ea-2>x<br><ea-2>y,#<list> | .L<br>.L | Listed registers → destination<br>Source → listed registers |
| MOVEQ | #<data>,Dx | .B → .L | Sign-extended immediate data → destination |
| MSAC | Ry,RxSF | .L - (.W × .W) → .L<br>.L - (.L × .L) → .L | ACC – (Ry × Rx){<< 1 \| >> 1} → ACC |
| MSACL | Ry,RxSF,<ea-1>y,Rw | .L - (.W × .W) → .L, .L<br>.L - (.L × .L) → .L, .L | ACC – (Ry × Rx){<< 1 \| >> 1} → ACC;<br>(<ea-1>y{&MASK}) → Rw |
| MULS | <ea>y,Dx | .W X .W → .L<br>.L X .L → .L | Source × destination → destination<br>Signed operation |
| MULU | <ea>y,Dx | .W X .W → .L<br>.L X .L → .L | Source × destination → destination<br>Unsigned operation |
| MVS | <ea>y,Dx | .B,.W | Sign-extended source → destination |
| MVZ | <ea-1>y,Dx | .B,.W | Zero-filled source → destination |
| NEG | Dx | .L | 0 – destination → destination |
| NEGX | Dx | .L | 0 – destination – X → destination |

**Table 2-8. User-Level Instruction Set Summary (Continued)**

| Instruction | Operand Syntax | Operand Size | Operation |
|---|---|---|---|
| NOP | none | Unsized | Synchronize pipelines; PC + 2 → PC |
| NOT | Dx | .L | ~ Destination → destination |
| OR | <ea>y,Dx<br>Dy,<ea>x | .L | Source \| destination → destination |
| ORI | #<data>,Dx | .L | Immediate data \| destination → destination |
| PEA | <ea-3>y | .L | SP – 4 → SP; Address of <ea> → (SP) |
| PULSE | none | Unsized | Set PST= 0x4 |
| REMS | <ea-1>,Dx | .L | Dx/<ea>y → Dw {32-bit remainder}<br>Signed operation |
| REMU | <ea-1>,Dx | .L | Dx/<ea>y → Dw {32-bit remainder}<br>Unsigned operation |
| RTS | none | Unsized | (SP) → PC; SP + 4 → SP |
| SATS | Dx | .L | If CCR.V=1,<br>then if Dx[31] =0<br>then 0x80000000 → Dx<br>else 0x7FFFFFFF→ Dx<br>else Dx is unchanged |
| Scc | Dx | .B | If condition true, then 1s — destination;<br>Else 0s → destination |
| SUB | <ea>y,Dx<br>Dy,<ea>x | .L<br>.L | Destination – source → destination |
| SUBA | <ea>y,Ax | .L | Destination – source → destination |
| SUBI | #<data>,Dx | .L | Destination – immediate data → destination |
| SUBQ | #<data>,<ea>x | .L | Destination – immediate data → destination |
| SUBX | Dy,Dx | .L | Destination – source – X → destination |
| SWAP | Dx | .W | MSW of Dx ←→ LSW of Dx |
| TAS | <ea>x | .B | Set CCR; 1→ Bit 7 of <ea>x |
| TRAP | #<vector> | Unsized | SP – 4 → SP;PC → (SP);<br>SP – 2 → SP;SR → (SP);<br>SP – 2 → SP; format → (SP);<br>Vector address → PC |
| TRAPF | None<br>#<data> | Unsized<br>.W<br>.L | PC + 2 → PC<br>PC + 4 → PC<br>PC + 6 → PC |
| TST | <ea>y | .B,.W,.L | Set condition codes |
| UNLK | Ax | Unsized | Ax →SP; (SP) → Ax; SP + 4 → SP |
| WDDATA | <ea>y | .B,.W,.L | <ea>y →DDATA port |

[1] By default the HALT instruction is a supervisor-level instruction; however, it can be configured to allow user-mode execution by setting CSR[UHE].

Table 2-9 describes supervisor-level instructions.

**Table 2-9. Supervisor-Level Instruction Set Summary**

| Instruction | Operand Syntax | Operand Size | Operation |
|---|---|---|---|
| CPUSHL | (An) | Unsized | Invalidate instruction cache line<br>Push and invalidate data cache line<br>Push data cache line and invalidate (I,D)-cache lines |
| HALT[1] | none | Unsized | Enter halted state |
| INTOUCH | (Ay) | Unsized | Touch instruction space at address Ay |
| MOVE from SR | SR, Dx | .W | SR → Dx |
| MOVE to SR | Dy,SR<br>#<data>,SR | .W | Source → SR |
| MOVEC | Ry,Rc | .L | Ry → Rc<br>**Rc** **Register Definition**<br>0x002 Cache control register (CACR)<br>0x004 Access control register 0 (ACR0)<br>0x005 Access control register 1 (ACR1)<br>0x006 Access control register 2 (ACR2)<br>0x007 Access control register 3 (ACR3)<br>0x801 Vector base register (VBR)<br>0xC04 RAM base address register 0 (RAMBAR0)<br>0xC05 RAM base address register 1 (RAMBAR1) |
| RTE | None | Unsized | (SP+2) → SR; SP+4 → SP; (SP) → PC; SP + formatfield — SP |
| STOP | #<data> | .W | Immediate data → SR; enter stopped state |
| WDEBUG | <ea-2>y | .L | <ea-2>y → debug module |

[1] The HALT instruction can be configured to allow user-mode execution by setting CSR[UHE].

## 2.7 Execution Timings

The timing data presented in this section assumes the following:

- Execution times are shown for individual instructions without assumptions regarding the OEP's ability to dispatch multiple instructions at a time. For sequences where instruction pairs are issued, the execution time of the two instructions is defined by the execution time of the first instruction; that is, the second instruction effectively executes in zero cycles.
- The OEP is loaded with the opword and all required extension words at the beginning of each instruction execution. This implies that the OEP spends no time waiting for the IFP to supply opwords and/or extension words.
- The OEP experiences no sequence-related pipeline stalls. For the MCF5407, the most common example of such a stall occurs when a register is modified in the EX compute engine and a subsequent instruction generating an address uses the previously modified register. The second instruction stalls in the OEP until the register is updated by the previous instruction. For example:

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

In this sequence, the second instruction is held for three cycles stalling for the multiply instruction to update d0. If consecutive instructions update a register and use that register as a base of index value with a scale factor of 1 (Xi.l*1) in an address calculation, a two-cycle pipeline stall occurs. Using the destination register as an index register with any other scale factor (Xi.l*2, Xi.l*4) causes a three-cycle stall.

Some instructions are optimized to ensure against causing such stalls on subsequent instructions. The destination register on the following instructions is always available for subsequent instructions:

```
lea      <ea>y,Ax
move.l   #<data>,Rx
mov.w    #<data>,Ax
moveq    #<data>,Dx
clr.l    Dx
mov3q.l  #<data>,Rx
<op>     (Ay)+,Rx
<op>     -(Ay),Rx
<op>     Ry,(Ax)+
<op>     Ry,-(Ax)
```

Note that the address register results from postincrement and predecrement modes are available to subsequent instructions without stalls.

- The OEP can complete all memory accesses without memory causing any stall conditions. Thus, timing details in this section assume an infinite zero-wait state memory attached to the core.
- Operand data accesses are assumed to be aligned on the same byte boundary as the operand size:
  - — 16-bit operands aligned on 0-modulo-2 addresses
  - — 32-bit operands aligned on 0-modulo-4 addresses

  Operands not meeting these guidelines are misaligned. Table 2-10 shows how the core decomposes a misaligned operand reference into a series of aligned accesses.

**Table 2-10. Misaligned Operand References**

| A[1:0] | Size | Bus Operations | Additional C(R/W) [1] | |
|---|---|---|---|---|
| | | | Read | Write |
| x1 | Word | Byte, Byte | 2(1/0) | 1(0/1) |
| x1 | Long | Byte, Word, Byte | 3(2/0) | 2(0/2) |
| 10 | Long | Word, Word | 2(1/0) | 1(0/1) |

[1] Each timing entry is presented as C(R/W), described as follows:

C is the number of processor clock cycles, including all applicable operand fetches and writes, as well as 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).

## 2.7.1 MOVE Instruction Execution Timing

Execution timing for the MOVE.{B,W,L} instructions are shown in the next tables. Table 2-13 shows the timing for the other generic move operations.

**NOTE:**

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

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

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

Table 2-11 lists execution times for MOVE.{B,W} instructions.

**Table 2-11. Move Byte and Word Execution Times**

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

Table 2-12 lists timings for MOVE.L.

**Table 2-12. Move Long Execution Times**

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

**Table 2-12. Move Long Execution Times (Continued)**

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

Table 2-13 gives execution times for MOVE.L instructions accessing program-visible registers of the MAC unit, along with other MOVE.L timings. Execution times for moving contents of the ACC or MACSR into a destination location represent the best-case scenario when the store instruction is executed and no load, MAC, or MSAC instructions are in the MAC execution pipeline. In general, these store operations require only one cycle for execution, but if they are preceded immediately by a load, MAC, or MSAC instruction, the MAC pipeline depth is exposed and execution time is 3 cycles.

**Table 2-13. Miscellaneous Move Execution Times**

| Opcode | <ea> | Effective Address | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Rn | (An) | (An)+ | -(An) | (d16,An) | (d8,An,Xi*SF) | (xxx).wl | #<xxx> |
| move.l | <ea>,ACC | 1(0/0) | — | — | — | — | — | — | 1(0/0) |
| move.l | <ea>,MACSR | 6(0/0) | — | — | — | — | — | — | 6(0/0) |
| move.l | <ea>,MASK | 5(0/0) | — | — | — | — | — | — | 5(0/0) |
| move.l | ACC,Rx | 1(0/0) | — | — | — | — | — | — | — |
| move.l | MACSR,CCR | 1(0/0) | — | — | — | — | — | — | — |
| move.l | MACSR,Rx | 1(0/0) | — | — | — | — | — | — | — |
| move.l | MASK,Rx | 1(0/0) | — | — | — | — | — | — | — |
| moveq | #imm,Dx | — | — | — | — | — | — | — | 1(0/0) |
| mov3q | #imm,<ea> | 1(0/0) | 1(1/0) | 1(1/0) | 1(1/0) | 1(1/0) | 2(1/0) | 1(1/0) | — |
| mvs | <ea>,Dx | 1(0/0) | 1(1/0) | 1(1/0) | 1(1/0) | 1(1/0) | 2(1/0) | 1(1/0) | 1(0/0) |
| mvz | <ea>,Dx | 1(0/0) | 1(1/0) | 1(1/0) | 1(1/0) | 1(1/0) | 2(1/0) | 1(1/0) | 1(0/0) |

## 2.7.2 Execution Timings—One-Operand Instructions

Table 2-14 shows standard timings for single-operand instructions.

**Table 2-14. One-Operand Instruction Execution Times**

| Opcode | <ea> | Effective Address | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Rn | (An) | (An)+ | -(An) | (d16,An) | (d8,An,Xi*SF) | (xxx).wl | #xxx |
| 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) | — | — | — | — | — | — | — |
| neg.l | Dx | 1(0/0) | — | — | — | — | — | — | — |
| negx.l | Dx | 1(0/0) | — | — | — | — | — | — | — |
| not.l | Dx | 1(0/0) | — | — | — | — | — | — | — |
| sats.l | Dx | 1(0/0) | — | — | — | — | — | — | — |
| scc | Dx | 1(0/0) | — | — | — | — | — | — | — |
| swap | Dx | 1(0/0) | — | — | — | — | — | — | — |
| tas | <ea> | 1(1/1) | 1(1/1) | 1(1/1) | 1(1/1) | 1(1/1) | 2(1/1) | 1(1/1) | — |
| tst.b | <ea> | 1(0/0) | 1(1/0) | 1(1/0) | 1(1/0) | 1(1/0) | 2(1/0) | 1(1/0) | 1(0/0) |
| tst.w | <ea> | 1(0/0) | 1(1/0) | 1(1/0) | 1(1/0) | 1(1/0) | 2(1/0) | 1(1/0) | 1(0/0) |
| tst.l | <ea> | 1(0/0) | 1(1/0) | 1(1/0) | 1(1/0) | 1(1/0) | 2(1/0) | 1(1/0) | 1(0/0) |

## 2.7.3 Execution Timings—Two-Operand Instructions

Table 2-15 shows standard timings for two-operand instructions.

**Table 2-15. Two-Operand Instruction Execution Times**

| Opcode | <ea> | Effective Address | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Rn | (An) | (An)+ | -(An) | (d16,An) | (d8,An,Xi*SF) | (xxx).wl | #<xxx> |
| add.l | <ea>,Rx | 1(0/0) | 1(1/0) | 1(1/0) | 1(1/0) | 1(1/0) | 2(1/0) | 1(1/0) | 1(0/0) |
| add.l | Dy,<ea> | — | 1(1/1) | 1(1/1) | 1(1/1) | 1(1/1) | 2(1/1) | 1(1/1) | — |
| addi.l | #imm,Dx | 1(0/0) | — | — | — | — | — | — | — |
| addq.l | #imm,<ea> | 1(0/0) | 1(1/1) | 1(1/1) | 1(1/1) | 1(1/1) | 2(1/1) | 1(1/1) | — |
| addx.l | Dy,Dx | 1(0/0) | — | — | — | — | — | — | — |
| and.l | <ea>,Rx | 1(0/0) | 1(1/0) | 1(1/0) | 1(1/0) | 1(1/0) | 2(1/0) | 1(1/0) | 1(0/0) |
| and.l | Dy,<ea> | — | 1(1/1) | 1(1/1) | 1(1/1) | 1(1/1) | 2(1/1) | 1(1/1) | — |
| andi.l | #imm,Dx | 1(0/0) | — | — | — | — | — | — | — |
| asl.l | <ea>,Dx | 1(0/0) | — | — | — | — | — | — | 1(0/0) |
| asr.l | <ea>,Dx | 1(0/0) | — | — | — | — | — | — | 1(0/0) |
| bchg | Dy,<ea> | 2(0/0) | 2(1/1) | 2(1/1) | 2(1/1) | 2(1/1) | 3(1/1) | 2(1/1) | — |

## Table 2-15. Two-Operand Instruction Execution Times (Continued)

| Opcode | <ea> | Effective Address | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Rn | (An) | (An)+ | -(An) | (d16,An) | (d8,An,Xi*SF) | (xxx).wl | #<xxx> |
| bchg | #imm,<ea> | 2(0/0) | 2(1/1) | 2(1/1) | 2(1/1) | 2(1/1) | — | — | — |
| bclr | Dy,<ea> | 2(0/0) | 2(1/1) | 2(1/1) | 2(1/1) | 2(1/1) | 3(1/1) | 2(1/1) | — |
| bclr | #imm,<ea> | 2(0/0) | 2(1/1) | 2(1/1) | 2(1/1) | 2(1/1) | — | — | — |
| bset | Dy,<ea> | 2(0/0) | 2(1/1) | 2(1/1) | 2(1/1) | 2(1/1) | 3(1/1) | 2(1/1) | — |
| bset | #imm,<ea> | 2(0/0) | 2(1/1) | 2(1/1) | 2(1/1) | 2(1/1) | — | — | — |
| btst | Dy,<ea> | 2(0/0) | 2(1/1) | 2(1/1) | 2(1/1) | 2(1/1) | 3(1/1) | 2(1/1) | — |
| btst | #imm,<ea> | 1(0/0) | 2(1/1) | 2(1/1) | 2(1/1) | 2(1/1) | — | — | 1(0/0) |
| cmp.b | <ea>,Rx | 1(0/0) | 1(1/0) | 1(1/0) | 1(1/0) | 1(1/0) | 2(1/0) | 1(1/0) | 1(0/0) |
| cmp.w | <ea>,Rx | 1(0/0) | 1(1/0) | 1(1/0) | 1(1/0) | 1(1/0) | 2(1/0) | 1(1/0) | 1(0/0) |
| cmp.l | <ea>,Rx | 1(0/0) | 1(1/0) | 1(1/0) | 1(1/0) | 1(1/0) | 2(1/0) | 1(1/0) | 1(0/0) |
| cmpi.b | #imm,Dx | 1(0/0) | — | — | — | — | — | — | — |
| cmpi.w | #imm,Dx | 1(0/0) | — | — | — | — | — | — | — |
| cmpi.l | #imm,Dx | 1(0/0) | — | — | — | — | — | — | — |
| divs.w | <ea>,Dx | 20(0/0) | 20(1/0) | 20(1/0) | 20(1/0) | 20(1/0) | 21(1/0) | 20(1/0) | 20(0/0) |
| divu.w | <ea>,Dx | 20(0/0) | 20(1/0) | 20(1/0) | 20(1/0) | 20(1/0) | 21(1/0) | 20(1/0) | 20(0/0) |
| divs.l | <ea>,Dx | 35(0/0) | 35(1/0) | 35(1/0) | 35(1/0) | 35(1/0) | — | — | — |
| divu.l | <ea>,Dx | 35(0/0) | 35(1/0) | 35(1/0) | 35(1/0) | 35(1/0) | — | — | — |
| eor.l | Dy,<ea> | 1(0/0) | 1(1/1) | 1(1/1) | 1(1/1) | 1(1/1) | 2(1/1) | 1(1/1) | — |
| eori.l | #imm,Dx | 1(0/0) | — | — | — | — | — | — | — |
| lea | <ea>,Ax | — | 1(0/0) | — | — | 1(0/0) | 2(0/0) | 1(0/0) | — |
| lsl.l | <ea>,Dx | 1(0/0) | — | — | — | — | — | — | 1(0/0) |
| lsr.l | <ea>,Dx | 1(0/0) | — | — | — | — | — | — | 1(0/0) |
| mac.w | Ry,Rx | 1(0/0) | — | — | — | — | — | — | — |
| mac.l | Ry,Rx | 3(0/0) | — | — | — | — | — | — | — |
| msac.w | Ry,Rx | 1(0/0) | — | — | — | — | — | — | — |
| msac.l | Ry,Rx | 3(0/0) | — | — | — | — | — | — | — |
| mac.w | Ry,Rx,ea,Rw | — | 1(1/0) | 1(1/0) | 1(1/0) | 1(1/0) | — | — | — |
| mac.l | Ry,Rx,ea,Rw | — | 3(1/0) | 3(1/0) | 3(1/0) | 3(1/0) | — | — | — |
| msac.w | Ry,Rx,ea,Rw | — | 1(1/0) | 1(1/0) | 1(1/0) | 1(1/0) | — | — | — |
| msac.l | Ry,Rx,ea,Rw | — | 3(1/0) | 3(1/0) | 3(1/0) | 3(1/0) | — | — | — |
| muls.w | <ea>,Dx | 3(0/0) | 3(1/0) | 3(1/0) | 3(1/0) | 3(1/0) | 4(1/0) | 3(1/0) | 3(0/0) |
| mulu.w | <ea>,Dx | 3(0/0) | 3(1/0) | 3(1/0) | 3(1/0) | 3(1/0) | 4(1/0) | 3(1/0) | 3(0/0) |
| muls.l | <ea>,Dx | 5(0/0) | 5(1/0) | 5(1/0) | 5(1/0) | 5(1/0) | — | — | — |
| mulu.l | <ea>,Dx | 5(0/0) | 5(1/0) | 5(1/0) | 5(1/0) | 5(1/0) | — | — | — |
| or.l | <ea>,Rx | 1(0/0) | 1(1/0) | 1(1/0) | 1(1/0) | 1(1/0) | 2(1/0) | 1(1/0) | 1(0/0) |

**Table 2-15. Two-Operand Instruction Execution Times (Continued)**

| Opcode | <ea> | Effective Address | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Rn | (An) | (An)+ | -(An) | (d16,An) | (d8,An,Xi*SF) | (xxx).wl | #<xxx> |
| or.l | Dy,<ea> | — | 1(1/1) | 1(1/1) | 1(1/1) | 1(1/1) | 2(1/1) | 1(1/1) | — |
| or.l | #imm,Dx | 1(0/0) | — | — | — | — | — | — | — |
| rems.l | <ea>,Dx | 35(0/0) | 35(1/0) | 35(1/0) | 35(1/0) | 35(1/0) | — | — | — |
| remu.l | <ea>,Dx | 35(0/0) | 35(1/0) | 35(1/0) | 35(1/0) | 35(1/0) | — | — | — |
| sub.l | <ea>,Rx | 1(0/0) | 1(1/0) | 1(1/0) | 1(1/0) | 1(1/0) | 2(1/0) | 1(1/0) | 1(0/0) |
| sub.l | Dy,<ea> | — | 1(1/1) | 1(1/1) | 1(1/1) | 1(1/1) | 2(1/1) | 1(1/1) | — |
| subi.l | #imm,Dx | 1(0/0) | — | — | — | — | — | — | — |
| subq.l | #imm,<ea> | 1(0/0) | 1(1/1) | 1(1/1) | 1(1/1) | 1(1/1) | 2(1/1) | 1(1/1) | — |
| subx.l | Dy,Dx | 1(0/0) | — | — | — | — | — | — | — |

## 2.7.4 Miscellaneous Instruction Execution Times

Table 2-16 lists timings for miscellaneous instructions.

**Table 2-16. Miscellaneous Instruction Execution Times**

| Opcode | <ea> | Effective Address | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Rn | (An) | (An)+ | -(An) | (d16,An) | (d8,An,Xi*SF) | (xxx).wl | #<xxx> |
| cpushl | (Ax) | — | 9(0/1) | — | — | — | — | — | — |
| intouch | (Ay) | — | 19(1/0) | | | | | | |
| link.w | Ay,#imm | 2(0/1) | — | — | — | — | — | — | — |
| move.w | CCR,Dx | 1(0/0) | — | — | — | — | — | — | — |
| move.w | <ea>,CCR | 1(0/0) | — | — | — | — | — | — | 1(0/0) |
| move.w | SR,Dx | 1(0/0) | — | — | — | — | — | — | — |
| move.w | <ea>,SR | 4(0/0) | — | — | — | — | — | — | 4(0/0) |
| movec | Ry,Rc | 20(0/1) | — | — | — | — | — | — | — |
| movem.l [1] | <ea>,&list | — | n(n/0) | — | — | n(n/0) | — | — | — |
| movem.l [1] | &list,<ea> | — | n(0/n) | — | — | n(0/n) | — | — | — |
| nop | | 6(0/0) | — | — | — | — | — | — | — |
| pea | <ea> | — | 1(0/1) | — | — | 1(0/1) | 2(0/1) | 1(0/1) | — |
| pulse | | 1(0/0) | — | — | — | — | — | — | — |
| stop | #imm | — | — | — | — | — | — | — | 6(0/0)[2] |
| trap | #imm | — | — | — | — | — | — | — | 18(1/2) |
| trapf | | 1(0/0) | — | — | — | — | — | — | — |
| trapf.w | | 1(0/0) | — | — | — | — | — | — | — |
| trapf.l | | 1(0/0) | — | — | — | — | — | — | — |

**Table 2-16. Miscellaneous Instruction Execution Times (Continued)**

| Opcode | <ea> | Effective Address | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Rn | (An) | (An)+ | -(An) | (d16,An) | (d8,An,Xi*SF) | (xxx).wl | #<xxx> |
| unlk | Ax | 1(1/0) | — | — | — | — | — | — | — |
| wddata.{b,w,l} | <ea> | — | 1(1/0) | 1(1/0) | 1(1/0) | 1(1/0) | 2(1/0) | 1(1/0) | — |
| wdebug.l | <ea> | — | 15(2/0) | — | — | 15(2/0) | — | — | — |

1 n is the number of registers moved by the MOVEM opcode

2 The execution time for STOP is the time required until the processor begins sampling continuously for interrupts.

## 2.7.5 Branch Instruction Execution Times

Table 2-17 shows general branch instruction timing.

**Table 2-17. Branch Instruction Execution Times**

| Opcode | <ea> | Effective Address | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Rn | (An) | (An)+ | -(An) | (d16,An) | (d8,An,Xi*SF) | (xxx).wl | #<xxx> |
| bra | | — | — | — | — | 1(0/1)[1] | — | — | — |
| bsr | | — | — | — | — | 1(0/1)[1] | — | — | — |
| jmp | <ea> | — | 5(0/0) | — | — | 5(0/0) | 6(0/0) | 1(0/0)[1] | — |
| jsr | <ea> | — | 5(0/1) | — | — | 5(0/1) | 6(0/1) | 1(0/1)[1] | — |
| rte | | — | — | 15(2/0) | — | — | — | — | — |
| rts | | — | — | 2(1/0)[2]<br>9(1/0)[3]<br>8(1/0)[4] | — | — | — | — | — |

1 Assumes branch acceleration.

2 If predicted correctly by the hardware return stack.

3 If mispredicted by the hardware return stack.

4 If not predicted by the hardware return stack.

Table 2-18 shows timing for Bcc instructions.

**Table 2-18. Bcc Instruction Execution Times**

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

## 2.8 Exception Processing Overview

Exception processing for ColdFire processors is streamlined for performance. Differences from previous M68000 Family processors 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 pointer

ColdFire processors use an instruction restart exception model but require more software support to recover from certain access errors. See Table 2-19 for details.

Exception processing can be defined as the time from the detection of the fault condition until the fetch of the first handler instruction has been initiated. It is comprised of the following four major steps:

1. The processor makes an internal copy of the SR and then enters supervisor mode by setting SR[S] and disabling trace mode by clearing SR[T]. The occurrence of an interrupt exception also forces SR[M] to be cleared and the interrupt priority mask to be set to the level of the current interrupt request.
2. 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 a peripheral device. The IACK cycle is mapped to a special acknowledge address space with the interrupt level encoded in the address.
3. The processor saves the current context by creating an exception stack frame on the system stack. ColdFire processors support a single stack pointer in the A7 address register; therefore, there is no notion of separate supervisor and user stack pointers. 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 in the exception stack frame defines the address of the faulting instruction (fault) or of the next instruction to be executed (next).
4. The processor acquires the address of the first instruction of the exception handler. The exception vector table is aligned on a 1-Mbyte boundary. This instruction address is obtained by fetching a value from the table at the address defined in the vector base register. The index into the exception table is calculated as 4 x vector_number. When the index value is generated, the vector table contents determine the address of the first instruction of the desired handler. After the fetch of the first opcode of the handler is initiated, exception processing terminates and normal instruction processing continues in the handler.

ColdFire processors support a 1024-byte vector table aligned on any 1-Mbyte address boundary; see Table 2-19. The table contains 256 exception vectors where the first 64 are

defined by Motorola; the remaining 192 are user-defined interrupt vectors.

**Table 2-19. Exception Vector Assignments**

| Vector Numbers | Vector Offset (Hex) | Stacked Program Counter [1] | Assignment |
|---|---|---|---|
| 0 | 000 | — | Initial stack pointer |
| 1 | 004 | — | Initial program counter |
| 2 | 008 | Fault | Access error |
| 3 | 00C | Fault | Address error |
| 4 | 010 | Fault | Illegal instruction |
| 5 | 014 | Fault | Divide by zero |
| 6–7 | 018–01C | — | Reserved |
| 8 | 020 | Fault | Privilege violation |
| 9 | 024 | Next | Trace |
| 10 | 028 | Fault | Unimplemented line-a opcode |
| 11 | 02C | Fault | Unimplemented line-f opcode |
| 12 | 030 | Next | Non-PC breakpoint debug interrupt |
| 13 | 034 | Next | PC breakpoint debug interrupt |
| 14 | 038 | Fault | Format error |
| 15 | 03C | Next | Uninitialized interrupt |
| 16–23 | 040–05C | — | Reserved |
| 24 | 060 | Next | Spurious interrupt |
| 25–31 | 064–07C | Next | Level 1–7 autovectored interrupts |
| 32–47 | 080–0BC | Next | Trap #0–15 instructions |
| 48–60 | 0C0–0F0 | — | Reserved |
| 61 | 0F4 | Fault | Unsupported instruction |
| 62–63 | 0F8–0FC | — | Reserved |
| 64–255 | 100–3FC | Next | User-defined interrupts |

[1] The term ‘fault’ refers to the PC of the instruction that caused the exception. The term ‘next’ refers to the PC of the instruction that immediately follows the instruction that caused the fault.

ColdFire processors inhibit sampling for interrupts 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.

## 2.8.1 Exception Stack Frame Definition

The exception stack frame is shown in Figure 2-1. The first longword of the exception stack frame contains the 16-bit format/vector word (F/V) and the 16-bit status register. The second longword contains the 32-bit program counter address.

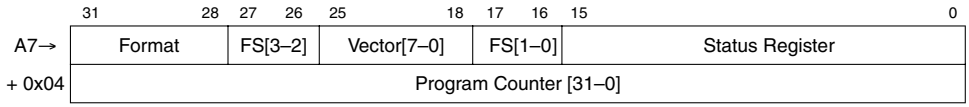



**Figure 2-1. Exception Stack Frame Form**

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

- Format field—This 4-bit field at the top of the system stack is always written with a value of {4,5,6,7} by the processor indicating a 2-longword frame format. See Table 2-20. This field records any longword misalignment of the stack pointer that may have existed when the exception occurred.

**Table 2-20. Format Field Encoding**

| Original A7 at Time of Exception, Bits 1–0 | A7 at First Instruction of Handler | Format Field Bits 31–28 |
|---|---|---|
| 00 | Original A[7–8] | 0100 |
| 01 | Original A[7–9] | 0101 |
| 10 | Original A[7–10] | 0110 |
| 11 | Original A[7–11] | 0111 |

- Fault status field—The 4-bit field, FS[3–0], at the top of the system stack is defined for access and address errors along with interrupted debug service routines. See Table 2-21.

**Table 2-21. Fault Status Encodings**

| FS[3–0] | Definition |
|---|---|
| 0000 | Not an access or address error nor an interrupted debug service routine |
| 0001 | Reserved |
| 0010 | Interrupt during a debug service routine |
| 0011 | Reserved |
| 0100 | Error on instruction fetch |
| 0101–011x | Reserved |
| 1000 | Error on operand write |
| 1001 | Attempted write to write-protected space |
| 101x | Reserved |
| 1100 | Error on operand read |
| 1101–111x | Reserved |

- Vector number—This 8-bit field, vector[7–0], defines the exception type. It is calculated by the processor for internal faults and is supplied by the peripheral for interrupts. See Table 2-19.

## 2.8.2 Processor Exceptions

Table 2-22 describes MCF5407 exceptions.

**Table 2-22. MCF5407 Exceptions**

| Exception | Description |
|---|---|
| Access Error | Access errors are reported only in conjunction with an attempted store to write-protected memory. Thus, access errors associated with instruction fetch or operand read accesses are not possible. The Version 4 processor, unlike the Version 2 and 3 processors, updates the condition code register if a write-protect error occurs during a CLR or MOV3Q operation to memory. |
| Address Error | Caused by an attempted execution transferring control to an odd instruction address (that is, if bit 0 of the target address is set), an attempted use of a word-sized index register (Xi.w) or a scale factor of 8 on an indexed effective addressing mode, or attempted execution of an instruction with a full-format indexed addressing mode.<br>If an address error occurs on a JSR instruction, the Version 4 processor first pushes the return address onto the stack and then calculates the target address. On Version 2 and 3 processors, these functions are reversed.<br>If an address error occurs on an RTS instruction, the Version 4 processor preserves the original return PC and writes the exception stack frame above this value. On Version 2 and 3 processors, the faulting return PC is overwritten by the address error stack frame. |
| Illegal Instruction | On Version 2 ColdFire implementations, only some illegal opcodes were decoded and generated an illegal instruction exception. Version 3 and Version 4 processors decode the full 16-bit opcode and generate this exception if execution of an unsupported instruction is attempted. Additionally, attempting to execute an illegal line A or line F opcode generates unique exception types: vectors 10 and 11, respectively.<br>ColdFire processors do not provide illegal instruction detection on extension words of any instruction, including MOVEC. Attempting to execute an instruction with an illegal extension word causes undefined results. |
| Divide by Zero | Attempted division by zero causes an exception (vector 5, offset = 0x014) except when the PC points to the faulting instruction (DIVU, DIVS, REMU, REMS). |
| Privilege Violation | Caused by attempted execution of a supervisor mode instruction while in user mode. The ColdFire Programmer's Reference Manual lists supervisor- and user-mode instructions. |
| Trace Exception | ColdFire processors provide instruction-by-instruction tracing. While the processor is in trace mode (SR[T] = 1), instruction completion signals a trace exception. This allows a debugger to monitor program execution.<br>The only exception to this definition is the STOP instruction. If the processor is in trace mode, the instruction before the STOP executes and then generates a trace exception. In the exception stack frame, the PC points to the STOP opcode. When the trace handler is exited, the STOP instruction is executed, loading the SR with the immediate operand from the instruction. The processor then generates a trace exception. The PC in the exception stack frame points to the instruction after STOP, and the SR reflects the just-loaded value.<br>If the processor is not in trace mode and executes a STOP instruction where the immediate operand sets the trace bit in the SR, hardware loads the SR and generates a trace exception. The PC in the exception stack frame points to the instruction after STOP, and the SR reflects the just-loaded value.<br>Because ColdFire processors do not support hardware stacking of multiple exceptions, it is the responsibility of the operating system to check for trace mode after processing other exception types. As an example, consider a TRAP instruction executing in trace mode. The processor initiates the TRAP exception and passes control to the corresponding handler. If the system requires that a trace exception be processed, the TRAP exception handler must 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. |

**Table 2-22. MCF5407 Exceptions (Continued)**

| Exception | Description |
|---|---|
| Debug Interrupt | Caused by a hardware breakpoint register trigger. Rather than generating an IACK cycle, the processor internally calculates the vector number (12 or 13, depending on the type of breakpoint trigger). Additionally, the M bit and the interrupt priority mask fields of the SR are unaffected by the interrupt. See Section 2.2.2.1, “Status Register (SR).”<br>The debug interrupt exception vector is expanded from Version 3 such that PC breakpoints are distinguishable from other triggers. The two unique entries occur when a PC breakpoint generates the 0x034 vector. In case of a two-level trigger, the last breakpoint event determines the vector.<br>The changes are described in more detail in Chapter 5, “Debug Support.” |
| RTE and Format Error Exceptions | When an RTE instruction executes, the processor first examines the 4-bit format field to validate the frame type. For a ColdFire processor, any attempted execution of an RTE 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 exception frame and the stacked PC points to RTE.The selection of the format value provides limited debug support for porting code from M68000 applications. On M68000 Family processors, the SR was at the top of the stack. Bit 30 of the longword addressed by the system stack pointer is typically zero; so, attempting an RTE using this old format generates a format error on a ColdFire processor.<br>If the format field defines a valid type, the processor does the following:<br>1 Reloads the SR operand.<br>2 Fetches the second longword operand.<br>3 Adjusts the stack pointer by adding the format value to the auto-incremented address after the first longword fetch.<br>4 Transfers control to the instruction address defined by the second longword operand in the stack frame. |
| TRAP | Executing TRAP always forces an exception and is useful for implementing system calls. The trap instruction may be used to change from user to supervisor mode. |
| Interrupt Exception | Interrupt exception processing, with interrupt recognition and vector fetching, includes uninitialized and spurious interrupts as well as those where the requesting device supplies the 8-bit interrupt vector. Autovectoring may optionally be configured through the system interface module (SIM). See Section 9.2.2, “Autovector Register (AVR).” |
| Reset Exception | Asserting the reset input signal ($\overline{\text{RSTI}}$) causes a reset exception. Reset has the highest exception priority; it provides for system initialization and recovery from catastrophic failure. When assertion of $\overline{\text{RSTI}}$ is recognized, current processing is aborted and cannot be recovered. The reset exception places the processor in supervisor mode by setting SR[S] and disables tracing by clearing SR[T]. This exception also clears SR[M] and sets the processor’s interrupt priority mask in the SR to the highest level (level 7). Next, the VBR is initialized to 0x0000_0000. Configuration registers controlling the operation of all processor-local memories are invalidated, disabling the memories.<br>Note: Other implementation-specific supervisor registers are also affected. Refer to each of the modules in this manual for details on these registers.<br>After $\overline{\text{RSTI}}$ is negated, the processor waits 16 cycles before beginning the actual reset exception process. During this time, certain events are sampled, including the assertion of the debug breakpoint signal. If the processor is not halted, it initiates the reset exception by performing two longword read bus cycles. The longword at address 0 is loaded into the stack pointer and the longword at address 4 is loaded into the PC. 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 executes, the processor enters the fault-on-fault halted state. |
| Unsupported Instruction Exception | If the MCF5407 attempts to execute a valid instruction but the required optional hardware module is not present in the OEP, a non-supported instruction exception is generated (vector 0x61). Control is then passed to an exception handler that can then process the opcode as required by the system. |

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.

## 2.9 ColdFire Instruction Set Architecture Enhancements

This section describes the new opcodes implemented as part of the Revision B enhancements to the basic ColdFire ISA. In some cases, the opcodes represent minor enhancements to existing ColdFire functions, while in other cases, the functionality is new and not covered in the existing ISA.

# Bcc — Branch Conditionally — Bcc

Operation: If Condition True
Then PC + $d_n$ → PC

Assembler Syntax: Bcc <label>

Attributes: Size = byte, word, long

Description: If the condition is true, execution continues at (PC) + displacement. PC holds the address of the instruction word for the Bcc instruction, plus two. The displacement is a two's-complement integer that represents the relative distance in bytes from the current PC to the destination PC. If the 8-bit displacement field is 0, a 16-bit displacement (the word after the instruction) is used. If the 8-bit displacement field is 0xFF, the 32-bit displacement (longword after the instruction) is used. Condition code specifies one of the following tests:

| Code | Condition | Code | Condition | Code | Condition | Code | Condition |
|---|---|---|---|---|---|---|---|
| CC(HI) | Carry clear | GT | Greater than | LT | Less than | VC | Overflow clear |
| CS(LO) | Carry set | HI | High | MI | Minus | VS | Overflow set |
| EQ | Equal | LE | Less or equal | NE | Not equal | | |
| GE | Greater or equal | LS | Low or same | PL | Plus | | |

Condition Codes: Not affected

Instruction Format:

| 15 | 14 | 13 | 12 | 11 10 9 8 | 7 6 5 4 3 2 1 0 |
|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | Condition | 8-bit displacement |
| 16-bit displacement if 8-bit displacement = 0x00 | | | | | |
| 32-bit displacement if 8-bit displacement = 0xFF | | | | | |

Instruction Fields:

- Condition field—Binary code for one of the conditions listed in the table.
- 8-bit displacement field—Two's complement integer specifying the number of bytes between the branch and the next instruction to be executed if the condition is met.
- 16-bit displacement field—Used when the 8-bit displacement contains 0x00.
- 32-bit displacement field—Used when the 8-bit displacement contains 0xFF.

**NOTE:**

A branch to the next immediate instruction uses 16-bit displacement because the 8-bit displacement field is 0x00.

| Bcc | V2, V3 Core | V4 Core |
|---|---|---|
| Opcode present | Yes | Yes |
| Operand sizes supported | .b, .w | .b, .w, .l |

# BRA — Branch Always — BRA

Operation: PC + $d_n$ → PC

Assembler Syntax: BRA <label>

Attributes: Size = byte, word, long

Description: Program execution continues at location (PC) + displacement. The PC contains the address of the instruction word of the BRA instruction, plus two. The displacement is a two's complement integer that represents the relative distance in bytes from the current PC to the destination PC. If the 8-bit displacement field in the instruction word is 0, a 16-bit displacement (the word immediately following the instruction) is used. If the 8-bit displacement field in the instruction word is all ones (0xFF), the 32-bit displacement (longword immediately following the instruction) is used.

Condition codes: Not affected

Instruction Format:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 8-bit displacement | | | | | | | |
| 16-bit displacement if 8-bit displacement = 0x00 | | | | | | | | | | | | | | | |
| 32-bit displacement if 8-bit displacement = 0xFF | | | | | | | | | | | | | | | |

Instruction Fields:

- 8-bit displacement field—Two's complement integer specifying the number of bytes between the branch instruction and the next instruction to be executed.
- 16-bit displacement field—Used for displacement when the 8-bit displacement contains 0x00.
- 32-bit displacement field—Used for displacement when the 8-bit displacement contains 0xFF.

**NOTE:**

A branch to the next immediate instruction automatically uses the 16-bit displacement format because the 8-bit displacement field contains 0x00 (zero offset).

| BRA | V2, V3 Core | V4 Core |
|---|---|---|
| Opcode present | Yes | Yes |
| Operand sizes supported | .b, .w | .b, .w, .l |

# BSR — Branch to Subroutine — BSR

Operation: SP – 4 → SP; PC → (SP); PC + $d_n$ → PC

Assembler Syntax: BSR <label>

Attributes: Size = byte, word, long

Description: Pushes the word address of the instruction immediately following the BSR instruction onto the system stack. The PC contains the address of the instruction word, plus two. Program execution then continues at location (PC) + displacement. The displacement is a two's complement integer that represents the relative distance in bytes from the current PC to the destination PC. If the 8-bit displacement field in the instruction word is 0, a 16-bit displacement (the word immediately following the instruction) is used. If the 8-bit displacement field in the instruction word is all ones (0xFF), the 32-bit displacement (longword immediately following the instruction) is used.

Condition Codes: Not affected

Instruction Format:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 8-bit displacement | | | | | | | |
| 16-bit displacement if 8-bit displacement = 0x00 | | | | | | | | | | | | | | | |
| 32-bit displacement if 8-bit displacement = 0xFF | | | | | | | | | | | | | | | |

Instruction Fields:

- 8-bit displacement field—Two's complement integer specifying the number of bytes between the branch instruction and the next instruction to be executed.
- 16-bit displacement field—Used for displacement when the 8-bit displacement contains 0x00.
- 32-bit displacement field—Used for displacement when the 8-bit displacement contains 0xFF.

**NOTE:**

A branch to the next immediate instruction automatically uses the 16-bit displacement format because the 8-bit displacement field contains 0x00 (zero offset).

| BSR | V2, V3 Core | V4 Core |
|---|---|---|
| Opcode present | Yes | Yes |
| Operand sizes supported | .b, .w | .b, .w, .l |

# CMP — Compare — CMP

Operation: `Destination – Source → cc`

Assembler Syntax: CMP <ea>y, Dx

Attributes: Size = byte, word, long

Description: Subtracts the source operand from the destination operand in the data register and sets condition codes according to the result; the data register is unchanged. The operation size may be a byte, word, or longword.

CMPA is used when the destination is an address register; CMPI is used when the source is immediate data. Most assemblers automatically make this distinction.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | * | * |

X Not affected
N Set if the result is negative; cleared otherwise
Z Set if the result is zero; cleared otherwise
V Set if an overflow occurs; cleared otherwise
C Set if a borrow occurs; cleared otherwise

Instruction Format:

| 15 | 14 | 13 | 12 | 11 10 9 | 8 7 6 | 5 4 3 2 1 0 | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | REGISTER | OPMODE | EFFECTIVE ADDRESS | |
| | | | | | | MODE | REGISTER |

Instruction Fields:

- Register field—specifies the destination register.
- Opmode field:

| Byte | Word | Long | Operation |
|---|---|---|---|
| 000 | 001 | 010 | Dx - <ea>y |

- Effective address field specifies the source operand; use addressing modes in the following table:

| Addressing Mode | Mode | Register | | Addressing Mode | Mode | Register |
|---|---|---|---|---|---|---|
| Dy | 000 | reg. number:Dy | | ($d_8$,Ay,Xi) | 110 | reg. number:Ay |
| Ay (word/longword operand only) | 001 | reg. number:Ay | | (xxx).W | 111 | 000 |
| (Ay) | 010 | reg. number:Ay | | (xxx).L | 111 | 001 |
| (Ay) + | 011 | reg. number:Ay | | #<data> | 111 | 100 |
| – (Ay) | 100 | reg. number:Ay | | ($d_{16}$,PC) | 111 | 010 |
| ($d_{16}$,Ay) | 101 | reg. number:Ay | | ($d_8$,PC,Xi) | 111 | 011 |

| CMP | V2, V3 Core | V4 Core |
|---|---|---|
| Opcode present | Yes | Yes |
| Operand sizes supported | .l | .b, .w, .l |

# CMPA Compare Address CMPA

Operation: `Destination – Source → cc`

Assembler Syntax: CMPA <ea>y, Ax

Attributes: Size = word, long

Description: Operates similarly to CMP, but is used when the destination register is an address register rather than a data register. The operation size can be word or longword. Word-length source operands are sign-extended to 32 bits for comparison.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | * | * |

X Not affected
N Set if the result is negative; cleared otherwise
Z Set if the result is zero; cleared otherwise
V Set if an overflow occurs; cleared otherwise
C Set if a borrow occurs; cleared otherwise

Instruction Format:

| 15 | 14 | 13 | 12 | 11 10 9 | 8 7 6 | 5 4 3 2 1 0 |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | REGISTER | OPMODE | EFFECTIVE ADDRESS |
| | | | | | | MODE / REGISTER |

Instruction Fields:

- Register field—Specifies the destination register.
- Opmode field:

| Byte | Word | Long | Operation |
|---|---|---|---|
| — | 011 | 111 | Ax - <ea>y |

- Effective address field specifies the source operand; use addressing modes in the following table:

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|---|---|---|---|---|---|
| Dy | 000 | reg. number:Dy | ($d_8$,Ay,Xi) | 110 | reg. number:Ay |
| Ay (word/longword operand only) | 001 | reg. number:Ay | (xxx).W | 111 | 000 |
| (Ay) | 010 | reg. number:Ay | (xxx).L | 111 | 001 |
| (Ay) + | 011 | reg. number:Ay | #<data> | 111 | 100 |
| – (Ay) | 100 | reg. number:Ay | ($d_{16}$,PC) | 111 | 010 |
| ($d_{16}$,Ay) | 101 | reg. number:Ay | ($d_8$,PC,Xi) | 111 | 011 |

| CMPA | V2, V3 Core | V4 Core |
|---|---|---|
| Opcode present | Yes | Yes |
| Operand sizes supported | .l | .w, .l |

# CMPI Compare Immediate CMPI

Operation: Destination – Immediate Data → cc

Assembler Syntax: CMPI #<data>, Dx

Attributes: Size = byte, word, long

Description: Operates similarly to CMP, but is used when the source operand is immediate data. The size of the operation may be specified as byte, word, or longword. The size of the immediate data matches the operation size.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | * | * |

X Not affected
N Set if the result is negative; cleared otherwise
Z Set if the result is zero; cleared otherwise
V Set if an overflow occurs; cleared otherwise
C Set if a borrow occurs; cleared otherwise

Instruction Format:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | SIZE | | 0 | 0 | 0 | REGISTER | | |
| UPPER WORD | | | | | | | | | | | | | | | |
| LOWER WORD | | | | | | | | | | | | | | | |

Instruction Fields:

- Register field—Destination data register.
- Size field:

| Byte | Word | Long | Operation |
|---|---|---|---|
| 00 | 01 | 10 | Dx - #<data> |

Note that if size = byte, the immediate is contained in bits [7:0] of the single extension word. If size = word, the immediate is contained in bits[15:0] of the single extension word. If size = long, the immediate is contained in the two extension words.

| CMPI | V2, V3 Core | V4 Core |
|---|---|---|
| Opcode present | Yes | Yes |
| Operand sizes supported | .l | .b, .w, .l |

# INTOUCH Instruction Fetch Touch INTOUCH

Operation:

```
If Supervisor State
          then Instruction Fetch Touch @ <Ay>
else TRAP
```

Assembler Syntax INTOUCH <Ay>

Attributes: Unsized

Description: Generates an instruction fetch reference at address (Ay). If the referenced address space is a cacheable region, this instruction can be used to prefetch a 16-byte packet into the processor's instruction cache. If the referenced instruction address is a non-cacheable space, the instruction effectively performs no operation.

The INTOUCH instruction can be used to prefetch, and with the later setting of CACR[11], lock specific memory lines in the processor's instruction cache. This function may be desirable in systems where deterministic real-time performance is critical.

Condition Codes: Not affected.

Instruction Format:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | REGISTER | | |

Instruction Fields:

- Register field—Specifies the destination address register number.

| INTOUCH | V2, V3 Core | V4 Core |
|---|---|---|
| Opcode present | No | Yes |
| Operand sizes supported | — | — |

# MOVE — Move Data from Source to Destination — MOVE

Operation: `Source → Destination`

Assembler Syntax: MOVE <ea>y, <ea>x

Attributes: Size = byte, word, long

Description: Moves the data at the source to the destination location and sets the condition codes according to the data. The size of the operation may be specified as byte, word, or longword.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | 0 | 0 |

X Not affected
N Set if the result is negative; cleared otherwise
Z Set if the result is zero; cleared otherwise
V Always cleared
C Always cleared

Instruction Format:

| 15 | 14 | 13 12 | 11 10 9 | 8 7 6 | 5 4 3 | 2 1 0 |
|---|---|---|---|---|---|---|
| 0 | 0 | SIZE | DESTINATION | | SOURCE | |
| | | | REGISTER | MODE | MODE | REGISTER |

Instruction fields:

- Size field—Specifies the size of the operand to be moved:

  01— byte operation
  11— word operation
  10— long operation

- Destination effective address field—Specifies destination location; the table below lists possible data alterable addressing modes. The restrictions on combinations of source and destination addressing modes are listed in the table at the bottom of the next page.

| Addressing Mode | Mode | Register | Addressing Mode | Mode | Register |
|---|---|---|---|---|---|
| Dx | 000 | reg. number:Dx | ($d_8$,Ax,Xi) | 110 | reg. number:Ax |
| Ax | — | — | (xxx).W | 111 | 000 |
| (Ax) | 010 | reg. number:Ax | (xxx).L | 111 | 001 |
| (Ax) + | 011 | reg. number:Ax | #<data> | — | — |
| – (Ax) | 100 | reg. number:Ax | ($d_{16}$,PC) | — | — |
| ($d_{16}$,Ax) | 101 | reg. number:Ax | ($d_8$,PC,Xi) | — | — |

- Source effective address field—Specifies source operand; the table below lists possible addressing modes. The ColdFire MOVE instruction has restrictions on combinations of source and destination addressing modes. The table at the end of this instruction description outlines the restrictions.

| Addressing Mode | Mode | Register | | Addressing Mode | Mode | Register |
|---|---|---|---|---|---|---|
| Dy | 000 | reg. number:Dy | | ($d_8$,Ay,Xi) | 110 | reg. number:Ay |
| Ay | 001 | reg. number:Ay | | (xxx).W | 111 | 000 |
| (Ay) | 010 | reg. number:Ay | | (xxx).L | 111 | 001 |
| (Ay) + | 011 | reg. number:Ay | | #<data> | 111 | 100 |
| – (Ay) | 100 | reg. number:Ay | | ($d_{16}$,PC) | 111 | 010 |
| ($d_{16}$,Ay) | 101 | reg. number:Ay | | ($d_8$,PC,Xi) | 111 | 011 |

**NOTE:**

Most assemblers use MOVEA when the destination is an address register.

Use MOVEQ to move an immediate 8-bit value to a data register. Use MOV3Q to move a 3-bit immediate value to any effective destination address.

Not all combinations of source/destination addressing modes are possible. The table below shows the possible combinations.

| Source Addressing Mode | Destination Addressing Mode |
|---|---|
| Dy, Ay, (Ay), (Ay)+,-(Ay) | All possible |
| ($d_{16}$, Ay), (d16, PC) | All possible except ($d_8$, Ax, Xi), (xxx).W, (xxx).L |
| (d8, Ay, Xi), (d8, PC, Xi), (xxx).W, (xxx).L, #<xxx> | All possible except ($d_8$, Ax, Xi), (xxx).W, (xxx).L |

Note: The combination of #<xxx>,d16(Ax) addressing modes can be used only on move byte and move word opcodes. Refer to the previous tables for valid source and destination addressing modes.

| MOVE | V2, V3 Core | V4 Core |
|---|---|---|
| Opcode present | Yes | Yes |
| Operand sizes supported | .b, .w, .l<br>except<br>move.x #<data>, d16(Ax) | .b, .w, .l<br>including<br>move.{b,w} #<data>, d16(Ax) |

# MOVEA Move Address from Source to Destination MOVEA

Operation: `Source → Destination`

Assembler Syntax: MOVEA <ea>y, Ax

Attributes: Size = word, long

Description: Moves the address at the source to the destination location and sets the condition codes according to the data. The size of the operation may be specified as word or longword.

Condition Codes: Not affected

Instruction Format:

| 15 | 14 | 13 12 | 11 10 9 | 8 7 6 | 5 4 3 | 2 1 0 |
|---|---|---|---|---|---|---|
| 0 | 0 | SIZE | DESTINATION | | SOURCE | |
| | | | REGISTER | MODE | MODE | REGISTER |

Instruction fields:

- Size field—specifies the size of the operand to be moved:

  11—word operation
  10—longword operation

- Destination effective address field—Specifies the destination location; the table below lists possible addressing modes.

| Addressing Mode | Mode | Register | | Addressing Mode | Mode | Register |
|---|---|---|---|---|---|---|
| Dx | — | — | | ($d_8$,Ax,Xi) | — | — |
| Ax | 001 | reg. number: Ax | | (xxx).W | — | — |
| (Ax) | — | — | | (xxx).L | — | — |
| (Ax) + | — | — | | #<data> | — | — |
| – (Ax) | — | — | | ($d_{16}$,PC) | — | — |
| ($d_{16}$,Ax) | — | — | | ($d_8$,PC,Xi) | — | — |

- Source effective address field—Specifies the source operand; the table below lists possible modes.

| Addressing Mode | Mode | Register | | Addressing Mode | Mode | Register |
|---|---|---|---|---|---|---|
| Dy | 000 | reg. number:Dy | | ($d_8$,Ay,Xi) | 110 | reg. number:Ay |
| Ay | 001 | reg. number:Ay | | (xxx).W | 111 | 000 |
| (Ay) | 010 | reg. number:Ay | | (xxx).L | 111 | 001 |
| (Ay) + | 011 | reg. number:Ay | | #<data> | 111 | 100 |
| – (Ay) | 100 | reg. number:Ay | | ($d_{16}$,PC) | 111 | 010 |
| ($d_{16}$,Ay) | 101 | reg. number:Ay | | ($d_8$,PC,Xi) | 111 | 011 |

| MOVEA | V2, V3 Core | V4 Core |
|---|---|---|
| Opcode present | Yes | Yes |
| Operand sizes supported | No differences | |

# MOV3Q — Move 3-Bit Data Quick — MOV3Q

Operation: `Immediate Data → Destination`

Assembler Syntax: MOV3Q #<data>,<ea>x

Attributes: Size = long

Description: Move the immediate data to the operand at the destination location. The data range is from -1 to 7, excluding 0. The immediate data is zero-filled to a long operand and all 32 bits are transferred to the destination location.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | 0 | 0 |

X Not affected
N Set if the result is negative; cleared otherwise
Z Set if the result is zero; cleared otherwise
V Always cleared
C Always cleared

Instruction Format:

| 15 | 14 | 13 | 12 | 11 10 9 | 8 | 7 | 6 | 5 4 3 | 2 1 0 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | DATA | 1 | 0 | 1 | MODE | REGISTER |

Instruction Fields:

- Data field—3 bits of data having a range {-1,1-7} where a data value of 0 represents -1.
- Effective Address field—Specifies the destination operand; use only data addressing modes listed in the following table:

| Addressing Mode | Mode | Register | | Addressing Mode | Mode | Register |
|---|---|---|---|---|---|---|
| Dx | 000 | reg. number:Dx | | ($d_8$,Ax,Xi) | 110 | reg. number:Ax |
| Ax | 001 | reg. number:Ax | | (xxx).W | 111 | 000 |
| (Ax) | 010 | reg. number:Ax | | (xxx).L | 111 | 001 |
| (Ax) + | 011 | reg. number:Ax | | #<data> | — | — |
| – (Ax) | 100 | reg. number:Ax | | ($d_{16}$,PC) | — | — |
| ($d_{16}$,Ax) | 101 | reg. number:Ax | | ($d_8$,PC,Xi) | — | — |

| MOV3Q | V2, V3 Core | V4 Core |
|---|---|---|
| Opcode present | No | Yes |
| Operand sizes supported | — | .l |

# MVS — Move with Sign Extend — MVS

Operation: (Source with sign extension) → Destination

Assembler Syntax: MVS <ea>y,Dx

Attributes: Size = byte, word

Description: Sign-extend the source operand and move to the destination register. For the byte operation, bit 7 of the source is copied to bits 31–8 of the destination. For the word operation, bit 15 of the source is copied to bits 31-16 of the destination.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | 0 | 0 |

X Not affected
N Set if the result is negative; cleared otherwise
Z Set if the result is zero; cleared otherwise
V Always cleared
C Always cleared

Instruction Format:

| 15 | 14 | 13 | 12 | 11 10 9 | 8 | 7 | 6 | 5 4 3 | 2 1 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | REGISTER | 1 | 0 | SIZE | EFFECTIVE ADDRESS | |
| | | | | | | | | MODE | REGISTER |

Instruction Fields:

- Size field—specifies the size of the operation
  0 byte operation
  1 word operation
- Register field—specifies a data register as the destination.
- Effective address field—specifies the source operand; use only data addressing modes from the following table:

| Addressing Mode | Mode | Register | | Addressing Mode | Mode | Register |
|---|---|---|---|---|---|---|
| Dy | 000 | reg. number:Dy | | ($d_8$,Ay,Xi) | 110 | reg. number:Ay |
| Ay | 001 | reg. number:Ay | | (xxx).W | 111 | 000 |
| (Ay) | 010 | reg. number:Ay | | (xxx).L | 111 | 001 |
| (Ay) + | 011 | reg. number:Ay | | #<data> | 111 | 100 |
| – (Ay) | 100 | reg. number:Ay | | ($d_{16}$,PC) | 111 | 010 |
| ($d_{16}$,Ay) | 101 | reg. number:Ay | | ($d_8$,PC,Xi) | 111 | 011 |

| MVS | V2, V3 Core | V4 Core |
|---|---|---|
| Opcode present | No | Yes |
| Operand sizes supported | — | .b, .w |

# MVZ — Move with Zero-Fill — MVZ

Operation: `(Source with zero fill) → Destination`

Assembler Syntax: MVZ <ea>y,Dx

Attributes: Size = byte, word

Description—Zero-fill the source operand and move to the destination register. For the byte operation, the source operand is moved to bits 7–0 of the destination and bits 31–8 are filled with zeros. For the word operation, the source operand is moved to bits 15–0 of the destination and bits 31–16 are filled with zeros.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | 0 | * | 0 | 0 |

X Not affected
N Always cleared
Z Set if the result is zero; cleared otherwise
V Always cleared
C Always cleared

Instruction Format:

| 15 | 14 | 13 | 12 | 11 10 9 | 8 | 7 | 6 | 5 4 3 | 2 1 0 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | REGISTER | 1 | 1 | SIZE | EFFECTIVE ADDRESS | |
| | | | | | | | | MODE | REGISTER |

Instruction Fields:

- Size field—Specifies the size of the operation
  0 byte operation
  1 word operation
- Register field—Specifies a data register as the destination.
- Effective address field—Specifies the source operand; use the following data addressing modes:

| Addressing Mode | Mode | Register | | Addressing Mode | Mode | Register |
|---|---|---|---|---|---|---|
| Dy | 000 | reg. number:Dy | | ($d_8$,Ay,Xi) | 110 | reg. number:Ay |
| Ay | 001 | reg. number:Ay | | (xxx).W | 111 | 000 |
| (Ay) | 010 | reg. number:Ay | | (xxx).L | 111 | 001 |
| (Ay) + | 011 | reg. number:Ay | | #<data> | 111 | 100 |
| – (Ay) | 100 | reg. number:Ay | | ($d_{16}$,PC) | 111 | 010 |
| ($d_{16}$,Ay) | 101 | reg. number:Ay | | ($d_8$,PC,Xi) | 111 | 011 |

| MVZ | V2, V3 Core | V4 Core |
|---|---|---|
| Opcode present | No | Yes |
| Operand sizes supported | — | .b, .w |

# SATS Signed Saturate SATS

Operation:

```
If CCR.V == 1,
then if Dx[31] == 0,
        then Dx[31:0] = 0x80000000
        else Dx[31:0] = 0x7FFFFFFF
else Dx[31:0] is unchanged
```

Assembler Syntax: SATS Dx

Attributes: Size = long

Description: Update the destination register only if the overflow bit of the CCR is set. If the operand is negative, then set the result to greatest positive number, otherwise set the result to the largest negative value. The condition codes are set according to the result.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | 0 | 0 |

X Not affected
N Set if the result is negative; cleared otherwise
Z Set if the result is zero; cleared otherwise
V Always cleared
C Always cleared

Instruction Format:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | REGISTER | | |

Instruction Fields:

- Register field—Specifies the destination data register.

| SATS | V2, V3 Core | V4 Core |
|---|---|---|
| Opcode present | No | Yes |
| Operand sizes supported | — | .l |

# TAS — Test and Set an Operand — TAS

Operation: Destination Tested → CCR; 1 → bit 7 of Destination

Assembler Syntax: TAS <ea>x

Attributes: Size = byte

Description: Tests and sets the byte operand addressed by the effective address field. The instruction tests the current value of the operand and sets the N and Z condition code bits appropriately. TAS also sets the high-order bit of the operand. The operand uses a read-modify-write memory cycle that completes the operation without interruption. This instruction supports use of a flag or semaphore to coordinate several processors.

Condition Codes:

| X | N | Z | V | C |
|---|---|---|---|---|
| — | * | * | 0 | 0 |

X Not affected
N Set if the msb of the operand is currently set; cleared otherwise
Z Set if the operand was zero; cleared otherwise
V Always cleared
C Always cleared

Instruction Format:

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | EFFECTIVE ADDRESS | | | | | |
| | | | | | | | | | | MODE | | | REGISTER | | |

Instruction Fields:

- Effective address field—specifies the destination location; the possible data alterable addressing modes are listed in the table below.

| Addressing Mode | Mode | Register | | Addressing Mode | Mode | Register |
|---|---|---|---|---|---|---|
| Dx | — | — | | ($d_8$,Ax,Xi) | 110 | reg. number:Ax |
| Ax | — | — | | (xxx).W | 111 | 000 |
| (Ax) | 010 | reg. number:Ax | | (xxx).L | 111 | 001 |
| (Ax) + | 011 | reg. number:Ax | | #<data> | — | — |
| – (Ax) | 100 | reg. number:Ax | | ($d_{16}$,PC) | — | — |
| ($d_{16}$,Ax) | 101 | reg. number:Ax | | ($d_8$,PC,Xi) | — | — |

| TAS | V2, V3 Core | V4 Core |
|---|---|---|
| Opcode present | No | Yes |
| Operand sizes supported | — | .b |

# Chapter 3
# Hardware Multiply/Accumulate (MAC) Unit

This chapter describes the MCF5407 multiply/accumulate (MAC) unit, which executes integer multiply, multiply-accumulate, and miscellaneous register instructions. The MAC is integrated into the operand execution pipeline (OEP).

## 3.1 Overview

The MAC unit provides hardware support for a limited set of digital signal processing (DSP) operations used in embedded code, while supporting the integer multiply instructions in the ColdFire microprocessor family.

The MAC unit provides signal processing capabilities for the MCF5407 in a variety of applications including digital audio and servo control. Integrated as an execution unit in the processor's OEP, the MAC unit implements a three-stage arithmetic pipeline optimized for 16 x 16 multiplies. Both 16- and 32-bit input operands are supported by this design in addition to a full set of extensions for signed and unsigned integers plus signed, fixed-point fractional input operands.

The MAC unit provides functionality in three related areas:

- Signed and unsigned integer multiplies
- Multiply-accumulate operations supporting signed, unsigned, and signed fractional operands
- Miscellaneous register operations

Each of the three areas of support is addressed in detail in the succeeding sections. Logic that supports this functionality is contained in a MAC module, as shown in Figure 3-1.

The MAC unit is tightly coupled to the OEP and features a three-stage execution pipeline. To minimize silicon costs, the ColdFire MAC is optimized for 16 x 16 multiply instructions. The OEP can issue a 16 x 16 multiply with a 32-bit accumulation and fetch a 32-bit operand in the same cycle. A 32 x 32 multiply with a 32-bit accumulation takes three cycles before the next instruction can be issued. Figure 3-1 shows the basic functionality of the ColdFire MAC. A full set of instructions is provided for signed and unsigned integers plus signed, fixed-point, fractional input operands.

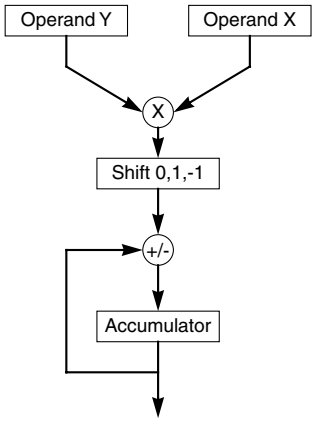



**Figure 3-1. ColdFire MAC Multiplication and Accumulation**

The MAC unit is an extension of the basic multiplier found on most microprocessors. It can perform operations native to signal processing algorithms in an acceptable number of cycles, given the application constraints. For example, small digital filters can tolerate some variance in the execution time of the algorithm; larger, more complicated algorithms such as orthogonal transforms may have more demanding speed requirements exceeding the scope of any processor architecture and requiring a fully developed DSP implementation.

The M68000 architecture was not designed for high-speed signal processing, and a large DSP engine would be excessive in an embedded environment. In striking a middle ground between speed, size, and functionality, the ColdFire MAC unit is optimized for a small set of operations that involve multiplication and cumulative additions. Specifically, the multiplier array is optimized for single-cycle, 16 x 16 multiplies producing a 32-bit result, with a possible accumulation cycle following. This is common in a large portion of signal processing applications. In addition, the ColdFire core architecture has been modified to allow for an operand fetch in parallel with a multiply, increasing overall performance for certain DSP operations.

### 3.1.0.1 MAC Programming Model

Figure 3-2 shows the registers in the MAC portion of the user programming model.

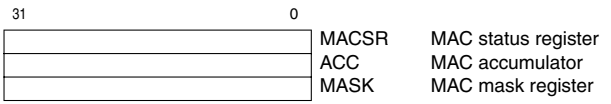



**Figure 3-2. MAC Programming Model**

These registers are described as follows:

- Accumulator (ACC)—This 32-bit, read/write, general-purpose register is used to accumulate the results of MAC operations.
- Mask register (MASK)—This 16-bit general-purpose register provides an optional address mask for MAC instructions that fetch operands from memory. It is useful in the implementation of circular queues in operand memory.
- MAC status register (MACSR)—This 8-bit register defines configuration of the MAC unit and contains indicator flags affected by MAC instructions. Unless noted otherwise, the setting of MACSR indicator flags is based on the final result, that is, the result of the final operation involving the product and accumulator.

### 3.1.0.2 General Operation

The MAC unit supports the ColdFire integer multiply instructions (MULS and MULU) and provides additional functionality for multiply-accumulate operations. The added MAC instructions to the ColdFire ISA provide for the multiplication of two numbers, followed by the addition or subtraction of this number to or from the value contained in the accumulator. The product may be optionally shifted left or right one bit before the addition or subtraction takes place. Hardware support for saturation arithmetic may be enabled to minimize software overhead when dealing with potential overflow conditions using signed or unsigned operands.

These MAC operations treat the operands as one of the following formats:

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

To maintain compactness, the MAC module is optimized for 16-bit multiplications. Two 16-bit operands produce a 32-bit product. Longword operations are performed by reusing the 16-bit multiplier array at the expense of a small amount of extra control logic. Again, the product of two 32-bit operands is a 32-bit result. For longword integer operations, only the least significant 32 bits of the product are calculated. For fractional operations, the entire 63-bit product is calculated and then either truncated or rounded to a 32-bit result using the round-to-nearest (even) method.

Because the multiplier array is implemented in a 3-stage pipeline, MAC instructions can have an effective issue rate of one clock for word operations, three for longword integer operations, and four for 32-bit fractional operations. Arithmetic operations use register-based input operands, and summed values are stored internally in the accumulator. Thus, an additional MOVE instruction is necessary to store data in a general-purpose register. MAC instructions can choose the upper or lower word of a register as the input, which helps filtering operations in which one data register is loaded with input data and another is loaded with coefficient data. Two 16-bit MAC operations can be performed without fetching additional operands between instructions by alternating the word choice during the calculations.

The need to move large amounts of data quickly can limit throughput in DSP engines. However, data can be moved efficiently by using the MOVEM instruction, which automatically generates line-sized burst references and is ideal for filling registers quickly with input data, filter coefficients, and output data. Loading an operand from memory into a register during a MAC operation makes some DSP operations, especially filtering and convolution, more manageable.

The MACSR has a 4-bit operational mode field and three condition flags. The operational mode bits control the overflow/saturation mode, whether operands are signed or unsigned, whether operands are treated as integers or fractions, and how rounding is performed. Negative, zero and overflow flags are also provided.

The three program-visible MAC registers, a 32-bit accumulator (ACC), the MAC mask register (MASK), and MACSR, are described in Section 3.1.0.1, “MAC Programming Model.”

### 3.1.0.3 MAC Instruction Set Summary

The MAC unit supports the integer multiply operations defined by the baseline ColdFire architecture, as well as the new multiply-accumulate instructions. Table 3-1 summarizes the MAC unit instruction set.

**Table 3-1. MAC Instruction Summary**

| Instruction | 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<br>MSAC Ry,RxSF | Multiplies two operands, then adds or subtracts the product to/from the accumulator |
| Multiply Accumulate with Load | MAC Ry,RxSF,Rw<br>MSAC Ry,RxSF,Rw | Multiplies two operands, then adds or subtracts the product to/from the accumulator while loading a register with the memory operand |
| Load Accumulator | MOV.L {Ry,#imm},ACC | Loads the accumulator with a 32-bit operand |
| Store Accumulator | MOV.L ACC,Rx | Writes the contents of the accumulator to a register |
| Load MACSR | MOV.L {Ry,#imm},MACSR | Writes a value to the MACSR |
| Store MACSR | MOV.L MACSR,Rx | Write the contents of MACSR to a register |
| Store MACSR to CCR | MOV.L MACSR,CCR | Write the contents of MACSR to the processor’s CCR register |
| Load MASK | MOV.L {Ry,#imm},MASK | Writes a value to MASK |
| Store MASK | MOV.L MASK,Rx | Writes the contents of MASK to a register |

### 3.1.0.4 Data Representation

The MAC unit supports three basic operand types:

- Two’s complement signed integer: In this format, an N-bit operand represents a number within the range $-2^{(N-1)} \leq \text{operand} \leq 2^{(N-1)} - 1$. The binary point is to the right of the least significant bit.

- Two's complement unsigned integer: In this format, an N-bit operand represents a number within the range $0 \leq$ operand $\leq 2^N - 1$. The binary point is to the right of the least significant bit.
- 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, $a_{N-1}a_{N-2}a_{N-3}... a_2a_1a_0$, its value is given by the following formula:

$$+\sum_{i=0}^{N-2} 2^{(i+1-N)} \cdot ai$$

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

For words and longwords, the greatest negative number that can be represented is -1, whose internal representation is 0x8000 and 0x0x8000_0000, respectively. The most positive word is 0x7FFF or $(1 - 2^{-15})$; the most positive longword is 0x7FFF_FFFF or $(1 - 2^{-31})$.

## 3.2 MAC Instruction Execution Timings

Table 3-2 shows standard timings for two-operand MAC instructions.

**Table 3-2. Two-Operand MAC Instruction Execution Times**

| Opcode | <ea> | Effective Address | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Rn | (An) | (An)+ | -(An) | (d16,An) | (d8,An,Xi*SF) | (xxx).wl | #<xxx> |
| mac.w | Ry,Rx | 1(0/0) | — | — | — | — | — | — | — |
| mac.l | Ry,Rx | 3(0/0) | — | — | — | — | — | — | — |
| msac.w | Ry,Rx | 1(0/0) | — | — | — | — | — | — | — |
| msac.l | Ry,Rx | 3(0/0) | — | — | — | — | — | — | — |
| mac.w | Ry,Rx,ea,Rw | — | 1(1/0) | 1(1/0) | 1(1/0) | 1(1/0) | — | — | — |
| mac.l | Ry,Rx,ea,Rw | — | 3(1/0) | 3(1/0) | 3(1/0) | 3(1/0) | — | — | — |
| msac.w | Ry,Rx,ea,Rw | — | 1(1/0) | 1(1/0) | 1(1/0) | 1(1/0) | — | — | — |
| msac.l | Ry,Rx,ea,Rw | — | 3(1/0) | 3(1/0) | 3(1/0) | 3(1/0) | — | — | — |
| muls.w | <ea>,Dx | 3(0/0) | 3(1/0) | 3(1/0) | 3(1/0) | 3(1/0) | 4(1/0) | 3(1/0) | 3(0/0) |
| mulu.w | <ea>,Dx | 3(0/0) | 3(1/0) | 3(1/0) | 3(1/0) | 3(1/0) | 4(1/0) | 3(1/0) | 3(0/0) |
| muls.l | <ea>,Dx | 5(0/0) | 5(1/0) | 5(1/0) | 5(1/0) | 5(1/0) | — | — | — |
| mulu.l | <ea>,Dx | 5(0/0) | 5(1/0) | 5(1/0) | 5(1/0) | 5(1/0) | — | — | — |

Table 3-3 shows standard timings for MAC move instructions.

## Table 3-3. MAC Move Instruction Execution Times

| Opcode | <ea> | Effective Address | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Rn | (An) | (An)+ | -(An) | (d16,An) | (d8,An,Xi*SF) | (xxx).wl | #<xxx> |
| move.l | <ea>,ACC | 1(0/0) | — | — | — | — | — | — | 1(0/0) |
| move.l | <ea>,MACSR | 6(0/0) | — | — | — | — | — | — | 6(0/0) |
| move.l | <ea>,MASK | 5(0/0) | — | — | — | — | — | — | 5(0/0) |
| move.l | ACC,Rx | 1(0/0) | — | — | — | — | — | — | — |
| move.l | MACSR,CCR | 1(0/0) | — | — | — | — | — | — | — |
| move.l | MACSR,Rx | 1(0/0) | — | — | — | — | — | — | — |
| move.l | MASK,Rx | 1(0/0) | — | — | — | — | — | — | — |

# Chapter 4
# Local Memory

This chapter describes the MCF5407 implementation of the ColdFire Version 4 local memory specification. It consists of two major sections.

- Section 4.2, “SRAM Overview,” describes the MCF5407 on-chip static RAM (SRAM) implementation. It covers general operations, configuration, and initialization. It also provides information and examples showing how to minimize power consumption when using the SRAM.
- Section 4.7, “Cache Overview,” describes the MCF5407 cache implementation, including organization, configuration, and coherency. It describes cache operations and how the cache interfaces with other memory structures.

## 4.1 Interactions between Local Memory Modules

Depending on configuration information, instruction fetches and data read accesses may be sent simultaneously to the RAM and cache controllers. This approach is required because all three controllers are memory-mapped devices and the hit/miss determination is made concurrently with the read data access. Power dissipation can be minimized by configuring the RAMBARs to mask unused address spaces whenever possible.

If the access address is mapped into the region defined by the RAM (and this region is not masked), the RAM provides the data back to the processor, and the cache data is discarded. Accesses from the RAM module are never cached. The complete definition of the processor’s local bus priority scheme for read references is as follows:

```
        if (RAM “hits”
)        RAM supplies data to the processor
               else if (data cache “hits”)
                         data cache supplies data to the processor
                  else system memory reference to access data
```

For data write references, the memory mapping into the local memories is resolved before the appropriate destination memory is accessed. Accordingly, only the targeted local memory is accessed for data write transfers.

## 4.2 SRAM Overview

The two 2-Kbyte on-chip SRAM modules provide pipelined, single-cycle access to memory mapped to these modules. Memory can be independently mapped to any

0-modulo-2K location in the 4-Gbyte address space and configured to respond to either instruction or data accesses. Time-critical functions can be mapped into instruction memory and the system stack. Other heavily-referenced data can be mapped into data memory.

The following summarizes features of the MCF5407 SRAM implementation:

- Two 2-Kbyte SRAMs, organized as 512 x 32 bits
- Single-cycle throughput. When the pipeline is full, one access can occur per clock cycle.
- Physical location on the processor's high-speed local bus with a user-programmed connection to the internal instruction or data bus
- Memory location programmable on any 0-modulo-2K address boundary
- Byte, word, and longword address capabilities
- The RAM base address registers (RAMBAR0 and RAMBAR1) define the logical base address, attributes, and access types for the two SRAM modules.

## 4.3 SRAM Operation

Each SRAM module provides a general-purpose memory block that the ColdFire processor can access with single-cycle throughput. The location of the memory block can be specified to any word-aligned address in the 4-Gbyte address space by RAMBAR*n*[BA], described in Section 4.4.1, "SRAM Base Address Registers (RAMBAR0/RAMBAR1)." The memory is ideal for storing critical code or data structures or for use as the system stack. Because the SRAM module connects physically to the processor's high-speed local bus, it can service processor-initiated accesses or memory-referencing debug module commands.

The Version 4 ColdFire processor core implements a Harvard memory architecture. Each SRAM module may be logically connected to either the processor's internal instruction or data bus. This logical connection is controlled by a configuration bit in the RAM base address registers (RAMBAR0 and RAMBAR1).

If an instruction fetch is mapped into the region defined by the SRAM, the SRAM sources the data to the processor and any cache data is discarded. Likewise, if a data access is mapped into the region defined by the SRAM, the SRAM services the access and the cache is not affected. Accesses from SRAM modules are never cached, and debug-initiated references are treated as data accesses.

Note also that the SRAMs cannot be accessed by the on-chip DMAs. The on-chip system configuration allows concurrent core and DMA execution, where the core can reference code or data from the internal SRAMs or caches while performing a DMA transfer.

Accesses are attempted in the following order:

1. SRAM
2. Cache (if space is defined as cacheable)
3. External access

## 4.4 SRAM Programming Model

The SRAM programming model consists of RAMBAR0 and RAMBAR1.

### 4.4.1 SRAM Base Address Registers (RAMBAR0/RAMBAR1)

The SRAM modules are configured through the RAMBARs, shown in Figure 4-1.

- Each RAMBAR holds the base address of the SRAM. The MOVEC instruction provides write-only access to this register from the processor.
- Each RAMBAR can be read or written from the debug module in a similar manner.
- All undefined RAMBAR bits are reserved. These bits are ignored during writes to the RAMBAR and return zeros when read from the debug module.
- The valid bits, RAMBAR*n*[V], are cleared at reset, disabling the SRAM modules. All other bits are unaffected.

| | 31 … 11 | 10 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | BA | — | WP | D/I | — | C/I | SC | SD | UC | UD | V |
| Reset | — | | | | | | | | | | 0 |
| R/W | W for CPU; R/W for debug | | | | | | | | | | |
| Address | CPU space + 0xC04 (RAMBAR0), CPU space + 0xC05 (RAMBAR1) | | | | | | | | | | |

**Figure 4-1. SRAM Base Address Registers (RAMBARn)**

RAMBAR*n* fields are described in detail in Table 4-1.

**Table 4-1. RAMBARn Field Description**

| Bits | Name | Description |
|---|---|---|
| 31–11 | BA | Base address. Defines the SRAM module's word-aligned base address. Each SRAM module occupies a 2-Kbyte space defined by the contents of BA. SRAM may reside on any 2-Kbyte boundary in the 4-Gbyte address space. |
| 10–9 | — | Reserved, should be cleared. |
| 8 | WP | Write protect. Controls read/write properties of the SRAM.<br>0 Allows read and write accesses to the SRAM module<br>1 Allows only read accesses to the SRAM module. Any attempted write reference generates an access error exception to the ColdFire processor core. |
| 7 | D/I | Data/instruction bus. Indicates whether SRAM is connected to the internal data or instruction bus.<br>0 Data bus<br>1 Instruction bus |

**Table 4-1. RAMBARn Field Description (Continued)**

| Bits | Name | Description |
|---|---|---|
| 6 | — | Reserved, should be cleared. |
| 5–1 | C/I, SC, SD, UC, UD | Address space masks (ASn). These fields allow certain types of accesses to be masked, or inhibited from accessing the SRAM module. These bits are useful for power management as described in Section 4.6, “Power Management.” In particular, C/I is typically set.<br>The address space mask bits are follows:<br>C/I = CPU space/interrupt acknowledge cycle mask. Note that C/I must be set if BA = 0.<br>SC = Supervisor code address space mask<br>SD = Supervisor data address space mask<br>UC = User code address space mask<br>UD = User data address space mask<br>For each ASn bit:<br>0 An access to the SRAM module can occur for this address space<br>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. |
| 0 | V | Valid. Enables/disables the SRAM module. V is cleared at reset.<br>0 RAMBAR contents are not valid.<br>1 RAMBAR contents are valid. |

The mapping of a given access into the RAM uses the following algorithm to determine if the access hits in the memory:

```
if (RAMBAR[0] = 1)

if (((access = instructionFetch) & (RAMBAR[7] = 1)) |
        ((access = dataReference)    & (RAMBAR[7] = 0)))
                if (requested address[31:10] = RAMBAR[31:10])
        if (requested address[31:n] = RAMBAR[31:n]
                if (ASn of the requested type = 0)
                        Access is mapped to the RAM module
                        if (access = read)
                                Read the RAM and return the data
                        if (access = write)
                                if (RAMBAR[8] = 0)
                                        Write the data into the RAM
                                else Signal a write-protect access error
```

AS*n* refers to the five address space mask bits: C/I, SC, SD, UC, and UD.

## 4.5 SRAM Initialization

After a hardware reset, the contents of each SRAM module are undefined. The valid bits, RAMBAR*n*[V], are cleared, disabling the SRAM modules. If the SRAM requires initialization with instructions or data, the following steps should be performed:

1. Load RAMBAR*n* with bit 7 = 0, mapping the SRAM module to the desired location. Clearing RAMBAR*n*[7] logically connects the SRAM module to the processor’s data bus.

2. Read the source data and write it to the SRAM. Various instructions support this function, including memory-to-memory move instructions and the move multiple instruction (MOVEM). MOVEM is optimized to generate line-sized burst fetches on line-aligned addresses, so it generally provides maximum performance.
3. After the data is loaded into the SRAM, it may be appropriate to revise the RAMBAR attribute bits, including the write-protect and address space mask fields. If the SRAM contains instructions, RAMBAR[D/I] must be set to logically connect the memory to the processor's internal instruction bus.

Remember that the SRAM cannot be accessed by the on-chip DMAs. The on-chip system configuration allows concurrent core and DMA execution where the core can execute code out of internal SRAM or cache during DMA access.

The ColdFire processor or an external emulator using the debug module can perform these initialization functions.

### 4.5.1 SRAM Initialization Code

The code segment below initializes the SRAM using RAMBAR0. The code sets the base address of the SRAM at 0x2000_0000 and then initializes the RAM to zeros.

```
RAMBASE         EQU     0x20000000      ;set this variable to 0x20000000
RAMVALID        EQU     0x00000035
move.l          #RAMBASE+RAMVALID,D0    ;load RAMBASE + valid bit into D0
movec.l         D0, RAMBAR0             ;load RAMBAR0 and enable SRAM
```

The following loop initializes the entire SRAM to zero:

```
lea.l           RAMBASE,A0              ;load pointer to SRAM
move.l          #512,D0                 ;load loop counter into D0


SRAM_INIT_LOOP:

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

The following function copies the number of bytesToMove from the source (*src) to the processor's local RAM at an offset relative to the SRAM base address defined by destinationOffset. The bytesToMove must be a multiple of 16. For best performance, source and destination SRAM addresses should be line-aligned (0-modulo-16).

```
; copyToCpuRam (*src, destinationOffset, bytesToMove)


RAMBASE         EQU     0x20000000      ;SRAM base address
RAMFLAGS        EQU     0x00000035      ;RAMBAR valid + mask bits

        lea.l       -12(a7),a7          ;allocate temporary space
        movem.l     #0x1c,(a7)          ;store D2/D3/D4 registers

; stack arguments and locations
```

```
; +0     saved d2
; +4     saved d3
; +8     saved d4
; +12    returnPc
; +16    pointer to source operand
; +20    destinationOffset
; +24    bytesToMove

        move.l      RAMBASE+RAMFLAGS,a0  ;define RAMBAR0 contents
        movec.l     a0,rambar0           ;load it

        move.l      16(a7),a0            ;load argument defining *src

        lea.l       RAMBASE,a1           ;memory pointer to RAM base
        add.l       20(a7),a1            ;include destinationOffset

        move.l      24(a7),d4            ;load byte count
        asr.l       #4,d4                ;divide by 16 to convert to loop count

        .align      4                    ;force loop on 0-mod-4 address
loop:   movem.l     (a0),#0xf            ;read 16 bytes from source
        movem.l     #0xf,(a1)            ;store into RAM destination
        lea.l       16(a0),a0            ;increment source pointer
        lea.l       16(a1),a1            ;increment destination pointer
        subq.l      #1,d4                ;decrement loop counter
        bne.b       loop                 ;if done, then exit, else continue

        movem.l     (a7),#0x1c           ;restore d2/d3/d4 registers
        lea.l       12(a7),a7            ;deallocate temporary space
        rts
```

## 4.6 Power Management

Because processor memory references may be simultaneously sent to an SRAM module and cache, power can be minimized by configuring RAMBAR address space masks as precisely as possible. For example, if an SRAM is mapped to the internal instruction bus and contains instruction data, setting the AS*n* mask bits associated with operand references can decrease power dissipation. Similarly, if the SRAM contains data, setting AS*n* bits associated with instruction fetches minimizes power.

Table 4-2 shows typical RAMBAR configurations.

**Table 4-2. Examples of Typical RAMBAR Settings**

| Data Contained in SRAM | RAMBAR[5–0] |
|---|---|
| Code only | 0x2B |
| Data only | 0x35 |
| Both code and data | 0x21 |

## 4.7 Cache Overview

This section describes the MCF5407 cache implementation, including organization, configuration, and coherency. It describes cache operations and how the cache interacts with other memory structures.

The MCF5407 implements a special branch instruction cache for accelerating branches, enabled by a bit in the cache access control register (CACR[BEC]). The branch cache is described in Section 2.1.2.1.1, “Branch Acceleration.”

The MCF5407 processor’s Harvard memory structure includes an 8-Kbyte data cache and a 16-Kbyte instruction cache. Both are nonblocking and 4-way set-associative with a 16-byte line. The cache improves system performance by providing single-cycle access to the instruction and data pipelines. This decouples processor performance from system memory performance, increasing bus availability for on-chip DMA or external devices. Figure 4-2 shows the organization and integration of the data cache.

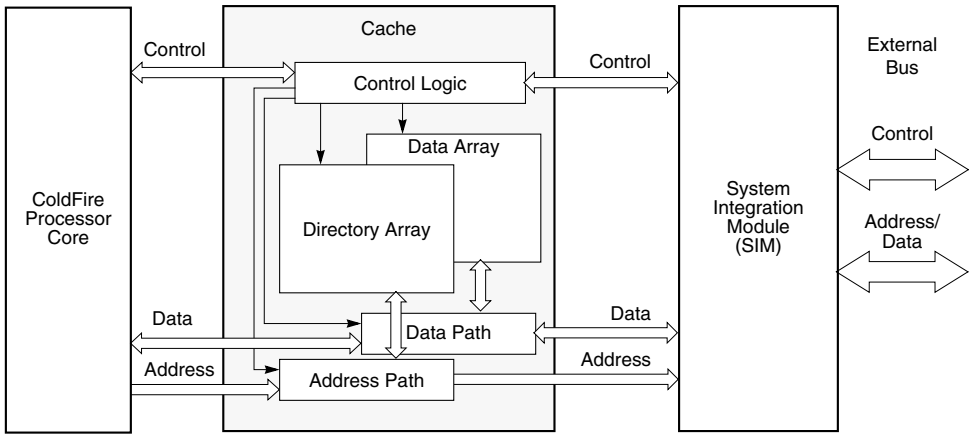



**Figure 4-2. Data Cache Organization**

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

A nonblocking cache services read hits or write hits from the processor while a fill (caused by a cache allocation) is in progress. As Figure 4-2 shows, accesses use a single bus connected to the cache.

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

The SRAM module does not implement bus snooping; cache coherency with other possible bus masters must be maintained in software.

# 4.8 Cache Organization

A four-way set associative cache is organized as four ways (levels). There are 128 sets in the 8-Kbyte data cache with each line containing 16 bytes (4 longwords). The 16-Kbyte instruction cache has 256 sets. Entire cache lines are loaded from memory by burst-mode accesses that cache 4 longwords of data or instructions. All 4 longwords must be loaded for the cache line to be valid.

Figure 4-3 shows data cache organization as well as terminology used.

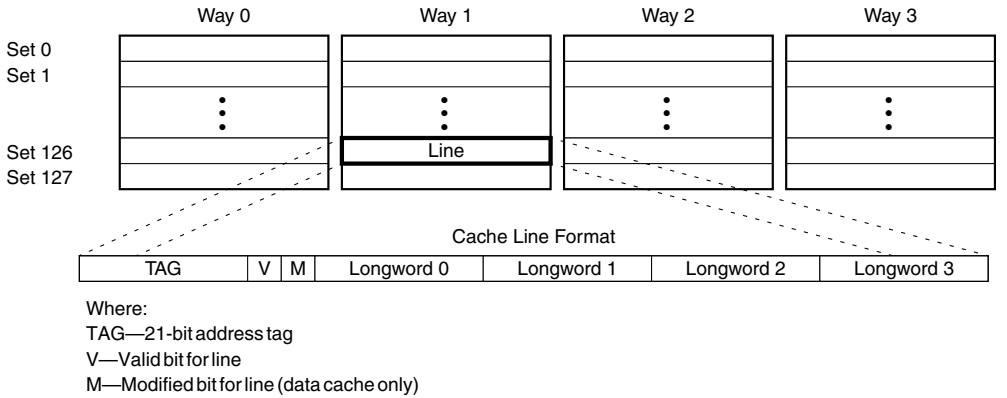



**Figure 4-3. Data Cache Organization and Line Format**

A set is a group of four lines (one from each level, or way), corresponding to the same index into the cache array.

## 4.8.1 Cache Line States: Invalid, Valid-Unmodified, and Valid-Modified

As shown in Table 4-3, a data cache line can be invalid, valid-unmodified (often called exclusive), or valid-modified. An instruction cache line can be valid or invalid.

**Table 4-3. Valid and Modified Bit Settings**

| V | M | Description |
|---|---|---|
| 0 | x | Invalid. Invalid lines are ignored during lookups. |
| 1 | 0 | Valid, unmodified. Cache line has valid data that matches system memory. |
| 1 | 1 | Valid, modified. Cache line contains most recent data, data at system memory location is stale. |

A valid line can be explicitly invalidated by executing a CPUSHL instruction.

### 4.8.2 The Cache at Start-Up

As Figure 4-4 (A) shows, after power-up, cache contents are undefined; V and M may be set on some lines even though the cache may not contain the appropriate data for start up. Because reset and power-up do not invalidate cache lines automatically, the cache should be cleared explicitly by setting CACR[DCINVA,ICINVA] before the cache is enabled (B).

After the entire cache is flushed, cacheable entries are loaded first in way 0. If way 0 is occupied, the cacheable entry is loaded into the same set in way 1, as shown in Figure 4-4 (D). This process is described in detail in Section 4.9, “Cache Operation.”

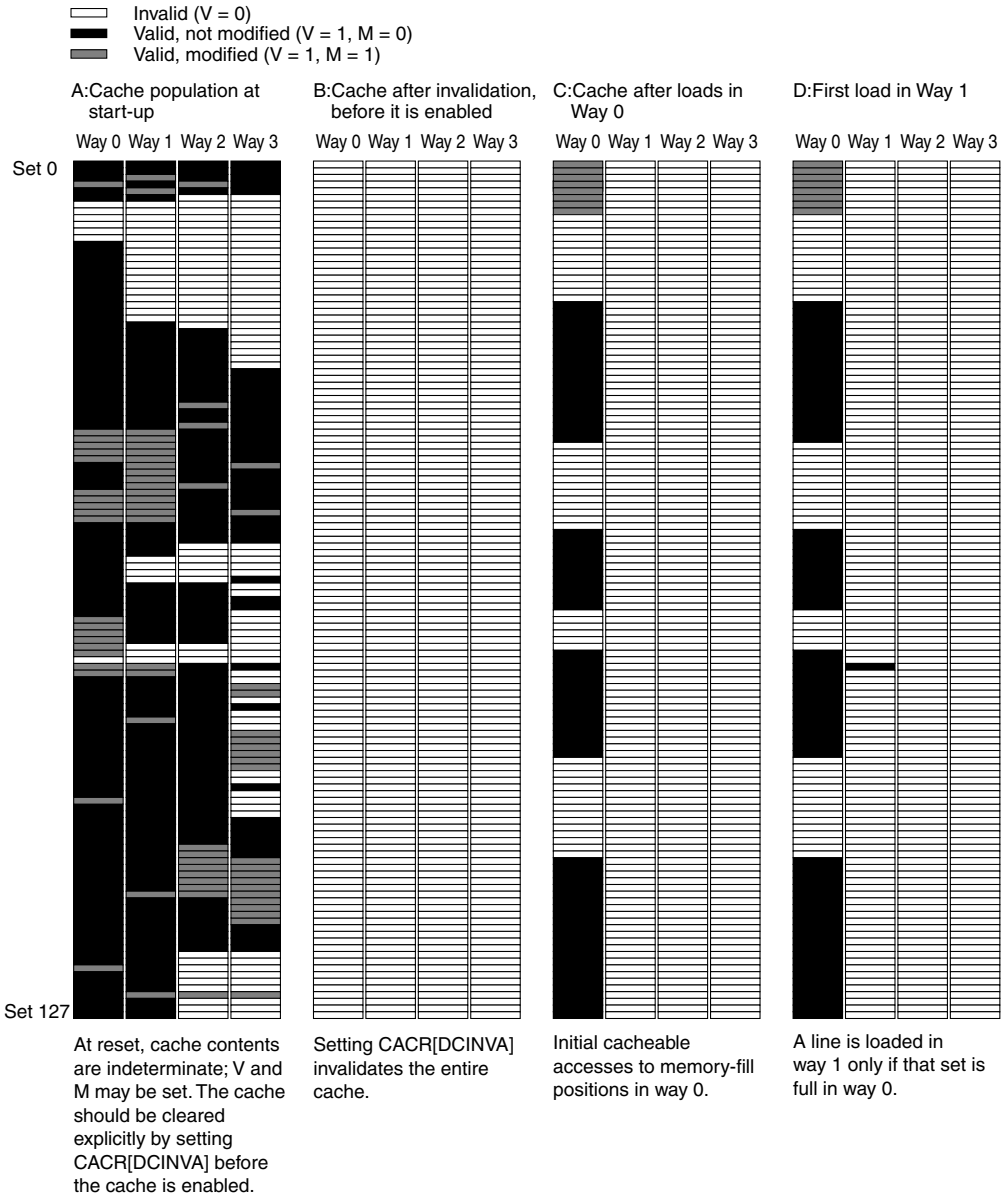



**Figure 4-4. Data Cache—A: at Reset, B: after Invalidation, C and D: Loading Pattern**

## 4.9 Cache Operation

Figure 4-5 shows the general flow of a caching operation using the 8-Kbyte data cache as an example. The discussion in this chapter assumes a data cache. Instruction cache operations are similar except that there is no support for writing to the cache; therefore such notions of modified cache lines and write allocation do not apply.

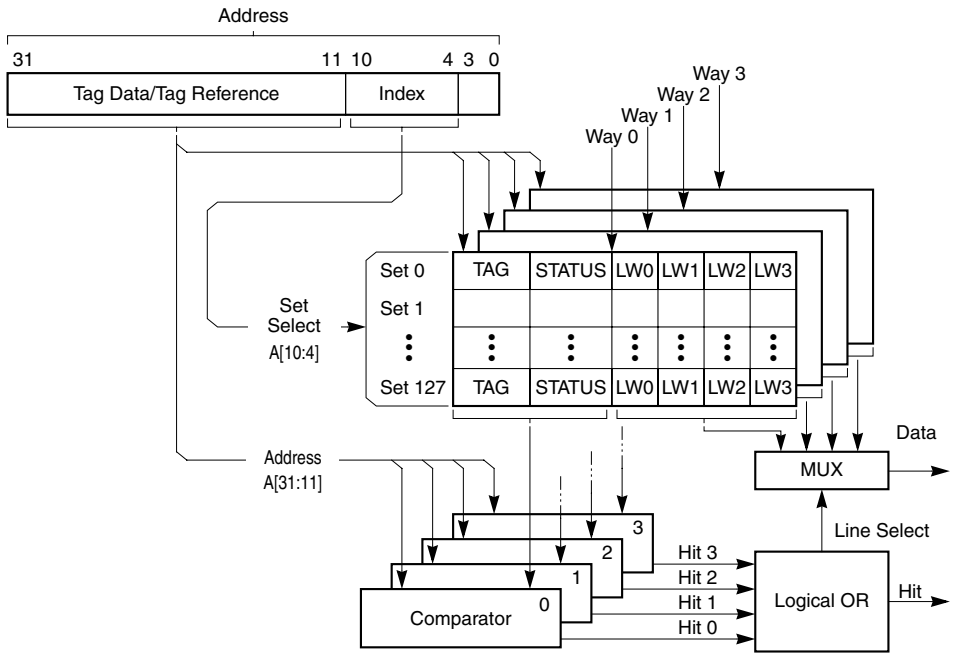



**Figure 4-5. Data Caching Operation**

The following steps determine if a data cache line is allocated for a given address:

1. The cache set index, A[10:4], selects one cache set.
2. A[31:11] and the cache set index are used as a tag reference or are used to update the cache line tag field. Note that A[31:11] can specify 21 possible addresses that can be mapped to one of the four ways.
3. The four tags from the selected cache set are compared with the tag reference. A cache hit occurs if a tag matches the tag reference and the V bit is set, indicating that the cache line contains valid data. If a cacheable write access hits in a valid cache line, the write can occur to the cache line without having to load it from memory.

   If the memory space is copyback, the updated cache line is marked modified (M = 1), because the new data has made the data in memory out of date. If the memory location is write-through, the write is passed on to system memory and the M bit is never used. Note that the tag does not have TT or TM bits.

To allocate a cache entry, the cache set index selects one of the cache's 128 sets. The cache control logic looks for an invalid cache line to use for the new entry. If none is available, the cache controller uses a pseudo-round-robin replacement algorithm to choose the line to be deallocated and replaced. First the cache controller looks for an invalid line, with way 0 the highest priority. If all lines have valid data, a 2-bit replacement counter is used to choose the way. After a line is allocated, the pointer increments to point to the next way.

Cache lines from ways 0 and 1 can be protected from deallocation by enabling half-cache locking. If CACR[DHLCK,IHLCK] = 1, the replacement pointer is restricted to way 2 or 3.

As part of deallocation, a valid, unmodified cache line is invalidated. It is consistent with system memory, so memory does not need to be updated. To deallocate a modified cache line, data is placed in a push buffer (for an external cache line push) before being invalidated. After invalidation, the new entry can replace it. The old cache line may be written after the new line is read.

When a cache line is selected to host a new cache entry, the following three things happen:

1. The new address tag bits A[31:11] are written to the tag.
2. The cache line is updated with the new memory data.
3. The cache line status changes to a valid state (V = 1).

Read cycles that miss in the cache allocate normally as previously described.

Write cycles that miss in the cache do not allocate on a cacheable write-through region, but do allocate for addresses in a cacheable copyback region**.**

A copyback byte, word, longword, or line write miss causes the following:

1. The cache initiates a line fill or flush.
2. Space is allocated for a new line.
3. V and M are both set to indicate valid and modified.
4. Data is written in the allocated space. No write to memory occurs.

Note the following:

- Read hits cannot change the status bits and no deallocation or replacement occurs; the data or instructions are read from the cache.
- If the cache hits on a write access, data is written to the appropriate portion of the accessed cache line. Write hits in cacheable, write-through regions generate an external write cycle and the cache line is marked valid, but is never marked modified. Write hits in cacheable copyback regions do not perform an external write cycle; the cache line is marked valid and modified (V = 1 and M = 1).
- Misaligned accesses are broken into at least two cache accesses.
- Validity is provided only on a line basis. Unless a whole line is loaded on a cache miss, the cache controller does not validate data in the cache line.

Write accesses designated as cache-inhibited by the CACR or ACR bypass the cache and perform a corresponding external write.

Normally, cache-inhibited reads bypass the cache and are performed on the external bus. The exception to this normal operation occurs when all of the following conditions are true during a cache-inhibited read:

- The cache-inhibited fill buffer bit, CACR[DNFB], is set.
- The access is an instruction read.
- The access is normal (that is, transfer type (TT) equals 0).

In this case, an entire line is fetched and stored in the fill buffer. It remains valid there, and the cache can service additional read accesses from this buffer until either another fill or a cache-invalidate-all operation occurs.

Valid cache entries that match during cache-inhibited address accesses are neither pushed nor invalidated. Such a scenario suggests that the associated cache mode for this address space was changed. To avoid this, it is generally recommended to use the CPUSHL instruction to push or invalidate the cache entry or set CACR[DCINVA] to invalidate the data cache before switching cache modes.

### 4.9.1 Caching Modes

For every memory reference generated by the processor or debug module, a set of effective attributes is determined based on the address and the ACRs. Caching modes determine how the cache handles an access. A data access can be cacheable in either write-through or copyback mode; it can be cache-inhibited in precise or imprecise modes. For normal accesses, the ACR*n*[CM] bit corresponding to the address of the access specifies the caching modes. If an address does not match an ACR, the default caching mode is defined by CACR[DDCM,IDCM]. The specific algorithm is as follows:

```
if (address == ACR0-address including mask)
        effective attributes = ACR0 attributes
else if (address == ACR1-address including mask)
                effective attributes = ACR1 attributes
        else effective attributes = CACR default attributes
```

Addresses matching an ACR can also be write-protected using ACR[W]. Addresses that do not match either ACR can be write-protected using CACR[DW].

Reset disables the cache and clears all CACR bits. As shown in Figure 4-4, reset does not automatically invalidate cache entries; they must be invalidated through software.

The ACRs allow the defaults selected in the CACR to be overridden. In addition, some instructions (for example, CPUSHL) and processor core operations perform accesses that have an implicit caching mode associated with them. The following sections discuss the different caching accesses and their associated cache modes.

#### 4.9.1.1 Cacheable Accesses

If ACR*n*[CM] or the default field of the CACR indicates write-through or copyback, the access is cacheable. A read access to a write-through or copyback region is read from the cache if matching data is found. Otherwise, the data is read from memory and the cache is updated. When a line is being read from memory for either a write-through or copyback read miss, the longword within the line that contains the core-requested data is loaded first and the requested data is given immediately to the processor, without waiting for the three remaining longwords to reach the cache.

The following sections describe write-through and copyback modes in detail. Note that some of this information applies to data caches only.

#### 4.9.1.2 Write-Through Mode (Data Cache Only)

Write accesses to regions specified as write-through are always passed on to the external bus, although the cycle can be buffered, depending on the state of CACR[DESB]. Writes in write-through mode are handled with a no-write-allocate policy—that is, writes that miss in the cache are written to the external bus but do not cause the corresponding line in memory to be loaded into the cache. Write accesses that hit always write through to memory and update matching cache lines. The cache supplies data to data-read accesses that hit in the cache; read misses cause a new cache line to be loaded into the cache.

#### 4.9.1.3 Copyback Mode (Data Cache Only)

Copyback regions are typically used for local data structures or stacks to minimize external bus use and reduce write-access latency. Write accesses to regions specified as copyback that hit in the cache update the cache line and set the corresponding M bit without an external bus access.

Be sure to flush the cache using the CPUSHL instruction before invalidating the cache in copyback mode. Modified cache data is written to memory only if the line is replaced because of a miss or a CPUSHL instruction pushes the line. If a byte, word, longword, or line write access misses in the cache, the required cache line is read from memory, thereby updating the cache. When a miss selects a modified cache line for replacement, the modified cache data moves to the push buffer. The replacement line is read into the cache and the push buffer contents are then written to memory.

### 4.9.2 Cache-Inhibited Accesses

Memory regions can be designated as cache-inhibited, which is useful for memory containing targets such as I/O devices and shared data structures in multiprocessing systems. It is also important to not cache the MCF5407 memory mapped registers. If the corresponding ACR*n*[CM] or CACR[DDCM] indicates cache-inhibited, precise or imprecise, the access is cache-inhibited. The caching operation is identical for both cache-inhibited modes, which differ only regarding recovery from an external bus error.

In determining whether a memory location is cacheable or cache-inhibited, the CPU checks memory-control registers in the following order:

1. RAMBARs
2. ACR0 and ACR2
3. ACR1 and ACR3
4. If an access does not hit in the RAMBARs or the ACRs, the default is provided for all accesses in CACR.

Cache-inhibited write accesses bypass the cache and a corresponding external write is performed. Cache-inhibited reads bypass the cache and are performed on the external bus, except when all of the following conditions are true:

- The cache-inhibited fill-buffer bit, CACR[DNFB], is set.
- The access is an instruction read.
- The access is normal (that is, TT = 0).

In this case, a fetched line is stored in the fill buffer and remains valid there; the cache can service additional read accesses from this buffer until another fill occurs or a cache-invalidate-all operation occurs.

If ACR*n*[CM] indicates cache-inhibited mode, precise or imprecise, the controller bypasses the cache and performs an external transfer. If a line in the cache matches the address and the mode is cache-inhibited, the cache does not automatically push the line if it is modified, nor does it invalidate the line if it is valid. Before switching cache mode, execute a CPUSHL instruction or set CACR[DCINVA,ICINVA] to invalidate the entire cache.

If ACR*n*[CM] indicates precise mode, the sequence of read and write accesses to the region is guaranteed to match the instruction sequence. In imprecise mode, the processor core allows read accesses that hit in the cache to occur before completion of a pending write from a previous instruction. Writes are not deferred past data-read accesses that miss the cache (that is, that must be read from the bus).

Precise operation forces data-read accesses for an instruction to occur only once by preventing the instruction from being interrupted after data is fetched. Otherwise, if the processor is not in precise mode, an exception aborts the instruction and the data may be accessed again when the instruction is restarted. These guarantees apply only when ACR*n*[CM] indicates precise mode and aligned accesses.

CPU space-register accesses, such as MOVEC, are treated as cache-inhibited and precise.

### 4.9.3 Cache Protocol

The following sections describe the cache protocol for processor accesses and assumes that the data is cacheable (that is, write-through or copyback). Note that the discussion of write operations applies to the data cache only.

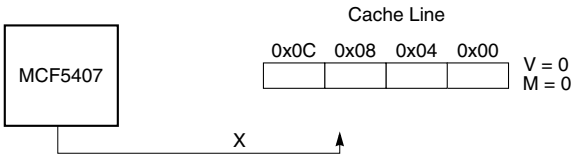
### 4.9.3.1 Read Miss

A processor read that misses in the cache requests the cache controller to generate a bus transaction. This bus transaction reads the needed line from memory and supplies the required data to the processor core. The line is placed in the cache in the valid state.

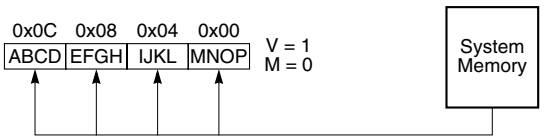
### 4.9.3.2 Write Miss (Data Cache Only)

The cache controller handles processor writes that miss in the data cache differently for write-through and copyback regions. Write misses to copyback regions cause the cache line to be read from system memory, as shown in Figure 4-6.

1. Writing character X to 0x0B generates a write miss. Data cannot be written to an invalid line.




2. The cache line (characters A–P) is updated from system memory, and line is marked valid.




3. After the cache line is filled, the write that initiated the write miss (the character X) completes to 0x0B.

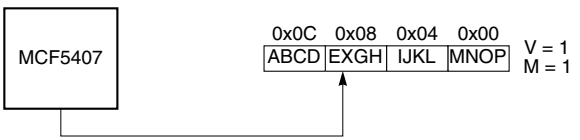



**Figure 4-6. Write-Miss in Copyback Mode**

The new cache line is then updated with write data and the M bit is set for the line, leaving it in modified state. Write misses to write-through regions write directly to memory without loading the corresponding cache line into the cache.

### 4.9.3.3 Read Hit

On a read hit, the cache provides the data to the processor core and the cache line state remains unchanged. If the cache mode changes for a specific region of address space, lines in the cache corresponding to that region that contain modified data are not pushed out to memory when a read hit occurs within that line. First execute a CPUSHL instruction or set CACR[DCINVA,ICINVA] before switching the cache mode.

#### 4.9.3.4 Write Hit (Data Cache Only)

The cache controller handles processor writes that hit in the data cache differently for write-through and copyback regions. For write hits to a write-through region, portions of cache lines corresponding to the size of the access are updated with the data. The data is also written to external memory. The cache line state is unchanged. For copyback accesses, the cache controller updates the cache line and sets the M bit for the line. An external write is not performed and the cache line state changes to (or remains in) the modified state.

### 4.9.4 Cache Coherency (Data Cache Only)

The MCF5407 provides limited cache coherency support in multiple-master environments. Both write-through and copyback memory update techniques are supported to maintain coherency between the cache and memory.

The cache does not support snooping (that is, cache coherency is not supported while external or DMA masters are using the bus). Therefore, on-chip DMAs of the MCF5407 cannot access local memory and do not maintain coherency with the data cache.

### 4.9.5 Memory Accesses for Cache Maintenance

The cache controller performs all maintenance activities that supply data from the cache to the core, including requests to the SIM for reading new cache lines and writing modified lines to memory. The following sections describe memory accesses resulting from cache fill and push operations. Chapter 18, “Bus Operation,” describes required bus cycles in detail.

#### 4.9.5.1 Cache Filling

When a new cache line is required, a line read is requested from the SIM, which generates a burst-read transfer by indicating a line access with the size signals, SIZ[1:0].

The responding device supplies 4 consecutive longwords of data. Burst operations can be inhibited or enabled through the burst read/write enable bits (BSTR/BSTW) in the chip-select control registers (CSCR0–CSCR7).

SIM line accesses implicitly request burst-mode operations from memory. For more information regarding external bus burst-mode accesses, see Chapter 18, “Bus Operation.”

The first cycle of a cache-line read loads the longword entry corresponding to the requested address. Subsequent transfers load the remaining longword entries.

A burst operation is aborted by an a write-protection fault, which is the only possible access error. Exception processing proceeds immediately. Note that unlike Version 2 and Version 3 access errors, the program counter stored on the exception stack frame points to the faulting instruction. See Section 2.8.2, “Processor Exceptions.”

### 4.9.5.2 Cache Pushes

Cache pushes occur for line replacement and as required for the execution of the CPUSHL instruction. To reduce the requested data's latency in the new line, the modified line being replaced is temporarily placed in the push buffer while the new line is fetched from memory. After the bus transfer for the new line completes, the modified cache line is written back to memory and the push buffer is invalidated.

#### 4.9.5.2.1 Push and Store Buffers

The 16-byte push buffer reduces latency for requested new data on a cache miss by holding a displaced modified data cache line while the new data is read from memory.

If a cache miss displaces a modified line, a miss read reference is immediately generated. While waiting for the response, the current contents of the cache location load into the push buffer. When the burst-read bus transaction completes, the cache controller can generate the appropriate line-write bus transaction to write the push buffer contents into memory.

In imprecise mode, the FIFO store buffer can defer pending writes to maximize performance. The store buffer can support as many as four entries (16 bytes maximum) for this purpose.

Data writes destined for the store buffer cannot stall the core. The store buffer effectively provides a measure of decoupling between the pipeline's ability to generate writes (one per cycle maximum) and the external bus's ability to retire those writes. In imprecise mode, writes stall only if the store buffer is full and a write operation is on the internal bus. The internal write cycle is held, stalling the data execution pipeline.

If the store buffer is not used (that is, store buffer disabled or cache-inhibited precise mode), external bus cycles are generated directly for each pipeline write operation. The instruction is held in the pipeline until external bus transfer termination is received. Therefore, each write is stalled for 5 cycles, making the minimum write time equal to 6 cycles when the store buffer is not used. See Section 2.1.2.2, "Operand Execution Pipeline (OEP)."

The data store buffer enable bit, CACR[DESB], controls the enabling of the data store buffer. This bit can be set and cleared by the MOVEC instruction. DESB is zero at reset and all writes are performed in order (precise mode). ACR*n*[CM] or CACR[DDCM] generates the mode used when DESB is set. Cacheable write-through and cache-inhibited imprecise modes use the store buffer.

The store buffer can queue data as much as 4 bytes wide per entry. Each entry matches the corresponding bus cycle it generates; therefore, a misaligned longword write to a write-through region creates two entries if the address is to an odd-word boundary. It creates three entries if it is to an odd-byte boundary—one per bus cycle.

#### 4.9.5.2.2 Push and Store Buffer Bus Operation

As soon as the push or store buffer has valid data, the internal bus controller uses the next available external bus cycle to generate the appropriate write cycles. In the event that

another cache fill is required (for example, cache miss to process) during the continued instruction execution by the processor pipeline, the pipeline stalls until the push and store buffers are empty, then generate the required external bus transaction.

Supervisor instructions, the NOP instruction, and exception processing synchronize the processor core and guarantee the push and store buffers are empty before proceeding. Note that the NOP instruction should be used only to synchronize the pipeline. The preferred no-operation function is the TPF instruction.

### 4.9.6 Cache Locking

Ways 0 and 1 of the data cache can be locked by setting CACR[DHLCK]; likewise, ways 0 and 1 of the instruction cache can be locked by setting CACR[IHLCK]. If a cache is locked, cache lines in ways 0 and 1 are not subject to being deallocated by normal cache operations.

As Figure 4-7 (B and C) shows, the algorithm for updating the cache and for identifying cache lines to be deallocated is otherwise unchanged. If ways 2 and 3 are entirely invalid, cacheable accesses are first allocated in way 2. Way 3 is not used until the location in way 2 is occupied.

Ways 0 and 1 are still updated on write hits (D in Figure 4-7) and may be pushed or cleared only explicitly by using specific cache push/invalidate instructions. However, new cache lines cannot be allocated in ways 0 and 1.

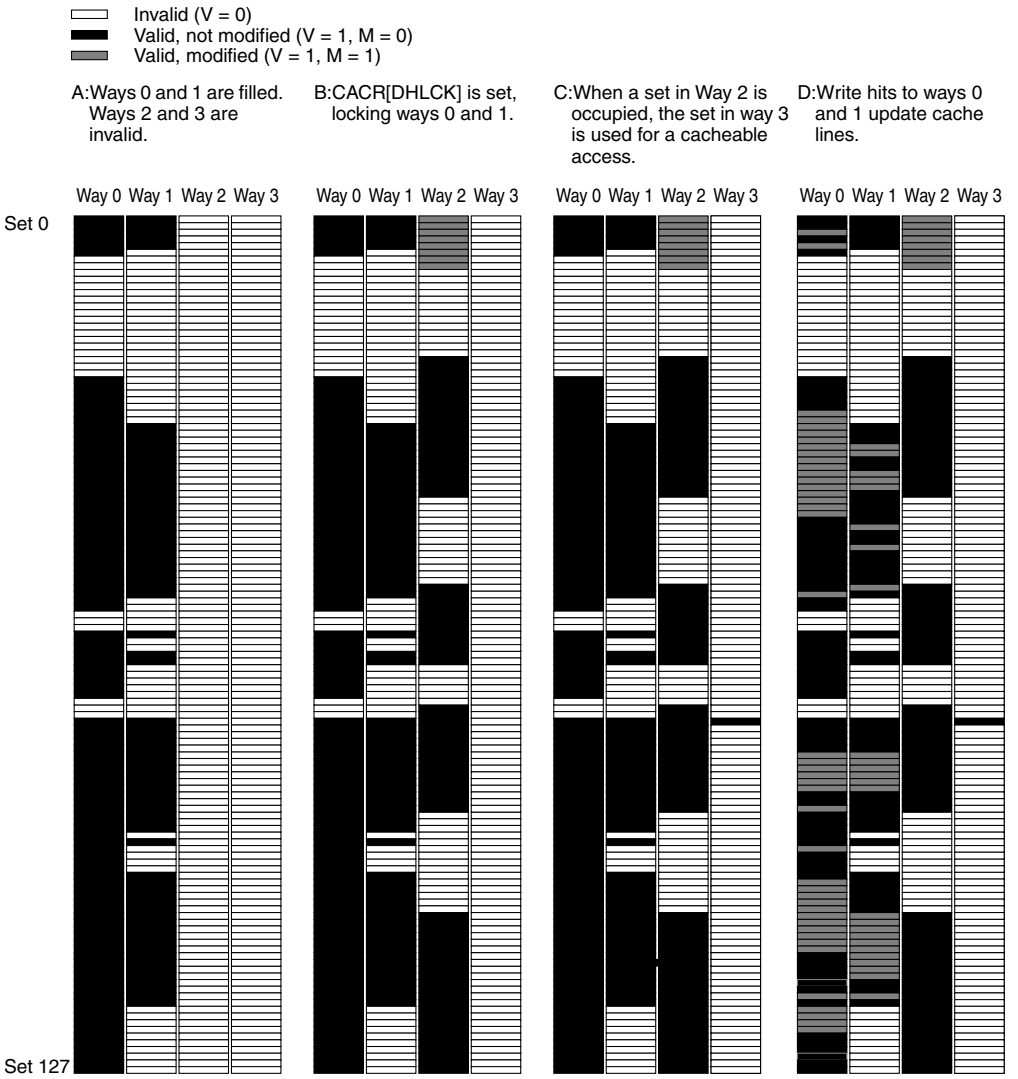



**Figure 4-7. Data Cache Locking**

# 4.10 Cache Registers

This section describes the MCF5407 implementation of the Version 4 cache registers.

## 4.10.1 Cache Control Register (CACR)

The CACR in Figure 4-8 contains bits for configuring the cache. It can be written by the MOVEC register instruction and can be read or written from the debug facility. A hardware reset clears CACR, which disables the cache; however, reset does not affect the tags, state information, or data in the cache.

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | DEC | DW | DESB | DDPI | DHLCK | DDCM | | DCINVA | — | | BEC | BCINVA | — | |
| Reset | 0000_0000_0000_0000 | | | | | | | | | | | | | |
| R/W | Write (R/W by debug module) | | | | | | | | | | | | | |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| Field | IEC | — | DNFB | IDPI | IHLCK | IDCM | — | ICINVA | — | |
| Reset | 0000_0000_0000_0000 | | | | | | | | | |
| R/W | Write (R/W by debug module) | | | | | | | | | |
| Rc | 0x002 | | | | | | | | | |

**Figure 4-8. Cache Control Register (CACR)**

Table 4-4 describes CACR fields.

**Table 4-4. CACR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31 | DEC | Enable data cache.<br>0 Cache disabled. The data cache is not operational, but data and tags are preserved.<br>1 Cache enabled. |
| 30 | DW | Data default write-protect. For normal operations that do not hit in the RAMBARs or ACRs, this field defines write-protection. See Section 4.9.1, “Caching Modes.”<br>0 Not write protected.<br>1 Write protected. Write operations cause an access error exception. |
| 29 | DESB | Enable data store buffer. Affects the precision of transfers. CACR[DESB] has precedence over CACR[9–8] and ACRn[9–8]; therefore, the store buffer must be disabled to use imprecise mode.<br>0 Imprecise-mode, write-through or cache-inhibited writes bypass the store buffer and generate bus cycles directly. Section 4.9.5.2.1, “Push and Store Buffers,” describes the associated performance penalty.<br>1 The four-entry FIFO store buffer is enabled; to maximize performance, this buffer defers pending imprecise-mode, write-through or cache-inhibited writes.<br>Precise-mode, cache-inhibited accesses always bypass the store buffer. Precise and imprecise modes are described in Section 4.9.2, “Cache-Inhibited Accesses.” |
| 28 | DDPI | Disable CPUSHL invalidation.<br>0 Normal operation. A CPUSHL instruction causes the selected line to be pushed if modified and then invalidated.<br>1 No clear operation. A CPUSHL instruction causes the selected line to be pushed if modified, then left valid. |

**Table 4-4. CACR Field Descriptions (Continued)**

| Bits | Name | Description |
|---|---|---|
| 27 | DHLCK | Half-data cache lock mode<br>0 Normal operation. The cache allocates the lowest invalid way. If all ways are valid, the cache allocates the way pointed at by the counter and then increments this counter modulo-4.<br>1 Half-cache operation. The cache allocates to the lower invalid way of levels 2 and 3; if both are valid, the cache allocates to way 2 if the high-order bit of the round-robin counter is zero; otherwise, it allocates way 3 and increments the round-robin counter modulo-2. This locks the content of ways 0 and 1. Ways 0 and 1 are still updated on write hits and may be pushed or cleared by specific cache push/invalidate instructions.<br>This implementation allows maximum use of available cache memory and provides the flexibility of setting DHLCK before, during, or after allocations occur. |
| 26–25 | DDCM | Default data cache mode. For normal operations that do not hit in the RAMBARs or ACRs, this field defines the effective cache mode.<br>00 Cacheable write-through imprecise<br>01 Cacheable copyback<br>10 Cache-inhibited precise<br>11 Cache-inhibited imprecise<br>Precise and imprecise accesses are described in Section 4.9.2, “Cache-Inhibited Accesses.” |
| 24 | DCINVA | Data cache invalidate all. Writing a 1 to this bit initiates entire cache invalidation. Once invalidation is complete, this bit automatically returns to 0; it is not necessary to clear it explicitly. Note the caches are not cleared on power-up or normal reset, as shown in Figure 4-4.<br>0 No invalidation is performed.<br>1 Initiate invalidation of the entire data cache. The cache controller sequentially clears V and M bits in all sets. Subsequent data accesses stall until the invalidation is finished, at which point, this bit is automatically cleared. In copyback mode, the cache should be flushed using a CPUSHL instruction before setting this bit. |
| 23–20 | — | Reserved, should be cleared. |
| 19 | BEC | Enable branch cache. The branch cache is described in Section 2.1.2.1.1, “Branch Acceleration.”<br>0 Branch cache disabled. This may be useful if code is unlikely to be reused.<br>1 Branch cache enabled. |
| 18 | BCINVA | Branch cache invalidate. Invalidation occurs when this bit is written as a 1. Note that branch caches are not cleared on power-up or normal reset.<br>0 No invalidation is performed.<br>1 Initiate an invalidation of the entire branch cache. |
| 17–16 | — | Reserved. These bits must be cleared; otherwise performance may be affected. |
| 15 | IEC | Enable instruction cache<br>0 Instruction cache disabled. All instructions and tags in the cache are preserved.<br>1 Instruction cache enabled. |
| 14 | — | Reserved, should be cleared. |
| 13 | DNFB | Default cache-inhibited fill buffer<br>0 Fill buffer does not store cache-inhibited instruction accesses (16 or 32 bits).<br>1 Fill buffer can store cache-inhibited accesses. The buffer is used only for normal (TT = 0) instruction reads of a cache-inhibited region. Instructions are loaded into the buffer by a burst access (line fill). They stay in the buffer until they are displaced; subsequent accesses may not appear on the external bus.<br>Setting DNFB can cause a coherency problem for self-modifying code. If a cache-inhibited access uses the buffer while DNFB = 1, instructions remain valid in the buffer until a cache-invalidate-all instruction, another cache-inhibited burst, or a miss that initiates a fill. A write to the line in the fill goes to the external bus without updating or invalidating the buffer. Subsequent reads of that written data are serviced by the fill buffer and receive stale information. |
| 12 | IDPI | Instruction CPUSHL invalidate disable.<br>0 Normal operation. A CPUSHL instruction causes the selected line to be invalidated.<br>1 No clear operation. A CPUSHL instruction causes the selected line to be left valid. |

**Table 4-4. CACR Field Descriptions (Continued)**

| Bits | Name | Description |
|---|---|---|
| 11 | IHLCK | Instruction cache half-lock.<br>0 Normal operation. The cache allocates to the lowest invalid way; if all ways are valid, the cache allocates to the way pointed at by the round-robin counter and then increments this counter modulo-4.<br>1 Half cache operation. The cache allocates to the lowest invalid way of ways 2 and 3; if both of these ways are valid, the cache allocates to way 2 if the high-order bit of the round-robin counter is zero; otherwise, it allocates way 3 and then increments the round-robin counter modulo-2. This locks the content of ways 0 and 1. Ways 0 and 1 are still updated on write hits and may be pushed or cleared by specific cache push/invalidate instructions.<br>This implementation allows maximum use of the available cache memory and also provides the flexibility of setting IHLCK before, during, or after the needed allocations occur. |
| 10 | IDCM | Instruction default cache mode. For normal operations that do not hit in the RAMBARs or ACRs, this field defines the effective cache mode.<br>0 Cacheable<br>1 Cache-inhibited |
| 9 | — | Reserved, should be cleared. |
| 8 | ICINVA | Instruction cache invalidate. Invalidation occurs when this bit is written as a 1. Note the caches are not cleared on power-up or normal reset.<br>0 No invalidation is performed.<br>1 Initiate invalidation of instruction cache. The cache controller sequentially clears all V bits. Subsequent local memory bus accesses stall until invalidation completes, at which point, ICINVA is cleared automatically without software intervention. For copyback mode, use CPUSHL before setting ICINVA. |
| 7–0 | — | Reserved. These bits must be cleared; otherwise, performance may be affected. |

## 4.10.2 Access Control Registers (ACR0–ACR3)

The ACRs, Figure 4-9, assign control attributes, such as cache mode and write protection, to specified memory regions. ACR0 and ACR1 control data attributes; ACR2 and ACR3 control instruction attributes. Registers are accessed with the MOVEC instruction with the Rc encodings in Figure 4-9.

For overlapping data regions, ACR0 takes priority; ACR2 takes priority for overlapping instruction regions. Data transfers to and from these registers are longword transfers. Bits 12–7, 4, 3, 1, and 0 are always read as zeros.

**NOTE:**

The SIM MBAR region should be mapped as cache-inhibited through an ACR.

| | 31 24 | 23 16 | 15 | 14 13 | 12 7 | 6 5 | 4 3 | 2 | 1 0 |
|---|---|---|---|---|---|---|---|---|---|
| Field | Address Base | Address Mask | E | S | — | CM | — | W[1] | — |
| Reset | Uninitialized | | 0 | Uninitialized | | | | | |
| R/W | Write (R/W by debug module) | | | | | | | | |
| Rc | ACR0: 0x004; ACR1: 0x005; ACR2: 0x006; ACR3: 0x007 | | | | | | | | |

[1] Reserved in ACR2 and ACR3.

**Figure 4-9. Access Control Register Format (ACRn)**

Table 4-5 describes ACR*n* fields.

**Table 4-5. ACRn Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–24 | Address base | Address base. Compared with address bits A[31:24]. Eligible addresses that match are assigned the access control attributes of this register. |
| 23–16 | Address mask | Address mask. Setting a mask bit causes the corresponding address base bit to be ignored. The low-order mask bits can be set to define contiguous regions larger than 16 Mbytes. The mask can define multiple noncontiguous regions of memory. |
| 15 | E | Enable. Enables or disables the other ACRn bits.<br>0 Access control attributes disabled<br>1 Access control attributes enabled |
| 14–13 | S | Supervisor mode. Specifies whether only user or supervisor accesses are allowed in this address range or if the type of access is a don't care.<br>00 Match addresses only in user mode<br>01 Match addresses only in supervisor mode<br>1x Execute cache matching on all accesses |
| 12–7 | — | Reserved; should be cleared. |
| 6–5 | CM | Cache mode. Selects the cache mode and access precision. Precise and imprecise modes are described in Section 4.9.2, "Cache-Inhibited Accesses."<br>00 Cacheable, write-through<br>01 Cacheable, copyback<br>10 Cache-inhibited, precise<br>11 Cache-inhibited, imprecise |
| 4–3 | — | Reserved, should be cleared. |
| 2 | W | ACR0/ACR1 only. Write protect. Selects the write privilege of the memory region. ACR2[2] and ACR3[2] are reserved.<br>0 Read and write accesses permitted<br>1 Write accesses not permitted |
| 1–0 | — | Reserved, should be cleared. |

## 4.11 Cache Management

The cache can be enabled and configured by using a MOVEC instruction to access CACR. A hardware reset clears CACR, disabling the cache and removing all configuration information; however, reset does not affect the tags, state information, and data in the cache.

Set CACR[DCINVA,ICINVA] to invalidate the caches before enabling them.

The privileged CPUSHL instruction supports cache management by selectively pushing and invalidating cache lines. The address register used with CPUSHL directly addresses the cache's directory array. The CPUSHL instruction flushes a cache line.

The value of CACR[DDPI,IDPI] determines whether CPUSHL invalidates a cache line after it is pushed. To push the entire cache, implement a software loop to index through all sets and through each of the four lines within each set (a total of 512 lines for the data cache and 1024 lines for the instruction cache). The state of CACR[DEC,IEC] does not affect the operation of CPUSHL or CACR[DCINVA,ICINVA]. Disabling a cache by setting CACR[IEC] or CACR[DEC] makes the cache nonoperational without affecting tags, state information, or contents.

The contents of A*n* used with CPUSHL specify cache row and line indexes. This differs from the MC68040 where a physical address is specified. Figure 4-11 shows the A*n* format for the data cache.

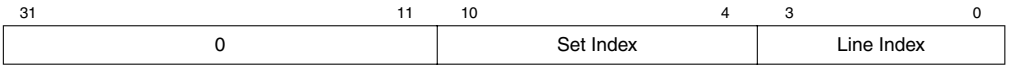



**Figure 4-10. An Format (Data Cache)**

Figure 4-11 shows the A*n* format for the instruction cache.

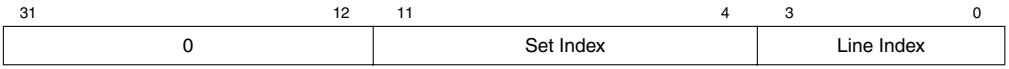



**Figure 4-11. An Format (Instruction Cache)**

The following code example flushes the entire data cache:

```
_cache_disable:
        nop
        move.w          #0x2700,SR      ;mask off IRQs
        jsr             _cache_flush    ;flush the cache completely
        clr.l           d0
        movec           d0,ACR0         ;ACR0 off
        movec           d0,ACR1         ;ACR1 off
        move.l          #0x01000000,d0  ;Invalidate and disable cache
        movec           d0,CACR
        rts

_cache_flush:
        nop                             ;synchronize—flush store buffer
        moveq.l         #0,d0           ;initialize way counter
        moveq.l         #0,d1           ;initialize set counter
        move.l          d0,a0           ;initialize cpushl pointer

setloop:
        cpushl          dc,(a0)         ;push cache line a0
        add.l           #0x0010,a0      ;increment set index by 1
        addq.l          #1,d1           ;increment set counter
        cmpi.l          #128,d1         ;are sets for this way done?
        bne             setloop

        moveq.l         #0,d1           ;set counter to zero again
```

```
        addq.l          #1,d0           ;increment to next way
        move.l          d0,a0           ;set = 0, way = d0
        cmpi.l          #4,d0           ;flushed all the ways?
        bne             setloop
        rts
```

The following CACR loads assume the instruction cache has been invalidated, the default instruction cache mode is cacheable, and the default data cache mode is copyback.

```
dataCacheLoadAndLock:

        move.l  #0xa3080800,d0; enable and invalidate data cache ...
        movec   d0,cacr ; ... in the CACR
```

The following code preloads half of the data cache (4 Kbytes). It assumes a contiguous block of data is to be mapped into the data cache, starting at a 0-modulo-4K address.

```
        move.l  #256,d0         ;256 16-byte lines in 4K space
        lea     data_,a0        ; load pointer defining data area
dataCacheLoop:
        tst.b   (a0)            ;touch location + load into data cache
        lea     16(a0),a0       ;increment address to next line
        subq.l  #1,d0           ;decrement loop counter
        bne.b   dataCacheLoop   ;if done, then exit, else continue

; A 4K region has been loaded into levels 0 and 1 of the 8K data cache. lock it!

        move.l  #0xaa088000,d0  ;set the data cache lock bit ...
        movec   d0,cacr         ; ... in the CACR
        rts

        align   16
```

The following CACR loads assume the data cache has been invalidated, the default instruction cache mode is cacheable and the default operand cache mode is copyback.

Note that this function must be mapped into a cache inhibited or SRAM space or these text lines will be prefetched into the instruction cache, which may displace some of the 8-Kbyte space being explicitly fetched.

```
instructionCacheLoadAndLock:

        move.l  #0xa2088100,d0  ;enable and invalidate the instruction
        movec   d0,cacr         ;cache in the CACR
```

The following code segments preload half of the instruction cache (8 Kbytes). It assumes a contiguous block of data is to be mapped, starting at a 0-modulo-8K address

```
        move.l  #512,d0         ;512 16-byte lines in 8K space
        lea     code_,a0        ;load pointer defining code area
instCacheLoop:
;       intouch (a0)            ;touch location + load into instruction cache


; Note in the assembler we use, there is no INTOUCH opcode. The following
; is used to produce the required binary representation

        cpushl  #nc,(a0)        ;touch location + load into
```

```
                                ;instruction cache
        lea     16(a0),a0       ;increment address to next line
        subq.l  #1,d0           ;decrement loop counter
        bne.b   instCacheLoop   ;if done, then exit, else continue
; A 8K region was loaded into levels 0 and 1 of the 16-Kbyte instruction cache.
; lock it!

        move.l  #0xa2088800,d0  ;set the instruction cache lock bit
        movec   d0,cacr         ;in the CACR
        rts
```

# 4.12 Cache Operation Summary

This section gives operational details for the cache and presents instruction and data cache-line state diagrams.

## 4.12.1 Instruction Cache State Transitions

Because the instruction cache does not support writes, it supports fewer operations than the data cache. As Figure 4-12 shows, an instruction cache line can be in one of two states, valid or invalid. Modified state is not supported. Transitions are labeled with a capital letter indicating the previous state and with a number indicating the specific case listed in Table 4-6. These numbers correspond to the equivalent operations on data caches, described in Section 4.12.2, “Data Cache State Transitions.”

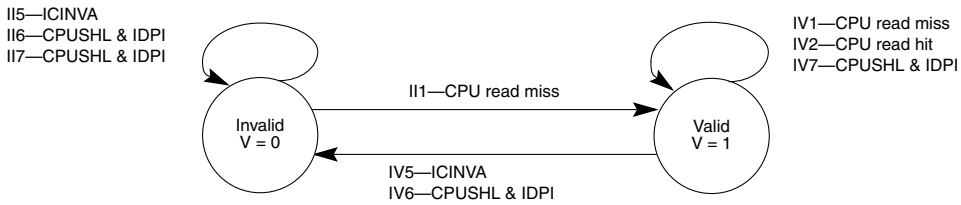



**Figure 4-12. Instruction Cache Line State Diagram**

Table 4-6 describes the instruction cache state transitions shown in Figure 4-12.

**Table 4-6. Instruction Cache Line State Transitions**

| Access | Current State | | | |
|---|---|---|---|---|
| | Invalid (V = 0) | | Valid (V = 1) | |
| Read miss | II1 | Read line from memory and update cache; supply data to processor; go to valid state. | IV1 | Read new line from memory and update cache; supply data to processor; stay in valid state. |
| Read hit | II2 | Not possible | IV2 | Supply data to processor; stay in valid state. |
| Write miss | II3 | Not possible | IV3 | Not possible |
| Write hit | II4 | Not possible | IV4 | Not possible |

**Table 4-6. Instruction Cache Line State Transitions (Continued)**

| Access | Current State | | | |
|---|---|---|---|---|
| | Invalid (V = 0) | | Valid (V = 1) | |
| Cache invalidate | II5 | No action; stay in invalid state. | IV5 | No action; go to invalid state. |
| Cache push | II6, II7 | No action; stay in invalid state. | IV6 | No action; go to invalid state. |
| | | | IV7 | No action; stay in valid state. |

## 4.12.2 Data Cache State Transitions

Using the V and M bits, the data cache supports a line-based protocol allowing individual cache lines to be invalid, valid, or modified. To maintain memory coherency, the data cache supports both write-through and copyback modes, specified by the corresponding ACR[CM], or CACR[DDCM] if no ACR matches.

Read or write misses to copyback regions cause the cache controller to read a cache line from memory into the cache. If available, tag and data from memory update an invalid line in the selected set. The line state then changes from invalid to valid by setting the V bit. If all lines in the row are already valid or modified, the pseudo-round-robin replacement algorithm selects one of the four lines and replaces the tag and data. Before replacement, modified lines are temporarily buffered and later copied back to memory after the new line has been read from memory.

Figure 4-13 shows the three possible data cache line states and possible processor-initiated transitions for memory configured as copyback. Transitions are labeled with a capital letter indicating the previous state and a number indicating the specific case listed in Table 4-7.

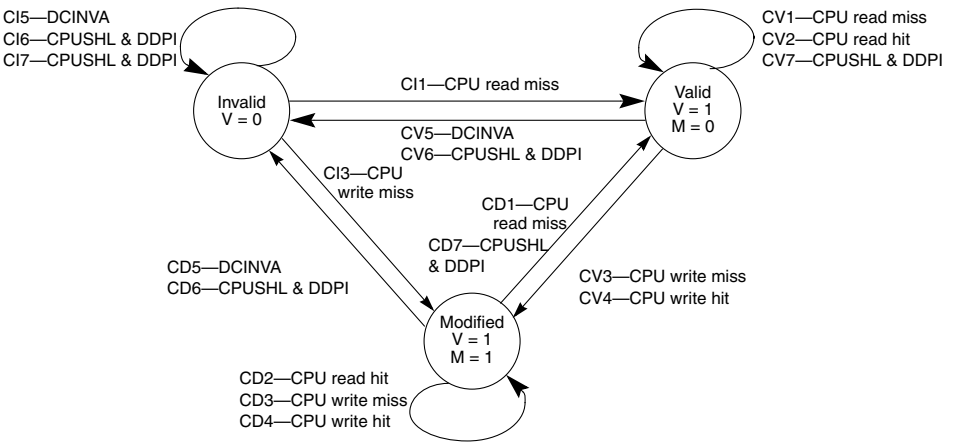



**Figure 4-13. Data Cache Line State Diagram—Copyback Mode**

Figure 4-14 shows the two possible states for a cache line in write-through mode.

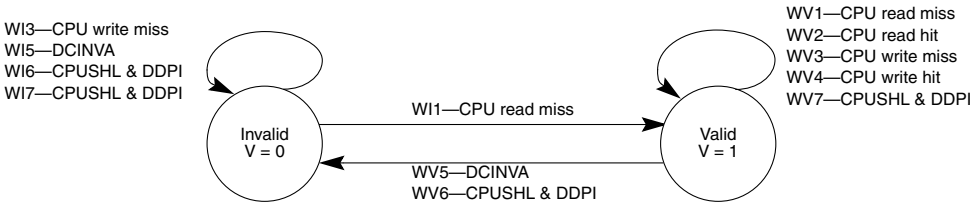



**Figure 4-14. Data Cache Line State Diagram—Write-Through Mode**

Table 4-7 describes data cache line transitions and the accesses that cause them.

**Table 4-7. Data Cache Line State Transitions**

| Access | Current State | | | | | |
|---|---|---|---|---|---|---|
| | Invalid (V = 0) | | Valid (V = 1, M = 0) | | Modified (V = 1, M = 1) | |
| Read miss | (C,W)I1 | Read line from memory and update cache;<br>supply data to processor;<br>go to valid state. | (C,W)V1 | Read new line from memory and update cache;<br>supply data to processor;<br>stay in valid state. | CD1 | Push modified line to buffer;<br>read new line from memory and update cache;<br>supply data to processor;<br>write push buffer contents to memory;<br>go to valid state. |
| Read hit | (C,W)I2 | Not possible. | (C,W)V2 | Supply data to processor;<br>stay in valid state. | CD2 | Supply data to processor;<br>stay in modified state. |
| Write miss (copy-back) | CI3 | Read line from memory and update cache;<br>write data to cache;<br>go to modified state. | CV3 | Read new line from memory and update cache;<br>write data to cache;<br>go to modified state. | CD3 | Push modified line to buffer;<br>read new line from memory and update cache;<br>write push buffer contents to memory;<br>stay in modified state. |
| Write miss (write-through) | WI3 | Write data to memory;<br>stay in invalid state. | WV3 | Write data to memory;<br>stay in valid state. | WD3 | Write data to memory;<br>stay in modified state.<br>Cache mode changed for the region corresponding to this line. To avoid this state, execute a CPUSHL instruction or set CACR[DCINVA,ICINVA] before switching modes. |
| Write hit (copy-back) | CI4 | Not possible. | CV4 | Write data to cache;<br>go to modified state. | CD4 | Write data to cache;<br>stay in modified state. |

**Table 4-7. Data Cache Line State Transitions (Continued)**

| Access | Current State | | | | | |
|---|---|---|---|---|---|---|
| | Invalid (V = 0) | | Valid (V = 1, M = 0) | | Modified (V = 1, M = 1) | |
| Write hit (write-through) | WI4 | Not possible. | WV4 | Write data to memory and to cache;<br>stay in valid state. | WD4 | Write data to memory and to cache;<br>go to valid state.<br>Cache mode changed for the region corresponding to this line. To avoid this state, execute a CPUSHL instruction or set CACR[DCINVA,ICINVA] before switching modes. |
| Cache invalidate | (C,W)I5 | No action;<br>stay in invalid state. | (C,W)V5 | No action;<br>go to invalid state. | CD5 | No action (modified data lost);<br>go to invalid state. |
| Cache push | (C,W)I6<br>(C,W)I7 | No action;<br>stay in invalid state. | (C,W)V6 | No action;<br>go to invalid state. | CD6 | Push modified line to memory;<br>go to invalid state. |
| | | | (C,W)V7 | No action;<br>stay in valid state. | CD7 | Push modified line to memory;<br>go to valid state. |

The following tables present the same information as Table 4-7, organized by the current state of the cache line. In Table 4-8 the current state is invalid.

**Table 4-8. Data Cache Line State Transitions (Current State Invalid)**

| Access | Response | |
|---|---|---|
| Read miss | (C,W)I1 | Read line from memory and update cache;<br>supply data to processor;<br>go to valid state. |
| Read hit | (C,W)I2 | Not possible |
| Write miss (copyback) | CI3 | Read line from memory and update cache;<br>write data to cache;<br>go to modified state. |
| Write miss (write-through) | WI3 | Write data to memory;<br>stay in invalid state. |
| Write hit (copyback) | CI4 | Not possible |
| Write hit (write-through) | WI4 | Not possible |
| Cache invalidate | (C,W)I5 | No action;<br>stay in invalid state. |
| Cache push | (C,W)I6 | No action;<br>stay in invalid state. |
| Cache push | (C,W)I7 | No action;<br>stay in invalid state. |

In Table 4-9 the current state is valid.

**Table 4-9. Data Cache Line State Transitions (Current State Valid)**

| Access | Response | |
|---|---|---|
| Read miss | (C,W)V1 | Read new line from memory and update cache;<br>supply data to processor; stay in valid state. |
| Read hit | (C,W)V2 | Supply data to processor;<br>stay in valid state. |
| Write miss (copyback) | CV3 | Read new line from memory and update cache;<br>write data to cache;<br>go to modified state. |
| Write miss (write-through) | WV3 | Write data to memory;<br>stay in valid state. |
| Write hit (copyback) | CV4 | Write data to cache;<br>go to modified state. |
| Write hit (write-through) | WV4 | Write data to memory and to cache;<br>stay in valid state. |
| Cache invalidate | (C,W)V5 | No action;<br>go to invalid state. |
| Cache push | (C,W)V6 | No action;<br>go to invalid state. |
| Cache push | (C,W)V7 | No action;<br>stay in valid state. |

In Table 4-10 the current state is modified.

**Table 4-10. Data Cache Line State Transitions (Current State Modified)**

| Access | Response | |
|---|---|---|
| Read miss | CD1 | Push modified line to buffer;<br>read new line from memory and update cache;<br>supply data to processor;<br>write push buffer contents to memory;<br>go to valid state. |
| Read hit | CD2 | Supply data to processor;<br>stay in modified state. |
| Write miss (copyback) | CD3 | Push modified line to buffer;<br>read new line from memory and update cache;<br>write push buffer contents to memory;<br>stay in modified state. |
| Write miss (write-through) | WD3 | Write data to memory;<br>stay in modified state.<br>Cache mode changed for the region corresponding to this line. To avoid this state, execute a CPUSHL instruction or set CACR[DCINVA,ICINVA] before switching modes. |
| Write hit (copyback) | CD4 | Write data to cache;<br>stay in modified state. |
| Write hit (write-through) | WD4 | Write data to memory and to cache;<br>go to valid state.<br>Cache mode changed for the region corresponding to this line. To avoid this state, execute a CPUSHL instruction or set CACR[DCINVA,ICINVA] before switching modes. |

**Table 4-10. Data Cache Line State Transitions (Current State Modified) (Continued)**

| Access | Response | |
|---|---|---|
| Cache invalidate | CD5 | No action (modified data lost);<br>go to invalid state. |
| Cache push | CD6 | Push modified line to memory;<br>go to invalid state. |
| Cache push | CD7 | Push modified line to memory;<br>go to valid state. |

## 4.13 Cache Initialization Code

The following example sets up the cache for FLASH or ROM space only.

```
move.l#0xA30C8100,D0 //enable cache, invalidate it,
                     //default mode is cache-inhibited imprecise
movecD0, CACR

move.l #0xFF00C000,D0//cache FLASH space, enable,
                     //ignore FC2, cacheable, writethrough
movecD0,ACR0
```

# Chapter 5
# Debug Support

This chapter describes the Revision C enhanced hardware debug support in the MCF5407. This revision of the ColdFire debug architecture encompasses the two earlier revisions.

## 5.1 Overview

The debug module is shown in Figure 5-1.

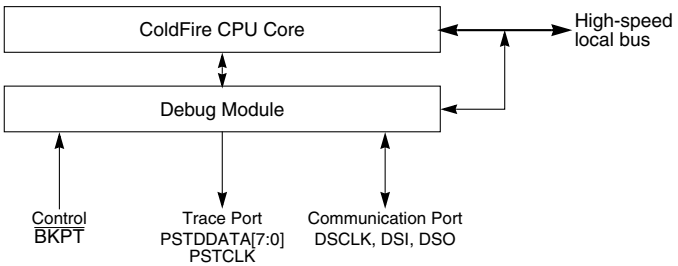



**Figure 5-1. Processor/Debug Module Interface**

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 5.3, “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 and registers. The external emulator uses a three-pin, serial, full-duplex channel. See Section 5.5, “Background Debug Mode (BDM),” and Section 5.4, “Programming Model.”
- Real-time debug support—BDM requires the processor to be halted, which many real-time embedded applications cannot do. Debug interrupts let real-time systems execute a unique service routine that can quickly save the contents of key registers and variables and return the system to normal operation. The emulator can access saved data because the hardware supports concurrent operation of the processor and BDM-initiated commands. See Section 5.6, “Real-Time Debug Support.”

The Version 2 ColdFire core implemented the original debug architecture, now called Revision A. Based on feedback from customers and third-party developers, enhancements have been added to succeeding generations of ColdFire cores. The Version 3 core implements the Revision B of the debug architecture, providing more flexibility for configuring the hardware breakpoint trigger registers and removing the restrictions involving concurrent BDM processing while hardware breakpoint registers are active.

The MCF5407 core implements Revision C of the debug architecture, which more than doubles the on-chip breakpoint registers and provides an ability to interrupt debug service routines. For Revision C, the revision level bit, CSR[HRL], is 2. See Section 5.4.4, “Configuration/Status Register (CSR).”

## 5.2 Signal Descriptions

Table 5-1 describes debug module signals. All ColdFire debug signals are unidirectional and related to a rising edge of the processor core’s clock signal. The standard 26-pin debug connector is shown in Section 5.7, “Motorola-Recommended BDM Pinout.”

**Table 5-1. Debug Module Signals**

| Signal | Description |
|---|---|
| Development Serial Clock (DSCLK) | Internally synchronized input that clocks the serial communication port to the debug module. Maximum frequency is 1/5 the processor CLK speed. At the synchronized rising edge of DSCLK, the data input on DSI is sampled and DSO changes state. The logic level on DSCLK is validated if it has the same value on two consecutive rising CLKIN edges. |
| 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 ($\overline{\text{BKPT}}$) | 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/debug data signals (PSTDDATA[7:0]) as the value 0xF. If CSR[BKD] is set (disabling normal $\overline{\text{BKPT}}$ functionality), asserting $\overline{\text{BKPT}}$ generates a debug interrupt exception in the processor. |
| Processor Status Clock (PSTCLK) | Half-speed version of the processor clock. Its rising edge appears in the center of the two processor-cycle window of valid PSTDDATA output. See Figure 5-2. Because debug trace port signals change on alternate processor cycles and are unrelated to external bus frequency, PSTCLK helps the development system sample PSTDDATA values.<br>If real-time trace is not used, setting CSR[PCD] keeps PSTCLK and PSTDDATA outputs from toggling without disabling triggers. Non-quiescent operation can be reenabled by clearing CSR[PCD], although the emulator must resynchronize with the PSTDDATA output.<br>PSTCLK starts clocking only when the first non-zero PST value (0xC, 0xD, or 0xF) occurs during system reset exception processing. Table 5-4 describes PST values. Chapter 7, “Phase-Locked Loop (PLL),” describes PSTCLK generation. |
| Processor Status/Debug Data (PSTDDATA[7:0]) | These outputs indicate both processor status and captured address and data values and are discussed more thoroughly in Section 5.2.1, “Processor Status/Debug Data (PSTDDATA[7:0]). |

Figure 5-2 shows PSTCLK timing.

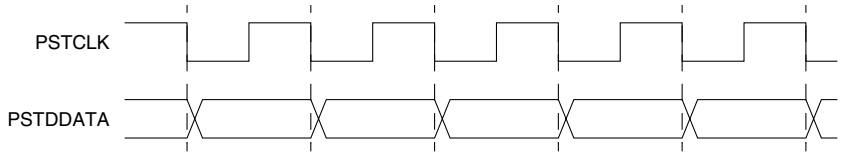



**Figure 5-2. PSTCLK Timing**

## 5.2.1 Processor Status/Debug Data (PSTDDATA[7:0])

Processor status data outputs are used to indicate both processor status and captured address and data values. They operate at half the processor's frequency. Given that real-time trace information appears as a sequence of 4-bit data values, there are no alignment restrictions; that is, the processor status (PST) values and operands may appear on either nibble of PSTDDATA[7:0]. The upper nibble (PSTDDATA[7:4]) is the most significant.

CSR controls capturing of data values. Executing the WDDATA instruction captures data displayed on PSTDDATA. These signals are updated each processor cycle and displayed two values at a time for two processor clock cycles. Table 5-2 shows the PSTDDATA output for the processor's sequential execution of single-cycle instructions (A, B, C, D...). Cycle counts are shown relative to processor frequency. These outputs indicate the current processor pipeline status and are not related to the current bus transfer.

**Table 5-2. PSTDDATA: Sequential Execution of Single-Cycle Instructions**

| Cycle | PSTDDATA[7:0] |
|---|---|
| T | {PST for A, PST for B} |
| T+1 | {PST for A, PST for B} |
| T+2 | {PST for C, PST for D} |
| T+3 | {PST for C, PST for D} |
| T+4 | {PST for E, PST for F} |
| T+5 | {PST for E, PST for F} |

The signal timing for the example in Table 5-2 is shown in Figure 5-3.

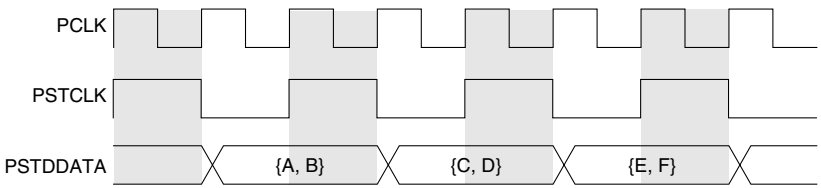



**Figure 5-3. PSTDDATA: Single-Cycle Instruction Timing**

Table 5-3 shows the case where a PSTDDATA module captures a memory operand on a simple load instruction: mov.l <mem>,Rx.

**Table 5-3. PSTDDATA: Data Operand Captured**

| Cycle | PSTDDATA[7:0] |
|---|---|
| T | {PST for mov.l, PST marker for captured operand) = {0x1, 0xB} |
| T+1 | {0x1, 0xB} |
| T+2 | {Operand[3:0], Operand[7:4]} |
| T+3 | {Operand[3:0], Operand[7:4]} |
| T+4 | {Operand[11:8], Operand[15:12]} |
| T+5 | {Operand[11:8], Operand[15:12]} |
| T+6 | {Operand[19:16], Operand[23:20]} |
| T+7 | {Operand[19:16], Operand[23:20]} |
| T+8 | {Operand[27:24], Operand[31:28]} |
| T+9 | {Operand[27:24], Operand[31:28]} |
| T+10 | (PST for next instruction) |
| T+11 | (PST for next instruction,...) |

A PST marker and its data display are transmitted contiguously. Except for this transmission, the IDLE status (0x0) may appear any time. Again, given the real-time trace information appears as a sequence of 4-bit values, there are no alignment restrictions. That is, PST values and operands may appear on either nibble of PSTDDATA.

## 5.3 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 8-bit port is partitioned into two consecutive 4-bit nibbles. Each nibble can either transmit information concerning the processor's execution status (PST) or debug data (DDATA). The processor status may not be related to the current bus transfer.

External development systems can use PSTDDATA outputs with an external image of the program to completely track the dynamic execution path. This tracking is complicated by any change in flow, especially when branch target address calculation is based on the contents of a program-visible register (variant addressing). PSTDDATA 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 5.3.1, "Begin Execution of Taken Branch (PST = 0x5)." Four 32-bit storage elements form a FIFO buffer connecting the processor's high-speed local bus to the external development system through PSTDDATA[7:0]. The buffer captures branch target addresses and certain data values for eventual display on the PSTDDATA port, two nibbles at a time starting with the lsb.

Execution speed is affected only when three storage elements have valid data to be dumped to the PSTDDATA port. This occurs only when two values are captured simultaneously in a read-modify-write operation; the core stalls until two FIFO entries are available.

Table 5-4 shows the encoding of these signals.

**Table 5-4. 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 clock cycles, subsequent clock cycles are indicated by driving PSTDDATA outputs with this encoding. |
| 0x1 | 0001 | Begin execution of one instruction. For most instructions, this encoding signals the first clock cycle of an instruction's execution. Certain change-of-flow opcodes, plus the PULSE and WDDATA instructions, generate different encodings. |
| 0x2 | 0010 | Begin execution of two instructions. For superscalar instruction dispatches, this encoding signals the first clock cycle of the simultaneous instructions' execution. |
| 0x3 | 0011 | Entry into user-mode. Signaled after execution of the instruction that caused the MCF5407 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 PSTDDATA port, independent of debug module configuration. When WDDATA is executed, a value of 0x4 is signaled, followed by the appropriate marker, and then the data transfer on the PSTDDATA port. Transfer length depends on the WDDATA operand size. |
| 0x5 | 0101 | Begin execution of taken branch. For some opcodes, a branch target address may be displayed on PSTDDATA depending on the CSR settings. CSR also controls the number of address bytes displayed, indicated by the PST marker value preceding the DDATA nibble that begins the data output. See Section 5.3.1, "Begin Execution of Taken Branch (PST = 0x5)." |
| 0x6 | 0110 | Begin execution of instruction plus a taken branch. The processor completes execution of a taken conditional branch instruction and simultaneously starts executing the target instruction. This is achieved through branch folding. |
| 0x7 | 0111 | Begin execution of return from exception (RTE) instruction. |
| 0x8–0xB | 1000–1011 | Indicates the size of the next consecutive nibbles. The encoding is driven onto the PSTDDATA port one clock cycle before the data is displayed on PSTDDATA.<br>0x8 Begin 1-byte transfer on PSTDDATA.<br>0x9 Begin 2-byte transfer on PSTDDATA.<br>0xA Begin 3-byte transfer on PSTDDATA.<br>0xB Begin 4-byte transfer on PSTDDATA. |
| 0xC | 1100 | Exception processing. Exceptions that enter emulation mode (debug interrupt or optionally trace) generate a different encoding, 0xD. Because the 0xC encoding defines a multiple-cycle mode, PSTDDATA outputs are driven with 0xC until exception processing completes. |
| 0xD | 1101 | Entry into emulator mode. Displayed during emulation mode (debug interrupt or optionally trace). Because this encoding defines a multiple-cycle mode, PSTDDATA outputs are driven with 0xD until exception processing completes. |
| 0xE | 1110 | A breakpoint state change causes this encoding to assert for one cycle only followed by the trigger status value. If the processor stops waiting for an interrupt, the encoding is asserted for multiple cycles. See Section 5.3.2, "Processor Stopped or Breakpoint State Change (PST = 0xE)." |
| 0xF | 1111 | Processor is halted. See Section 5.3.3, "Processor Halted (PST = 0xF)." |

### 5.3.1 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 PSTDDATA 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 MCF5407 uses the debug pins to output the following sequence of information on successive processor clock cycles:

1. Use PSTDDATA (0x5) to identify that a taken branch was executed.
2. Optionally signal the target address to be displayed sequentially on the PSTDDATA pins. Encodings 0x9–0xB identify the number of bytes displayed.
3. The new target address is optionally available on subsequent cycles using the PSTDDATA 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 5-4 shows when the PSTDDATA outputs that indicate when a JMP (A0) executed, assuming the CSR was programmed to display the lower 2 bytes of an address.

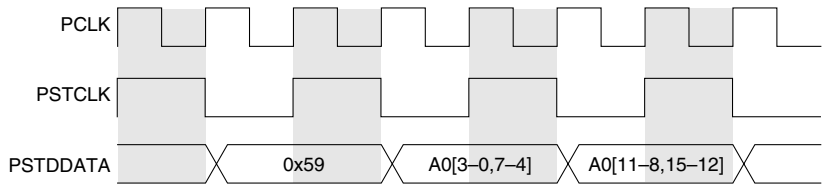



**Figure 5-4. Example JMP Instruction Output on PSTDDATA**

PSTDDATA is driven two nibbles at a time with a 0x59; 0x5 indicates a taken branch and the marker value 0x9 indicates a 2-byte address. Thus, the remaining 4 nibbles display the lower 2 bytes of address register A0 in least-to-most-significant nibble order. The PSTDDATA output after the JMP instruction continues with the next instruction.

### 5.3.2 Processor Stopped or Breakpoint State Change (PST = 0xE)

The 0xE encoding is generated either as a one- or multiple-cycle issue as follows:

- When the MCF5407 is stopped by a STOP instruction, this encoding appears in multiple-cycle format. The ColdFire processor remains stopped until an interrupt occurs, thus PSTDDATA outputs display 0xE until stopped mode is exited.
- When a breakpoint status change is to be output on PSTDDATA, 0xE is displayed for one cycle, followed immediately with the 4-bit value of the current trigger status, where the trigger status is left justified rather than in the CSR[BSTAT] description. Section 5.4.4, “Configuration/Status Register (CSR),” shows that status is right justified. That is, the displayed trigger status on PSTDDATA after a single 0xE is as follows:
  - 0x0 = no breakpoints enabled
  - 0x2 = waiting for level-1 breakpoint
  - 0x4 = level-1 breakpoint triggered
  - 0xA = waiting for level-2 breakpoint
  - 0xC = level-2 breakpoint triggered

Thus, 0xE can indicate multiple events, based on the next value, as Table 5-5 shows.

**Table 5-5. 0xE Status Posting**

| PSTDDATA Stream Includes | Result |
|---|---|
| {0xE, 0x2} | Breakpoint state changed to waiting for level-1 trigger |
| {0xE, 0x4} | Breakpoint state changed to level-1 breakpoint triggered |
| {0xE, 0xA} | Breakpoint state changed to waiting for level-2 trigger |
| {0xE, 0xC} | Breakpoint state changed to level-2 breakpoint triggered |
| {0xE, 0xE} | The MCF5407 is in stopped mode. |

### 5.3.3 Processor Halted (PST = 0xF)

PST is 0xF when the processor is halted (see Section 5.5.1, “CPU Halt”). Because this encoding defines a multiple-cycle mode, the PSTDDATA outputs display 0xF until the processor is restarted or reset.

HALT can be distinguished from a data output 0xFF by counting 0xFF occurrences on PSTDDATA. Because data always follows a marker (0x8, 0x9, 0xA, or 0xB), the longest occurrence in PSTDDATA of 0xFF, in a data output, is four 0xFFs.

Two scenarios exist for data—0xFFFF_FFFF

- A B marker occurs on the left nibble of PSTDDATA with the data of 0xFF following:

  PSTDDATA[7:0]
  0xBF
  0xFF
  0xFF
  0xFF
  0xFX (X indicates that the next PST value is guaranteed to not be 0xF.)

- A B marker occurs on the right nibble of PSTDDATA with the data of 0xFF following:

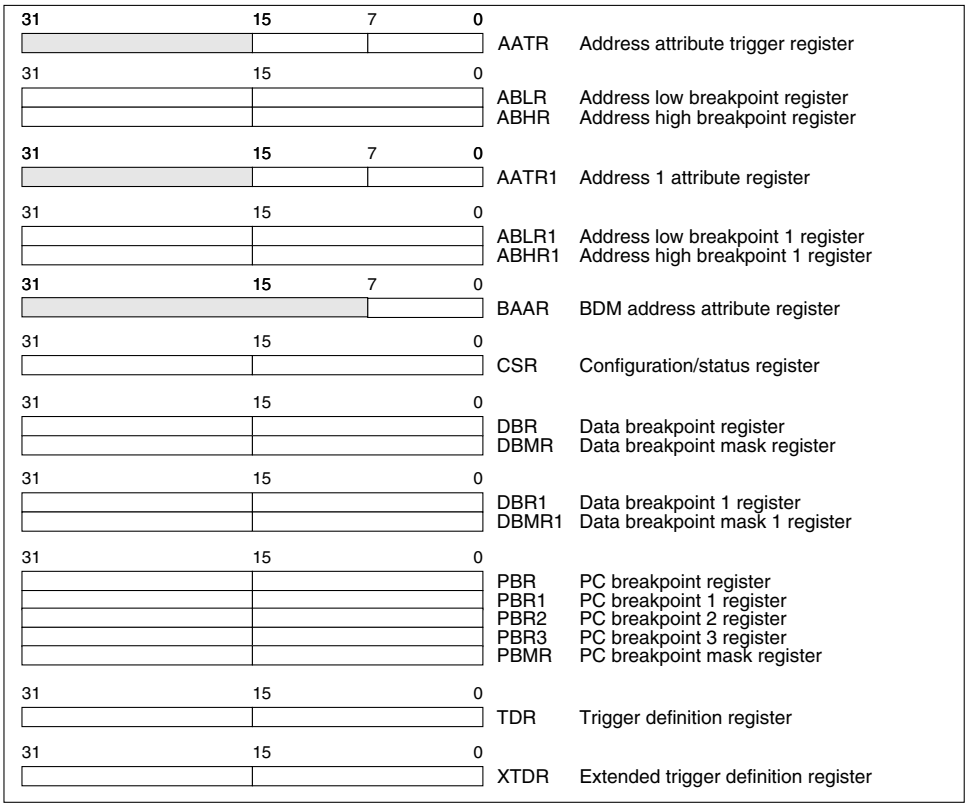
  PSTDDATA[7:0]
  0xYB
  0xFF
  0xFF
  0xFF
  0xFF
  0xXY (X indicates the PST value is guaranteed not to be 0xF, and Y signifies a PSTDDATA value that doesn't affect the 0xFF count.)

Thus, a count of either nine or more sequential single 0xF values or five or more sequential 0xFF values signifies the HALT condition.

## 5.4 Programming Model

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. Thus, the breakpoint hardware in the debug module can be accessed 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 MCF5407 is using the WDEBUG instruction to access debug module registers or the resulting behavior is undefined.

These registers, shown in Figure 5-5, are treated as 32-bit quantities, regardless of the number of implemented bits.




Note: Each debug register is accessed as a 32-bit register; shaded fields above are not used (don't care). All debug control registers are writable from the external development system or the CPU via the WDEBUG instruction.
CSR is write-only from the programming model as debug control register 0x00 using the supervisor-mode WDEBUG instruction. It can be read from and written through the BDM port using the RDMREG and WDMREG commands.

**Figure 5-5. Debug Programming Model**

These registers are accessed through the BDM port by new BDM commands, WDMREG and RDMREG, described in Section 5.5.3.3, "Command Set Descriptions." These commands contain a 5-bit field, DRc, that specifies the register, as shown in Table 5-6.

**Table 5-6. BDM/Breakpoint Registers**

| DRc[4–0] | Register Name | Abbreviation | Initial State | Page |
|---|---|---|---|---|
| 0x00 | Configuration/status register | CSR | 0x0020_0000 | p. 5-13 |
| 0x01–0x04 | Reserved | — | — | — |
| 0x05 | BDM address attribute register | BAAR | 0x0000_0005 | p. 5-12 |
| 0x06 | Address attribute trigger register | AATR | 0x0000_0005 | p. 5-10 |

**Table 5-6. BDM/Breakpoint Registers (Continued)**

| DRc[4–0] | Register Name | Abbreviation | Initial State | Page |
|---|---|---|---|---|
| 0x07 | Trigger definition register | TDR | 0x0000_0000 | p. 5-18 |
| 0x08 | Program counter breakpoint register | PBR | — | p. 5-16 |
| 0x09 | Program counter breakpoint mask register | PBMR | — | p. 5-16 |
| 0x0A–0x0B | Reserved | — | — | — |
| 0x0C | Address breakpoint high register | ABHR | — | p. 5-12 |
| 0x0D | Address breakpoint low register | ABLR | — | p. 5-12 |
| 0x0E | Data breakpoint register | DBR | — | p. 5-15 |
| 0x0F | Data breakpoint mask register | DBMR | — | p. 5-15 |
| 0x10–0x15 | Reserved | — | — | — |
| 0x16 | Address attribute trigger register 1 | AATR1 | 0x0000_0005 | p. 5-10 |
| 0x17 | Extended trigger definition register | XTDR | 0x0000_0000 | p. 5-19 |
| 0x18 | Program counter breakpoint 1 register | PBR1 | 0x0000_0000 | p. 5-16 |
| 0x19 | Reserved | — | — | — |
| 0x1A | Program counter breakpoint register 2 | PBR2 | 0x0000_0000 | p. 5-16 |
| 0x1B | Program counter breakpoint register 3 | PBR3 | 0x0000_0000 | p. 5-16 |
| 0x1C | Address high breakpoint register 1 | ABHR1 | — | p. 5-12 |
| 0x1D | Address low breakpoint register 1 | ABLR1 | — | p. 5-12 |
| 0x1E | Data breakpoint register 1 | DBR1 | — | p. 5-15 |
| 0x1F | Data breakpoint mask register 1 | DBMR1 | — | p. 5-15 |

**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 as debug control register 0x00 using the supervisor-mode WDEBUG instruction. It can be read from and written through the BDM port using the RDMREG and WDMREG commands.

## 5.4.1 Address Attribute Trigger Registers (AATR, AATR1)

The address attribute trigger registers (AATR, AATR1) define address attributes and a mask to be matched in the trigger. The register value is compared with address attribute signals from the processor's local high-speed bus, as defined by the setting of the trigger definition register (TDR) for AATR and the extended trigger definition register (XTDR) for AATR1.

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | RM | SZM | | TTM | | TMM | | | R | SZ | | TT | | TM | | |
| Reset | 0000_0000_0000_0101 | | | | | | | | | | | | | | | |
| R/W | AATR and AATR1 are accessible in supervisor mode as debug control register 0x06 and 0x16 respectively, and using the WDEBUG instruction and through the BDM port using the WDMREG command. | | | | | | | | | | | | | | | |
| DRc[4–0] | 0x06 (AATR); 0x16 (AATR1) | | | | | | | | | | | | | | | |

**Figure 5-6. Address Attribute Trigger Registers (AATR, AATR1)**

Table 5-7 describes AATR and AATR1 fields.

**Table 5-7. AATR and AATR1 Field Descriptions**

| Bits | Name | 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/$\overline{W}$ signal of the processor's local bus. |
| 6–5 | SZ | Size. Compared to the processor's local bus size signals.<br>00 Longword<br>01 Byte<br>10 Word<br>11 Reserved |
| 4–3 | TT | Transfer type. Compared with the local bus transfer type signals.<br>00 Normal processor access<br>01 Reserved<br>10 Emulator mode access<br>11 Acknowledge/CPU space access<br>These bits also define the TT encoding for BDM memory commands. In this case, the 01 encoding indicates an external or DMA access (for backward compatibility). These bits affect the TM bits. |
| 2–0 | TM | Transfer modifier. Compared with the local bus transfer modifier signals, which give supplemental information for each transfer type.<br>TT = 00 (normal mode):<br>000 Data and instruction cache line push<br>001 User data access<br>010 User code access<br>011 Instruction cache invalidate<br>100 Data cache push<br>101 Supervisor data access<br>110 Supervisor code access<br>111 INTOUCH instruction access<br>TT = 10 (emulator mode):<br>0xx–100 Reserved<br>101 Emulator mode data access<br>110 Emulator mode code access<br>111 Reserved<br>TT = 11 (acknowledge/CPU space transfers):<br>000 CPU space access<br>001–111 Interrupt acknowledge levels 1–7<br>These bits also define the TM encoding for BDM memory commands (for backward compatibility). |

## 5.4.2 Address Breakpoint Registers (ABLR/ABLR1, ABHR/ABHR1)

The address breakpoint low and high registers (ABLR, ABLR1, ABHR, and ABHR1), Figure 5-7, 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. TDR determines if the trigger is in the address in ABLR or either inside or outside of the range bound by ABLR and ABHR. XTDR determines the same for ABLR1 and ABHR1.

| | 31 | 0 |
|---|---|---|
| Field | Address | |
| Reset | — | |
| R/W | ABHR and ABHR1 are accessible in supervisor mode as debug control registers 0x0C and 0x1C, using the WDEBUG instruction and via the BDM port using the RDMREG and WDMREG commands.<br>ABLR and ABLR1 are accessible in supervisor mode as debug control register 0x0D and 0x1D, using the WDEBUG instruction and via the BDM port using the WDMREG command. | |
| DRc[4–0] | 0x0D (ABLR); 0x1D (ABLR1); 0x0C (ABHR); 0x1C (ABHR1) | |

**Figure 5-7. Address Breakpoint Registers (ABLR, ABHR, ABLR1, ABHR1)**

Table 5-8 describes ABLR and ABLR1 fields.

**Table 5-8. ABLR and ABLR1 Field Description**

| Bits | Name | 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 or ABLR1. |

Table 5-9 describes ABHR and ABHR1 fields.

**Table 5-9. ABHR and ABHR1 Field Description**

| Bits | Name | Description |
|---|---|---|
| 31–0 | Address | High address. Holds the 32-bit address marking the upper bound of the address breakpoint range. |

## 5.4.3 BDM Address Attribute Register (BAAR)

The BAAR defines the address space for memory-referencing BDM commands. See Figure 5-8. The reset value of 0x5 sets supervisor data as the default address space.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field | R | SZ | | TT | | TM | | |
| Reset | 0000_0101 | | | | | | | |
| R/W | BAAR[R,SZ] are loaded directly from the BDM command; BAAR[TT,TM] can be programmed as debug control register 0x05 from the external development system. For compatibility with Rev. A, BAAR is loaded each time AATR is written. | | | | | | | |
| DRc[4–0] | 0x05 | | | | | | | |

**Figure 5-8. BDM Address Attribute Register (BAAR)**

Table 5-10 describes BAAR fields.

**Table 5-10. BAAR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7 | R | Read/write<br>0 Write<br>1 Read |
| 6–5 | SZ | Size<br>00 Longword<br>01 Byte<br>10 Word<br>11 Reserved |
| 4–3 | TT | Transfer type. See the TT definition in Table 5-7. |
| 2–0 | TM | Transfer modifier. See the TM definition in Table 5-7. |

## 5.4.4 Configuration/Status Register (CSR)

The configuration/status register (CSR) defines the debug configuration for the processor and memory subsystem and contains status information from the breakpoint logic.

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | BSTAT | | | | FOF | TRG | HALT | BKPT | HRL | | | | — | BKD | PCD | IPW |
| Reset | 0000 | | | | 0 | 0 | 0 | 0 | 0010 | | | | — | — | — | 0 |
| R/W[1] | R | | | | R | R | R | R | R | | | | — | — | — | R/W |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | MAP | TRC | EMU | DDC | | UHE | BTB | | —[2] | NPL | — | SSM | — | | | |
| Reset | 0 | 0 | 0 | 00 | | 0 | 00 | | 0 | 0 | — | 0 | — | | | |
| R/W | R/W | R/W | R/W | R/W | | R/W | R/W | | R | R/W | — | R/W | — | | | |
| DRc[4–0] | 0x00 | | | | | | | | | | | | | | | |

1 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.

2 Bit 7 is reserved for Motorola use and must be written as a zero.

**Figure 5-9. Configuration/Status Register (CSR)**

Table 5-11 describes CSR fields.

**Table 5-11. CSR Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 31–28 | BSTAT | Breakpoint status. Provides read-only status information concerning hardware breakpoints. Also output on PSTDDATA when it is not displaying PST or other processor data. BSTAT is cleared by a TDR or XTDR 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.<br>0000 No breakpoints enabled<br>0001 Waiting for level-1 breakpoint<br>0010 Level-1 breakpoint triggered<br>0101 Waiting for level-2 breakpoint<br>0110 Level-2 breakpoint triggered |
| 27 | FOF | Fault-on-fault. If FOF is set, a catastrophic halt occurred and forced entry into BDM. |
| 26 | TRG | Hardware breakpoint trigger. If TRG is set, a hardware breakpoint halted the processor core and forced entry into BDM. Reset and the debug GO command clear TRG. |
| 25 | HALT | Processor halt. If HALT is set, the processor executed a HALT and forced entry into BDM. Reset and the debug GO command reset HALT. |
| 24 | BKPT | Breakpoint assert. If BKPT is set, $\overline{\text{BKPT}}$ was asserted, forcing the processor into BDM. Reset and the debug GO command clears this bit. |
| 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.<br>0000 Initial debug functionality (Revision A)<br>0001 Revision B<br>0010 Revision C (this is the only valid value for the MCF5407) |
| 19 | — | Reserved, should be cleared. |
| 18 | BKD | Breakpoint disable. Used to disable the normal $\overline{\text{BKPT}}$ input functionality and to allow the assertion of $\overline{\text{BKPT}}$ to generate a debug interrupt.<br>0 Normal operation<br>1 $\overline{\text{BKPT}}$ is edge-sensitive: a high-to-low edge on $\overline{\text{BKPT}}$ signals a debug interrupt to the processor. The processor makes this interrupt request pending until the next sample point, 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. Setting PCD disables generation of PSTCLK and PSTDDATA outputs and forces them to remain quiescent. |
| 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.<br>0 All emulator-mode references are mapped into supervisor code and data spaces.<br>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. |
| 13 | EMU | Force emulation mode. If EMU = 1, the processor begins executing in emulator mode. See Section 5.6.1.1, "Emulator Mode." |

**Table 5-11. CSR Field Descriptions (Continued)**

| Bit | Name | Description |
|---|---|---|
| 12–11 | DDC | Debug data control. Controls operand data capture for PSTDDATA, which displays the number of bytes defined by the operand reference size before the actual data; byte displays 8 bits, word displays 16 bits, and long displays 32 bits (one nibble at a time across multiple clock cycles). See Table 5-4.<br>00 No operand data is displayed.<br>01 Capture all write data.<br>10 Capture all read data.<br>11 Capture all read and write data. |
| 10 | UHE | User halt enable. Selects the CPU privilege level required to execute the HALT instruction.<br>0 HALT is a supervisor-only instruction.<br>1 HALT is a supervisor/user instruction. |
| 9–8 | BTB | Branch target bytes. Defines the number of bytes of branch target address PSTDDATA displays.<br>00 0 bytes<br>01 Lower 2 bytes of the target address<br>10 Lower 3 bytes of the target address<br>11 Entire 4-byte target address<br>See Section 5.3.1, “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.<br>0 Pipelined mode<br>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. Instruction folding is disabled. Given an average execution latency of 1.6, throughput in non-pipeline mode would be 6.6, approximately 25% or less compared to pipelined performance.<br>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.<br>An address or data breakpoint should always occur before the next instruction begins execution. Therefore the occurrence of the address/data breakpoints should be guaranteed. |
| 5 | — | Reserved, should be cleared. |
| 4 | SSM | Single-step mode. Setting SSM puts the processor in single-step mode.<br>0 Normal mode.<br>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. |

## 5.4.5 Data Breakpoint/Mask Registers (DBR/DBR1, DBMR/DBMR1)

The data breakpoint registers (DBR/DBR1), Figure 5-10, specify data patterns used as part of the trigger into debug mode. Only DBR*n* bits not masked with a corresponding zero in DBMR*n* are compared with the data from the processor’s local bus, as defined in TDR.

| | 31 … 0 |
|---|---|
| Field | Data (DBR/DBR1); Mask (DBMR/DBMR1) |
| Reset | Uninitialized |
| R/W | DBR and DBR1 are accessible in supervisor mode as debug control register 0x0E and 0x1E, using the WDEBUG instruction and through the BDM port using the RDMREG and WDMREG commands.<br>DBMR and DBMR1 are accessible in supervisor mode as debug control register 0x0F and 0x0F1 using the WDEBUG instruction and via the BDM port using the WDMREG command. |
| DRc[4–0] | 0x0E (DBR), 0x1E (DBR1); 0x0F (DBMR), 0x1F (DBMR1) |

**Figure 5-10. Data Breakpoint/Mask Registers (DBR/DBR1 and DBMR/DBMR1)**

Table 5-12 describes DBR*n* fields.

**Table 5-12. DBRn Field Descriptions**

| Bits | Name | 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 5-13 describes DBMR*n* fields.

**Table 5-13. DBMRn Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–0 | Mask | Data breakpoint mask. The 32-bit mask for the data breakpoint trigger. Clearing a DBRn bit allows the corresponding DBRn bit to be compared to the appropriate bit of the processor's local data bus. Setting a DBMRn bit causes that bit to be ignored. |

DBRs support both aligned and misaligned references. Table 5-14 shows relationships between processor address, access size, and location within the 32-bit data bus.

**Table 5-14. 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] |

## 5.4.6 Program Counter Breakpoint/Mask Registers (PBR, PBR1, PBR2, PBR3, PBMR)

Each PC breakpoint register (PBR, PBR1, PBR2, PBR3) defines an instruction address for use as part of the trigger. These registers' contents are compared with the processor's

program counter register when the appropriate valid bit is set and TDR and/or XTDR are configured appropriately. PBR bits are masked by clearing corresponding PBMR bits. Results are compared with the processor's program counter register, as defined in TDR and/or XTDR. PBR1–PBR3 are not masked. Figure 5-11 shows the PC breakpoint register.

| | 31 | 1 | 0 |
|---|---|---|---|
| Field | Program Counter | | V [1] |
| Reset | — | | 0 |
| R/W | Write. PC breakpoint registers are accessible in supervisor mode using the WDEBUG instruction and through the BDM port using the RDMREG and WDMREG commands using values shown in Section 5.5.3.3, "Command Set Descriptions." | | |
| DRc[4–0] | 0x08 (PBR); 0x18 (PBR1); 0x1A (PBR2); 0x1B (PBR3) | | |

[1] PBR does not have a valid bit. PBR[0] is read as 0 and should be cleared.

**Figure 5-11. Program Counter Breakpoint Registers (PBR, PBR1, PBR2, PBR3)**

Table 5-15 describes PBR, PBR1, PBR2, and PBR3 fields.

**Table 5-15. PBR, PBR1, PBR2, PBR3 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–1 | Address | PC breakpoint address. The 31-bit address to be compared with the PC as a breakpoint trigger. PBR does not have a valid bit. |
| 0 | V | Valid. Breakpoint registers are compared with the processor's program counter register when the appropriate valid bit is set and TDR and/or XTDR are configured appropriately. This bit is not implemented on PBR. |

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 5-12 shows PBMR.

| | 31 | 1 | 0 |
|---|---|---|---|
| Field | Mask | | — |
| Reset | — | | — |
| R/W | Write. PBMR is accessible in supervisor mode as debug control register 0x09 using the WDEBUG instruction and via the BDM port using the wdmreg command. | | |
| DRc[4–0] | 0x09 | | |

**Figure 5-12. Program Counter Breakpoint Mask Register (PBMR)**

Table 5-16 describes PBMR fields.

**Table 5-16. PBMR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–0 | Mask | PC breakpoint mask. A zero in a bit position causes the corresponding PBR bit to be compared to the appropriate PC bit. Set PBMR bits cause PBR bits to be ignored. |

## 5.4.7 Trigger Definition Register (TDR)

The TDR, shown in Table 5-13, configures the operation of the hardware breakpoint logic that corresponds with the ABHR/ABLR/AATR, PBR/PBR1/PBR2/PBR3/PBMR, and DBR/DBMR registers within the debug module. In conjunction with the XTDR and its associated debug registers, TDR controls the actions taken under the defined conditions. Breakpoint logic may be configured as one- or two-level triggers. TDR[31–16] and/or XTDR[31–16] define second-level triggers and bits 15–0 define first-level triggers.

**NOTE:**

The debug module has no hardware interlocks, so to prevent spurious breakpoint triggers while the breakpoint registers are being loaded, disable TDR and XTDR (by clearing TDR[29,13] and XTDR[29,13]) before defining triggers.

A write to TDR clears the CSR trigger status bits, CSR[BSTAT].

Section 5.4.9, “Resulting Set of Possible Trigger Combinations,” describes how to handle multiple breakpoint conditions.

Second-Level Triggers

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | TRC | | EBL | EDLW | EDWL | EDWU | EDLL | EDLM | EDUM | EDUU | DI | EAI | EAR | EAL | EPC | PCI |
| Reset | 0000_0000_0000_0000 | | | | | | | | | | | | | | | |
| R/W | Accessible in supervisor mode as debug control register 0x07 using the WDEBUG instruction and through the BDM port using the WDMREG command. | | | | | | | | | | | | | | | |

First-Level Triggers

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | — | | EBL | EDLW | EDWL | EDWU | EDLL | EDLM | EDUM | EDUU | DI | EAI | EAR | EAL | EPC | PCI |
| Reset | 0000_0000_0000_0000 | | | | | | | | | | | | | | | |
| R/W | 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 | | | | | | | | | | | | | | | |

**Figure 5-13. Trigger Definition Register (TDR)**

Table 5-17 describes TDR fields.

**Table 5-17. TDR Field Descriptions**

| Bits | Name | Description | |
|---|---|---|---|
| 31–30 | TRC | Trigger response control. Determines how the processor responds to a completed trigger condition. The trigger response is always displayed on PSTDDATA.<br>00 Display on PSTDDATA only<br>01 Processor halt<br>10 Debug interrupt<br>11 Reserved | |
| 29/13 | EBL | Enable breakpoint. Global enable for the breakpoint trigger. Setting TDR[EBL] or XTDR[EBL] enables a breakpoint trigger; clearing both disables all breakpoints. | |
| 28–22<br>12–6 | EDx | 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. | |
| 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/<br>4–2 | EAx | Enable address bits. Setting an EA bit enables the corresponding address breakpoint. Clearing all three bits disables the breakpoint. | |
| 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 | Enable PC breakpoint. If set, this bit enables the PC breakpoint. | |
| 16/0 | PCI | Breakpoint invert. If set, this bit allows execution outside a given region as defined by PBR/PBR1/PBR2/PBR3 and PBMR to enable a trigger. If cleared, the PC breakpoint is defined within the region defined by PBR/PBR1/PBR2/PBR3 and PBMR. | |

## 5.4.8 Extended Trigger Definition Register (XTDR)

The XTDR configures the operation of the hardware breakpoint logic that corresponds with the ABHR1/ABLR1/AATR1 and DBR1/DBMR1 registers within the debug module and, in conjunction with the TDR and its associated debug registers, controls the actions taken under the defined conditions. The breakpoint logic may be configured as a one- or two-level trigger, where TDR[31–16] and/or XTDR[31–16] define the second-level trigger and bits 15–0 define the first-level trigger. The XTDR is accessible in supervisor mode as debug control register 0x17 using the WDEBUG instruction and via the BDM port using the WDMREG command.

**NOTE:**

The debug module has no hardware interlocks, so to prevent spurious breakpoint triggers while the breakpoint registers are being loaded, disable TDR and XTDR (by clearing TDR[29,13] and XTDR[29,13]) before defining triggers.

A write to the XTDR clears the trigger status bits, CSR[BSTAT].

Section 5.4.9, “Resulting Set of Possible Trigger Combinations,” describes how to handle multiple breakpoint conditions.

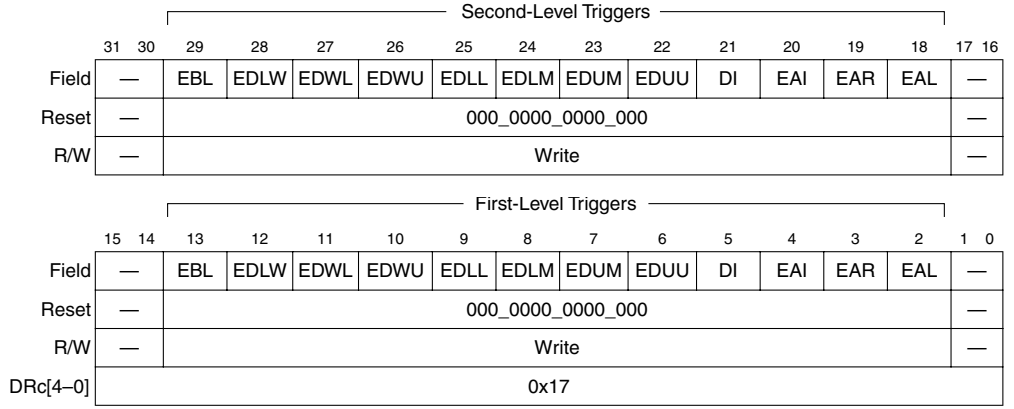



**Figure 5-14. Extended Trigger Definition Register (XTDR)**

Table 5-18 describes XTDR fields.

**Table 5-18. XTDR Field Descriptions**

| Bits | Name | Description | |
|---|---|---|---|
| 29/13 | EBL | Enable breakpoint level. If set, EBL is the global enable for the breakpoint trigger; that is, if TDR[EBL] or XTDR[EBL] is set, a breakpoint trigger is enabled. Clearing both disables all breakpoints. | |
| 28–22<br>12–6 | EDx | 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. | |
| 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. |

**Table 5-18. XTDR Field Descriptions (Continued)**

| Bits | Name | Description | |
|---|---|---|---|
| 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 DBR1 contents. | |
| 20–18/ 4–2 | EAx | Enable address bits. Setting an EAx bit enables the corresponding address breakpoint. If all three bits are cleared, this breakpoint is disabled. | |
| 20/4 | | EAI | Enable address breakpoint inverted. Breakpoint is based outside the range between ABLR1 and ABHR1. |
| 19/3 | | EAR | Enable address breakpoint range. Breakpoint is based on the range defined between ABLR1 and ABHR1. |
| 18/2 | | EAL | Enable address breakpoint low. The breakpoint is based on the address in ABLR1. |
| 17–16, 1–0 | — | Reserved, should be cleared. | |

## 5.4.9 Resulting Set of Possible Trigger Combinations

The resulting set of possible breakpoint trigger combinations consist of the following options where || denotes logical OR, && denotes logical AND, and {} denotes an optional additional trigger term:

```
One-level triggers of the form:

if      (PC_breakpoint)
if      (PC_breakpoint||Address_breakpoint{&& Data_breakpoint})
if      (PC_breakpoint||Address_breakpoint{&& Data_breakpoint}
                ||      Address1_breakpoint{&& Data1_breakpoint})

if      (Address_breakpoint     {&& Data_breakpoint})
if      ((Address_breakpoint    {&& Data_breakpoint})
                ||      (Address1_breakpoint{&& Data1_breakpoint}))

if      (Address1_breakpoint    {&& Data1_breakpoint})


Two-level triggers of the form:
if      (PC_breakpoint)
        then if          (Address_breakpoint{&& Data_breakpoint})

if      (PC_breakpoint)
        then if          (Address_breakpoint{&& Data_breakpoint}
                ||      Address1_breakpoint{&& Data1_breakpoint})

if      (PC_breakpoint)
        then if         (Address1_breakpoint{&& Data1_breakpoint})

if      (Address_breakpoint     {&& Data_breakpoint})
        then if         (Address1_breakpoint{&& Data1_breakpoint})

if      (Address1_breakpoint    {&& Data1_breakpoint})
        then if         (Address_breakpoint{&& Data_breakpoint})

if      (Address_breakpoint     {&& Data_breakpoint})
        then if         (PC_breakpoint)

if      (Address1_breakpoint    {&& Data1_breakpoint})
```

```
        then if         (PC_breakpoint)

if      (Address_breakpoint     {&& Data_breakpoint})
        then if         (PC_breakpoint
                ||      Address1_breakpoint{&& Data1_breakpoint})

if      (Address1_breakpoint    {&& Data1_breakpoint})
        then if         (PC_breakpoint
                ||      Address_breakpoint{&& Data_breakpoint})
```

In this example, PC_breakpoint is the logical summation of the PBR/PBMR, PBR1, PBR2, and PBR3 breakpoint registers; Address_breakpoint is a function of ABHR, ABLR, and AATR; Data_breakpoint is a function of DBR and DBMR; Address1_breakpoint is a function of ABHR1, ABLR1, and AATR1; and Data1_breakpoint is a function of DBR1 and DBMR1. In all cases, the data breakpoints can be included with an address breakpoint to further qualify a trigger event as an option.

## 5.5 Background Debug Mode (BDM)

The ColdFire Family implements a low-level system debugger in the microprocessor hardware. Communication with the development system is handled through a dedicated, high-speed serial command interface. The ColdFire architecture implements the BDM controller in a dedicated hardware module. 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.

### 5.5.1 CPU Halt

Although many BDM operations can occur in parallel with CPU operations, unrestricted BDM operation requires the CPU to be halted. The sources that can cause the CPU to halt are listed below in order of priority:

1. A catastrophic fault-on-fault condition automatically halts the processor.
2. A hardware breakpoint can be configured to generate a pending halt condition similar to the assertion of $\overline{\text{BKPT}}$. 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 5.6.1, “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 $\overline{\text{BKPT}}$ 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 $\overline{\text{BKPT}}$ should be considered in the following two special cases:

- After the system reset signal is negated, the processor waits for 16 clock cycles before beginning reset exception processing. If the $\overline{\text{BKPT}}$ input is asserted within eight cycles after $\overline{\text{RSTI}}$ is negated, the processor enters the halt state, signaling halt status (0xF) on the PSTDDATA outputs. While the processor is in this state, all resources accessible through the debug module can be referenced. This is the only chance to force the processor into emulation mode through CSR[EMU].

  After system initialization, the processor's response to the GO command depends on the set of BDM commands performed while it is halted for a breakpoint. Specifically, if the PC register was loaded, the GO command causes the processor to exit halted state and pass control to the instruction address in the PC, bypassing normal reset exception processing. If the PC was not loaded, the GO command causes the processor to exit halted state and continue reset exception processing.
- The ColdFire architecture also handles a special case of $\overline{\text{BKPT}}$ 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.

CSR[27–24] indicates the halt source, showing the highest priority source for multiple halt conditions. Debug module Revisions A and B clear CSR[27–24] upon a read of the CSR, but Revision C (in the MCF5407) does not. The debug GO command clears CSR[26–24].

HALT can be recognized by counting 0xFF occurrences on PSTDDATA. The count is necessary to determine between a possible data output value of 0xFF and the HALT condition. Because data always follows a marker (0x8, 0x9, 0xA, or 0xB), PSTDDATA can display no more than four data 0xFFs. Two such scenarios exist:

- A B marker occurs on the left nibble of PSTDDATA with the data of 0xFF following:

  PSTDDATA[7:0]

  0xBF
  0xFF
  0xFF
  0xFF
  0xFX (X indicates that the next PST value is guaranteed to not be 0xF)

- A B marker occurs on the right nibble of PSTDDATA with the data of 0xFF following:

  PSTDDATA[7:0]

  0xYB
  0xFF
  0xFF
  0xFF
  0xFF
  0xXY (X indicates that the PST value is guaranteed to not be 0xF; and Y indicates a PSTDDATA value that doesn't affect the 0xFF count).

Thus, a count of either nine or more sequential single 0xF values or five or more sequential 0xFF values signifies the HALT condition.

## 5.5.2 BDM Serial Interface

When the CPU is halted and PSTDDATA reflects the halt status, the development system can send unrestricted commands to the debug module. The debug module implements a synchronous protocol using two inputs (DSCLK and DSI) and one output (DSO), where DSCLK and DSI must meet the required input setup and hold timings and the DSO is specified as a delay relative to the rising edge of the processor clock. See Table 5-1. 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 processor 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 5-15, all state transitions are enabled on a rising edge of the processor clock when DSCLK is high; that is, DSI is sampled and DSO is driven.

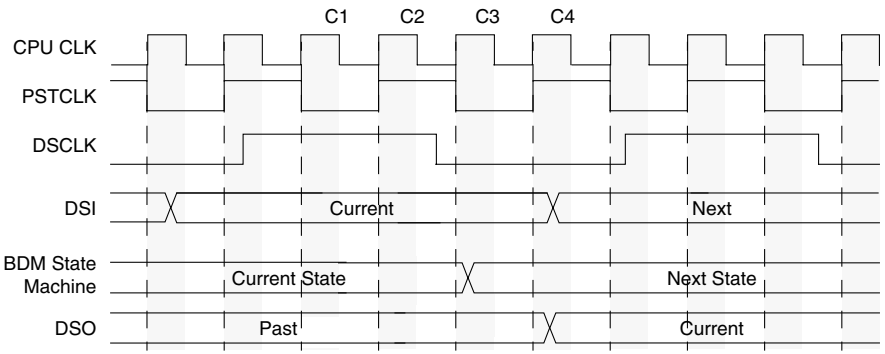



**Figure 5-15. BDM Serial Interface Timing**

DSCLK and DSI are synchronized inputs. DSCLK acts as a pseudo clock enable and is sampled on the rising edge of the processor CLK 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, DSDO 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 DSI (DSCLK is high).
- C2—Second synchronization cycle for DSI (DSCLK is high).
- C3—BDM state machine changes state depending upon DSI 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.

### 5.5.2.1 Receive Packet Format

The basic receive packet, Figure 5-16, consists of 16 data bits and 1 status bit.

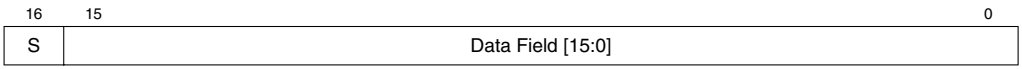



**Figure 5-16. Receive BDM Packet**

Table 5-19 describes receive BDM packet fields.

**Table 5-19. Receive BDM Packet Field Description**

| Bits | Name | 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.<br>S Data Message<br>0 xxxx Valid data transfer<br>0 0xFFFF Status OK<br>1 0x0000 Not ready with response; come again<br>1 0x0001 Error—Terminated bus cycle; data invalid<br>1 0xFFFF 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. |

### 5.5.2.2 Transmit Packet Format

The basic transmit packet, Figure 5-17, consists of 16 data bits and 1 control bit.

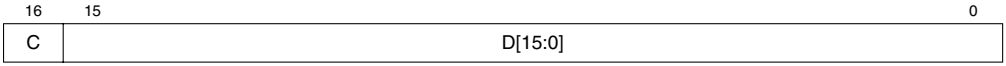



**Figure 5-17. Transmit BDM Packet**

Table 5-20 describes transmit BDM packet fields.

**Table 5-20. Transmit BDM Packet Field Description**

| Bits | Name | Description |
|---|---|---|
| 16 | C | Control. This bit is reserved. Command and data transfers initiated by the development system should clear C. |
| 15–0 | Data | Contains the data to be sent from the development system to the debug module. |

## 5.5.3 BDM Command Set

Table 5-21 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 5-21. 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 | 5.5.3.3.1 | 0x218 {A/D, Reg[2:0]} |
| Write A/D register | WAREG/ WDREG | Write the data operand to the specified address or data register. | Halted | 5.5.3.3.2 | 0x208 {A/D, Reg[2:0]} |
| Read memory location | READ | Read the data at the memory location specified by the longword address. | Steal | 5.5.3.3.3 | 0x1900—byte<br>0x1940—word<br>0x1980—lword |
| Write memory location | WRITE | Write the operand data to the memory location specified by the longword address. | Steal | 5.5.3.3.4 | 0x1800—byte<br>0x1840—word<br>0x1880—lword |
| 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 | 5.5.3.3.5 | 0x1D00—byte<br>0x1D40—word<br>0x1D80—lword |
| 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 | 5.5.3.3.6 | 0x1C00—byte<br>0x1C40—word<br>0x1C80—lword |
| Resume execution | GO | The pipeline is flushed and refilled before resuming instruction execution at the current PC. | Halted | 5.5.3.3.7 | 0x0C00 |
| No operation | NOP | Perform no operation; may be used as a null command. | Parallel | 5.5.3.3.8 | 0x0000 |

**Table 5-21. BDM Command Summary (Continued)**

| Command | Mnemonic | Description | CPU State[1] | Section | Command (Hex) |
|---|---|---|---|---|---|
| Output the current PC | SYNC_PC | Capture the current PC and display it on the PSTDDATA output pins. | Parallel | 5.5.3.3.9 | 0x0001 |
| Read control register | RCREG | Read the system control register. | Halted | 5.5.3.3.10 | 0x2980 |
| Write control register | WCREG | Write the operand data to the system control register. | Halted | 5.5.3.3.11 | 0x2880 |
| Read debug module register | RDMREG | Read the debug module register. | Parallel | 5.5.3.3.12 | 0x2D {0x4[2] DRc[4:0]} |
| Write debug module register | WDMREG | Write the operand data to the debug module register. | Parallel | 5.5.3.3.13 | 0x2C {0x4[2] 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 0x4 is a three-bit field.

Unassigned command opcodes are reserved by Motorola. All unused command formats within any revision level perform a NOP and return the illegal command response.

### 5.5.3.1 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 5-18.

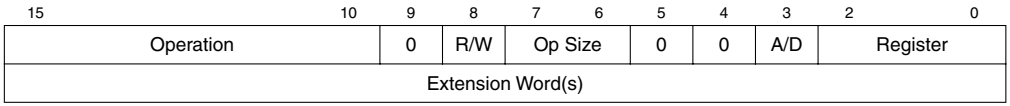



**Figure 5-18. BDM Command Format**

Table 5-22 describes BDM fields.

**Table 5-22. BDM Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 15–10 | Operation | Specifies the command. These values are listed in Table 5-21. |
| 9 | 0 | Reserved |
| 8 | R/W | Direction of operand transfer.<br>0 Data is written to the CPU or to memory from the development system.<br>1 The transfer is from the CPU to the development system. |

**Table 5-22. BDM Field Descriptions (Continued)**

| Bit | Name | Description |
|---|---|---|
| 7–6 | Operand 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.<br>Operand Size Bit Values<br>00 Byte 8 bits<br>01 Word 16 bits<br>10 Longword 32 bits<br>11 Reserved — |
| 5–4 | 00 | Reserved |
| 3 | A/D | Address/data. Determines whether the register field specifies a data or address register.<br>0 Indicates a data register.<br>1 Indicates an address register. |
| 2–0 | Register | Contains the register number in commands that operate on processor registers. |

#### 5.5.3.1.1 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.

### 5.5.3.2 Command Sequence Diagrams

The command sequence diagram in Figure 5-19 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.

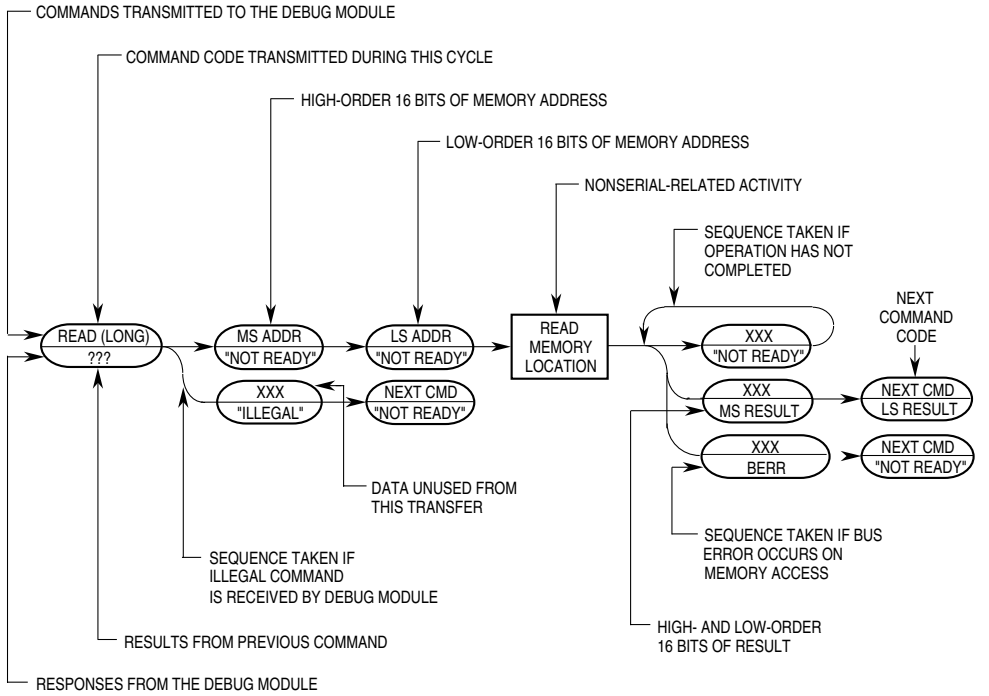



**Figure 5-19. Command Sequence Diagram**

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.

### 5.5.3.3 Command Set Descriptions

The following sections describe the commands summarized in Table 5-21.

**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 5.5.2, "BDM Serial Interface," describes the receive packet format.

Motorola reserves unassigned command opcodes for future expansion. Unused command formats in any revision level perform a NOP and return an illegal command response.

#### 5.5.3.3.1 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:

| | 15–12 | 11–8 | 7–4 | 3 | 2–0 |
|---|---|---|---|---|---|
| Command | 0x2 | 0x1 | 0x8 | A/D | Register |
| Result | D[31:16] | | | | |
| | D[15:0] | | | | |

**Figure 5-20. RAREG/RDREG Command Format**

Command Sequence:

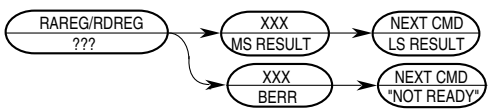



**Figure 5-21. RAREG/RDREG Command Sequence**

Operand Data: None

Result Data: The contents of the selected register are returned as a longword value, most-significant word first.

#### 5.5.3.3.2 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 12 | 11 8 | 7 4 | 3 | 2 0 |
|---|---|---|---|---|---|
| Command | 0x2 | 0x0 | 0x8 | A/D | Register |
| Result | D[31:16] | | | | |
| | D[15:0] | | | | |

**Figure 5-22. WAREG/WDREG Command Format**

Command Sequence

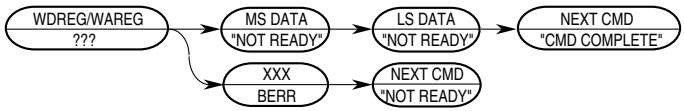



**Figure 5-23. WAREG/WDREG Command Sequence**

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.

### 5.5.3.3.3 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:

| | | 15 12 | 11 8 | 7 4 | 3 0 |
|---|---|---|---|---|---|
| Byte | Command | 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 5-24. READ Command/Result Formats**

Command Sequence:

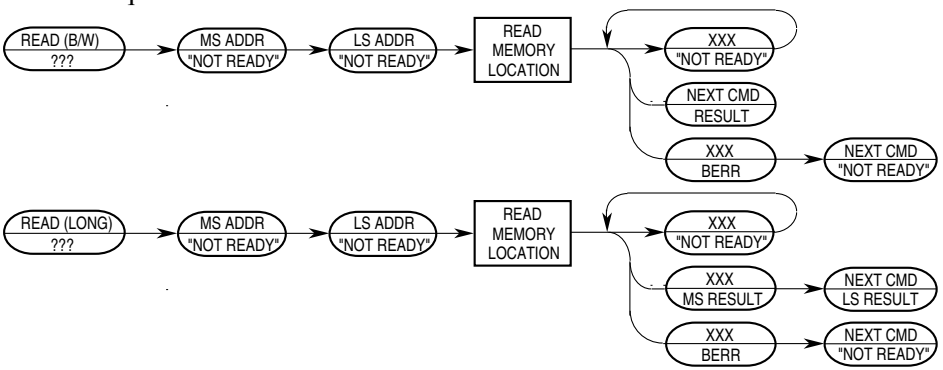



**Figure 5-25. READ Command Sequence**

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.

#### 5.5.3.3.4 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 12 | 11 8 | 7 4 | 3 1 |
|---|---|---|---|---|
| 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 5-26. WRITE Command Format**

Command Sequence:

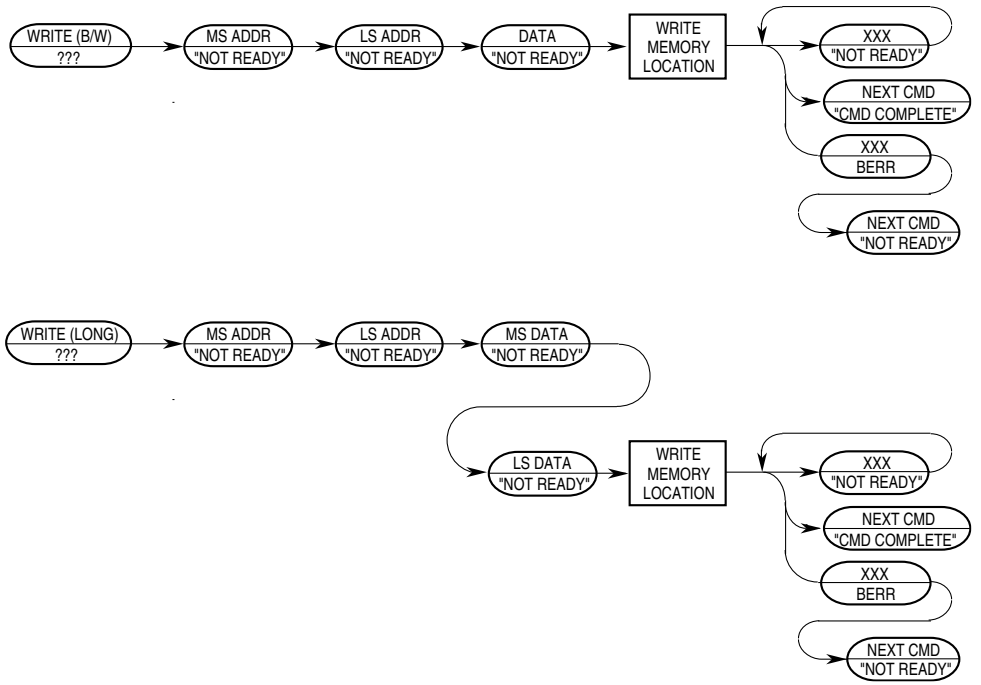



**Figure 5-27. WRITE Command Sequence**

| | |
|---|---|
| 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. |

#### 5.5.3.3.5 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:

| | | 15 12 | 11 8 | 7 4 | 3 0 |
|---|---|---|---|---|---|
| Byte | Command | 0x1 | 0xD | 0x0 | 0x0 |
| | Result | X 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 5-28. DUMP Command/Result Formats**

Command Sequence:

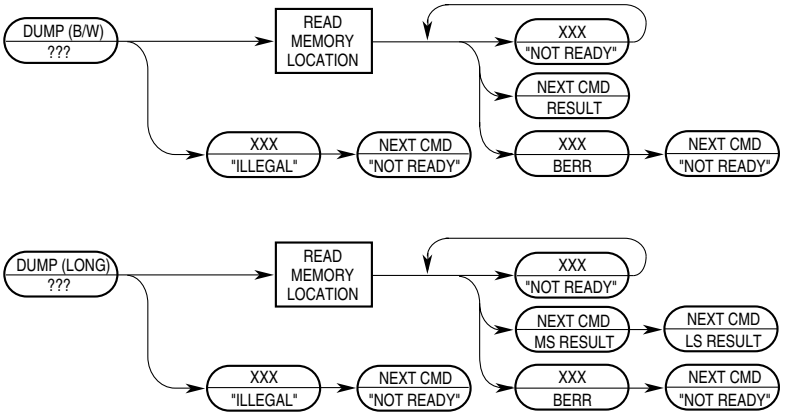



**Figure 5-29. DUMP Command Sequence**

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.

#### 5.5.3.3.6 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 12 | 11 8 | 7 4 | 3 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 5-30. FILL Command Format**

Command Sequence:

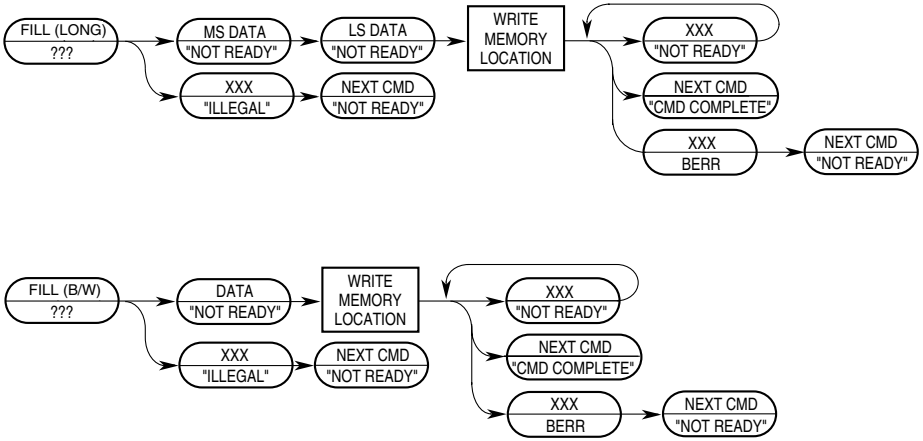



**Figure 5-31. FILL Command Sequence**

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.

#### 5.5.3.3.7 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.

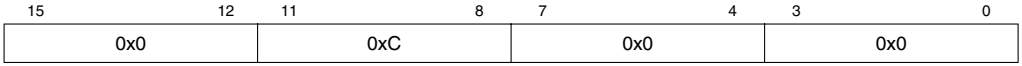



**Figure 5-32. GO Command Format**

Command Sequence:

**Figure 5-33. GO Command Sequence**

Operand Data: None

Result Data: The command-complete response (0xFFFF) is returned during the next shift operation.

### 5.5.3.3.8 No Operation (NOP)

NOP performs no operation and may be used as a null command where required.

Command Formats:

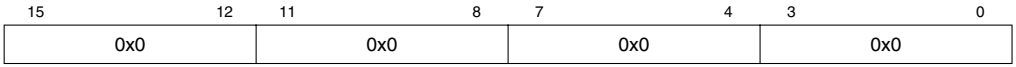



**Figure 5-34. NOP Command Format**

Command Sequence:

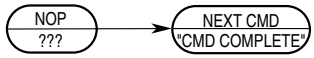



**Figure 5-35. NOP Command Sequence**

Operand Data: None

Result Data: The command-complete response, 0xFFFF (with S cleared), is returned during the next shift operation.

#### 5.5.3.3.9 Synchronize PC to the PSTDDATA Lines (SYNC_PC)

The SYNC_PC command captures the current PC and displays it on the PSTDDATA outputs. After the debug module receives the command, it sends a signal to the ColdFire processor that the current PC must be displayed. The processor then forces an instruction fetch at the next PC with the address being captured in the DDATA logic under control of CSR[BTB]. The specific sequence of PSTDDATA values is as follows:

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 = 0x5 value indicating a taken branch and signals the capture of DDATA.
4. The instruction address corresponding to the PC is captured.
5. The PST marker (0x9–0xB) is generated and displayed as defined by CSR[BTB] followed by the captured PC address.

The SYNC_PC command can be used to dynamically access the PC for performance monitoring. The execution of this command is considerably less obtrusive to the real-time operation of an application than a HALT-CPU/READ-PC/RESUME command sequence.

Command Formats:

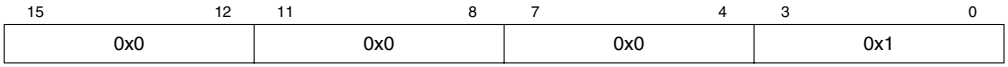



**Figure 5-36. SYNC_PC Command Format**

Command Sequence:

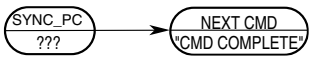



**Figure 5-37. SYNC_PC Command Sequence**

Operand Data: None

Result Data: Command complete status (0xFFFF) is returned when the register write is complete.

### 5.5.3.3.10 Read Control Register (RCREG)

Read the selected control register and return the 32-bit result. Accesses to the processor/memory control registers are always 32 bits wide, regardless of register width. The second and third words of the command form a 32-bit address, which the debug module uses to generate a special bus cycle to access the specified control register. The 12-bit Rc field is the same as that used by the MOVEC instruction.

Command/Result Formats:

| | 15 12 | 11 8 | 7 4 | 3 0 |
|---|---|---|---|---|
| Command | 0x2 | 0x9 | 0x8 | 0x0 |
| | 0x0 | 0x0 | 0x0 | 0x0 |
| | 0x0 | Rc | | |
| Result | D[31:16] | | | |
| | D[15:0] | | | |

**Figure 5-38. RCREG Command/Result Formats**

Rc encoding:

**Table 5-23. Control Register Map**

| Rc | Register Definition |
|---|---|
| 0x002 | Cache control register (CACR) |
| 0x004 | Access control register 0 (ACR0) |
| 0x005 | Access control register 1 (ACR1) |
| 0x006 | Access control register 2 (ACR2) |
| 0x007 | Access control register 2 (ACR3) |
| 0x801 | Vector base register (VBR) |
| 0x804 | MAC status register (MACSR) |

| Rc | Register Definition |
|---|---|
| 0x805 | MAC mask register (MASK) |
| 0x806 | MAC accumulator (ACC) |
| 0x80E | Status register (SR) |
| 0x80F | Program register (PC) |
| 0xC04 | RAM base address register (RAMBAR0) |
| 0xC05 | RAM base address register (RAMBAR1) |
| 0xC0F | Memory base address (MBAR) |

Command Sequence:

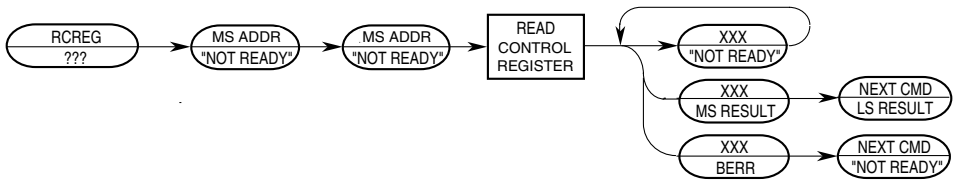



**Figure 5-39. RCREG Command Sequence**

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.

#### 5.5.3.3.11 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:

| | 15–12 | 11–8 | 7–4 | 3–0 |
|---|---|---|---|---|
| Command | 0x2 | 0x8 | 0x8 | 0x0 |
| | 0x0 | 0x0 | 0x0 | 0x0 |
| | 0x0 | Rc | | |
| Result | D[31:16] | | | |
| | D[15:0] | | | |

**Figure 5-40. WCREG Command/Result Formats**

Command Sequence:

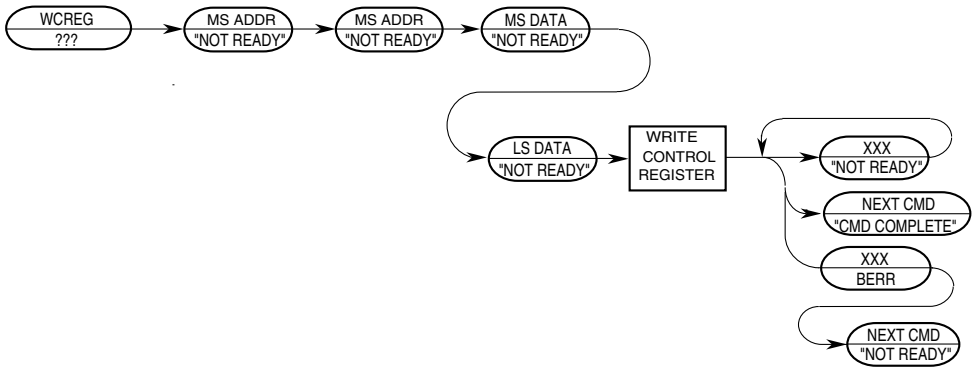



**Figure 5-41. WCREG Command Sequence**

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.

### 5.5.3.3.12 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 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:

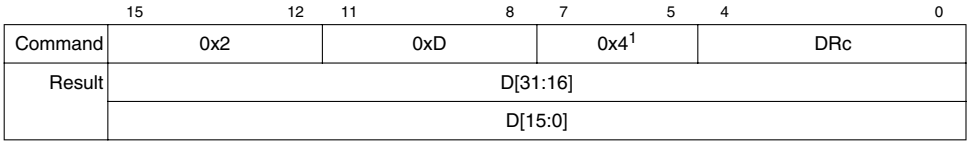



**Figure 5-42. RDMREG BDM Command/Result Formats**

[1] Note 0x4 is a 3-bit field

Table 5-24 shows the definition of DRc encoding.

**Table 5-24. Definition of DRc Encoding—Read**

| DRc[4:0] | Debug Register Definition | Mnemonic | Initial State | Page |
|---|---|---|---|---|
| 0x00 | Configuration/Status | CSR | 0x0 | p. 5-13 |
| 0x01–0x1F | Reserved | — | — | — |

Command Sequence:

**Figure 5-43. 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.

#### 5.5.3.3.13 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:

**Figure 5-44. WDMREG BDM Command Format**

| 15 12 | 11 8 | 7 5 | 4 0 |
|---|---|---|---|
| 0x2 | 0xC | 0x4[1] | DRc |
| D[31:16] | | | |
| D[15:0] | | | |

1 Note: 0x4 is a three-bit field

Table 5-6 shows the definition of the DRc write encoding.

Command Sequence:

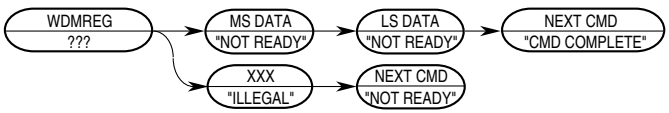



**Figure 5-45. WDMREG Command Sequence**

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.

## 5.6 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 three types of breakpoints—PC with 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.

## 5.6.1 Theory of Operation

Breakpoint hardware can be configured to respond to triggers in several ways. The response desired is programmed into TDR. As shown in Table 5-25, when a breakpoint is triggered, an indication (CSR[BSTAT]) is provided on the PSTDDATA output port when it is not displaying captured processor status, operands, or branch addresses. See Section 5.3.2, “Processor Stopped or Breakpoint State Change (PST = 0xE).”

**Table 5-25. PSTDDATA Nibble/CSR[BSTAT] Breakpoint Response**

| PSTDDATA Nibble/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 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 either TDR or XTDR.

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 this 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 from the vector table. Table 5-26 describes the two unique entries that distinguish PC breakpoints from other trigger events.

**Table 5-26. Exception Vector Assignments**

| Vector Number | Vector Offset (Hex) | Stacked Program Counter | Assignment |
|---|---|---|---|
| 12 | 0x030 | Next | Non-PC-breakpoint debug interrupt |
| 13 | 0x034 | Next | PC-breakpoint debug interrupt |

In the case of a two-level trigger, the last breakpoint event determines the exception vector; however, if the second-level trigger is PC || Address {&& Data} (as shown in the last condition in the code example in Section 5.4.9, “Resulting Set of Possible Trigger Combinations”), the vector taken is determined by the first condition that occurs after the first-level trigger—vector 13 if PC occurs first or vector 12 if Address {&& Data} occurs first. If both occur simultaneously, the non-PC-breakpoint debug interrupt is taken (vector number 12).

Execution continues at the instruction address in the vector corresponding to the breakpoint triggered. The debug interrupt handler can use supervisor instructions to save the necessary context such as the state of all program-visible registers into a reserved memory area.

During a debug interrupt service routine, all normal interrupt requests are evaluated and sampled once per instruction. If any exception occurs, the processor responds as follows:

1. It saves a copy of the current value of the emulator mode state bit and then exits emulator mode by clearing the actual state.
2. Bit 1 of the fault status field (FS1) in the next exception stack frame is set to indicate the processor was in emulator mode when the interrupt occurred. This corresponds to bit 17 of the longword at the top of the system stack. See Section 2.8.1, “Exception Stack Frame Definition.”
3. It passed control to the appropriate exception handler.
4. It executes an RTE instruction when the exception handler finishes. During the processing of the RTE, FS1 is reloaded from the system stack. If this bit is set, the processor sets the emulator mode state and resumes execution of the original debug interrupt service routine. This is signaled externally by the generation of the PST value that originally identified the debug interrupt exception, that is, PST = 0xD.

Fault status encodings are listed in Table 2-21. Implementation of this debug interrupt handling fully supports the servicing of a number of normal interrupt requests during a debug interrupt service routine. The emulator mode state bit is essentially changed to be a program-visible value, stored into memory during exception stack frame creation, and loaded from memory by the RTE instruction.

When debug interrupt operations complete, the RTE instruction executes and the processor exits emulator mode. After the debug interrupt handler completes execution, the external

development system can use BDM commands to read the reserved memory locations.

The generation of another debug interrupt during the first instruction after the RTE exits emulator mode is inhibited. This behavior is consistent with the existing logic involving trace mode where the first instruction executes before another trace exception is generated. Thus, all hardware breakpoints are disabled until the first instruction after the RTE completes execution, regardless of the programmed trigger response.

#### 5.6.1.1 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 5.5.1, “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:

- Unmasked interrupt requests are serviced. The resulting interrupt exception stack frame has FS[1] set to indicate the interrupt occurred while in emulator mode.
- 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 = 2, TM = 5 or 6. This includes stack frame writes and the vector fetch for the exception that forced entry into this mode.

The return-from-exception (RTE) instruction exits emulation mode. The processor status output port provides a unique encoding for emulator mode entry (0xD) and exit (0x7).

### 5.6.2 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

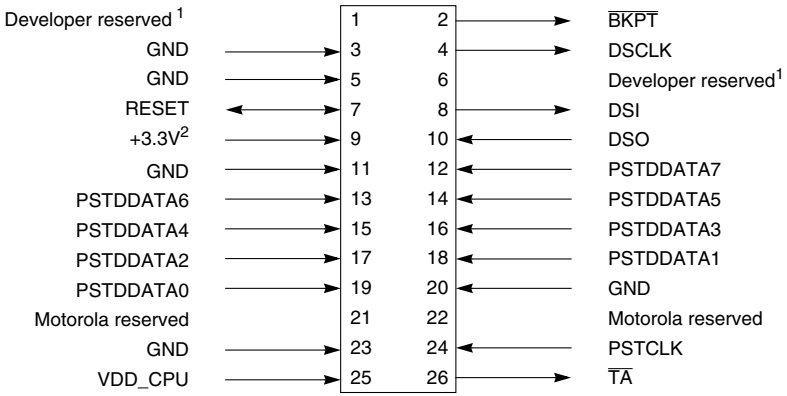
For BDM commands that access memory, the debug module requests the processor’s local bus. The processor responds by stalling the instruction fetch pipeline and waiting for current bus activity to complete before freeing the local bus for the debug module to perform its access. After the debug module 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 and XTDR should be disabled while breakpoint registers are loaded, after which TDR and XTDR can be written to define the exact trigger. This prevents spurious breakpoint triggers.

Because there are no hardware interlocks in the debug unit, no BDM operations are allowed while the CPU is writing the debug's registers (DSCLK must be inactive).

## 5.7 Motorola-Recommended BDM Pinout

The ColdFire BDM connector, Figure 5-46, is a 26-pin Berg connector arranged 2 x 13.




1 Pins reserved for BDM developer use.
2 Supplied by target.

**Figure 5-46. Recommended BDM Connector**

## 5.8 Debug C Definition of PSTDDATA Outputs

This section specifies the ColdFire processor and debug module's generation of the PSTDDATA output on an instruction basis. In general, the PSTDDATA output for an instruction is defined as PSTDDATA = 1, {[89B], operand} where the {...} definition is optional operand information defined by the setting of the CSR. The [89B] signifies a PST value that is a marker identifying the presence and size of valid data to follow. A PST value of 0x8 (1 byte of data), 0x9 (2 bytes), or 0xB (4 bytes) is displayed before the data output.

The CSR allows operands to be displayed based on reference type (read, write, or both). A PST value {0x8, 0x9, or 0xB} identifies the size and presence of valid data to follow on the PSTDDATA output {1, 2, or 4 bytes}. Additionally, for certain change-of-flow branch instructions, CSR[9,8] provides the ability to display {0x2, 0x3, or 0x4} bytes of the target instruction address. A PST value {0x9, 0xA, or 0xB} provides the marker identifying the size and presence of a valid target address on the PSTDDATA output.

## 5.8.1 User Instruction Set

Table 5-27 shows the PSTDDATA specification for user-mode instructions.

**Table 5-27. PSTDDATA Specification for User-Mode Instructions**

| Instruction | Syntax | PSTDDATA |
|---|---|---|
| add.l | <ea>y,Rx | PSTDDATA = 1, {B, source operand} |
| add.l | Dy,<ea>x | PSTDDATA = 1, {B, source}, {B, destination} |
| addi.l | #imm,Dx | PSTDDATA = 1 |
| addq.l | #imm,<ea>x | PSTDDATA = 1, {B, source}, {B, destination} |
| addx.l | Dy,Dx | PSTDDATA = 1 |
| and.l | <ea>y,Dx | PSTDDATA = 1, {B, source operand} |
| and.l | Dy,<ea>x | PSTDDATA = 1, {B, source}, {B, destination} |
| andi.l | #imm,Dx | PSTDDATA = 1 |
| asl.l | {Dy,#imm},Dx | PSTDDATA = 1 |
| asr.l | {Dy,#imm},Dx | PSTDDATA = 1 |
| bcc.{b,w,l} | | if taken, then PSTDDATA = 5, else PSTDDATA = 1 |
| bchg | Dy,<ea>x | PSTDDATA = 1, {8, source}, {8, destination} |
| bchg | #imm,<ea>x | PSTDDATA = 1, {8, source}, {8, destination} |
| bclr | Dy,<ea>x | PSTDDATA = 1, {8, source}, {8, destination} |
| bclr | #imm,<ea>x | PSTDDATA = 1, {8, source}, {8, destination} |
| bra.{b,w,l} | | PSTDDATA = 5 |
| bset | Dy,<ea>x | PSTDDATA = 1, {8, source}, {8, destination} |
| bset | #imm,<ea>x | PSTDDATA = 1, {8, source}, {8, destination} |
| bsr.{b,w,l} | | PSTDDATA = 5, {B, destination operand} |
| btst | Dy,<ea>x | PSTDDATA = 1, {8, source operand} |
| btst | #imm,<ea>x | PSTDDATA = 1, {8, source operand} |
| clr.b | <ea>x | PSTDDATA = 1, {8, destination operand} |
| clr.w | <ea>x | PSTDDATA = 1, {9, destination operand} |
| clr.l | <ea>x | PSTDDATA = 1, {B, destination operand} |
| cmp.b | <ea>y,Rx | PSTDDATA = 1, {8, source operand} |
| cmp.w | <ea>y,Rx | PSTDDATA = 1, {9, source operand} |
| cmp.l | <ea>y,Rx | PSTDDATA = 1, {B, source operand} |
| cmpi.b | #imm,Dx | PSTDDATA = 1 |
| cmpi.w | #imm,Dx | PSTDDATA = 1 |
| cmpi.l | #imm,Dx | PSTDDATA = 1 |
| divs.l | <ea>y,Dx | PSTDDATA = 1, {B, source operand} |
| divs.w | <ea>y,Dx | PSTDDATA = 1, {9, source operand} |
| divu.l | <ea>y,Dx | PSTDDATA = 1, {B, source operand} |

## Table 5-27. PSTDDATA Specification for User-Mode Instructions (Continued)

| Instruction | Syntax | PSTDDATA |
|---|---|---|
| divu.w | <ea>y,Dx | PSTDDATA = 1, {9, source operand} |
| eor.l | Dy,<ea>x | PSTDDATA = 1, {B, source}, {B, destination} |
| eori.l | #imm,Dx | PSTDDATA = 1 |
| ext.w | Dx | PSTDDATA = 1 |
| ext.l | Dx | PSTDDATA = 1 |
| extb.l | Dx | PSTDDATA = 1 |
| jmp | <ea>x | PSTDDATA = 5, {[9AB], target address} [1] |
| jsr | <ea>x | PSTDDATA = 5, {[9AB], target address},{B, destination operand} [1] |
| lea | <ea>y,Ax | PSTDDATA = 1 |
| link.w | Ay,#imm | PSTDDATA = 1, {B, destination operand} |
| lsl.l | {Dy,#imm},Dx | PSTDDATA = 1 |
| lsr.l | {Dy,#imm},Dx | PSTDDATA = 1 |
| mac.l | Ry,Rx | PSTDDATA = 1 |
| mac.l | Ry,Rx,ea,Rw | PSTDDATA = 1, {B, source operand} |
| mac.w | Ry,Rx | PSTDDATA = 1 |
| mac.w | Ry,Rx,ea,Rw | PSTDDATA = 1, {B, source operand} |
| mov3q.l | #imm,<ea>x | PSTDDATA = 1, {B, destination operand} |
| move.b | <ea>y,<ea>x | PSTDDATA = 1, {8, source}, {8, destination} |
| move.w | <ea>y,<ea>x | PSTDDATA = 1, {9, source}, {9, destination} |
| move.l | <ea>y,<ea>x | PSTDDATA = 1, {B, source}, {B, destination} |
| move.l | <ea>y,ACC | PSTDDATA = 1 |
| move.l | <ea>y,MACSR | PSTDDATA = 1 |
| move.l | <ea>y,MASK | PSTDDATA = 1 |
| move.l | ACC,Rx | PSTDDATA = 1 |
| move.l | MACSR,CCR | PSTDDATA = 1 |
| move.l | MACSR,Rx | PSTDDATA = 1 |
| move.l | MASK,Rx | PSTDDATA = 1 |
| move.w | CCR,Dx | PSTDDATA = 1 |
| move.w | {Dy,#imm},CCR | PSTDDATA = 1 |
| movem.l | <ea>y,#list | PSTDDATA = 1, {B, source},... [2] |
| movem.l | #list,<ea>x | PSTDDATA = 1, {B, destination},...[2] |
| moveq | #imm,Dx | PSTDDATA = 1 |
| msac.l | Ry,Rx | PSTDDATA = 1 |
| msac.l | Ry,Rx,ea,Rw | PSTDDATA = 1, {B, source operand} |
| msac.w | Ry,Rx | PSTDDATA = 1 |
| msac.w | Ry,Rx,ea,Rw | PSTDDATA = 1, {B, source operand} |

## Table 5-27. PSTDDATA Specification for User-Mode Instructions (Continued)

| Instruction | Syntax | PSTDDATA |
|---|---|---|
| muls.w | <ea>y,Dx | PSTDDATA = 1, {9, source operand} |
| mulu.w | <ea>y,Dx | PSTDDATA = 1, {9, source operand} |
| muls.l | <ea>y,Dx | PSTDDATA = 1, {B, source operand} |
| mulu.l | <ea>y,Dx | PSTDDATA = 1, {B, source operand} |
| mvs.b | <ea>y,Dx | PSTDDATA = 1, {8, source operand} |
| mvs.w | <ea>y,Dx | PSTDDATA = 1, {9, source operand} |
| mvz.b | <ea>y,Dx | PSTDDATA = 1, {8, source operand} |
| mvz.w | <ea>y,Dx | PSTDDATA = 1, {9, source operand} |
| neg.l | Dx | PSTDDATA = 1 |
| negx.l | Dx | PSTDDATA = 1 |
| nop | | PSTDDATA = 1 |
| not.l | Dx | PSTDDATA = 1 |
| or.l | <ea>y,Dx | PSTDDATA = 1, {B, source operand} |
| or.l | Dy,<ea>x | PSTDDATA = 1, {B, source}, {B, destination} |
| ori.l | #imm,Dx | PSTDDATA = 1 |
| pea | <ea>y | PSTDDATA = 1, {B, destination operand} |
| pulse | | PSTDDATA = 4 |
| rems.l | <ea>y,Dx:Dw | PSTDDATA = 1, {B, source operand} |
| remu.l | <ea>y,Dx:Dw | PSTDDATA = 1, {B, source operand} |
| rts (not predicted) | | PSTDDATA=1, {B, source operand},<br>PSTDDATA=5, {[9AB], target address} |
| rts (predicted) [3] | | PSTDDATA=1, {B, source operand}, 5 |
| sats.l | Dx | PSTDDATA = 1 |
| scc | Dx | PSTDDATA = 1 |
| sub.l | <ea>y,Rx | PSTDDATA = 1, {B, source operand} |
| sub.l | Dy,<ea>x | PSTDDATA = 1, {B, source}, {B, destination} |
| subi.l | #imm,Dx | PSTDDATA = 1 |
| subq.l | #imm,<ea>x | PSTDDATA = 1, {B, source}, {B, destination} |
| subx.l | Dy,Dx | PSTDDATA = 1 |
| swap | Dx | PSTDDATA = 1 |
| tas | <ea>x | PSTDDATA = 1, {8, source}, {8, destination} |
| trap | #imm | PSTDDATA = 1 [4] |
| tpf | | PSTDDATA = 1 |
| tpf.w | | PSTDDATA = 1 |
| tpf.l | | PSTDDATA = 1 |
| tst.b | <ea>x | PSTDDATA = 1, {8, source operand} |

**Table 5-27. PSTDDATA Specification for User-Mode Instructions (Continued)**

| Instruction | Syntax | PSTDDATA |
|---|---|---|
| tst.w | <ea>x | PSTDDATA = 1, {9, source operand} |
| tst.l | <ea>x | PSTDDATA = 1, {B, source operand} |
| unlk | Ax | PSTDDATA = 1, {B, destination operand} |
| wddata.b | <ea>y | PSTDDATA = 4, 8, source operand |
| wddata.w | <ea>y | PSTDDATA = 4, 9, source operand |
| wddata.l | <ea>y | PSTDDATA = 4, B, 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 address reaches a cache line (0-modulo-16) boundary four or more registers are to be transferred. For these line-sized transfers, the operand data is never captured nor displayed, regardless of the CSR value.

Automatic line-sized burst transfers maximize performance for these sequential memory accesses.

3 The source operand in predicted RTS is displayed if CSR[12] and/or (CSR[9] or CSR[8]) is set.

4 During normal exception processing, PSTDDATA outputs are driven to 0xC. The exception stack write operands and the vector read and target address of the exception handler may also be displayed.

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.

```
Exception Processing

        PSTDDATA = C,{B, destination},// stack frame
                     {B, destination},// stack frame
                     {B, source},// vector read
        PSTDDATA = 5,{[9AB], target}// PC of handler
```

The PSTDDATA specification for the reset exception is shown below:

```
Exception Processing

        PSTDDATA = C,
        PSTDDATA = 5,{[9AB], target}// initial 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 PSTDDATA = 0xC value is driven at all times, unless the PSTDDATA output is needed for one of the optional marker values or for the taken branch indicator (0x5).

## 5.8.2 Supervisor Instruction Set

The supervisor instruction set has complete access to the user mode instructions plus the opcodes shown below. Table 5-28 shows the PSTDDATA specification for these opcodes.

**Table 5-28. PSTDDATA Specification for Supervisor-Mode Instructions**

| Instruction | Syntax | PSTDDATA |
|---|---|---|
| cpushl | | PSTDDATA = 1 |
| halt | | PSTDDATA = 1,<br>PSTDDATA = F |
| intouch | | PSTDDATA = 1 |
| move.w | SR,Dx | PSTDDATA = 1 |
| move.w | {Dy,#imm},SR | PSTDDATA = 1, {3} |
| movec | Ry,Rc | PSTDDATA = 1 |
| rte | | PSTDDATA = 7, {B, source operand}, {3},{B, source operand}, {DD},<br>PSTDDATA = 5, {[9AB], target address} |
| stop | #imm | PSTDDATA = 1,<br>PSTDDATA = E |
| wdebug | <ea>y | PSTDDATA = 1, {B, source, B, source} |

The move-to-SR and RTE instructions include an optional PSTDDATA = 0x3 value, indicating an entry into user mode. Additionally, if the execution of a RTE instruction returns the processor to emulator mode, a multiple-cycle status of 0xD is signaled.

Similar to the exception processing mode, the stopped state (PSTDDATA = 0xE) and the halted state (PSTDDATA = 0xF) display this status throughout the entire time the ColdFire processor is in the given mode.

# Part II
# System Integration Module (SIM)

## Intended Audience

Part II is intended for users who need to understand the interface between the ColdFire core processor complex, described in Part I, and internal peripheral devices, described in Part III. It includes a general description of the SIM and individual chapters that describe components of the SIM, such as the phase-lock loop (PLL) timing source, interrupt controller for both on-chip and external peripherals, configuration and operation of chip selects, and the SDRAM controller.

## Contents

Part II contains the following chapters:

- Chapter 6, “SIM Overview,” describes the SIM programming model, bus arbitration, and system-protection functions for the MCF5407.
- Chapter 7, “Phase-Locked Loop (PLL),” describes configuration and operation of the PLL module. It describes in detail the registers and signals that support the PLL implementation.
- Chapter 8, “I2C Module,” describes the MCF5407 I2C module, including I2C protocol, clock synchronization, and the registers in the I2C programing model. It also provides extensive programming examples.
- Chapter 9, “Interrupt Controller,” describes operation of the interrupt controller portion of the SIM. Includes descriptions of the registers in the interrupt controller memory map and the interrupt priority scheme.
- Chapter 10, “Chip-Select Module,” describes the MCF5407 chip-select implementation, including the operation and programming model, which includes the chip-select address, mask, and control registers.
- Chapter 11, “Synchronous/Asynchronous DRAM Controller Module,” describes configuration and operation of the synchronous/asynchronous DRAM controller component of the SIM. It begins with a general description and brief glossary, and

includes a description of signals involved in DRAM operations. The remainder of the chapter is divided between descriptions of asynchronous and synchronous operations.

## Suggested Reading

The following literature may be helpful with respect to the topics in Part II:

- *The $I^2C$ Bus Specification, Version 2.1* (January 2000)

## Acronyms and Abbreviations

Table II-i contains acronyms and abbreviations are used in Part II.

**Table II-i. Acronyms and Abbreviated Terms**

| Term | Meaning |
|---|---|
| ADC | Analog-to-digital conversion |
| BDM | Background debug mode |
| CODEC | Code/decode |
| DAC | Digital-to-analog conversion |
| DMA | Direct memory access |
| DSP | Digital signal processing |
| EDO | Extended data output (DRAM) |
| FIFO | First-in, first-out |
| GPIO | General-purpose I/O |
| $I^2C$ | Inter-integrated circuit |
| IEEE | Institute for Electrical and Electronics Engineers |
| IPL | Interrupt priority level |
| JEDEC | Joint Electron Device Engineering Council |
| LIFO | Last-in, first-out |
| LRU | Least recently used |
| LSB | Least-significant byte |
| lsb | Least-significant bit |
| MBAR | Memory base address register |
| MSB | Most-significant byte |
| msb | Most-significant bit |
| Mux | Multiplex |

## Table II-i. Acronyms and Abbreviated Terms (Continued)

| Term | Meaning |
|---|---|
| NOP | No operation |
| PCLK | Processor clock |
| PLL | Phase-locked loop |
| POR | Power-on reset |
| Rx | Receive |
| SIM | System integration module |
| SOF | Start of frame |
| TAP | Test access port |
| TTL | Transistor-to-transistor logic |
| Tx | Transmit |
| UART | Universal asynchronous/synchronous receiver transmitter |

# Chapter 6
# SIM Overview

This chapter provides detailed operation information regarding the system integration module (SIM). It describes the SIM programming model, bus arbitration, and system-protection functions for the MCF5407.

## 6.1 Features

The SIM, shown in Figure 6-1, provides overall control of the bus and serves as the interface between the ColdFire core processor complex and the internal peripheral devices.

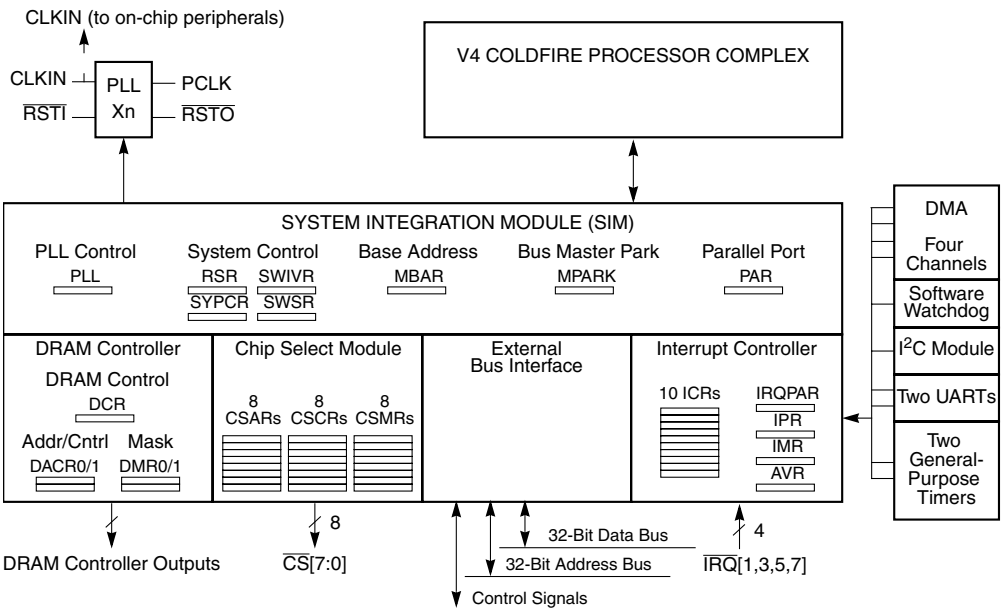



**Figure 6-1. SIM Block Diagram**

The following is a list of the key SIM features:

- Module base address register (MBAR)
  - Base address location of all internal peripherals and SIM resources
  - Address space masking to internal peripherals and SIM resources
- Phase-locked loop (PLL) clock control register (PLLCR) for CPU STOP instruction
  - Control for turning off clocks to core and interrupt levels that turn clocks back on

  Chapter 7, “Phase-Locked Loop (PLL).”
- Interrupt controller
  - Programmable interrupt level (1–7) for internal peripheral interrupts
  - Programmable priority level (0–3) within each interrupt level
  - Four external interrupts; one set to interrupt level 7; three others programmable to two interrupt levels

  See Chapter 9, “Interrupt Controller.”
- Chip select module
  - Eight independent, user-programmable chip-select signals ($\overline{CS}$[7:0]) that can interface with SRAM, PROM, EPROM, EEPROM, Flash, and peripherals
  - Address masking for 64-Kbyte to 4-Gbyte memory block sizes
  - Programmable wait states and port sizes
  - External master access to chip selects

  See Chapter 10, “Chip-Select Module.”
- System protection and reset status
  - Reset status indicating the cause of last reset
  - Software watchdog timer with programmable secondary bus monitor

  See Section 6.2.4, “Software Watchdog Timer.”
- Pin assignment register (PAR) configures the parallel port. See Section 6.2.9, “Pin Assignment Register (PAR).”
- Bus arbitration
  - Default bus master park register (MPARK) controls internal and external bus arbitration and enables display of internal accesses on the external bus for debugging
  - Supports several arbitration algorithms

  See Section 6.2.10, “Bus Arbitration Control.”

## 6.2 Programming Model

The following sections describe the registers incorporated into the SIM.

### 6.2.1 SIM Register Memory Map

Table 6-1 shows the memory map for the SIM registers. The internal registers in the SIM are memory-mapped registers offset from the MBAR address pointer defined in MBAR[BA]. This supervisor-level register is described in Section 6.2.2, “Module Base Address Register (MBAR).” Because SIM registers depend on the base address defined in MBAR[BA], MBAR must be programmed before SIM registers can be accessed.

**NOTE:**

Although external masters cannot access the MCF5407’s on-chip memories or MBAR, they can access any of the SIM memory map and peripheral registers, such as those belonging to the interrupt controller, chip-select module, UARTs, timers, DMA, and $I^2C$.

**Table 6-1. SIM Registers**

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| 0x000 | Reset status register (RSR) [p. 6-5] | System protection control register (SYPCR) [p. 6-8] | Software watchdog interrupt vector register (SWIVR) [p. 6-9] | Software watchdog service register (SWSR) [p. 6-9] |
| 0x004 | Pin assignment register (PAR) [p. 6-10] | | Interrupt port assignment register (IRQPAR) [p. 9-7] | Reserved |
| 0x008 | PLL control (PLLCR) [p. 7-3] | Reserved | | |
| 0x00C | Default bus master park register (MPARK) [p. 6-11] | Reserved | | |
| 0x010–0x03C | Reserved | | | |
| **Interrupt Controller Registers [p. 9-2]** | | | | |
| 0x040 | Interrupt pending register (IPR) [p. 9-6] | | | |
| 0x044 | Interrupt mask register (IMR) [p. 9-6] | | | |
| 0x048 | Reserved | | | Autovector register (AVR) [p. 9-5] |
| Interrupt Control Registers (ICRs) [p. 9-3] | | | | |

**Table 6-1. SIM Registers (Continued)**

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| 0x04C | Software watchdog timer (ICR0) [p. 9-3] | Timer0 (ICR1) [p. 9-3] | Timer1 (ICR2) [p. 9-3] | $I^2C$ (ICR3) [p. 9-3] |
| 0x050 | UART0 (ICR4) [p. 9-3] | UART1 (ICR5) [p. 9-3] | DMA0 (ICR6) [p. 9-3] | DMA1 (ICR7) [p. 9-3] |
| 0x054 | DMA2 (ICR8) [p. 9-3] | DMA3 (ICR9) [p. 9-3] | Reserved | |

## 6.2.2 Module Base Address Register (MBAR)

The supervisor-level MBAR, Figure 6-2, specifies the base address and allowable access types for all internal peripherals. It is written with a MOVEC instruction using the CPU address 0xC0F. (See the *ColdFire Family Programmer's Reference Manual*.) MBAR can be read or written through the debug module as a read/write register, as described in Chapter 5, "Debug Support." Only the debug module can read MBAR.

The valid bit, MBAR[V], is cleared at system reset to prevent incorrect references before MBAR is written; other MBAR bits are uninitialized at reset. To access internal peripherals, write MBAR with the appropriate base address (BA) and set MBAR[V] after system reset.

All internal peripheral registers occupy a single relocatable memory block along 4-Kbyte boundaries. If MBAR[V] is set, MBAR[BA] is compared to the upper 20 bits of the full 32-bit internal address to determine if an internal peripheral is being accessed. MBAR masks specific address spaces using the address space fields. Attempts to access a masked address space generate an external bus access.

Addresses hitting overlapping memory spaces take the following priority:

1. MBAR
2. SRAM and caches
3. Chip select

**NOTE:**

The MBAR region must be mapped to non-cacheable space.

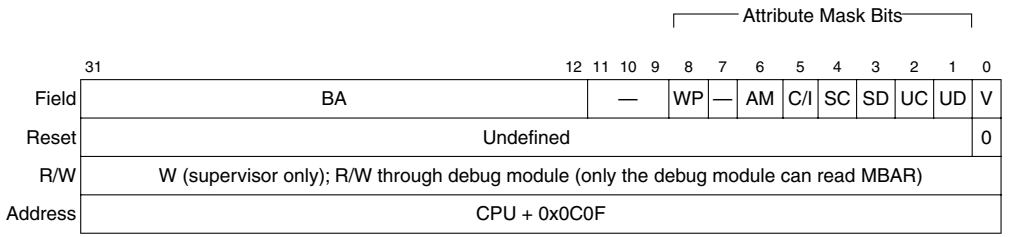



**Figure 6-2. Module Base Address Register (MBAR)**

Table 6-2 describes MBAR fields.

**Table 6-2. MBAR Field Descriptions**

| Bits | Field | Description |
|---|---|---|
| 31–12 | BA | Base address. Defines the base address for a 4-Kbyte address range. |
| 11–9 | — | Reserved, should be cleared. |
| 8 | WP | Write protect. Mask bit for write cycles in the MBAR-mapped register address range.<br>0 Module address range is read/write.<br>1 Module address range is read only. |
| 7 | — | Reserved, should be cleared. |
| 6 | AM | Alternate master mask. When AM = 0 and an alternate master (external master or DMA) accesses MBAR-mapped registers, MBAR[SC,SD,UC,UD] are ignored in address decoding. These fields mask address space, placing the MBAR-mapped register in a specific address space or spaces. |
| 5 | C/I | Mask CPU space and interrupt acknowledge cycles.<br>0 Activates the corresponding MBAR-mapped register<br>1 Regular external bus access |
| 4 | SC | Setting masks supervisor code space in MBAR address range |
| 3 | SD | Setting masks supervisor data space in MBAR address range |
| 2 | UC | Setting masks user code space in MBAR address range |
| 1 | UD | Setting masks user data space in MBAR address range |
| 0 | V | Valid. Determines whether MBAR settings are valid.<br>0 MBAR contents are invalid.<br>1 MBAR contents are valid. |

The following example shows how to set the MBAR to location 0x1000_0000 using the D0 register. Setting MBAR[V] validates the MBAR location. This example assumes all accesses are valid:

```
move.l #0x10000001,DO
movec DO,MBAR
```

## 6.2.3 Reset Status Register (RSR)

The reset status register (RSR), Figure 6-3, contains two status bits, HRST and SWTR. Reset control logic sets one of the bits depending on whether the last reset was caused by an external device asserting $\overline{\text{RSTI}}$ (HRST = 1) or by the software watchdog timer (SWTR = 1). Only one RSR bit can be set at any time. If a reset occurs, reset control logic sets only the bit that indicates the cause of reset.

| | 7 | 6 | 5 | 4 … 0 |
|---|---|---|---|---|
| Field | HRST | — | SWTR | — |
| Reset | 1/0 | 0 | 1/0 | 0_0000 |
| R/W | Read/Write | | | |
| Address | MBAR + 0x000 | | | |

**Figure 6-3. Reset Status Register (RSR)**

Table 6-3 describes RSR fields.

**Table 6-3. RSR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7 | HRST | Hardware or system reset<br>1 An external device driving $\overline{\text{RSTI}}$ caused the last reset. Assertion of reset by an external device causes the core processor to take a reset exception. All registers in internal peripherals and the SIM are reset. |
| 6 | — | Reserved, should be cleared. |
| 5 | SWTR | Software watchdog timer reset<br>1 The last reset was caused by the software watchdog timer. If SYPCR[SWRI] = 1 and the software watchdog timer times out, a hardware reset occurs. |
| 4–0 | — | Reserved, should be cleared. |

## 6.2.4 Software Watchdog Timer

The software watchdog timer prevents system lockup should the software become trapped in loops with no controlled exit. The software watchdog timer can be enabled or disabled through SYPCR[SWE]. If enabled, the watchdog timer requires the periodic execution of a software watchdog servicing sequence. If this periodic servicing action does not occur, the timer times out, resulting in a watchdog timer $\overline{\text{IRQ}}$ or hardware reset with $\overline{\text{RSTO}}$ driven low, as programmed by SYPCR[SWRI].

If the timer times out and the software watchdog transfer acknowledge enable bit (SYPCR[SWTA]) is set, a watchdog timer $\overline{\text{IRQ}}$ is asserted. Note that the software watchdog timer IACK cycle cannot be autovectored.

If a software watchdog timer IACK cycle has not occurred after another timeout, SWT $\overline{\text{TA}}$ is asserted in an attempt to terminate the bus cycle and allow the IACK cycle to proceed. The setting of SYPCR[SWTAVAL] indicates that the watchdog timer $\overline{\text{TA}}$ was asserted. Figure 6-4 shows termination of a locked bus.

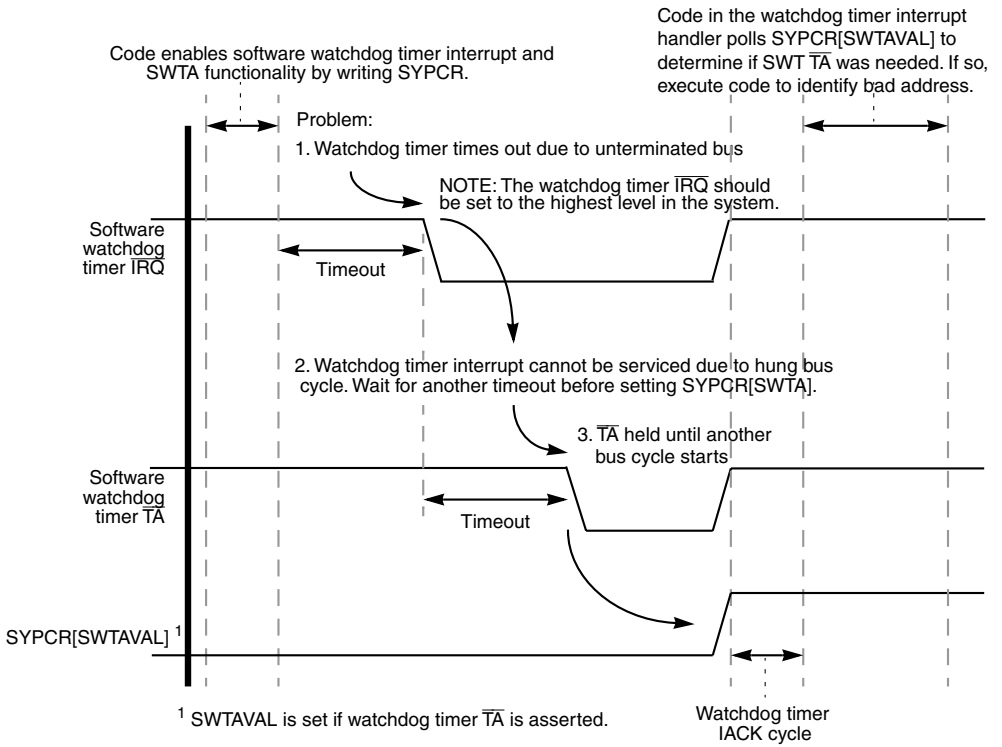



**Figure 6-4. MCF5407 Embedded System Recovery from Unterminated Access**

When the watchdog timer times out and SYPCR[SWRI] is programmed for a software reset, an internal reset is asserted and RSR[SWTR] is set.

To prevent the watchdog timer from interrupting or resetting, the SWSR must be serviced by performing the following sequence:

1. Write 0x55 to SWSR.
2. Write 0xAA to the SWSR.

Both writes must occur in order before the timeout, but any number of instructions or SWSR accesses can be executed between the two writes. This order allows interrupts and exceptions to occur, if necessary, between the two writes.

Caution should be exercised when changing SYPCR values after the software watchdog timer has been enabled with the setting of SYPCR[SWE], because it is difficult to determine the state of the watchdog timer while it is running. The countdown value is constantly compared with the timeout period specified by SYPCR[SWP,SWT]. Therefore, altering SWP and SWT improperly causes unpredictable processor behavior. The following steps must be taken to change SWP or SWT:

1. Disable the software watchdog timer by clearing SYPCR[SWE].
2. Reset the counter by writing 0x55 and then 0xAA to SWSR.
3. Update SYPCR[SWT,SWP].
4. Reenable the watchdog timer by setting SYPCR[SWE]. This can be done in step 3.

## 6.2.5 System Protection Control Register (SYPCR)

The SYPCR, Figure 6-5, controls the software watchdog timer, timeout periods, and software watchdog timer transfer acknowledge. The SYPCR can be read at any time, but can be written only if a software watchdog timer $\overline{IRQ}$ is not pending. At system reset, the software watchdog timer is disabled.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field | SWE | SWRI | SWP | SWT | | SWTA | SWTAVAL | — |
| Reset | 0000_0000 | | | | | | | |
| R/W | R/W | | | | | | | |
| Address | MBAR + 0x01 | | | | | | | |

**Figure 6-5. System Protection Control Register (SYPCR)**

Table 6-4 describes SYPCR fields.

**Table 6-4. SYPCR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7 | SWE | Software watchdog timer enable<br>0 Software watchdog timer disabled<br>1 Software watchdog timer enabled |
| 6 | SWRI | Software watchdog reset/interrupt select<br>0 If a timeout occurs, the watchdog timer generates an interrupt to the core processor at the level programmed into ICR0[IL].<br>1 The software watchdog timer causes soft reset to be asserted for all modules of the part except for the PLL (reset mode selects, such as PP_RESET_SEL or chip-select settings, should not change). |
| 5 | SWP | Software watchdog prescaler. This bit interacts with SYPCR[SWT].<br>0 Software watchdog timer clock not prescaled.<br>1 Software watchdog timer clock prescaled by 8192. |
| 4–3 | SWT | Software watchdog timing delay. SWT and SWP select the timeout period for the watchdog timer. At system reset, the software watchdog timer is set to the minimum timeout period.<br><br>SWP = 0 / SWP = 1<br>00 $2^9$/system frequency / 00 $2^{22}$/system frequency<br>01 $2^{11}$/system frequency / 01 $2^{24}$/system frequency<br>10 $2^{13}$/system frequency / 10 $2^{26}$/system frequency<br>11 $2^{15}$/system frequency / 11 $2^{28}$/system frequency<br><br>Note that if SWP and SWT are modified to select a new software timeout, the software service sequence must be performed (0x55 followed by 0xAA written to the SWSR) before the new timeout period takes effect. |

**Table 6-4. SYPCR Field Descriptions (Continued)**

| Bits | Name | Description |
|---|---|---|
| 2 | SWTA | Software watchdog transfer acknowledge enable<br>0 SWTA transfer acknowledge disabled<br>1 SWTA asserts transfer acknowledge enabled. After one timeout period of the unacknowledged assertion of the software watchdog timer interrupt, the software watchdog transfer acknowledge asserts, which allows the watchdog timer to terminate a bus cycle and allow the IACK to occur. |
| 1 | SWTAVAL | Software watchdog transfer acknowledge valid<br>0 SWTA transfer acknowledge has not occurred.<br>1 SWTA transfer acknowledge has occurred. Write a 1 to clear this flag bit. |

## 6.2.6 Software Watchdog Interrupt Vector Register (SWIVR)

The SWIVR, shown in Figure 6-6, contains the 8-bit interrupt vector (SWIV) that the SIM returns during an interrupt-acknowledge cycle in response to a software watchdog timer-generated interrupt. SWIVR is set to the uninitialized vector 0x0F at system reset.

| | 7 | 0 |
|---|---|---|
| Field | SWIV | |
| Reset | 0000_1111 | |
| R/W | Supervisor write only | |
| Address | MBAR + 0x002 | |

**Figure 6-6. Software Watchdog Interrupt Vector Register (SWIVR)**

Note that the software watchdog interrupt cannot be autovectored.

## 6.2.7 Software Watchdog Service Register (SWSR)

The SWSR, shown in Figure 6-7, is where the software watchdog timer servicing sequence should be written. To prevent a watchdog timer timeout, the software service sequence must be performed (0x55 followed by 0xAA written to the SWSR). Both writes must be performed in order before the timeout, but any number of instructions or accesses to the SWSR can be executed between the two writes. If the timer has timed out, writing to SWSR does not cancel the interrupt (that is, IPR[SWT] remains set). The interrupt is cancelled (and SWT is cleared) automatically when the IACK cycle is run.

| | 7 | 0 |
|---|---|---|
| Field | SWSR | |
| Reset | Undetermined | |
| R/W | Supervisor write only | |
| Address | MBAR + 0x003 | |

**Figure 6-7. Software Watchdog Service Register (SWSR)**

## 6.2.8 PLL Clock Control for CPU STOP Instruction

The SIM contains the PLL clock control register, which is described in detail in Section 7.2.4, “PLL Control Register (PLLCR).” PLLCR[ENBSTOP,PLLIPL] are significant to the operation of the SIM, and are described as follows:

- PLLCR[ENBSTOP] must be set for the ColdFire CPU STOP instruction to be acknowledged. This bit is cleared at reset and must be set for the MCF5407 to enter low-power modes. The CPU STOP instruction stops only clocks to the core processor. All internal modules remain clocked and can generate interrupts to restart the ColdFire core. For example, the on-chip timer can be used to interrupt the processor after a given timer countdown.
- PLLCR[PLLIPL] determines the minimum level at which an interrupt (decoded as an interrupt priority level or IPL) must occur to awaken the PLL. The PLL then turns clocks back on to the core processor and interrupt exception processing takes place. Table 6-5 describes PLLIPL settings to be compared against the interrupt ranges that awaken the core processor from a CPU STOP instruction.

**Table 6-5. PLLIPL Settings**

| PLLIPL | Description |
|---|---|
| 000 | Any interrupts can wake core |
| 001 | Interrupts 2–7 |
| 010 | Interrupts 3–7 |
| 011 | Interrupts 4–7 |
| 100 | Interrupts 5–7 |
| 101 | Interrupts 6–7 |
| 110 | Interrupt 7 only |
| 111 | No interrupts can wake core |

## 6.2.9 Pin Assignment Register (PAR)

The pin assignment register (PAR), Figure 6-8, allows the selection of pin assignments.

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | PAR15 | PAR14 | PAR13 | PAR12 | PAR11 | PAR10 | PAR9 | PAR8 | PAR7 | PAR6 | PAR5 | PAR4 | PAR3 | PAR2 | PAR1 | PAR0 |
| PARn = 0 | PP15 | PP14 | PP13 | PP12 | PP11 | PP10 | PP9 | PP8 | PP7 | PP6 | PP5 | PP4 | PP3 | PP2 | PP1 | PP0 |
| PARn = 1 | A31 | A30 | A29 | A28 | A27 | A26 | A25 | A24 | $\overline{\text{TIP}}$ | $\overline{\text{DREQ0}}$ | $\overline{\text{DREQ1}}$ | TM2 | TM1/ $\overline{\text{DACK1}}$ | TM0/ $\overline{\text{DACK0}}$ | TT1 | TT0 |
| Reset | Determined by driving D4/ADDR_CONFIG with a 1 or 0 when $\overline{\text{RSTI}}$ negates. The system is configured as PP[15:0] if D4 is low; otherwise alternate pin functions selected by PAR = 1 are used. | | | | | | | | | | | | | | | |
| R/W | R/W | | | | | | | | | | | | | | | |
| Address | Address MBAR + 0x004 | | | | | | | | | | | | | | | |

**Figure 6-8. Pin Assignment Register (PAR)**

## 6.2.10 Bus Arbitration Control

This section describes the bus arbitration register and the four arbitration schemes.

### 6.2.10.1 Default Bus Master Park Register (MPARK)

The MPARK, shown in Figure 6-9, determines the default bus master arbitration between internal transfers (core and DMA module) and between internal and external transfers to internal resources. This arbitration is needed because external masters can access internal registers within the MCF5407 peripherals.

| | 7 | 6 | 5 | 4 | 3 | 2 | 0 |
|---|---|---|---|---|---|---|---|
| Field | PARK | | IARBCTRL | EARBCTRL | SHOWDATA | — | |
| Reset | 0000_0000 | | | | | | |
| R/W | R/W | | | | | | |
| Address | MBAR + 0x0C | | | | | | |

**Figure 6-9. Default Bus Master Register (MPARK)**

Table 6-6 describes MPARK bits.

**Table 6-6. MPARK Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–6 | PARK | Park. Indicates the arbitration priority of internal transfers among MCF5407 resources.<br>00 Round-robin between DMA and ColdFire core<br>01 Park on master ColdFire core<br>10 Park on master DMA module<br>11 Park on current master<br>Use of this field is described in detail in Section 6.2.10.1.1, “Arbitration for Internally Generated Transfers (MPARK[PARK]).” |
| 5 | IARBCTRL | Internal bus arbitration control. Controls external device access to the MCF5407 internal bus.<br>0 Arbitration disabled (single-master system)<br>1 Arbitration enabled. IARBCTRL must be set if external masters are using internal resources like the DRAM controller or chip selects.<br>Use of this bit depends on whether the system has single or multiple masters, as follows:<br>• In a single-master system, IARBCTRL should stay cleared, disabling internal arbitration by external masters. In this scenario, MPARK[PARK] applies only to priority of internal masters over one another. Note that the internal DMA (master 3) has priority over the ColdFire core (master 2), if internal DMA bandwidth is at its maximum (BWC = 000).<br>• In multiple master systems that expect to use internal resources like the DRAM controller or chip selects, internal arbitration should be enabled. The external master defaults to the highest priority internal master anytime $\overline{BG}$ is negated. |
| 4 | EARBCTRL | External bus arbitration control. Enables internal register memory space to external bus arbitration. Internal registers are those accessed at offsets to the MBAR. These include the SIM, DMA, chip selects, timers, UARTs, $I^2C$, and parallel port registers. These registers do not include the MBAR; only the core can access the MBAR.<br>0 Arbitration disabled<br>1 Arbitration enabled<br>The use of this field is described in detail in Section 6.2.10.1.2, “Arbitration between Internal and External Masters for Accessing Internal Resources.” |

**Table 6-6. MPARK Field Descriptions (Continued)**

| Bits | Name | Description |
|---|---|---|
| 3 | SHOWDATA | Enable internal register data bus to be driven on external bus. EARBCTRL must be set for this function to work. Section 6.2.10.1.2, “Arbitration between Internal and External Masters for Accessing Internal Resources,” describes the proper use of SHOWDATA.<br>0 Do not drive internal register data bus values to external bus.<br>1 Drive internal register data bus values to external bus. |
| 2–0 | — | Reserved, should be cleared. |

### 6.2.10.1.1 Arbitration for Internally Generated Transfers (MPARK[PARK])

MPARK[PARK] prioritizes internal transfers, which can be initiated by the core and the on-chip DMA module, which contains all four DMA channels. Priority among the four DMA channels in the module is determined by the BWC bits in their respective DMA control registers (see Chapter 12, “DMA Controller Module”).

The four arbitration schemes for internally generated transfers are described as follows:

- Round-robin scheme (PARK = 00)—Figure 6-10 shows round-robin arbitration between the core and DMA module. Bus mastership alternates between the core and DMA module.

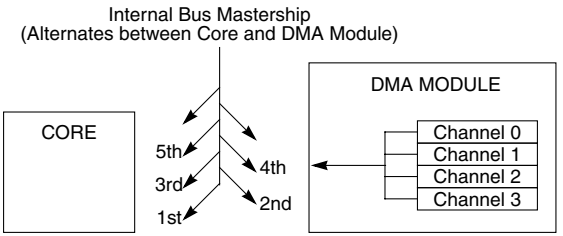



**Figure 6-10. Round Robin Arbitration (PARK = 00)**

The DMA module presents only the highest-priority DMA request, and bus mastership alternates between the core and DMA channel as long as both are requesting bus mastership. Section 12.5.4.1, “External Request and Acknowledge Operation,” includes a timing diagram showing a lower-priority DMA transfer.

When the processor is initialized, the core has first priority. If DMA channels 0 and 1 (both set to BWC = 010) assert an internal bus request during a core-generated bus transfer, DMA channel 0 would gain bus mastership next. However, if the core requests the bus during this DMA transfer, bus mastership returns to the core rather than being granted to DMA channel 1.

Note that the internal DMA has higher priority than the core if the internal DMA has its bandwidth BWC bits set to 000 (maximum bandwidth).

- Park on master core priority (PARK = 01)—The core retains bus mastership as long as it needs it. After it negates its internal bus request, the core does not have to rearbitrate for the bus unless the DMA module has requested the bus when it is idle. The DMA module can be granted bus mastership only when the core is not asserting its bus request. See Figure 6-11.

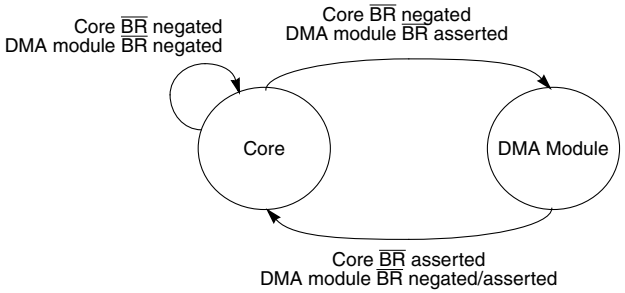



**Figure 6-11. Park on Master Core Priority (PARK = 01)**

- Park on master DMA priority (PARK = 10)—The DMA module retains bus mastership as long as it needs it. After it negates its internal bus request, the DMA module does not have to rearbitrate for the bus unless the core has requested the bus when it is idle. The core can be granted bus mastership only when the DMA module is not asserting its bus request. See Figure 6-12.

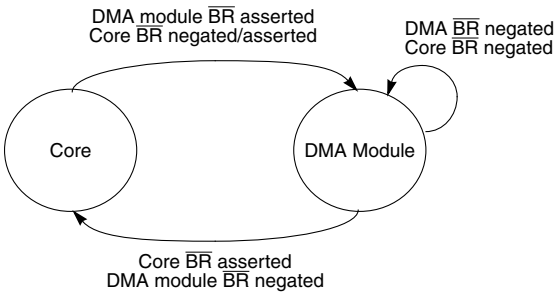



**Figure 6-12. Park on DMA Module Priority (PARK = 10)**

- Park on current master priority (PARK = 11)—The current bus master retains mastership as long as it needs the bus. The other device can become the bus master only when the bus is idle. For example, if the core is bus master out of reset, it retains mastership as long as it needs the bus. It loses mastership only when it negates its bus request signal and the DMA asserts its internal bus request signal. At this point the DMA module is the bus master, and retains bus mastership as long as it needs the bus. See Figure 6-13.

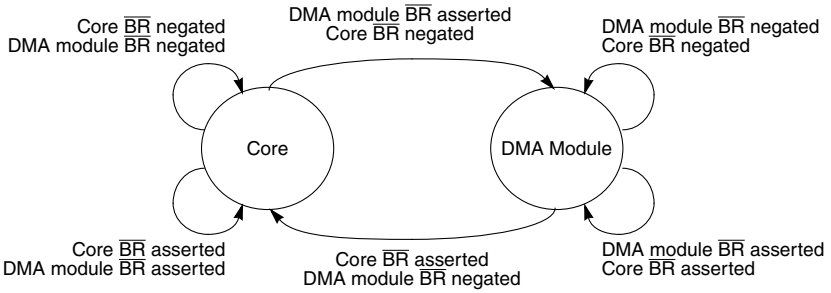



**Figure 6-13. Park on Current Master Priority (PARK = 01)**

### 6.2.10.1.2 Arbitration between Internal and External Masters for Accessing Internal Resources

If an external device is programmed to access internal MCF5407 resources (EARBCTRL = 1), the external device can gain bus mastership only when $\overline{BG}$ is negated. This means neither the core nor the DMA controller can access the external bus until the external device asserts $\overline{BG}$. After the external master finishes its bus transfer and asserts $\overline{BG}$, the core has priority on the next available bus cycle regardless of the value of PARK. Thus if the core asserts its internal bus request on this first bus cycle, it executes a bus cycle even if PARK indicates the DMA should have priority. Then, after the bus transfer, the PARK scheme returns to programmed functioning and the DMA is given bus mastership.

**NOTE:**

In all arbitration modes, if $\overline{BG}$ is negated, the external master interface has highest priority. In this case, the ColdFire core has second-highest priority, until the internal bus grant is asserted.

- In a single-master system, the setting of EARBCTRL does not affect arbitration performance. Typically, $\overline{BG}$ is tied low and the MCF5407 always owns the external bus and internal register transfers are already shown on the external bus. In a system where MCF5407 is the only master, this bit may remain cleared.

  If the system needs external visibility of the data bus values during internal register transfers for system debugging, both EARBCTRL and SHOWDATA must be set.

  Note that when an internal register transfer is driven externally, $\overline{TA}$ becomes an output, which is asserted (normally an input) to prevent external devices and

memories from responding to internal register transfers that go to the external bus. The $\overline{AS}$ signal and all chip-select-related strobe signals are not asserted.

Do not immediately follow a cycle in which SHOWDATA is set with a cycle using fast termination.

- In multiple-master systems, disabling arbitration with EARBCTRL allows performance improvement because internal register bus transfer cycles do not interfere with the external bus.

  Having internal transfers go external may affect performance in two ways:

  — If the internal device does not control the bus immediately, the core stalls until it wins arbitration of the external bus.

  — If the core wins arbitration instantly, it may kick the external master off of the external bus unnecessarily for a transfer that did not need the external bus. For debug, where this performance penalty is not a concern, setting EARBCTRL and SHOWDATA provides external visibility of the internal bus cycles.

# Chapter 7
# Phase-Locked Loop (PLL)

This chapter describes configuration and operation of the phase-locked loop (PLL) module. It describes in detail the registers and signals that support the PLL implementation.

## 7.1 Overview

The basic features of the MCF5407 PLL implementation are as follows:

- The MCF5407 PLL is enhanced to support faster processor clock (PCLK) frequencies than the MCF5307. It also offers a wider range of clock input ratios.
- A buffered processor status clock (PSTCLK) is half the PCLK frequency, as indicated in Figure 7-1. This signal is made available for system development.

The PLL module has the following three modes of operation:

- Reset mode—In reset mode, the core/bus frequency ratio and other configuration information is sampled. At reset, the PLL asserts the reset out signal, $\overline{\text{RSTO}}$.
- Normal mode—During normal operations, the divide ratio is programmed at reset and is clock-multiplied to provide the processor clock frequency. These frequencies are described in the electrical specifications."
- Reduced-power mode—In reduced-power mode, the high-speed processor core clocks are turned off without losing the register contents so that the system can be reenabled by an unmasked interrupt or reset.

Figure 7-1 shows the frequency relationships of PLL module clock signals.

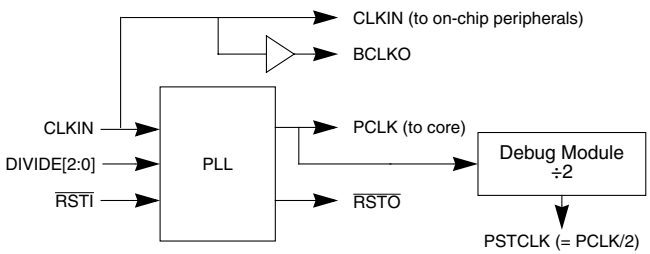



**Figure 7-1. PLL Module Block Diagram**

Motorola recommends using CLKIN for the system clock. BCLKO is provided only for compatibility with slower MCF5307 designs. Regardless of the CLKIN frequency driven at power-up, CLKIN (and BCLKO) have the same ratio value to the PCLK. Although either signal can be used as a clock reference, CLKIN leaves more room to meet the bus specifications than BCLKO, which is generated as a phase-aligned signal to CLKIN.

### 7.1.1 PLL:PCLK Ratios

The PLL for the MCF5407 is enhanced to support faster processor clock (PCLK) frequencies. While the MCF5307 supports various PCLK frequencies listed in the electrical specifications with a clock input (CLKIN) of 1/2 PCLK, the MCF5407 offers a wider range of clock input ratios and a higher performance processor clock.

Like the MCF5307, the MCF5407 samples clock ratio encodings on the lower data bus bits at reset to determine the CLKIN-to-PCLK ratio. These bits are DIVIDE[1:0] on the MCF5307 and are multiplexed with data bits D[1:0]. Because the MCF5407 offers more divide ratio than the MCF5307, three bits, D[2:0]/DIVIDE[2:0], are provided to offer more programming options at reset. Also, note that only specific CLKIN ranges are allowed for each divide ratio on the MCF5407. Table 7-1 shows MCF5407 divide ratio encodings.

**Table 7-1. Divide Ratio Encodings**

| D[2:0]/DIVIDE[2:0] | Multiplier |
|---|---|
| 00x–010 | Reserved |
| 011 | 3 |
| 100 | 4 |
| 101 | 5 |
| 110 | 6 |
| 111 | Reserved |

## 7.2 PLL Operation

The following sections provide detailed information about the three PLL modes.

### 7.2.1 Reset/Initialization

The PLL receives $\overline{\text{RSTI}}$ as an input directly from the pin. Additionally, signals are multiplexed with D[2:0]/DIVIDE[2:0] while $\overline{\text{RSTI}}$ is asserted. These signals are sampled during reset and registered by the PLL on the negation of $\overline{\text{RSTI}}$ to provide initialization information. DIVIDE[2:0] are used by the PLL to set the CLKIN/PCLK ratio.

### 7.2.2 Normal Mode

CLKIN should be used as the system bus clock in 5407 systems. The CLKIN frequency is

multiplied up as determined by the logic level of the multiplexed D[2:0]/DIVIDE[2:0] pins during reset to create PCLK.

## 7.2.3 Reduced-Power Mode

The PCLK can be turned off in a predictable manner to conserve system power. To allow fast restart of the MCF5407 processor core, the PLL continues to operate at the frequency configured at reset. PCLK is disabled using the CPU STOP instruction and resumes normal operation on interrupt, as described in Section 7.2.4, “PLL Control Register (PLLCR).”

## 7.2.4 PLL Control Register (PLLCR)

The PLL control register (PLLCR), Figure 7-2, provides control over the PLL.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field | ENBSTOP | PLLIPL | | | DISBCLKO | — | | |
| Reset | 0000_0000 | | | | | | | |
| R/W | R/W | | | | | | | |
| Address | MBAR + 0x08 | | | | | | | |

**Figure 7-2. PLL Control Register (PLLCR)**

Table 7-2 describes PLLCR bits.

**Table 7-2. PLLCR Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 7 | ENBSTOP | Enable CPU STOP instruction. Must be set for the ColdFire CPU STOP instruction to be acknowledged. Cleared at reset and must be subsequently set for the processor to enter low-power modes. Only clocks to the core are turned off because of the CPU STOP instruction. Internal modules remain clocked and can generate interrupts to restart the ColdFire core.<br>0 Disable CPU STOP<br>1 Enable CPU STOP; STOP instruction turns off clocks to the ColdFire core. |
| 6–4 | PLLIPL | PLL interrupt priority level to wake up from CPU STOP. Determines the minimum level an interrupt (decoded as an interrupt priority level) must be to waken the PLL. The PLL then turns clocks back on to the core processor and interrupt exception processing occurs.<br>000 Any interrupts can wake core<br>001 Interrupts 2–7<br>010 Interrupts 3–7<br>011 Interrupts 4–7<br>100 Interrupts 5–7<br>101 Interrupts 6–7<br>110 Interrupt 7 only<br>111 No interrupts can wake core. Any reset, including a watchdog reset, can wake the core. No PLL phase lock time is required. |
| 3 | DISBCLKO | BCLKO disable. Determines whether BCLKO is driven.<br>0 BCLKO is driven.<br>1 BCLKO is three-stated. BCLKO can be reenabled only by a reset. |
| 2–0 | — | Reserved, should be cleared. |

## 7.3 PLL Port List

Table 7-3 describes PLL module inputs.

**Table 7-3. PLL Module Input SIgnals**

| SIgnal | Description |
|---|---|
| CLKIN | Input clock to the PLL. Input frequency must not be changed during operation. Changes are recognized only at reset. |
| $\overline{\text{RSTI}}$ | Active-low asynchronous input that, when asserted, indicates PLL is to enter reset mode. As long as $\overline{\text{RSTI}}$ is asserted, the PLL is held in reset and does not begin to lock. |
| DIVIDE[2:0] | TheMCF5407 samples clock ratio encodings on the lower data bits of the bus to determine the CLKIN-to-processor clock ratio. D[2:0]/DIVIDE[2:0] support the divide-ratio combinations. Note that only specific CLKIN ranges are allowed for each divide ratio on the MCF5407. See the electrical specifications for valid frequencies. |

Table 7-4 describes PLL module outputs.

**Table 7-4. PLL Module Output Signals**

| Output | Description |
|---|---|
| BCLKO | This bus clock output provides a divided version of the processor clock frequency, determined by DIVIDE[2:0]. BCLKO is provided for MCF5307 compatibility (slower-speed designs). |
| PSTCLK | Provides a buffered processor status clock. PSTCLK is half the frequency of PCLK. See Section 7.4.1, “PCLK, PSTCLK, and BCLKO,” and Figure 7-1. |
| $\overline{\text{RSTO}}$ | This output provides an external reset for peripheral devices. |

## 7.4 Timing Relationships

The MCF5407 CLKIN frequency can be 1/3, 1/4, 1/5, or 1/6 the processor clock. In this document, bus timings are referenced from CLKIN.

Regardless of the CLKIN frequency driven at power-up, CLKIN and BCLKO have the same ratio value to the PCLK. Although either signal can be used as a clock reference, CLKIN leaves more room to meet the bus specifications than BCLKO, which is generated as a phase-aligned signal to CLKIN.

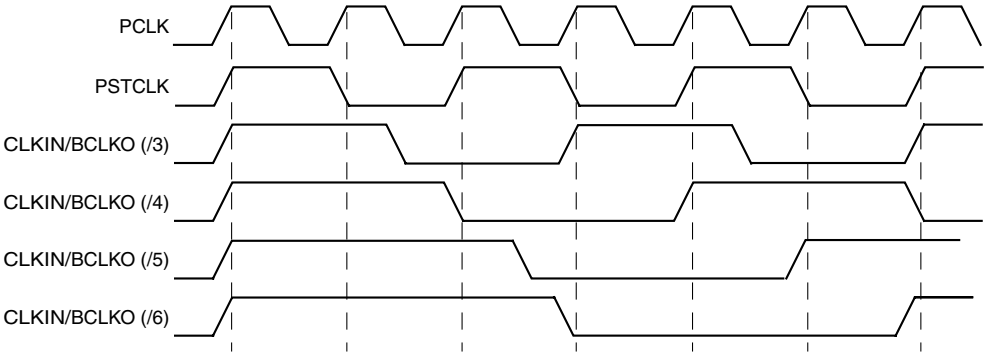
Although the CLKIN duty cycle remains the same for the MCF5307 and MCF5407, caution should be used when interfacing signals on the falling edge of CLKIN with only a 4-nS window to work from at high frequencies. Also, note that the MCF5407 CLKIN rise time is reduced to 2 nS (5 nS in the MCF5307).

If signals are referenced from CLKIN only, setting PLLCR[DISBCLKO] and disabling BCLKO reduces power consumption. See Section 7.2.4, “PLL Control Register (PLLCR).”

### 7.4.1 PCLK, PSTCLK, and BCLKO

Figure 7-3 shows the frequency relationships between PCLK, PSTCLK, and the four

possible versions of CLKIN/BCLKO. This figure does not show the skew between CLKIN and PCLK, PSTCLK, and BCLKO. PSTCLK is half the frequency of PCLK. Similarly, the skew between PCLK and BCLKO is unspecified.




NOTE: The clock signals are shown with edges aligned to show frequency relationships only. Actual signal edges have some skew between them.

**Figure 7-3. CLKIN, PCLK, PSTCLK, and BCLKO Timing**

## 7.4.2 $\overline{\text{RSTI}}$ Timing

Figure 7-4 shows PLL timing during reset. As shown, $\overline{\text{RSTI}}$ must be asserted for at least 16 CLKIN cycles to give the MCF5407 time to begin its initialization sequence. At this time, the configuration pins should be asserted (D[2:0] for DIVIDE[2:0]), meeting the minimum setup and hold times to $\overline{\text{RSTI}}$ given in Chapter 20, “Electrical Specifications.”

On the rising edge of CLKIN before the rising edge of $\overline{\text{RSTI}}$, the data on D[7:0] is latched and the PLL begins ramping to its final operating frequency. During this ramp and lock time, BCLKO and PSTCLK are held low. The PLL locks in about 2 mS or less depending on the CLKIN frequency, at which time BCLKO begins normal operation in the specified mode. The PLL requires 50,000 CLKIN cycles to guarantee PLL lock. To allow for reset of external peripherals requiring a clock source, $\overline{\text{RSTO}}$ remains asserted for a number of CLKIN cycles, as shown in Figure 7-4. PSTCLK will begin oscillating a minimum of10 clock cycles after $\overline{\text{RSTO}}$ is negated.

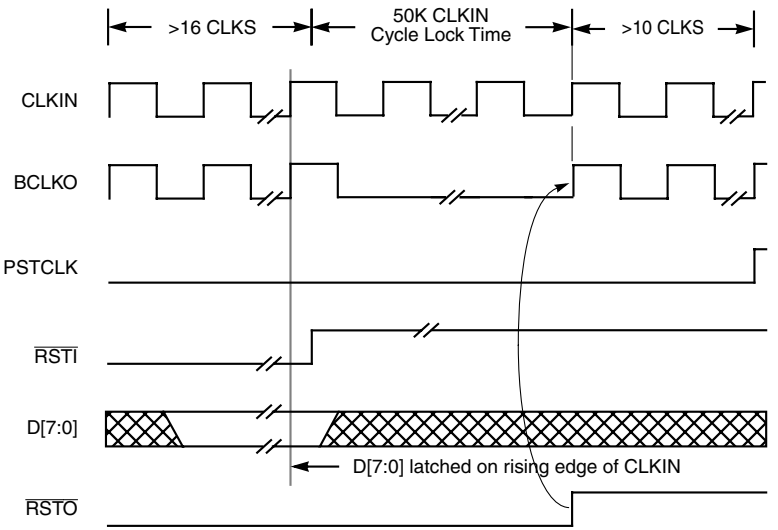



**Figure 7-4. Reset and Initialization Timing**

## 7.5 PLL Power Supply Filter Circuit

To ensure PLL stability, the power supply to the PLL power pin should be filtered using a circuit similar to the one in Figure 7-5. The circuit should be placed as close as possible to the PLL power pin to ensure maximum noise filtering.

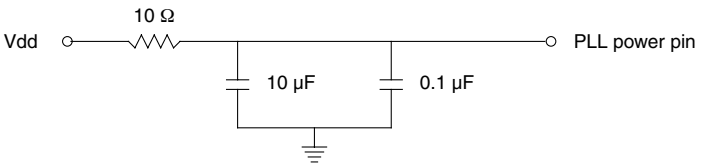



**Figure 7-5. PLL Power Supply Filter Circuit**

# Chapter 8
# I²C Module

This chapter describes the MCF5407 I²C module, including I²C protocol, clock synchronization, and the registers in the I²C programing model. It also provides extensive programming examples.

## 8.1 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 requiring occasional communications over a short distance between many devices. The flexible I²C allows additional devices to be connected to the bus for expansion and system development.

The I²C system is a true multiple-master bus including arbitration and collision detection that prevents data corruption if 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 that I²C is defined for 5-V operation and the MCF5407 is a 3.3-V device. Care must be taken to interface to 5-V peripherals.

## 8.2 Interface Features

The I²C module has the following key features:

- Compatibility with I²C bus standard
- Support for 3.3-V tolerant devices
- Multiple-master operation
- Software-programmable for one of 64 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 8-1 is a block diagram of the I$^2$C module.

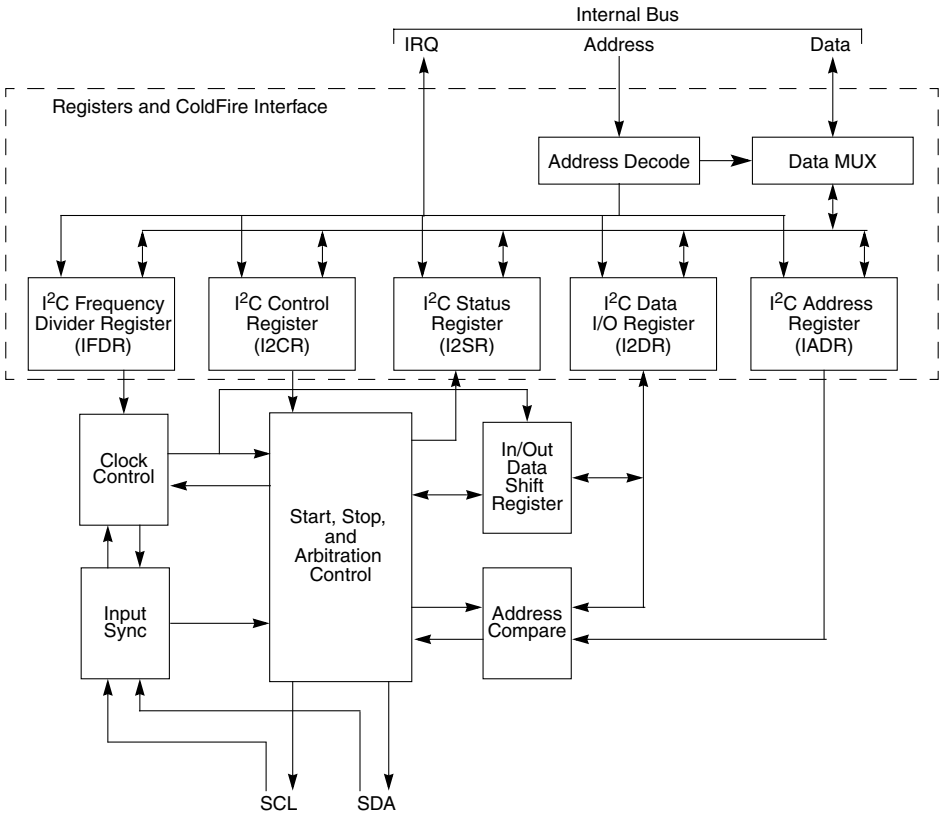



**Figure 8-1. I$^2$C Module Block Diagram**

Figure 8-1 shows the relationships of the I$^2$C registers, listed below:

- I$^2$C address register (IADR)
- I$^2$C frequency divider register (IFDR)
- I$^2$C control register (I2CR)
- I$^2$C status register (I2SR)
- I$^2$C data I/O register (I2DR)

These registers are described in Section 8.5, “Programming Model.”

## 8.3 I²C System Configuration

The I²C module uses a serial data line (SDA) and a serial clock line (SCL) for data transfer. For I²C compliance, all devices connected to these two signals must have open drain or open collector outputs. (There is no such requirement for inputs.) The logic AND function is exercised on both lines with external pull-up resistors.

Out of reset, the I²C default is as 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 8.6.1, “Initialization Sequence,” for exceptions.

**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*.

## 8.4 I²C Protocol

Normally, a standard communication is composed of the following parts:

1. START signal—When no other device is bus master (both SCL and SDA lines are at logic high), a device can initiate communication by sending a START signal (see A in Figure 8-2). A START signal is defined as a high-to-low transition of SDA while SCL is high. This signal denotes the beginning of a data transfer (each data transfer can be several bytes long) and awakens all slaves.

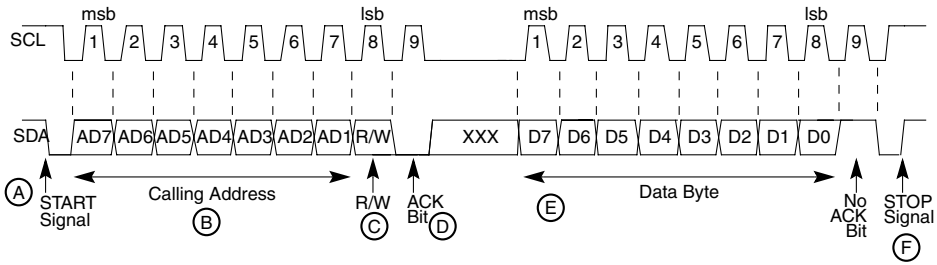



**Figure 8-2. I²C Standard Communication Protocol**

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.

Each slave must have a unique address. An $I^2C$ master must not transmit an address that is the same as its slave address; it cannot be master and slave at the same time.

The slave whose address matches that sent by the master pulls SDA low at the ninth clock (D) to return an acknowledge bit.

3. Data transfer—When successful slave addressing is achieved, the data transfer can proceed (E) 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 SCL is low and must be held stable while SCL is high, as Figure 8-2 shows. SCL is pulsed once for each data bit, with the msb being sent first. The receiving device must acknowledge each byte by pulling SDA low at the ninth clock; therefore, a data byte transfer takes nine clock pulses.

   If it does not acknowledge the master, the slave receiver must leave 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 8-3) to start a new calling sequence.

   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 SDA for the master to generate a STOP or START signal.

4. 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 SDA while SCL is at logical high (F). Note that a master can generate a STOP even if the slave has made an acknowledgment, at which point the slave must release the bus.

Instead of signalling a STOP, the master can repeat the START signal, followed by a calling command, (A in Figure 8-3). A repeated START occurs when a START signal is generated without first generating a STOP signal to end the communication.

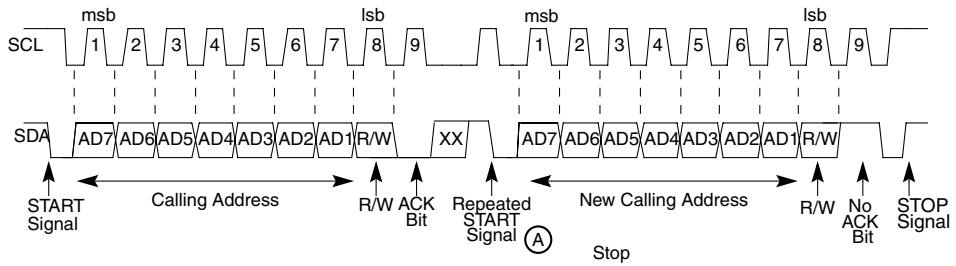



**Figure 8-3. Repeated START**

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.

## 8.4.1 Arbitration Procedure

If multiple devices simultaneously request the bus, the bus clock is determined by a

synchronization procedure in which the low period equals the longest clock-low period among the devices and the high period equals the shortest. A data arbitration procedure determines the relative priority of competing devices. A device loses arbitration if it sends logic high while another sends logic low; it immediately switches to slave-receive mode and stops driving SDA. 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.

## 8.4.2 Clock Synchronization

Because wire-AND logic is used, a high-to-low transition on SCL affects devices connected to the bus. Devices start counting their low period when the master drives SCL low. When a device clock goes low, it holds SCL low until the clock high state is reached. However, the low-to-high change in this device clock may not change the state of SCL if another device clock is still in its low period. Therefore, the device with the longest low period holds the synchronized clock SCL low. Devices with shorter low periods enter a high wait state during this time (See Figure 8-4). When all devices involved have counted off their low period, the synchronized clock SCL is released and pulled high. There is then no difference between device clocks and the state of SCL, so all of the devices start counting their high periods. The first device to complete its high period pulls SCL low again.

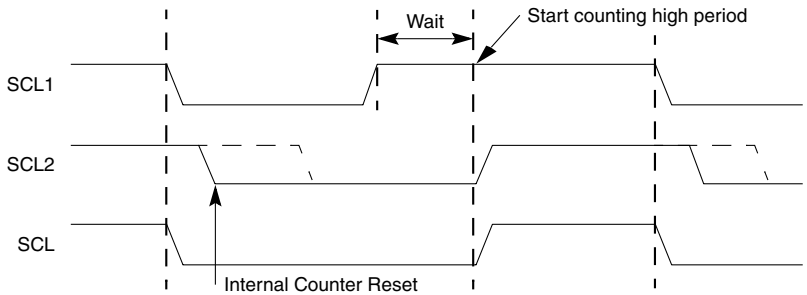



**Figure 8-4. Synchronized Clock SCL**

## 8.4.3 Handshaking

The clock synchronization mechanism can be used as a handshake in data transfers. Slave devices can hold SCL low after completing one byte transfer (9 bits). In such a case, the clock mechanism halts the bus clock and forces the master clock into wait states until the slave releases SCL.

## 8.4.4 Clock Stretching

Slaves can use the clock synchronization mechanism to slow down the transfer bit rate. After the master has driven SCL low, the slave can drive SCL low for the required period and then release it. If the slave SCL low period is longer than the master SCL low period,

the resulting SCL bus signal low period is stretched.

## 8.5 Programming Model

Table 8-1 lists the configuration registers used in the I²C interface.

**Table 8-1. I²C Interface Memory Map**

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| 0x280 | I²C address register (IADR) [p. 8-6] | Reserved | | |
| 0x284 | I²C frequency divider register (IFDR) [p. 8-6] | Reserved | | |
| 0x288 | I²C control register (I2CR) [p. 8-7] | Reserved | | |
| 0x28C | I²C status register (I2SR) [p. 8-8] | Reserved | | |
| 0x290 | I²C data I/O register (I2DR) [p. 8-9] | Reserved | | |

**NOTE:**

External masters cannot access the MCF5407's on-chip memories or MBAR, but can access any I²C module register.

### 8.5.1 I²C Address Register (IADR)

The IADR 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.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field | ADR | | | | | | | — |
| Reset | 0000_0000 | | | | | | | |
| R/W | Read/Write | | | | | | | |
| Address | MBAR + 0x280 | | | | | | | |

**Figure 8-5. I²C Address Register (IADR)**

Table 8-2 describes IADR fields.

**Table 8-2. I²C Address Register Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–1 | ADR | Slave address. Contains the specific slave address to be used by the I²C module. Slave mode is the default I²C mode for an address match on the bus. |
| 0 | — | Reserved, should be cleared. |

### 8.5.2 I²C Frequency Divider Register (IFDR)

The IFDR, Figure 8-6, provides a programmable prescaler to configure the clock for

bit-rate selection.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field | — | — | IC | IC | IC | IC | IC | IC |
| Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R/W | Read/Write | | | | | | | |
| Address | MBAR + 0x284 | | | | | | | |

**Figure 8-6. I²C Frequency Divider Register (IFDR)**

Table 8-3 describes IFDR[IC].

**Table 8-3. IFDR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–6 | — | Reserved, should be cleared. |
| 5–0 | IC | I²C clock rate. Prescales the clock for bit-rate selection. Due to potentially slow SCL and SDA rise and fall times, bus signals are sampled at the prescaler frequency. The serial bit clock frequency is equal to CLKIN divided by the divider shown below. Note that IC can be changed anywhere in a program. |

| IC | Divider | IC | Divider | IC | Divider | IC | Divider |
|---|---|---|---|---|---|---|---|
| 0x00 | 28 | 0x10 | 288 | 0x20 | 20 | 0x30 | 160 |
| 0x01 | 30 | 0x11 | 320 | 0x21 | 22 | 0x31 | 192 |
| 0x02 | 34 | 0x12 | 384 | 0x22 | 24 | 0x32 | 224 |
| 0x03 | 40 | 0x13 | 480 | 0x23 | 26 | 0x33 | 256 |
| 0x04 | 44 | 0x14 | 576 | 0x24 | 28 | 0x34 | 320 |
| 0x05 | 48 | 0x15 | 640 | 0x25 | 32 | 0x35 | 384 |
| 0x06 | 56 | 0x16 | 768 | 0x26 | 36 | 0x36 | 448 |
| 0x07 | 68 | 0x17 | 960 | 0x27 | 40 | 0x37 | 512 |
| 0x08 | 80 | 0x18 | 1152 | 0x28 | 48 | 0x38 | 640 |
| 0x09 | 88 | 0x19 | 1280 | 0x29 | 56 | 0x39 | 768 |
| 0x0A | 104 | 0x1A | 1536 | 0x2A | 64 | 0x3A | 896 |
| 0x0B | 128 | 0x1B | 1920 | 0x2B | 72 | 0x3B | 1024 |
| 0x0C | 144 | 0x1C | 2304 | 0x2C | 80 | 0x3C | 1280 |
| 0x0D | 160 | 0x1D | 2560 | 0x2D | 96 | 0x3D | 1536 |
| 0x0E | 192 | 0x1E | 3072 | 0x2E | 112 | 0x3E | 1792 |
| 0x0F | 240 | 0x1F | 3840 | 0x2F | 128 | 0x3F | 2048 |

## 8.5.3 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.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field | IEN | IIEN | MSTA | MTX | TXAK | RSTA | — | |
| Reset | 0000_0000 | | | | | | | |
| R/W | Read/Write | | | | | | | |
| Address | MBAR + 0x288 | | | | | | | |

**Figure 8-7. I²C Control Register (I2CR)**

Table 8-4 describes I2CR fields.

**Table 8-4. I2CR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7 | IEN | I²C enable. Controls the software reset of the entire I²C 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 I²C module to lose arbitration, after which bus operation returns to normal.<br>0 The module is disabled, but registers can still be accessed.<br>1 The I²C module is enabled. This bit must be set before any other I2CR bits have any effect. |
| 6 | IIEN | I²C interrupt enable.<br>0 I²C module interrupts are disabled, but currently pending interrupt condition are not cleared.<br>1 I²C module interrupts are enabled. An I²C 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.<br>0 Slave mode. Changing MSTA from 1 to 0 generates a STOP and selects slave mode.<br>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.<br>0 Receive<br>1 Transmit. When a slave is addressed, software should set MTX according to I2SR[SRW]. In master mode, MTX should be set according to the type of transfer required. Therefore, for address cycles, MTX is always 1. |
| 3 | TXAK | Transmit acknowledge enable. Specifies the value driven onto SDA during acknowledge cycles for both master and slave receivers. Note that writing TXAK applies only when the I²C bus is a receiver.<br>0 An acknowledge signal is sent to the bus at the ninth clock bit after receiving one byte of data.<br>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.<br>0 No repeat start<br>1 Generates a repeated START condition. |
| 1–0 | — | Reserved, should be cleared. |

## 8.5.4 I²C Status Register (I2SR)

This I2SR contains bits that indicate transaction direction and status.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field | ICF | IAAS | IBB | IAL | — | SRW | IIF | RXAK |
| Reset | 1000_0001 | | | | | | | |
| R/W | R | | | R/W | R | | R/W | R |
| Address | MBAR + 0x28C | | | | | | | |

**Figure 8-8. I²CR Status Register (I2SR)**

Table 8-5 describes I2SR fields.

**Table 8-5. I2SR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7 | ICF | Data transferring bit. While one byte of data is transferred, ICF is cleared.<br>0 Transfer in progress<br>1 Transfer complete. Set by the falling edge of the ninth clock of a byte transfer. |
| 6 | IAAS | I²C 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.<br>0 Not addressed.<br>1 Addressed as a slave. Set when its own address (IADR) matches the calling address. |
| 5 | IBB | I²C bus busy bit. Indicates the status of the bus.<br>0 Bus is idle. If a STOP signal is detected, IBB is cleared.<br>1 Bus is busy. When START is detected, IBB is set. |
| 4 | IAL | Arbitration lost. Set by hardware in the following circumstances. (IAL must be cleared by software by writing zero to it.)<br>• SDA sampled low when the master drives high during an address or data-transmit cycle.<br>• SDA sampled low when the master drives high during the acknowledge bit of a data-receive cycle.<br>• A start cycle is attempted when the bus is busy.<br>• A repeated start cycle is requested in slave mode.<br>• 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 I²C module is a slave and has an address match.<br>0 Slave receive, master writing to slave.<br>1 Slave transmit, master reading from slave. |
| 1 | IIF | I²C interrupt. Must be cleared by software by writing a zero to it in the interrupt routine.<br>0 No I²C interrupt pending<br>1 An interrupt is pending, which causes a processor interrupt request (if IIEN = 1). Set when one of the following occurs:<br>• Complete one byte transfer (set at the falling edge of the ninth clock)<br>• Reception of a calling address that matches its own specific address in slave-receive mode<br>• Arbitration lost |
| 0 | RXAK | Received acknowledge. The value of SDA during the acknowledge bit of a bus cycle.<br>0 An acknowledge signal was received after the completion of 8-bit data transmission on the bus<br>1 No acknowledge signal was detected at the ninth clock. |

## 8.5.5 I²C Data I/O Register (I2DR)

In master-receive mode, reading the I2DR, Figure 8-9, allows a read to occur and initiates

next byte data receiving. In slave mode, the same function is available after it is addressed.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field | D | | | | | | | |
| Reset | 0000_0000 | | | | | | | |
| R/W | Read/Write | | | | | | | |
| Address | MBAR + 0x290 | | | | | | | |

**Figure 8-9. I²C Data I/O Register (I2DR)**

# 8.6 I²C Programming Examples

The following examples show programming for initialization, signalling START, post-transfer software response, signalling STOP, and generating a repeated START.

## 8.6.1 Initialization Sequence

Before the interface can transfer serial data, registers must be initialized, as follows:

1. Set IFDR[IC] to obtain SCL frequency from the system bus clock. See Section 8.5.2, “I2C Frequency Divider Register (IFDR).”
2. Update the IADR to define its slave address.
3. Set I2CR[IEN] to enable the I²C bus interface system.
4. Modify the I2CR to select master/slave mode, transmit/receive mode, and interrupt-enable or not.

**NOTE:**

If IBSR[IBB] when the I²C bus module is enabled, execute the following code 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 = 0xA
dummy read of I2DR
IBSR = 0x0
I2CR = 0x0
```

## 8.6.2 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, IBSR[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 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)+
        BNE.S CHFLAG;If I2SR[MBB] = 1, wait until it is clear
TXSTART 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;
```

## 8.6.3 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 IIF in the interrupt routine. 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 IIF 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 (I2DR[R/W], 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 8-10).

```
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
TRANSMITMOVE.B DATABUF,-(A7);Stack data byte
        MOVE.B (A7)+, I2DR;Transmit next byte of data
```

## 8.6.4 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
```

## 8.6.5 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 signalling 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
```

### 8.6.6 Slave Mode

In the slave interrupt service routine, the module addressed as slave bit (IAAS) should be tested to check if a calling of its own address has just been received. 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 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 SCL so that the master can generate a STOP signal.

### 8.6.7 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 SDA stops, but 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 do a START, hardware inhibits the transmission, clears MSTA without signalling a STOP, generates an interrupt to the CPU, and sets 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.

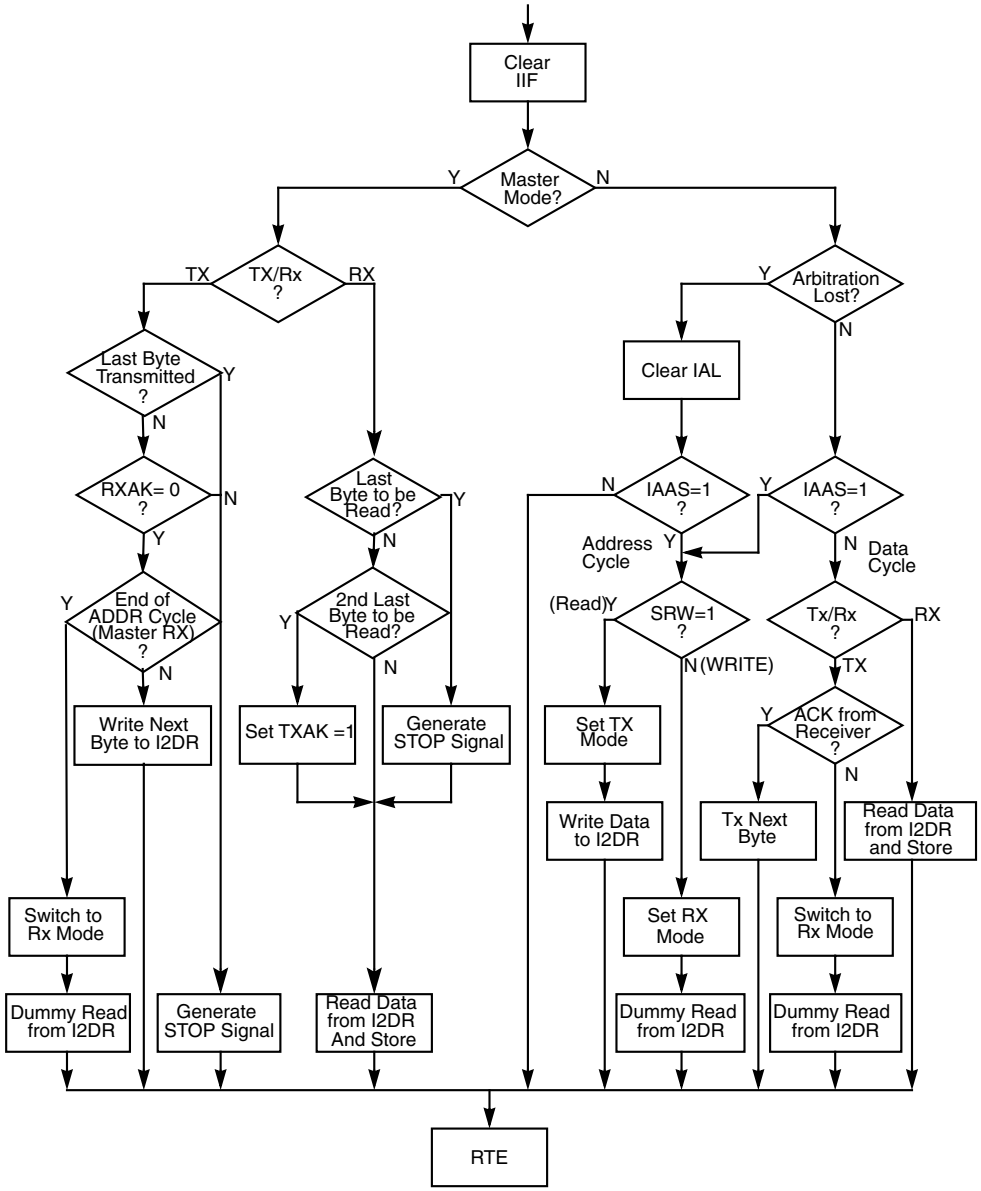



**Figure 8-10. Flow-Chart of Typical I$^2$C Interrupt Routine**

# Chapter 9
# Interrupt Controller

This chapter describes the operation of the interrupt controller portion of the system integration module (SIM). It includes descriptions of the registers in the interrupt controller memory map and the interrupt priority scheme.

## 9.1 Overview

The SIM provides a centralized interrupt controller for all MCF5407 interrupt sources, which consist of the following:

- External interrupts
- Software watchdog timer
- Timer modules
- $I^2C$ module
- UART modules
- DMA module

Figure 9-1 is a block diagram of the interrupt controller.

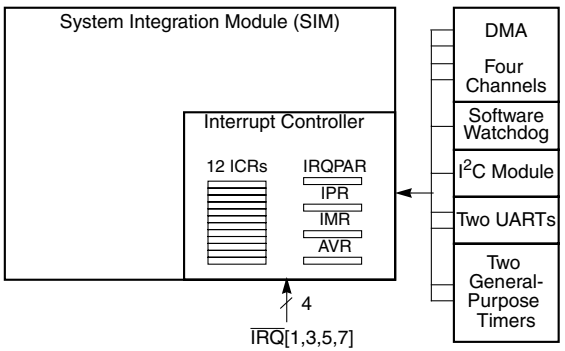



**Figure 9-1. Interrupt Controller Block Diagram**

The SIM provides the following registers for managing interrupts:

- Each potential interrupt source is assigned one of the 10 interrupt control registers (ICR0–ICR9), which are used to prioritize the interrupt sources.
- The interrupt mask register (IMR) provides bits for masking individual interrupt sources.
- The interrupt pending register (IPR) provides bits for indicating when an interrupt request is being made (regardless of whether it is masked in the IMR).
- The autovector register (AVEC) controls whether the SIM supplies an autovector or executes an external interrupt acknowledge cycle for each IRQ.
- The interrupt port assignment register (IRQPAR) provides the level assignment of the primary external interrupt pins—IRQ5, IRQ3, and IRQ1.

## 9.2 Interrupt Controller Registers

The interrupt controller register portion of the SIM memory map is shown in Table 9-2.

**Table 9-1. Interrupt Controller Registers**

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| 0x040 | Interrupt pending register (IPR) [p. 9-6] | | | |
| 0x044 | Interrupt mask register (IMR) [p. 9-6] | | | |
| 0x048 | Reserved | | | Autovector register (AVR) [p. 9-5] |
| **Interrupt Control Registers (ICRs) [p. 9-3]** | | | | |
| 0x04C | Software watchdog timer (ICR0) [p. 9-3] | Timer0 (ICR1) [p. 9-3] | Timer1 (ICR2) [p. 9-3] | $I^2C$ (ICR3) [p. 9-3] |
| 0x050 | UART0 (ICR4) [p. 9-3] | UART1 (ICR5) [p. 9-3] | DMA0 (ICR6) [p. 9-3] | DMA1 (ICR7) [p. 9-3] |
| 0x054 | DMA2 (ICR8) [p. 9-3] | DMA3 (ICR9) [p. 9-3] | Reserved | |

Each internal interrupt source has its own interrupt control register (ICR0–ICR9), shown in Table 9-2 and described in Section 9.2.1, “Interrupt Control Registers (ICR0–ICR9).”

**Table 9-2. Interrupt Control Registers**

| MBAR Offset | Register | Name |
|---|---|---|
| 0x04C | ICR0 | Software watchdog timer |
| 0x04D | ICR1 | Timer0 |
| 0x04E | ICR2 | Timer1 |
| 0x04F | ICR3 | $I^2C$ |
| 0x050 | ICR4 | UART0 |
| 0x051 | ICR5 | UART1 |
| 0x052 | ICR6 | DMA0 |

**Table 9-2. Interrupt Control Registers (Continued)**

| MBAR Offset | Register | Name |
|---|---|---|
| 0x053 | ICR7 | DMA1 |
| 0x054 | ICR8 | DMA2 |
| 0x055 | ICR9 | DMA3 |

Internal interrupts are programmed to a level and priority. Each internal interrupt has a unique ICR. Each of the 7 interrupt levels has 5 priorities, for a total of 35 possible priority levels, encompassing internal and external interrupts. The four external interrupt pins offer seven possible settings at a fixed interrupt level and priority.

The IRQPAR determines these settings for external interrupt request levels. External interrupts can be programmed to supply an autovector or execute an external interrupt acknowledge cycle. This is described in Section 9.2.2, “Autovector Register (AVR).”

## 9.2.1 Interrupt Control Registers (ICR0–ICR9)

The interrupt control registers (ICR0–ICR9) provide bits for defining the interrupt level and priority for the interrupt source assigned to the ICR, shown in Table 9-2.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field | AVEC | — | | IL | | | IP | |
| Reset | 0 | — | | 0_00 | | | 00 | |
| R/W | R/W | | | | | | | |
| Address | MBAR + 0x04C (ICR0); 0x04D (ICR1); 0x04E (ICR2); 0x04F (ICR3); 0x050 (ICR4); 0x051 (ICR5); 0x052 (ICR6); 0x053 (ICR7); 0x054 (ICR8); 0x055 (ICR9) | | | | | | | |

**Figure 9-2. Interrupt Control Registers (ICR0–ICR9)**

Table 9-3 describes ICR fields.

**Table 9-3. ICRn Field Descriptions**

| Bits | Field | Description |
|---|---|---|
| 7 | AVEC | Autovector enable. Determines whether the interrupt-acknowledge cycle input (for the internal interrupt level indicated in IL for each interrupt) requires an autovector response.<br>0 Interrupting source returns vector during interrupt-acknowledge cycle.<br>1 SIM generates autovector during interrupt acknowledge cycle. |
| 6–5 | — | Reserved, should be cleared. |
| 4–2 | IL | Interrupt level. Indicates the interrupt level assigned to each interrupt input. See Table 9-4. |
| 1–0 | IP | Interrupt priority. Indicates the interrupt priority for internal modules within the interrupt-level assignment. See Table 9-4.<br>00 Lowest<br>01 Low<br>10 High<br>11 Highest |

**NOTE:**

Assigning the same interrupt level and priority to multiple ICRs causes unpredictable system behavior.

Table 9-4 shows possible priority schemes for internal and external sources of the MCF5407. The internal module interrupt source in this table can be any internal interrupt source programmed to the given level and priority.

This table shows how external interrupts are prioritized with respect to internal interrupt sources within the same level. For example, UART0 and UART1 sources are programmed to IL = 110; in this case, UART0 is given lower priority than UART1, so ICR4[IP] = 01 and the ICR5[IP] = 10. $\overline{IRQ3}$ is programmed to level 6. If all three assert an interrupt request at the same time, they are serviced in the following order:

1. ICR5[IL] = 110 and ICR5[IP] = 10, so UART1 is serviced first (priority 7 in Table 9-4).
2. External interrupt IRQ3, set to level 6, is serviced next (priority 8).
3. ICR4[IL] = 110 and ICR5[IP] = 01, so UART0 is serviced last (priority 9).

**Table 9-4. Interrupt Priority Scheme**

| Priority | Interrupt Level | ICR | | Interrupt Source | IRQPAR[IRQPAR] |
|---|---|---|---|---|---|
| | | IL | IP | | |
| 1 | 7 | 111 | 11 | Internal module | xxx |
| 2 | | 111 | 10 | | xxx |
| 3 | | xxx | xx | External interrupt pin IRQ7 | xxx |
| 4 | | 111 | 01 | Internal module | xxx |
| 5 | | 111 | 00 | | xxx |
| 6 | 6 | 110 | 11 | Internal module | xxx |
| 7 | | 110 | 10 | | xxx |
| 8 | | xxx | xx | External interrupt pin IRQ3 (programmed as IRQ6) | x1x |
| 9 | | 110 | 01 | Internal module | xxx |
| 10 | | 110 | 00 | | xxx |
| 11 | 5 | 101 | 11 | Internal module | xxx |
| 12 | | 101 | 10 | | xxx |
| 13 | | xxx | xx | External interrupt pin IRQ5 | 0xx |
| 14 | | 101 | 01 | Internal module | xxx |
| 15 | | 101 | 00 | | xxx |

**Table 9-4. Interrupt Priority Scheme (Continued)**

| Priority | Interrupt Level | ICR | | Interrupt Source | IRQPAR[IRQPAR] |
|---|---|---|---|---|---|
| | | IL | IP | | |
| 16 | 4 | 100 | 11 | Internal module | xxx |
| 17 | | 100 | 10 | | xxx |
| 18 | | xxx | xx | External interrupt pin IRQ5 (programmed as IRQ4) | 1xx |
| 19 | | 100 | 01 | Internal module | xxx |
| 20 | | 100 | 00 | | xxx |
| 21 | 3 | 011 | 11 | Internal module | xxx |
| 22 | | 011 | 10 | | xxx |
| 23 | | xxx | xx | External interrupt pin IRQ3 | x0x |
| 24 | | 011 | 01 | Internal module | xxx |
| 25 | | 011 | 00 | | xxx |
| 26 | 2 | 010 | 11 | Internal module | xxx |
| 27 | | 010 | 10 | | xxx |
| 28 | | xxx | xx | External interrupt pin IRQ1 (programmed as IRQ2) | xx1 |
| 29 | | 010 | 01 | Internal module | xxx |
| 30 | | 010 | 00 | | xxx |
| 31 | 1 | 001 | 11 | Internal module | xxx |
| 32 | | 001 | 10 | | xxx |
| 33 | | xxx | xx | External interrupt pin IRQ1 | xx0 |
| 34 | | 001 | 01 | Internal module | xxx |
| 35 | | 001 | 00 | | xxx |

## 9.2.2 Autovector Register (AVR)

The autovector register (AVR), shown in Figure 9-3, enables external interrupt sources to be autovectored, using the vector offset defined in Table 2-19 in Section 2.8, “Exception Processing Overview.” Note that the autovector enable for internal interrupt sources applies for respective ICRs.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field | AVEC | | | | | | | BLK |
| Reset | 0000_0000 | | | | | | | |
| R/W | R/W | | | | | | | |
| Address | MBAR + 0x04B | | | | | | | |

**Figure 9-3. Autovector Register (AVR)**

Table 9-5 describes AVR fields.

**Table 9-5. AVR Field Descriptions**

| Bit | Name | Description |
|---|---|---|
| 7–1 | AVEC | Autovector control. Determines whether the external interrupt at that level is autovectored.<br>0 Interrupting source returns vector during interrupt-acknowledge cycle.<br>1 SIM generates autovector during interrupt-acknowledge cycle. |
| 0 | BLK | Block address strobe ($\overline{AS}$) for external AVEC access. Available for users who use $\overline{AS}$ as a global chip select for peripherals and do not want to enable them during an AVEC cycle.<br>0 Do not block address strobe.<br>1 Block address strobe from asserting. |

Table 9-6 shows the correlation between AVR[AVEC] and the external interrupts. Note that an AVEC*n* bit is valid only when the corresponding external interrupt request level is enabled in the IRQPAR.

**Table 9-6. Autovector Register Bit Assignments**

| Autovector Interrupt Source | Autovector Register Bit Location | Vector Offset |
|---|---|---|
| External interrupt request 1 | AVEC1 | 0x64 |
| External interrupt request 2 | AVEC2 | 0x68 |
| External interrupt request 3 | AVEC3 | 0x6C |
| External interrupt request 4 | AVEC4 | 0x70 |
| External interrupt request 5 | AVEC5 | 0x74 |
| External interrupt request 6 | AVEC6 | 0x78 |
| External interrupt request 7 | AVEC7 | 0x7C |

## 9.2.3 Interrupt Pending and Mask Registers (IPR and IMR)

The interrupt pending register (IPR), Figure 9-4, makes visible the interrupt sources that have an interrupt pending. The interrupt mask register (IMR), also shown in Figure 9-4, is used to mask the internal and external interrupt sources.

**NOTE:**

To mask interrupt sources, first set the core's status register interrupt mask level to that of the source being masked in the IMR. Then, the IMR bit can be masked.

An interrupt is masked by setting, and enabled by clearing, the corresponding IMR bit. When a masked interrupt occurs, the corresponding IPR bit is still set, but no interrupt request is passed to the core.

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | — | | | | | | | | | | | | | | DMA3 | DMA2 |
| Reset | — | | | | | | | | | | | | | | 1 | 1 |
| R/W | Read-only (IPR); R/W (IMR) | | | | | | | | | | | | | | | |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | DMA1 | DMA0 | UART1 | UART0 | I2C | TIMER2 | TIMER1 | SWT | EINT7 | EINT6 | EINT5 | EINT4 | EINT3 | EINT2 | EINT1 | — |
| Reset | 1111 | | | | 1111 | | | | 1111 | | | | 1 | 1 | 1 | — |
| R/W | Read-only (IPR); R/W (IMR) | | | | | | | | | | | | | | | |
| Addr | MBAR + 0x040 (IPR); + 0x044 (IMR) | | | | | | | | | | | | | | | |

**Figure 9-4. Interrupt Pending Register (IPR) and Interrupt Mask Register (IMR)**

Table 9-7 describes IPR and IMR fields.

**Table 9-7. IPR and IMR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–18 | — | Reserved, should be cleared. |
| 17–1 | See Figure 9-4 | Interrupt pending/mask. Each bit corresponds to an interrupt source defined by the ICR. The corresponding IMR bit determines whether an interrupt condition can generate an interrupt. At every clock, the IPR samples the signal generated by the interrupting source. The corresponding IPR bit reflects the state of the interrupt signal even if the corresponding IMR bit is set.<br>0 The corresponding interrupt source is not masked (IMR) and has no interrupt pending (IPR).<br>1 The corresponding interrupt source is masked (IMR) and has an interrupt pending (IPR) |

## 9.2.4 Interrupt Port Assignment Register (IRQPAR)

The interrupt port assignment register (IRQPAR), shown in Figure 9-5, provides the level assignment of the primary external interrupt pins—IRQ5, IRQ3, and IRQ1. The setting of IRQPAR2–IRQPAR0 determines the interrupt level of these external interrupt pins.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field | IRQPAR2 | IRQPAR1 | IRQPAR0 | — | | | ENBDACK1 | ENBDACK0 |
| Reset | 0000_0000 | | | | | | | |
| R/W | R/W | | | | | | | |
| Address | MBAR + 0x06 | | | | | | | |

**Figure 9-5. Interrupt Port Assignment Register (IRQPAR)**

Table 9-8 describes IRQPAR fields.

**Table 9-8. IRQPAR Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–5 | IRQPARn | Configures the IRQ pin assignments and priorities<br>IRQPARn External Pin IRQPARn = 0 IRQPARn = 1<br>IRQPAR2 IRQ5 Level 5 Level 4<br>IRQPAR1 IRQ3 Level 3 Level 6<br>IRQPAR0 IRQ1 Level 1 Level 2 |
| 4–2 | — | Reserved, should be cleared. |
| 1–0 | ENBDACKn | Enable $\overline{\text{DACK1}}$ and $\overline{\text{DACK0}}$. Determines the functionality of the respective TMn/$\overline{\text{DACK}}$n pins.<br>0 TM1 and TM0 are driven instead of $\overline{\text{DACK1}}$ and $\overline{\text{DACK0}}$.<br>1 If the PAR register is programmed to enable TMn, the $\overline{\text{DACK}}$n signal for DMA channel n is driven in place of TMn for DMA transfers. |

# Chapter 10
# Chip-Select Module

This chapter describes the MCF5407 chip-select module, including the operation and programming model of the chip-select registers, which include the chip-select address, mask, and control registers.

## 10.1 Overview

The following list summarizes the key chip-select features:

- Eight independent, user-programmable chip-select signals ($\overline{CS}$[7:0]) that can interface with SRAM, PROM, EPROM, EEPROM, Flash, and peripherals
- Address masking for 64-Kbyte to 4-Gbyte memory block sizes
- Programmable wait states and port sizes
- External master access to chip selects

## 10.2 Chip-Select Module Signals

Table 10-1 lists signals used by the chip-select module.

**Table 10-1. Chip-Select Module Signals**

| Signal | Description |
|---|---|
| Chip Selects ($\overline{CS}$[7:0]) | Each $\overline{CSn}$ can be independently programmed for an address location as well as for masking, port size, read/write burst-capability, wait-state generation, and internal/external termination. Only $\overline{CS0}$ is initialized at reset when it acts as a global chip select that allows boot ROM to be at any defined address space. Port size and termination (internal versus external) and byte enables for $\overline{CS0}$ are configured by the logic levels of D[7:5] when $\overline{RSTI}$ negates. |
| Output Enable ($\overline{OE}$) | Interfaces to memory or to peripheral devices and enables a read transfer. It is asserted and negated on the falling edge of the clock. $\overline{OE}$ is asserted only when one of the chip selects matches for the current address decode. |
| Byte Enables/ Byte Write Enables ($\overline{BE}$[3:0]/ $\overline{BWE}$[3:0]) | These multiplexed signals are individually programmed through the byte enable mode bit, CSCRn[BEM], described in Section 10.4.1.3, “Chip-Select Control Registers (CSCR0–CSCR7).” These generated signals provide byte data select signals, which are decoded from the transfer size, A1, and A0 signals in addition to the programmed port size and burstability of the memory accessed, as Table 10-2 shows. |

Table 10-2 shows the interaction of the byte enable/byte-write enables with related signals.

**Table 10-2. Byte Enables/Byte Write Enable Signal Settings**

| Transfer Size | Port Size | A1 | A0 | $\overline{\text{BE0/BWE0}}$ | $\overline{\text{BE1/BWE1}}$ | $\overline{\text{BE2/BWE2}}$ | $\overline{\text{BE3/BWE3}}$ |
|---|---|---|---|---|---|---|---|
| | | | | D[31:24] | D[23:16] | D[15:8] | D[7:0] |
| Byte | 8-bit | 0 | 0 | 0 | 1 | 1 | 1 |
| | | 0 | 1 | 0 | 1 | 1 | 1 |
| | | 1 | 0 | 0 | 1 | 1 | 1 |
| | | 1 | 1 | 0 | 1 | 1 | 1 |
| | 16-bit | 0 | 0 | 0 | 1 | 1 | 1 |
| | | 0 | 1 | 1 | 0 | 1 | 1 |
| | | 1 | 0 | 0 | 1 | 1 | 1 |
| | | 1 | 1 | 1 | 0 | 1 | 1 |
| | 32-bit | 0 | 0 | 0 | 1 | 1 | 1 |
| | | 0 | 1 | 1 | 0 | 1 | 1 |
| | | 1 | 0 | 1 | 1 | 0 | 1 |
| | | 1 | 1 | 1 | 1 | 1 | 0 |
| Word | 8-bit | 0 | 0 | 0 | 1 | 1 | 1 |
| | | 0 | 1 | 0 | 1 | 1 | 1 |
| | | 1 | 0 | 0 | 1 | 1 | 1 |
| | | 1 | 1 | 0 | 1 | 1 | 1 |
| | 16-bit | 0 | 0 | 0 | 0 | 1 | 1 |
| | | 1 | 0 | 0 | 0 | 1 | 1 |
| | 32-bit | 0 | 0 | 0 | 0 | 1 | 1 |
| | | 1 | 0 | 1 | 1 | 0 | 0 |
| Longword | 8-bit | 0 | 0 | 0 | 1 | 1 | 1 |
| | | 0 | 1 | 0 | 1 | 1 | 1 |
| | | 1 | 0 | 0 | 1 | 1 | 1 |
| | | 1 | 1 | 0 | 1 | 1 | 1 |
| | 16-bit | 0 | 0 | 0 | 0 | 1 | 1 |
| | | 1 | 0 | 0 | 0 | 1 | 1 |
| | 32-bit | 0 | 0 | 0 | 0 | 0 | 0 |
| Line | 8-bit | 0 | 0 | 0 | 1 | 1 | 1 |
| | | 0 | 1 | 0 | 1 | 1 | 1 |
| | | 1 | 0 | 0 | 1 | 1 | 1 |
| | | 1 | 1 | 0 | 1 | 1 | 1 |
| | 16-bit | 0 | 0 | 0 | 0 | 1 | 1 |
| | | 1 | 0 | 0 | 0 | 1 | 1 |
| | 32-bit | 0 | 0 | 0 | 0 | 0 | 0 |

## 10.3 Chip-Select Operation

Each chip select has a dedicated set of the following registers for configuration and control.

- Chip-select address registers (CSAR*n*) control the base address space of the chip select. See Section 10.4.1.1, “Chip-Select Address Registers (CSAR0–CSAR7).”

- Chip-select mask registers (CSMR*n*) provide 16-bit address masking and access control. See Section 10.4.1.2, “Chip-Select Mask Registers (CSMR0–CSMR7).”
- Chip-select control registers (CSCR*n*) provide port size and burst capability indication, wait-state generation, and automatic acknowledge generation features. See Section 10.4.1.3, “Chip-Select Control Registers (CSCR0–CSCR7).”

Each $\overline{CS}n$ can assert during specific CPU space accesses such as interrupt-acknowledge cycles and each can be accessed by an external master. $\overline{CS0}$ is a global chip select after reset and provides relocatable boot ROM capability.

## 10.3.1 General Chip-Select Operation

When a bus cycle is initiated, the MCF5407 first compares its address with the base address and mask configurations programmed for chip selects 0–7 (configured in CSCR0–CSCR7) and DRAM block 0 and 1 address and control registers (configured in DACR0 and DACR1). If the driven address matches a programmed chip select or DRAM block, the appropriate chip select is asserted or the DRAM block is selected using the specifications programmed in the respective configuration register. Otherwise, the following occurs:

- If the address and attributes do not match in CSCR or DACR, the MCF5407 runs an external burst-inhibited bus cycle with a default of external termination on a 32-bit port.
- Should an address and attribute match in multiple CSCRs, the matching chip-select signals are driven; however, the MCF5407 runs an external burst-inhibited bus cycle with external termination on a 32-bit port.
- Should an address and attribute match both DACRs or a DACR and a CSCR, the operation is undefined.

Table 10-3 shows the type of access as a function of match in the CSCRs and DACRs.

**Table 10-3. Accesses by Matches in CSCRs and DACRs**

| Number of CSCR Matches | Number of DACR Matches | Type of Access |
|---|---|---|
| 0 | 0 | External |
| 1 | 0 | Defined by CSCR |
| Multiple | 0 | External, burst-inhibited, 32-bit |
| 0 | 1 | Defined by DACRs |
| 1 | 1 | Undefined |
| Multiple | 1 | Undefined |
| 0 | Multiple | Undefined |
| 1 | Multiple | Undefined |
| Multiple | Multiple | Undefined |

### 10.3.1.1 8-, 16-, and 32-Bit Port Sizing

Static bus sizing is programmable through the port size bits, CSCR[PS]. See Section 10.4.1.3, “Chip-Select Control Registers (CSCR0–CSCR7).” Figure 10-1 shows the correspondence between data byte lanes and the external chip-select memory. Note that all lanes are driven, although unused lines are undefined.

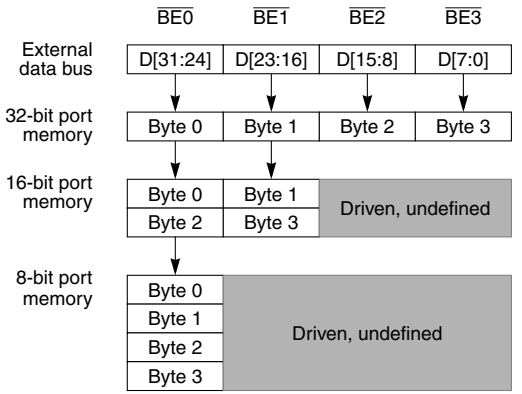



**Figure 10-1. Connections for External Memory Port Sizes**

### 10.3.1.2 Global Chip-Select Operation

$\overline{CS0}$, 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, $\overline{CS0}$ is asserted for every external access. No other chip-select can be used until the valid bit, CSMR0[V], is set, at which point $\overline{CS0}$ functions as configured and $\overline{CS}$[7:1] can be used. At reset, the port size, byte enable, and automatic acknowledge functions of the global chip-select are determined by the logic levels of the inputs on D[7:5,3]. Table 10-4 through Table 10-6 list the various reset encodings for the configuration signals multiplexed with D[7:5,3].

**Table 10-4. D7/AA, Automatic Acknowledge of Boot $\overline{CS0}$**

| D7/AA | Boot $\overline{CS0}$ AA Configuration at Reset |
|---|---|
| 0 | Disabled |
| 1 | Enable with 15 wait states |

**Table 10-5. D[6:5]/PS[1:0], Port Size of Boot $\overline{CS0}$**

| D[6:5]/PS[1:0] | Boot $\overline{CS0}$ Port Size at Reset |
|---|---|
| 00 | 32-bit port |
| 01 | 8-bit port |
| 1x | 16-bit port |

**Table 10-6. D3/BE_CONFIG0, $\overline{BE}$[3:0] Boot Configuration**

| D3/BE_CONFIG0 | Configuration of Byte Enables for Boot $\overline{CS0}$ |
|---|---|
| 0 | $\overline{BE}$[3:0] is enabled as byte write enables only. |
| 1 | $\overline{BE}$[3:0] is enabled as byte enables for reads and writes. |

Provided the required address range is in the chip-select address register (CSAR0), $\overline{CS0}$ can be programmed to continue decoding for a range of addresses after the CSMR0[V] is set, after which the global chip-select can be restored only by a system reset.

## 10.4 Chip-Select Registers

Table 10-7 is the chip-select register memory map. Reading reserved locations returns zeros.

**Table 10-7. Chip-Select Registers**

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| 0x080 | Chip-select address register—bank 0 (CSAR0) [p. 10-6] | | Reserved[1] | |
| 0x084 | Chip-select mask register—bank 0 (CSMR0) [p. 10-7] | | | |
| 0x088 | Reserved[1] | | Chip-select control register—bank 0 (CSCR0) [p. 10-8] | |
| 0x08C | Chip-select address register—bank 1 (CSAR1) [p. 10-6] | | Reserved[1] | |
| 0x090 | Chip-select mask register—bank 1 (CSMR1) [p. 10-7] | | | |
| 0x094 | Reserved[1] | | Chip-select control register—bank 1 (CSCR1) [p. 10-8] | |
| 0x098 | Chip-select address register—bank 2 (CSAR2) [p. 10-6] | | Reserved[1] | |
| 0x09C | Chip-select mask register—bank 2 (CSMR2) [p. 10-7] | | | |
| 0x0A0 | Reserved[1] | | Chip-select control register—bank 2 (CSCR2) [p. 10-8] | |
| 0x0A4 | Chip-select address register—bank 3 (CSAR3) [p. 10-6] | | Reserved[1] | |
| 0x0A8 | Chip-select mask register—bank 3 (CSMR3) [p. 10-7] | | | |
| 0x0AC | Reserved[1] | | Chip-select control register—bank 3 (CSCR3) [p. 10-8] | |
| 0x0B0 | Chip-select address register—bank 4 (CSAR4) [p. 10-6] | | Reserved[1] | |
| 0x0B4 | Chip-select mask register—bank 4 (CSMR4) [p. 10-7] | | | |

**Table 10-7. Chip-Select Registers (Continued)**

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| 0x0B8 | Reserved[1] | | Chip-select control register—bank 4 (CSCR4) [p. 10-8] | |
| 0x0BC | Chip-select address register—bank 5 (CSAR5) [p. 10-6] | | Reserved[1] | |
| 0x0C0 | Chip-select mask register—bank 5 (CSMR5) [p. 10-7] | | | |
| 0x0C4 | Reserved | | Chip-select control register—bank 5 (CSCR5) [p. 10-8] | |
| 0x0C8 | Chip-select address register—bank 6 (CSAR6) [p. 10-6] | | Reserved[1] | |
| 0x0CC | Chip-select mask register—bank 6 (CSMR6) [p. 10-7] | | | |
| 0x0D0 | Reserved[1] | | Chip-select control register—bank 6 (CSCR6) [p. 10-8] | |
| 0x0D4 | Chip-select address register—bank 7 (CSAR7) [p. 10-6] | | Reserved[1] | |
| 0x0D8 | Chip-select mask register—bank 7 (CSMR7) [p. 10-7] | | | |
| 0x0DC | Reserved[1] | | Chip-select control register—bank 7 (CSCR7) [p. 10-8] | |

[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.

**NOTE:**

External masters cannot access MCF5407 on-chip memories or MBAR, but can access any of the chip-select module registers.

## 10.4.1 Chip-Select Module Registers

The chip-select module is programmed through the chip select address registers (CSAR0–CSAR7), chip select mask registers (CSMR0–CSMR7), and the chip select control registers (CSCR0–CSCR7).

### 10.4.1.1 Chip-Select Address Registers (CSAR0–CSAR7)

Chip select address registers, Figure 10-2, specify the chip select base addresses.

| | 15 … 0 |
|---|---|
| Field | BA |
| Reset | Uninitialized |
| R/W | R/W |
| Addr | 0x080 (CSAR0); 0x08C (CSAR1); 0x098 (CSAR2); 0x0A4 (CSAR3); 0x0B0 (CSAR4); 0x0BC (CSAR5); 0x0C8 (CSAR6); 0x0D4 (CSAR7) |

**Figure 10-2. Chip Select Address Registers (CSAR0–CSAR7)**

Table 10-8 describes CSAR[BA].

**Table 10-8. CSARn Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–0 | BA | Base address. Defines the base address for memory dedicated to chip select $\overline{CS}$[7:0]. BA is compared to bits 31–16 on the internal address bus to determine if chip-select memory is being accessed. |

### 10.4.1.2 Chip-Select Mask Registers (CSMR0–CSMR7)

The chip select mask registers, Figure 10-3, are used to specify the address mask and allowable access types for the respective chip selects.

| | 31 16 | 15 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | BAM | — | WP | — | AM | C/I | SC | SD | UC | UD | V |
| Reset | Unitialized | | | | | | | | | | 0 |
| R/W | R/W | | | | | | | | | | |
| Addr | 0x084 (CSMR0); 0x090 (CSMR1); 0x09C (CSMR2); 0x0A8 (CSMR3); 0x0B4 (CSMR4); 0x0C0 (CSMR5); 0x0CC (CSMR6); 0x0D8 (CSMR7) | | | | | | | | | | |

**Figure 10-3. Chip Select Mask Registers (CSMRn)**

Table 10-9 describes CSMR fields.

**Table 10-9. CSMRn Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–16 | BAM | Base address mask. Defines the chip select block by masking address bits. Setting a BAM bit causes the corresponding CSAR bit to be ignored in the decode.<br>0 Corresponding address bit is used in chip-select decode.<br>1 Corresponding address bit is a don't care in chip-select decode.<br>The block size for $\overline{CS}$[7:0] is $2^n$; n = (number of bits set in respective CSMR[BAM]) + 16.<br>So, if CSAR0 = 0x0000 and CSMR0[BAM] = 0x0008, $\overline{CS0}$ would address two discontinuous 64-Kbyte memory blocks: one from 0x0000–0xFFFF and one from 0x8_0000–0x8_FFFF.<br>Likewise, for $\overline{CS0}$ to access 32 Mbytes of address space starting at location 0x0, $\overline{CS1}$ must begin at the next byte after $\overline{CS0}$ for a 16-Mbyte address space. Then CSAR0 = 0x0000, CSMR0[BAM] = 0x01FF, CSAR1 = 0x0200, and CSMR1[BAM] = 0x00FF. |
| 8 | WP | Write protect. Controls write accesses to the address range in the corresponding CSAR. Attempting to write to the range of addresses for which CSARn[WP] = 1 results in the appropriate chip select not being selected. No exception occurs.<br>0 Both read and write accesses are allowed.<br>1 Only read accesses are allowed. |
| 7 | — | Reserved, should be cleared. |
| 6 | AM | Alternate master. When AM = 0 during an external master or DMA access, SC, SD, UC, and UD are don't cares in the chip-select decode. |

**Table 10-9. CSMRn Field Descriptions (Continued)**

| Bits | Name | Description |
|---|---|---|
| 5–1 | C/I, SC, SD, UC, UD | Address space mask bits. These bits determine whether the specified accesses can occur to the address space defined by the BAM for this chip select.<br><br>C/I CPU space and interrupt acknowledge cycle mask<br>SC Supervisor code address space mask<br>SD Supervisor data address space mask<br>UC User code address space mask<br>UD User data address space mask<br><br>0 The address space assigned to this chip select. is available to the specified access type.<br>1 The address space assigned to this chip select. is not available (masked) to the specified access type. If this address space is accessed, chip select is not activated and a regular external bus cycle occurs.<br>Note that if if AM = 0, SC, SD, UC, and UD are ignored in the chip select decode on external master or DMA access. |
| 0 | V | Valid bit. Indicates whether the corresponding CSAR, CSMR, and CSCR contents are valid. Programmed chip selects do not assert until V is set (except for $\overline{CS0}$, which acts as the global chip select). Reset clears each CSMRn[V].<br>0 Chip select invalid<br>1 Chip select valid |

### 10.4.1.3 Chip-Select Control Registers (CSCR0–CSCR7)

Each chip-select control register, Figure 10-4, controls the auto acknowledge, external master support, port size, burst capability, and activation of each chip select. Note that to support the global chip select, $\overline{CS0}$, the CSCR0 reset values differ from the other CSCRs. $\overline{CS0}$ allows address decoding for boot ROM before system initialization.

| | 15 | 14 | 13 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | — | | WS | | — | AA | PS1 | PS0 | BEM | BSTR | BSTW | — | |
| Reset: CSCR0 | — | | 11_11 | | — | D7 | D6 | D5 | D3 | — | | | |
| Reset: Other CSCRs | Unitialized | | | | | | | | | | | | |
| R/W | R/W | | | | | | | | | | | | |
| Address | 0x08A (CSCR0); 0x096 (CSCR1); 0x0A2 (CSCR2); 0x0AE (CSCR3); 0x0BA (CSCR4); 0x0C6 (CSCR5); 0x0D2 (CSCR6); 0x0DE (CSCR7) | | | | | | | | | | | | |

**Figure 10-4. Chip-Select Control Registers (CSCR0–CSCR7)**

Table 10-10 describes CSCR*n* fields.

**Table 10-10. CSCRn Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 15–14 | — | Reserved, should be cleared. |
| 13–10 | WS | Wait states. The number of wait states inserted before an internal transfer acknowledge is generated (WS = 0 inserts zero wait states, WS = 0xF inserts 15 wait states). If AA = 0, $\overline{TA}$ 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 $\overline{TA}$ supersedes the generation of an internal $\overline{TA}$. |
| 9 | — | Reserved, should be cleared. |

**Table 10-10. CSCRn Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 8 | AA | Auto-acknowledge enable. Determines the assertion of the internal transfer acknowledge for accesses specified by the chip-select address.<br>0 No internal $\overline{TA}$ is asserted. Cycle is terminated externally.<br>1 Internal $\overline{TA}$ is asserted as specified by WS. Note that if AA = 1 for a corresponding $\overline{CS}$n and the external system asserts an external $\overline{TA}$ before the wait-state countdown asserts the internal $\overline{TA}$, the cycle is terminated. Burst cycles increment the address bus between each internal termination. |
| 7–6 | PS | Port size. Specifies the width of the data associated with each chip select. It determines where data is driven during write cycles and where data is sampled during read cycles. See Section 10.3.1.1, “8-, 16-, and 32-Bit Port Sizing.”<br>00 32-bit port size. Valid data sampled and driven on D[31:0]<br>01 8-bit port size. Valid data sampled and driven on D[31:24]<br>1x 16-bit port size. Valid data sampled and driven on D[31:16] |
| 5 | BEM | 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 in support of these SRAMs.<br>0 Neither $\overline{BE}$ nor $\overline{BWE}$ is asserted for read. $\overline{BWE}$ is generated for data write only.<br>1 $\overline{BE}$ is asserted for read; $\overline{BWE}$ is asserted for write. |
| 4 | BSTR | Burst read enable. Specifies whether burst reads are used for memory associated with each $\overline{CS}$n.<br>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.<br>1 Enables data burst reads larger than the specified port size, including longword reads from 8- and 16-bit ports, word reads from 8-bit ports, and line reads from 8-, 16-, and 32-bit ports. |
| 3 | BSTW | Burst write enable. Specifies whether burst writes are used for memory associated with each $\overline{CS}$n.<br>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.<br>1 Enables burst write of data larger than the specified port size, including longword writes to 8 and 16-bit ports, word writes to 8-bit ports and line writes to 8-, 16-, and 32-bit ports. |
| 2–0 | — | Reserved, should be cleared. |

### 10.4.1.4 Code Example

The code below provides an example of how to initialize the chip-selects. Only chip selects 0, 1, 2, and 3 are programmed here; chip selects 4, 5, 6, and 7 are left invalid. MBARx defines the base of the module address space.

```
CSAR0 EQU MBARx+0x080   ;Chip select 0 address register
CSMR0 EQU MBARx+0x084   ;Chip select 0 mask register
CSCR0 EQU MBARx+0x08A   ;Chip select 0 control register

CSAR1 EQU MBARx+0x08C   ;Chip select 1 address register
CSMR1 EQU MBARx+0x090   ;Chip select 1 mask register
CSCR1 EQU MBARx+0x096   ;Chip select 1 control register

CSAR2 EQU MBARx+0x098   ;Chip select 2 address register
CSMR2 EQU MBARx+0x09C   ;Chip select 2 mask register
CSCR2 EQU MBARx+0x0A2   ;Chip select 2 control register

CSAR3 EQU MBARx+0x0A4   ;Chip select 3 address register
CSMR3 EQU MBARx+0x0A8   ;Chip select 3 mask register
CSCR3 EQU MBARx+0x0AE   ;Chip select 3 control register

CSAR4 EQU MBARx+0x0B0   ;Chip select 4 address register
CSAR4 EQU MBARx+0x0B4   ;Chip select 4 mask register
CSMR4 EQU MBARx+0x0BA   ;Chip select 4 control register
```

```
CSAR5 EQU MBARx+0x0BC   ;Chip select 5 address register
CSMR5 EQU MBARx+0x0C0   ;Chip select 5 mask register
CSCR5 EQU MBARx+0x0C6   ;Chip select 5 control register

CSAR6 EQU MBARx+0x0C8   ;Chip select 6 address register
CSMR6 EQU MBARx+0x0CC   ;Chip select 6 mask register
CSCR6 EQU MBARx+0x0D2   ;Chip select 6 control register

CSAR7 EQU MBARx+0x0D4   ;Chip select 7 address register
CSMR7 EQU MBARx+0x0D8   ;Chip select 7 mask register
CSCR7 EQU MBARx+0x0DE   ;Chip select 7 control register

; All other chip selects should be programmed and made valid before global
; chip select is de-activated by validating CS0

; Program Chip Select 3 Registers
move.w  #0x0040,D0      ;CSAR3 base address 0x00400000
move.w  D0,CSAR3

move.w  #0x00A0,D0      ;CSCR3 = no wait states, AA=0, PS=16-bit, BEM=1,
move.w  D0,CSCR3        ;BSTR=0, BSTW=0

move.l #0x001F016B,D0   ;Address range from 0x00400000 to 0x005FFFFF
move.l D0,CSMR3         ;WP,EM,C/I,SD,UD,V=1; SC,UC=0

; Program Chip Select 2 Registers

move.w  #0x0020,D0      ;CSAR2 base address 0x00200000 (to 0x003FFFFF)
move.w  D0,CSAR2

move.w  #0x0538,D0      ;CSCR2 = 1 wait state, AA=1, PS=32-bit, BEM=1,
move.w  D0,CSCR2        ;BSTR=1, BSTW=1

move.l  #0x001F0001,D0  ;Address range from 0x00200000 to 0x003FFFFF
move.l  D0,CSMR2        ;WP,EM,C/I,SC,SD,UC,UD=0; V=1

; Program Chip Select 1 Registers

move.w  #0x0000,D0      ;CSAR1 base addresses 0x00000000 (to 0x001FFFFF)
move.w  D0,CSAR1        ;and 0x80000000 (to 0x801FFFFF)

move.w  #0x09B0,D0      ;CSCR1 = 2 wait states, AA=1, PS=16-bit, BEM=1,
move.w  D0,CSCR1        ;BSTR=1, BSTW=0

move.l  #0x801F0001,D0  ;Address range from 0x00000000 to 0x001FFFFF and
move.l  D0,CSMR1        ;0x80000000 to 0x801FFFFF
                        ;WP, EM, C/I, SC, SD, UC, UD=0, V=1

; Program Chip Select 0 Registers

move.w  #0x0080,D0      ;CSAR0 base address 0x00800000 (to 0x009FFFFF)
move.w  D0,CSAR0

move.w  #0x0D80,D0      ;CSCR0 = three wait states, AA=1, PS=16-bit, BEM=0,
move.w  D0,CSCR0        ;BSTR=0, BSTW=0

; Program Chip Select 0 Mask Register (validate chip selects)

move.l  #0x001F0001,D0  ;Address range from 0x00800000 to 0x009FFFFF
move.l  D0,CSMR0        ;WP,EM,C/I,SC,SD,UC,UD=0; V=1
```

# Chapter 11
# Synchronous/Asynchronous DRAM Controller Module

This chapter describes configuration and operation of the synchronous/asynchronous DRAM controller component of the system integration module (SIM). It begins with a general description and brief glossary, and includes a description of signals involved in DRAM operations. The remainder of the chapter consists of the two following parts:

- Section 11.3, “Asynchronous Operation,” describes the programming model and signal timing for the four basic asynchronous modes.
  - Non-page mode
  - Burst page mode
  - Continuous page mode
  - Extended data-out mode
- Section 11.4, “Synchronous Operation,” describes the programming model and signal timing, as well as the command set required for synchronous operations. This section also includes extensive examples the designer can follow to better understand how to configure the DRAM controller for synchronous operations.

## 11.1 Overview

The DRAM controller module provides glueless integration of DRAM with the ColdFire product. The key features of the DRAM controller include the following:

- Support for two independent blocks of DRAM
- Interface to standard synchronous/asynchronous dynamic random access memory (ADRAM/SDRAM) components
- Programmable $\overline{\text{SRAS}}$, $\overline{\text{SCAS}}$, and refresh timing
- Support for page mode
- Support for 8-, 16-, and 32-bit wide DRAM blocks
- Support for synchronous and asynchronous DRAMs, including EDO DRAM, SDRAM, and fast page mode

### 11.1.1 Definitions

The following terminology is used in this chapter:

- A/SDRAM block—Any group of DRAM memories selected by one of the MCF5407 $\overline{\text{RAS}}$[1:0] signals. Thus, the MCF5407 can support two independent memory blocks. The base address of each block is programmed in the DRAM address and control registers (DACR0 and DACR1).
- SDRAM—RAMs that operate like asynchronous DRAMs but with a synchronous clock, a pipelined, multiple-bank architecture, and faster speed.
- 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 SDRAM component's bank select lines.

### 11.1.2 Block Diagram and Major Components

The basic components of the DRAM controller are shown in Figure 11-1.

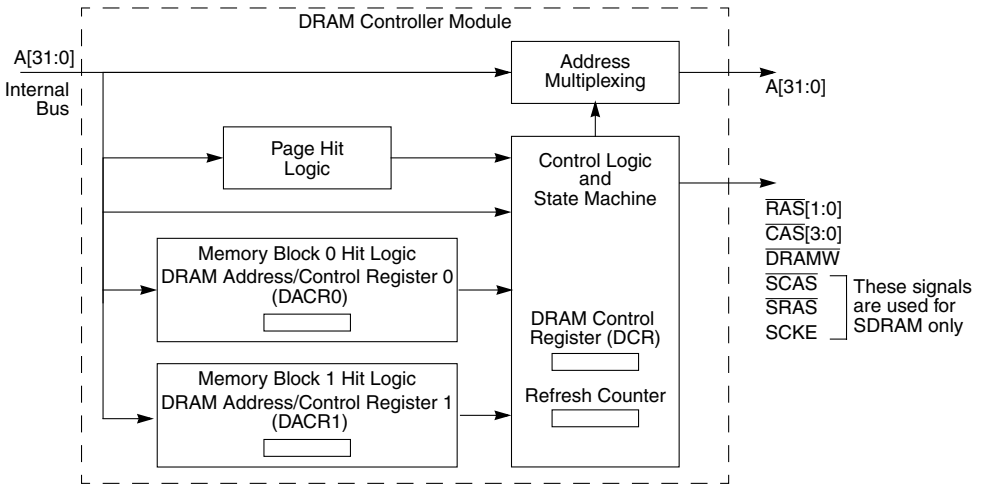



**Figure 11-1. Asynchronous/Synchronous DRAM Controller Block Diagram**

The DRAM controller's major components, shown in Figure 11-1, are described as follows:

- DRAM address and control registers (DACR0 and DACR1)—The DRAM controller consists of two configuration register units, one for each supported memory block. DACR0 is accessed at MBAR + 0x0108; DACR1 is accessed at 0x010. The register information is passed on to the hit logic.

- Control logic and state machine—Generates all DRAM signals, taking bus cycle characteristic data from the block logic, along with hit information to generate DRAM accesses. Handles refresh requests from the refresh counter.
  - DRAM control register (DCR)—Contains data to control refresh operation of the DRAM controller. Both memory blocks are refreshed concurrently as controlled by DCR[RC].
  - Refresh counter—Determines when refresh should occur, determined by the value of DCR[RC]. It generates a refresh request to the control block.
- Hit logic—Compares address and attribute signals of a current DRAM bus cycle to both DACRs to determine if a DRAM block is being accessed. Hits are passed to the control logic along with characteristics of the bus cycle to be generated.
- Page hit logic—Determines if the next DRAM access is in the same DRAM page as the previous one. This information is passed on to the control logic.
- Address multiplexing—Multiplexes addresses to allow column and row addresses to share pins. This allows glueless interface to DRAMs.

## 11.2 DRAM Controller Operation

The DRAM controller mode is programmed through DCR[SO]. Asynchronous mode (SO = 0) includes support for page mode and EDO DRAMs. Synchronous mode is designed to work with industry-standard SDRAMs. These modes act very differently from one another, especially regarding the use of DRAM registers and pins. Memory blocks cannot operate in different modes; both are either synchronous or asynchronous.

### 11.2.1 DRAM Controller Registers

The DRAM controller registers memory map, Table 11-1, is the same regardless of whether asynchronous or synchronous DRAM is used, although bit configurations may vary.

**Table 11-1. DRAM Controller Registers**

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| 0x100 | DRAM control register (DCR) [p. 11-4] | | Reserved | |
| 0x104 | Reserved | | | |
| 0x108 | DRAM address and control register 0 (DACR0) [p. 11-5] | | | |
| 0x10C | DRAM mask register block 0 (DMR0) [p. 11-7] | | | |
| 0x110 | DRAM address and control register 1 (DACR1) [p. 11-5] | | | |
| 0x114 | DRAM mask register block 1 (DMR1) [p. 11-7] | | | |

**NOTE:**

External masters cannot access MCF5407 on-chip memories or MBAR, but they can access DRAM controller registers.

## 11.3 Asynchronous Operation

The DRAM controller supports asynchronous DRAMs for cost-effective systems. Typical access times for the DRAM controller interfacing to ADRAM are 4-3-3-3. The DRAM controller supports the following four asynchronous modes:

- Non-page mode
- Burst page mode
- Continuous page mode
- Extended data-out mode

In asynchronous mode, $\overline{RAS}$ and $\overline{CAS}$ always transition at the falling clock edge. As summarized previously, operation and timing of each ADRAM block is controlled by separate registers, but refresh is the same for both. All ADRAM accesses should be terminated by the DRAM controller. There is no priority encoding between memory blocks, so programming blocks to overlap with other blocks or with other internal resources causes undefined behavior.

### 11.3.1 DRAM Controller Signals in Asynchronous Mode

Table 11-2 summarizes DRAM signals used in asynchronous mode.

**Table 11-2. SDRAM Signal Summary**

| Signal | Description |
|---|---|
| $\overline{RAS}$[1:0] | Row address strobes. Interface to $\overline{RAS}$ inputs on industry-standard ADRAMs. When SDRAMs are used, these signals interface to the chip-select lines within an SDRAM's memory block. Thus, there is one $\overline{RAS}$ line for each of the two blocks. |
| $\overline{CAS}$[3:0] | Column address strobes. Interface to $\overline{CAS}$ inputs on industry-standard DRAMs. These provide $\overline{CAS}$ for a given ADRAM block. When SDRAMs are used, $\overline{CAS}$[3:0] control the byte enables (DQMx) for standard SDRAMs. $\overline{CAS}$[3:0] strobes data in least-to-most significant byte order ($\overline{CAS0}$ is MSB). |
| $\overline{DRAMW}$ | DRAM read/write. Asserted when a DRAM write cycle is underway. Negated for read bus cycles. |

### 11.3.2 Asynchronous Register Set

The following register configurations apply when DCR[SO] is 0, indicating the DRAM controller is interfacing to asynchronous DRAMs.

#### 11.3.2.1 DRAM Control Register (DCR) in Asynchronous Mode

The DCR provides programmable options for the refresh logic as well as the control bit to determine if the module is operating with synchronous or asynchronous DRAMs. The DCR is shown in Figure 11-2.

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 0 |
|---|---|---|---|---|---|---|---|---|---|
| Field | SO | — | NAM | RRA | | RRP | | RC | |
| Reset | 0 | Uninitialized | | | | | | | |
| R/W | R/W | | | | | | | | |
| Address | MBAR + 0x100 | | | | | | | | |

**Figure 11-2. DRAM Control Register (DCR) (Asynchronous Mode)**

Table 11-3 describes DCR fields.

**Table 11-3. DCR Field Descriptions (Asynchronous Mode)**

| Bits | Name | Description |
|---|---|---|
| 15 | SO | Synchronous operation. Selects synchronous or asynchronous mode. A DRAM controller in synchronous mode can be switched to ADRAM mode only by resetting the MCF5407.<br>0 Asynchronous DRAMs. Default at reset.<br>1 Synchronous DRAMs |
| 14 | — | Reserved, should be cleared. |
| 13 | NAM | No address multiplexing. Some implementations require external multiplexing. For example, when linear addressing is required, the DRAM should not multiplex addresses on DRAM accesses.<br>0 The DRAM controller multiplexes the external address bus to provide column addresses.<br>1 The DRAM controller does not multiplex the external address bus to provide column addresses. |
| 12–11 | RRA | Refresh $\overline{\text{RAS}}$ asserted. Determines how long $\overline{\text{RAS}}$ is asserted during a refresh operation.<br>00 2 clocks<br>01 3 clocks<br>10 4 clocks<br>11 5 clocks |
| 10–9 | RRP | Refresh $\overline{\text{RAS}}$ precharge. Controls how many clocks $\overline{\text{RAS}}$ is precharged after a refresh operation before accesses are allowed to DRAM.<br>00 1 clock<br>01 2 clocks<br>10 3 clocks<br>11 4 clocks |
| 8–0 | RC | Refresh count. Controls refresh frequency. The number of bus clocks between refresh cycles is (RC + 1) * 16. Refresh can range from 16–8192 bus clocks to accommodate both standard and low-power DRAMs with bus clock operation from less than 2 MHz to greater than 50 MHz.<br>The following example calculates RC for an auto-refresh period for 4096 rows to receive 64 mS of refresh every 15.625 µs for each row (625 bus clocks at 40 MHz).<br># of bus clocks = 625 = (RC field + 1) * 16<br>RC = (625 bus clocks/16) -1 = 38.06, which rounds to 38; therefore, RC = 0x26. |

### 11.3.2.2 DRAM Address and Control Registers (DACR0/DACR1)

DACR0 and DACR1, Figure 11-3, contain the base address compare value and the control bits for memory blocks 0 and 1. Address and timing are also controlled by these registers. Memory areas defined for each block should not overlap; operation is undefined for accesses in overlapping regions.

| | 31 18 | 17 16 | 15 | 14 | 13 12 | 11 10 | 9 | 8 | 7 | 6 | 5 4 | 3 2 | 1 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | BA | — | RE | — | CAS | RP | RNCN | RCD | — | EDO | PS | PM | — |
| Reset | Unitialized | | 0 | Unitialized | | | | | | | | | |
| R/W | R/W | | | | | | | | | | | | |
| Addr | MBAR + 0x10C (DACR0); 0x110 (DACR1) | | | | | | | | | | | | |

**Figure 11-3. DRAM Address and Control Registers (DACR0/DACR1)**

Table 11-4 describes DACR*n* fields.

**Table 11-4. DACR0/DACR1 Field Description**

| Bits | Name | Description |
|---|---|---|
| 31–18 | BA | Base address. Used with DMR[BAM] to determine the address range in which the associated DRAM block is located. Each BA bit is compared with the corresponding address of the bus cycle in progress. If each bit matches, or if bits that do not match are masked in the BAM, the address selects the associated DRAM block. |
| 17–16 | — | Reserved, should be cleared. |
| 15 | RE | Refresh enable. Determines whether the DRAM controller generates a refresh to the associated DRAM block. DRAM contents are not preserved during hard reset or software watchdog reset.<br>0 Do not refresh associated DRAM block. (Default at reset)<br>1 Refresh associated DRAM block. |
| 14 | — | Reserved, should be cleared. |
| 13–12 | CAS | $\overline{CAS}$ timing. Determines how long $\overline{CAS}$ is asserted during a DRAM access.<br>00 1 clock cycle<br>01 2 clock cycles<br>10 3 clock cycles<br>11 4 clock cycles |
| 11–10 | RP | $\overline{RAS}$ precharge timing. Determines how long $\overline{RAS}$ is precharged between accesses. Note that RP is different from DCR[RRP].<br>00 1 clock cycle<br>01 2 clock cycles<br>10 3 clock cycles<br>11 4 clock cycles |
| 9 | RNCN | $\overline{RAS}$-negate-to-$\overline{CAS}$-negate. Controls whether $\overline{RAS}$ and $\overline{CAS}$ negate concurrently or one clock apart. RNCN is ignored if $\overline{CAS}$ is asserted for only one clock and both $\overline{RAS}$ and $\overline{CAS}$ are negated. RNCN is used only for non-page-mode accesses and single accesses in page mode.<br>0 $\overline{RAS}$ negates concurrently with $\overline{CAS}$.<br>1 $\overline{RAS}$ negates one clock before $\overline{CAS}$. |
| 8 | RCD | $\overline{RAS}$-to-$\overline{CAS}$ delay. Determines the number of system clocks between assertions of $\overline{RAS}$ and $\overline{CAS}$.<br>0 1 clock cycle<br>1 2 clock cycles |
| 7 | — | Reserved, should be cleared. |
| 6 | EDO | Extended data out. Determines whether the DRAM block operates in a mode to take advantage of industry-standard EDO DRAMs. Do not use EDO mode with non-EDO DRAM.<br>0 EDO operation disabled.<br>1 EDO operation enabled. |

**Table 11-4. DACR0/DACR1 Field Description (Continued)**

| Bits | Name | Description |
|---|---|---|
| 5–4 | PS | Port size. Determines the port size of the associated DRAM block. For example, if two 16-bit wide DRAM components form one DRAM block, the port size is 32 bits. Programming PS allows the DRAM controller to execute dynamic bus sizing for associated accesses.<br>00 32-bit port<br>01 8-bit port<br>1x 16-bit port |
| 3–2 | PM | Page mode. Configures page-mode operation for the memory block.<br>00 No page mode<br>01 Burst page mode (page mode for bursts only)<br>10 Reserved<br>11 Continuous page mode |
| 1–0 | — | Reserved, should be cleared. |

### 11.3.2.3 DRAM Controller Mask Registers (DMR0/DMR1)

The DRAM controller mask registers (DMR0 and DMR1), shown in Figure 11-4, include mask bits for the base address and for address attributes.

| | 31 18 | 17 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | BAM | — | WP | — | C/I | AM | SC | SD | UC | UD | V |
| Reset | Uninitialized | | | | | | | | | | 0 |
| R/W | R/W | | | | | | | | | | |
| Addr | MBAR + 0x10C (DMR0), 0x114 (DMR1) | | | | | | | | | | |

**Figure 11-4. DRAM Controller Mask Registers (DMR0 and DMR1)**

Table 11-5 describes DMR*n* fields.

**Table 11-5. DMR0/DMR1 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31–18 | BAM | Base address mask. Masks the associated DACRn[BA]. Lets the DRAM controller connect to various DRAM sizes. Mask bits need not be contiguous (see Section 11.5, “SDRAM Example.”)<br>0 The associated address bit is used in decoding the DRAM hit to a memory block.<br>1 The associated address bit is not used in the DRAM hit decode. |
| 17–9 | — | Reserved, should be cleared. |
| 8 | WP | Write protect. Determines whether the associated block of DRAM is write protected.<br>0 Allow write accesses<br>1 Ignore write accesses. The DRAM controller ignores write accesses to the memory block and an address exception occurs. Write accesses to a write-protected DRAM region are compared in the chip select module for a hit. If no hit occurs, an external bus cycle is generated. If this external bus cycle is not acknowledged, an access exception occurs. |
| 7 | — | Reserved, should be cleared. |

**Table 11-5. DMR0/DMR1 Field Descriptions (Continued)**

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>6–1</td><td>AMx</td><td>Address modifier masks. Determine which accesses can occur in a given DRAM block.<br>0 Allow access type to hit in DRAM<br>1 Do not allow access type to hit in DRAM
<table>
<tr><th>Bit</th><th>Associated Access Type</th><th>Access Definition</th></tr>
<tr><td>C/I</td><td>CPU space/interrupt acknowledge</td><td>MOVEC instruction or interrupt acknowledge cycle</td></tr>
<tr><td>AM</td><td>Alternate master</td><td>External or DMA master</td></tr>
<tr><td>SC</td><td>Supervisor code</td><td>Any supervisor-only instruction access</td></tr>
<tr><td>SD</td><td>Supervisor data</td><td>Any data fetched during the instruction access</td></tr>
<tr><td>UC</td><td>User code</td><td>Any user instruction</td></tr>
<tr><td>UD</td><td>User data</td><td>Any user data</td></tr>
</table></td></tr>
<tr><td>0</td><td>V</td><td>Valid. Cleared at reset to ensure that the DRAM block is not erroneously decoded.<br>0 Do not decode DRAM accesses.<br>1 Registers controlling the DRAM block are initialized; DRAM accesses can be decoded.</td></tr>
</table>

## 11.3.3 General Asynchronous Operation Guidelines

The DRAM controller provides control for $\overline{\text{RAS}}$, $\overline{\text{CAS}}$, and $\overline{\text{DRAMW}}$ signals, as well as address multiplexing and bus cycle termination. Whether the mode is synchronous or asynchronous determines signal control and termination. To reduce complexity, multiplexing is the same for both modes. Table 11-6 shows the scheme for DRAM configurations. This scheme works for symmetric configurations (in which the number of rows equals the number of columns) as well as asymmetric configurations (in which the number of rows and columns are different).

**Table 11-6. Generic Address Multiplexing Scheme**

| Address Pin | Row Address | Column Address | Notes Relating to Port Sizes |
|---|---|---|---|
| 17 | 17 | 0 | 8-bit port only |
| 16 | 16 | 1 | 8- and 16-bit ports only |
| 15 | 15 | 2 | |
| 14 | 14 | 3 | |
| 13 | 13 | 4 | |
| 12 | 12 | 5 | |
| 11 | 11 | 6 | |
| 10 | 10 | 7 | |
| 9 | 9 | 8 | |
| 17 | 17 | 16 | 32-bit port only |
| 18 | 18 | 17 | 16-bit port only or 32-bit port with only 8 column address lines |
| 19 | 19 | 18 | 16-bit port only when at least 9 column address lines are used |

**Table 11-6. Generic Address Multiplexing Scheme (Continued)**

| Address Pin | Row Address | Column Address | Notes Relating to Port Sizes |
|---|---|---|---|
| 20 | 20 | 19 | |
| 21 | 21 | 20 | |
| 22 | 22 | 21 | |
| 23 | 23 | 22 | |
| 24 | 24 | 23 | |
| 25 | 25 | 24 | |

Note the following:

- Each MCF5407 address bit drives both a row address and a column address bit.
- As the user upgrades ADRAM, corresponding MCF5407 address bits must be connected. This multiplexing scheme allows various memory widths to be connected to the address bus.
- Some differences exist for each of the three possible port sizes. Note that only 8-bit ports use an A0 address from the MCF5407. Because 16- and 32-bit ports issue either words or longwords when accessed, they do not use the MCF5407 A0 signal.

  Likewise, the configuration for 32-bit ports uses neither A0 or A1. This presents a slight problem because DRAM address signal A0 is issued on physical pin A17 of the MCF5407 along with the ADRAM address signal A17. Although A0 is not used for larger ports, A17 is still needed. The MCF5407 DRAM controller provides for this by changing the column address that appears on physical pin A17 of the processor whenever an 8-bit port is not selected. This is determined by the DACR*n*[PS] settings. For 8-bit ports, MCF5407 physical pin A17 drives logical address A0 during the CAS cycle. When 16- or 32-bit port sizes are programmed, the CAS cycle pin A17 drives logical address A16, as indicated in the generic connection scheme.
- If a 32-bit port is used with only eight column address lines, A18 must drive DRAM address bit A18. Otherwise, in 32-bit port configurations, the MCF5407 physical address line is not connected with more than eight column address lines.
- All ADRAM blocks have a fixed page size of 512 bytes for page-mode operation. The addresses are connected differently for various width combinations.

Table 11-7, Table 11-8, and Table 11-9 show how 8-, 16-, and 32-bit symmetrical ADRAM memories are connected to the address bus. The memory sizes show what DRAM size is accessed if the corresponding bits are connected to the memory. In each case, there is a base memory size. This limitation exists to allow simple page-mode multiplexing. Notice also that MCF5407 pin 17 is treated differently in byte-wide operations. In byte-wide operations, address bits 16 and 17 are driven on MCF5407 physical address pins 16 and 17, rather than the two bits being driven solely on A17, as they are for 32-wide memories.

**Table 11-7. DRAM Addressing for Byte-Wide Memories**

| MCF5407 Address Pin | MCF5407 Address Bit Driven for RAS | MCF5407 Address Bit Driven when CAS is Asserted | Memory Size |
|---|---|---|---|
| 17 | 17 | 0 | Base memory size of 256 Kbytes |
| 16 | 16 | 1 | |
| 15 | 15 | 2 | |
| 14 | 14 | 3 | |
| 13 | 13 | 4 | |
| 12 | 12 | 5 | |
| 11 | 11 | 6 | |
| 10 | 10 | 7 | |
| 9 | 9 | 8 | |
| 19 | 19 | 18 | 1 Mbyte |
| 21 | 21 | 20 | 4 Mbytes |
| 23 | 23 | 22 | 16 Mbytes |
| 25 | 25 | 24 | 64 Mbytes |

Note that in Table 11-8, MCF5407 pin A19 is not connected because DRAM address bit 18 is already provided on MCF5407 pin A18; thus, the next MCF5407 pin used should be A20.

**Table 11-8. DRAM Addressing for 16-Bit Wide Memories**

| MCF5407 Address Pin | MCF5407 Address Bit Driven for RAS | MCF5407 Address Bit Driven when CAS is Asserted | Memory Size |
|---|---|---|---|
| 16 | 16 | 1 | Base memory size of 128 Kbytes |
| 15 | 15 | 2 | |
| 14 | 14 | 3 | |
| 13 | 13 | 4 | |
| 12 | 12 | 5 | |
| 11 | 11 | 6 | |
| 10 | 10 | 7 | |
| 9 | 9 | 8 | |
| 18 | 18 | 17 | 512 Kbytes |
| 20 | 20 | 19 | 2 Mbytes |
| 22 | 22 | 21 | 8 Mbytes |
| 24 | 24 | 23 | 32 Mbytes |

**Table 11-9. DRAM Addressing for 32-Bit Wide Memories**

| MCF5407 Address Pin | MCF5407 Address Bit Driven for $\overline{RAS}$ | MCF5407 Address Bit Driven when CAS is Asserted | Memory Size |
|---|---|---|---|
| 15 | 15 | 2 | Base Memory Size of 64 Kbytes |
| 14 | 14 | 3 | |
| 13 | 13 | 4 | |
| 12 | 12 | 5 | |
| 11 | 11 | 6 | |
| 10 | 10 | 7 | |
| 9 | 9 | 8 | |
| 17 | 17 | 16 | 256 Kbytes |
| 19 | 19 | 18 | 1 Mbyte |
| 21 | 21 | 20 | 4 Mbytes |
| 23 | 23 | 22 | 16 Mbytes |
| 25 | 25 | 24 | 64 Mbytes |

### 11.3.3.1 Non-Page-Mode Operation

In non-page mode, the simplest mode, the DRAM controller provides termination and runs a separate bus cycle for each data transfer. Figure 11-5 shows a non-page-mode access in which a DRAM read is followed by a write. Addresses for a new bus cycle are driven at the rising clock edge.

For a DRAM block hit, the associated $\overline{RAS}$ is driven at the next falling edge. Here DACR*n*[RCD] = 0, so the address is multiplexed at the next rising edge to provide the column address. The required $\overline{CAS}$ signals are then driven at the next falling edge and remain asserted for the period programmed in DACR*n*[CAS]. Here, DACR*n*[RNCN] = 1, so it is precharged one clock before $\overline{CAS}$ is negated. On a read, data is sampled on the last rising edge of the clock that $\overline{CAS}$ is valid.

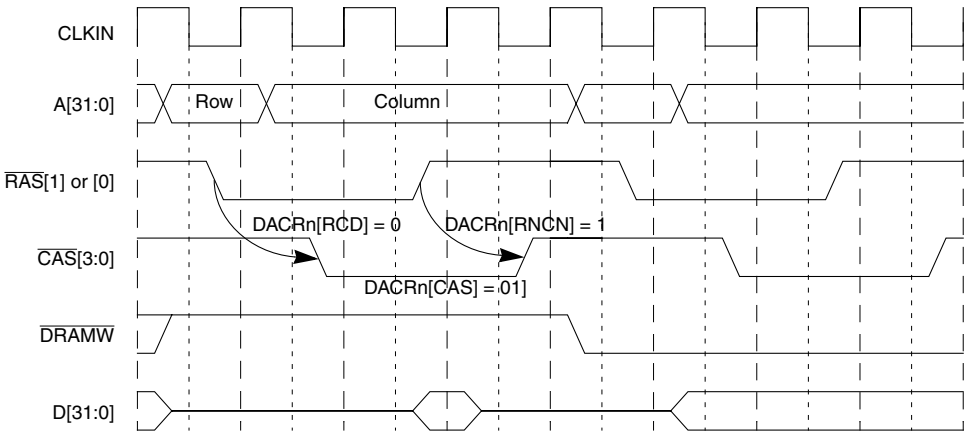



**Figure 11-5. Basic Non-Page-Mode Operation RCD = 0, RNCN = 1 (4-4-4-4)**

Figure 11-6 shows a variation of the basic cycle. In this case, RCD is 1, so there are two clocks between $\overline{RAS}$ and $\overline{CAS}$. Note that the address is multiplexed on the rising clock immediately before $\overline{CAS}$ is asserted. Because RNCN = 0, $\overline{RAS}$ and $\overline{CAS}$ are negated together. The next bus cycle is initiated, but because DACR*n*[RP] requires $\overline{RAS}$ to be precharged for two clocks, $\overline{RAS}$ is delayed for a clock in the bus cycle. Note that this does not delay the address signals, only $\overline{RAS}$.

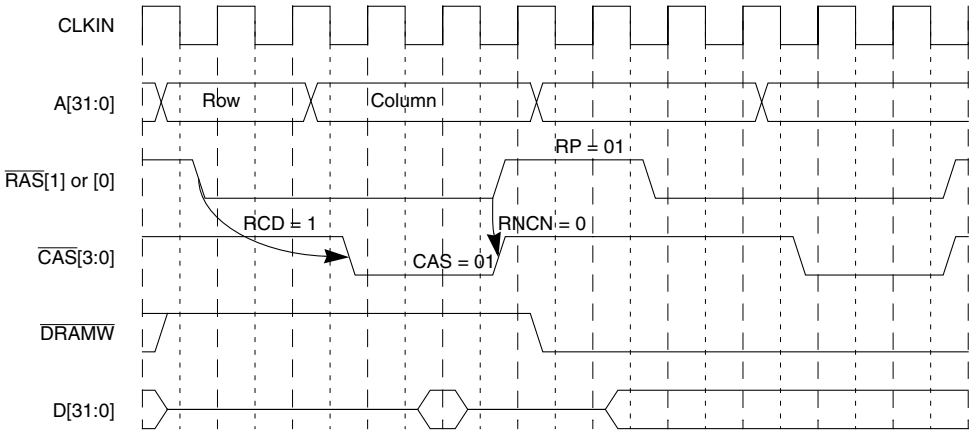



**Figure 11-6. Basic Non-Page-Mode Operation RCD = 1, RNCN = 0 (5-5-5-5)**

### 11.3.3.2 Burst Page-Mode Operation

Burst page-mode operation (DACRn[PM] = 01) optimizes memory accesses in page mode by allowing a row address to remain registered in the DRAM while accessing data in different columns. This eliminates the setup and hold times associated with the need to precharge and assert RAS. Therefore, only the first bus cycle in the page takes the full access time; subsequent accesses are streamlined. Single accesses look the same as non-page-mode accesses.

Burst page-mode accesses of any size—byte, word, longword, or line—are assumed to reside in the same page. In this mode, the DRAM controller generates a burst transfer only when the operand is larger than the DRAM block port size (such as, a line transfer to a 32-bit port or a longword transfer to an 8-bit port). The primary cycle asserts $\overline{RAS}$ and $\overline{CAS}$; subsequent cycles assert only $\overline{CAS}$. At the end of the access, $\overline{RAS}$ is precharged. The DRAM controller increments addresses between cycles.

Figure 11-7 shows a read access in burst page mode. Four accesses take place, which could be a 32-bit access to an 8-bit port or a line access to a 32-bit port. Other burst page-mode operations may be from 2 to 16 accesses long, depending on the access and port sizes. In those cases, timing is similar with more or fewer accesses.

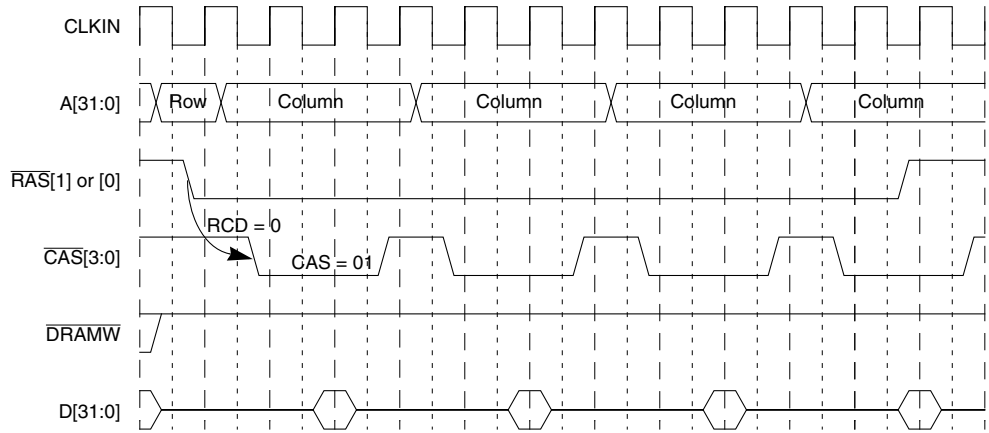



**Figure 11-7. Burst Page-Mode Read Operation (4-3-3-3)**

Figure 11-8 shows the write operation with the same configuration.

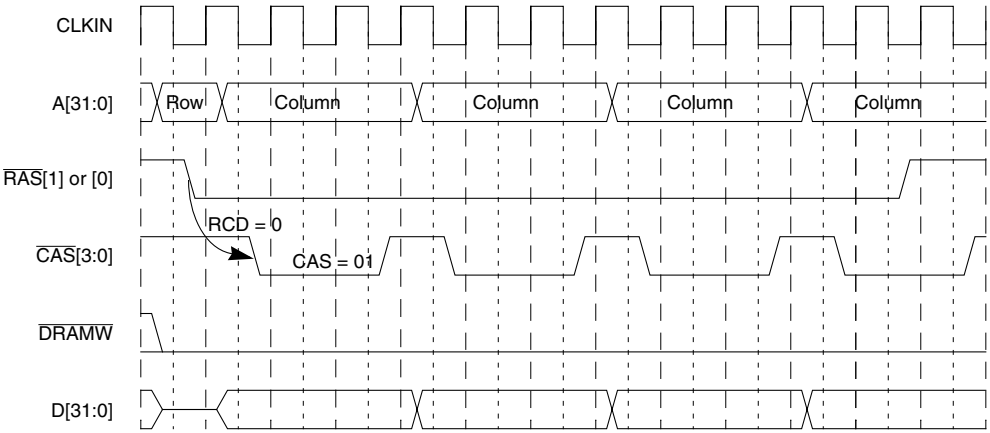



**Figure 11-8. Burst Page-Mode Write Operation (4-3-3-3)**

### 11.3.3.3 Continuous Page Mode

Continuous page mode (DACRn[PM] = 11) is a type of page mode that balances performance, complexity, and size. In typical page-mode implementations, sequential addresses are checked for multiple hits in a DRAM block. On a hit, $\overline{\text{RAS}}$ remains asserted and $\overline{\text{CAS}}$ is asserted with the new column address. On a miss, $\overline{\text{RAS}}$ must be precharged again before the bus cycle begins.

Continuous page mode supports page-mode operation without requiring an address holding register per memory block and eliminates the delay for a miss-to-precharge $\overline{\text{RAS}}$ for the upcoming bus cycle. Because the internal MCF5407 address bus is pipelined, addresses for

the next bus cycle are often available before the current cycle completes. The two addresses are compared at the end of the cycle to determine if the next address hits the same page. If so, $\overline{RAS}$ remains asserted. If not, or if no access is pending, $\overline{RAS}$ is precharged before the next bus cycle is active on the external bus. As a result, a page miss suffers no penalty. Single accesses not followed by a hit in the page look like non-page-mode accesses.

Figure 11-9 shows a write cycle followed by a read cycle in continuous page mode. The read hits in the same page as the write so $\overline{RAS}$ is not negated before the second cycle. Note that the row address does not appear on the pins for a bus cycle that hits in the page. Column addresses are immediately multiplexed onto the pins. The third bus cycle is a page miss, so $\overline{RAS}$ is precharged before the end of the bus cycle and no extra precharge delay is incurred.

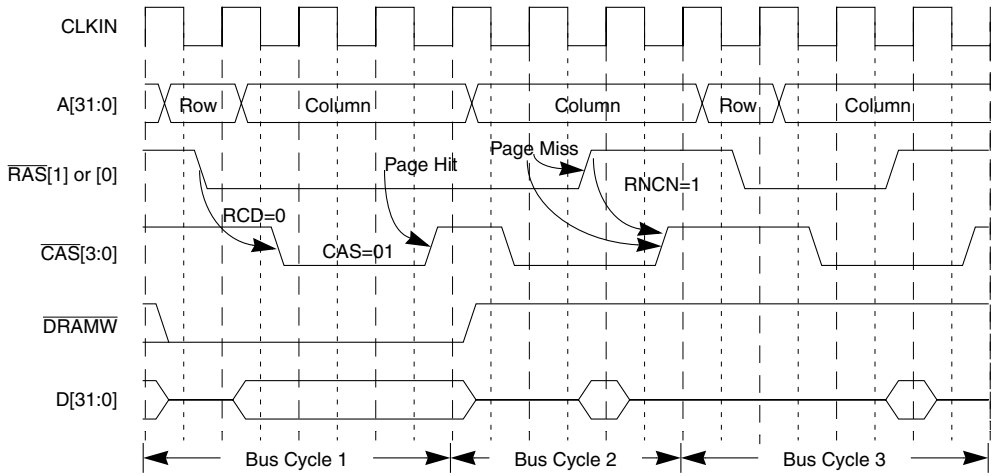



**Figure 11-9. Continuous Page-Mode Operation**

If a write cycle hits in the page, $\overline{CAS}$ must be delayed by one clock to allow data to become valid, as shown in Figure 11-10.

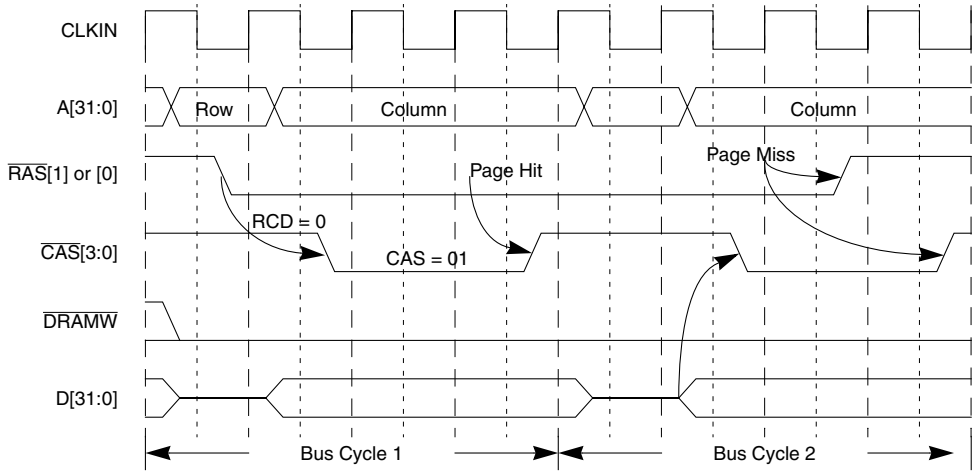



**Figure 11-10. Write Hit in Continuous Page Mode**

### 11.3.3.4 Extended Data Out (EDO) Operation

EDO is a variation of page mode that allows the DRAM to continue driving data out of the device while $\overline{CAS}$ is precharging. To support EDO DRAMs, the DRAM controller delays internal termination of the cycle by one clock so data can continue to be captured as $\overline{CAS}$ is being precharged. For data to be driven by the DRAMs, $\overline{RAS}$ is held after $\overline{CAS}$ is negated. EDO operation does not affect write operations. EDO DRAMs can be used in continuous page or burst page modes. Single accesses not followed by a hit in the page look like non-page-mode accesses.

Figure 11-11 shows four consecutive EDO accesses. Note that data is sampled after $\overline{CAS}$ is negated and that on the last page access, $\overline{CAS}$ is held until after data is sampled to assure that the data is driven. This allows $\overline{RAS}$ to be precharged before the end of the cycle.

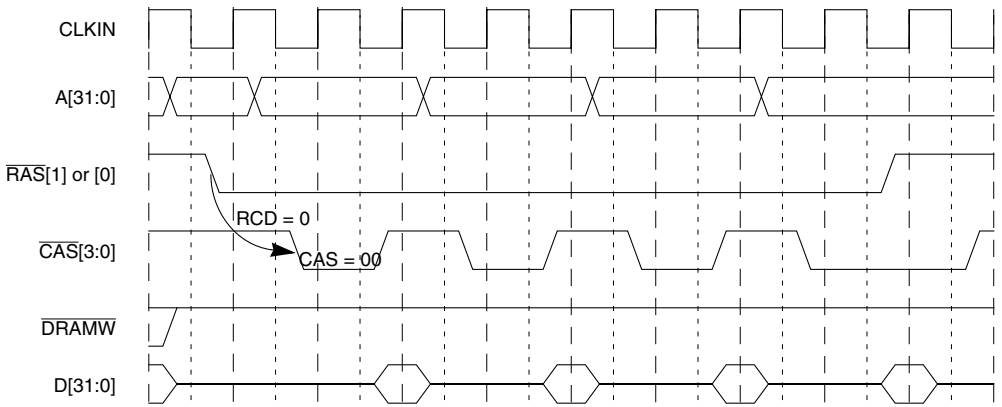



**Figure 11-11. EDO Read Operation (3-2-2-2)**

### 11.3.3.5 Refresh Operation

The DRAM controller supports $\overline{\text{CAS}}$-before-$\overline{\text{RAS}}$ refresh operations that are not synchronized to bus activity. A special $\overline{\text{DRAMW}}$ pin is provided so refresh can occur regardless of the state of the processor bus.

When the refresh counter rolls over, it sets an internal flag to indicate that a refresh is pending. If that happens during a continuous page-mode access, the page is closed ($\overline{\text{RAS}}$ precharged) when the data transfer completes to allow the refresh to occur. The flag is cleared when the refresh cycle is run. Both memory blocks are simultaneously refreshed as determined by the DCR. DRAM accesses are delayed during refresh. Only an active bus access to a DRAM block can delay refresh.

Figure 11-12 shows a bus cycle delayed by a refresh operation. Notice that $\overline{\text{DRAMW}}$ is forced high during refresh. The row address is held until the pending DRAM access.

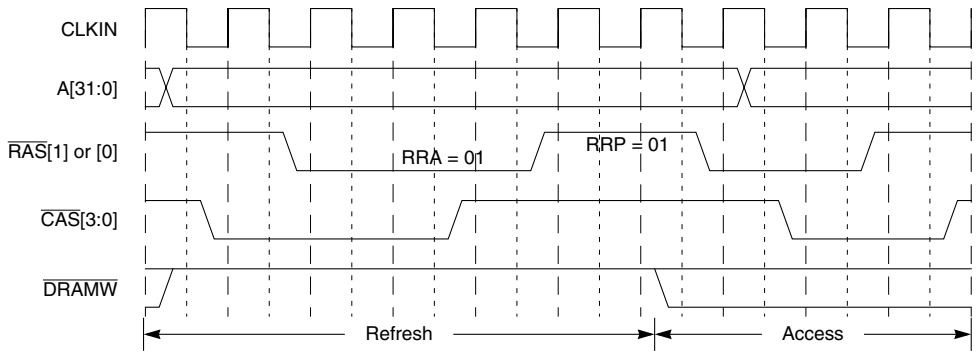



**Figure 11-12. DRAM Access Delayed by Refresh**

## 11.4 Synchronous Operation

By running synchronously with the system clock instead of responding to asynchronous control signals, SDRAM can (after an initial latency period) be accessed on every clock; 5-1-1-1 is a typical MCF5407 burst rate to SDRAM.

Note that because the MCF5407 cannot have more than one page open at a time, it does not support interleaving.

SDRAM controllers are more sophisticated than asynchronous DRAM controllers. Not only must they manage addresses and data, but they must send special commands for such functions as precharge, read, write, burst, auto-refresh, and various combinations of these functions. Table 11-10 lists common SDRAM commands.

**Table 11-10. SDRAM Commands**

| Command | Definition |
|---|---|
| ACTV | Activate. Executed before READ or WRITE executes; SDRAM registers and decodes row address. |
| MRS | Mode register set. |
| NOP | No-op. Does not affect SDRAM state machine; DRAM controller control signals negated; $\overline{\text{RAS}}$ asserted. |
| PALL | Precharge all. Precharges all internal banks of an SDRAM component; executed before new page is opened. |
| READ | Read access. SDRAM registers column address and decodes that a read access is occurring. |
| REF | Refresh. Refreshes internal bank rows of an SDRAM component. |
| SELF | Self refresh. Refreshes internal bank rows of an SDRAM component when it is in low-power mode. |
| SELFX | Exit self refresh. This command is sent to the DRAM controller when DCR[IS] is cleared. |
| WRITE | Write access. SDRAM registers column address and decodes that a write access is occurring. |

SDRAMs operate differently than asynchronous DRAMs, particularly in the use of data pipelines and commands to initiate special actions. Commands are issued to memory using specific encodings on address and control pins. Soon after system reset, a command must be sent to the SDRAM mode register to configure SDRAM operating parameters. Note that, after synchronous operation is selected by setting DCR[SO], DRAM controller registers reflect the synchronous operation and there is no way to return to asynchronous operation without resetting the processor.

## 11.4.1 DRAM Controller Signals in Synchronous Mode

Table 11-11 shows the behavior of DRAM signals in synchronous mode.

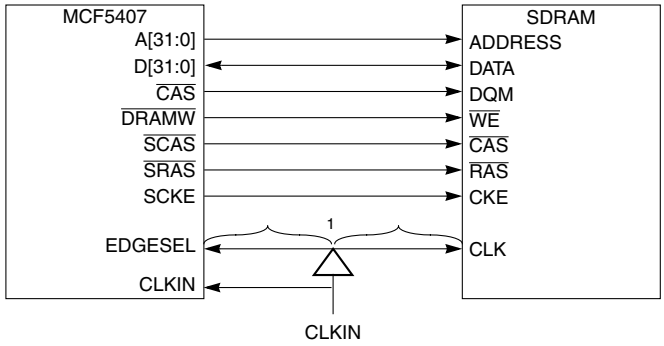
**Table 11-11. Synchronous DRAM Signal Connections**

| Signal | Description |
|---|---|
| $\overline{\text{SRAS}}$ | Synchronous row address strobe. Indicates a valid SDRAM row address is present and can be latched by the SDRAM. $\overline{\text{SRAS}}$ should be connected to the corresponding SDRAM $\overline{\text{SRAS}}$. Do not confuse $\overline{\text{SRAS}}$ with the DRAM controller's $\overline{\text{RAS}}$[1:0], which should not be interfaced to the SDRAM $\overline{\text{SRAS}}$ signals. |
| $\overline{\text{SCAS}}$ | Synchronous column address strobe. Indicates a valid column address is present and can be latched by the SDRAM. $\overline{\text{SCAS}}$ should be connected to the corresponding signal labeled $\overline{\text{SCAS}}$ on the SDRAM. Do not confuse $\overline{\text{SCAS}}$ with the DRAM controller's $\overline{\text{CAS}}$[3:0] signals. |
| $\overline{\text{DRAMW}}$ | DRAM read/write. Asserted for write operations and negated for read operations. |
| $\overline{\text{RAS}}$[1:0] | Row address strobe. Select each memory block of SDRAMs connected to the MCF5407. One $\overline{\text{RAS}}$ signal selects one SDRAM block and connects to the corresponding $\overline{\text{CS}}$ signals. |
| SCKE | Synchronous DRAM clock enable. Connected directly to the CKE (clock enable) signal of SDRAMs. Enables and disables the clock internal to SDRAM. When CKE is low, memory can enter a power-down mode where operations are suspended or they can enter self-refresh mode. SCKE functionality is controlled by DCR[COC]. For designs using external multiplexing, setting COC allows SCKE to provide command-bit functionality. |
| $\overline{\text{CAS}}$[3:0] | Column address strobe. For synchronous operation, $\overline{\text{CAS}}$[3:0] function as byte enables to the SDRAMs. They connect to the DQM signals (or mask qualifiers) of the SDRAMs. |

**Table 11-11. Synchronous DRAM Signal Connections (Continued)**

| Signal | Description |
|---|---|
| CLKIN | Bus clock output. Connects to the CLK input of SDRAMs. |
| EDGESEL | Synchronous edge select. Provides additional output hold time for signals that interface to external SDRAMs. EDGESEL supports the three following modes for SDRAM interface signals:<br>• Tied high. Signals change on the rising edge of CLKIN.<br>• Tied low. Signals change on the falling edge of CLKIN.<br>• Tied to buffered CLKIN. Signals change on the rising edge of the buffered clock.<br>EDGESEL can provide additional output hold time for SDRAM interface signals, however the SDRAM clock and CLKIN frequencies must be the same. See Section 11.4.2, “Using Edge Select (EDGESEL).” |

Figure 11-13 shows a typical signal configuration for synchronous mode.




[1] Trace length from buffer to CLK must equal length from buffer to EDGESEL.

**Figure 11-13. MCF5407 SDRAM Interface**

## 11.4.2 Using Edge Select (EDGESEL)

EDGESEL can ease system-level timings (note that the optional buffer in Figure 11-13 is for memories that need extra delay). The clock at the input to the SDRAM is monitored and data is held until the next edge of the bus clock, adding required output hold time to the address, data, and control signals.

To generate SDRAM interface timing, address, data, and control signals are clocked through a two-stage shift register. The first stage is clocked on the rising edge of CLKIN; the second is clocked on the falling edge. This makes the signal available for up to an additional half bus clock cycle, of which only a small amount is needed for proper timing.

Using the connection shown in Figure 11-13 ensures that data remains held for a longer time after the rising edge of the SDRAM clock input. This helps to match the MCF5407 output timing with the SDRAM clock.

Figure 11-14 shows the output wave forms for the interface signals changing on the rising edge (A) and falling edge (B) of CLKIN as determined by whether EDGESEL is tied high or low. It also shows timing (C) with EDGESEL tied to buffered CLKIN.

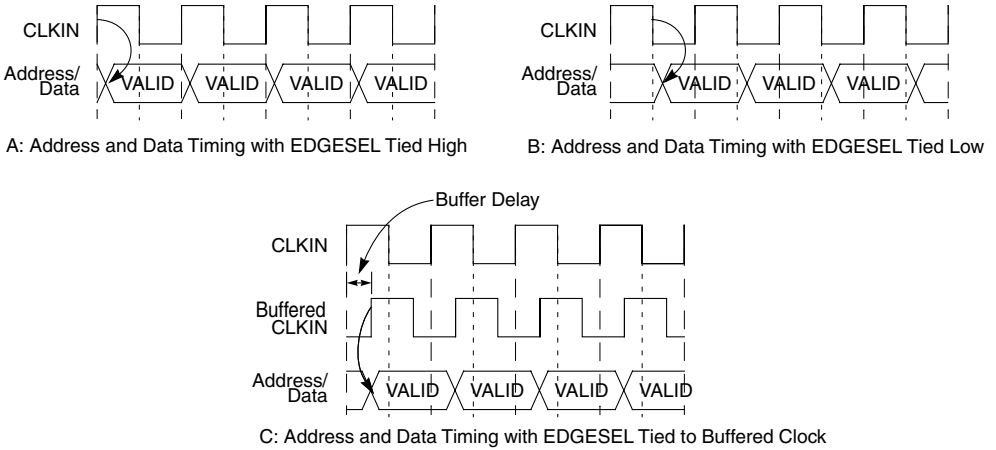



**Figure 11-14. Using EDGESEL to Change Signal Timing**

## 11.4.3 Synchronous Register Set

The memory map in Table 11-1 is the same for both synchronous and asynchronous operation. However, some bits are different, as noted in the following sections.

### 11.4.3.1 DRAM Control Register (DCR) in Synchronous Mode

The DRAM control register (DCR), Figure 11-15, controls refresh logic.

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | SO | — | NAM | COC | IS | RTIM | | RC | | | | | | | | |
| Reset | 0 | Uninitialized | | | | | | | | | | | | | | |
| R/W | R/W | | | | | | | | | | | | | | | |
| Addr | MBAR + 0x100 | | | | | | | | | | | | | | | |

**Figure 11-15. DRAM Control Register (DCR) (Synchronous Mode)**

Table 11-12 describes DCR fields.

**Table 11-12. DCR Field Descriptions (Synchronous Mode)**

| Bits | Name | Description |
|---|---|---|
| 15 | SO | Synchronous operation. Selects synchronous or asynchronous mode. When in synchronous mode, the DRAM controller can be switched to ADRAM mode only by resetting the MCF5407.<br>0 Asynchronous DRAMs. Default at reset.<br>1 Synchronous DRAMs |
| 14 | — | Reserved, should be cleared. |
| 13 | NAM | No address multiplexing. Some implementations require external multiplexing. For example, when linear addressing is required, the DRAM should not multiplex addresses on DRAM accesses.<br>0 The DRAM controller multiplexes the external address bus to provide column addresses.<br>1 The DRAM controller does not multiplex the external address bus to provide column addresses. |

**Table 11-12. DCR Field Descriptions (Synchronous Mode) (Continued)**

| Bits | Name | Description |
|---|---|---|
| 12 | COC | Command on SDRAM clock enable (SCKE). Implementations that use external multiplexing (NAM = 1) must support command information to be multiplexed onto the SDRAM address bus.<br>0 SCKE functions as a clock enable; self-refresh is initiated by the DRAM controller through DCR[IS].<br>1 SCKE drives command information. Because SCKE is not a clock enable, self-refresh cannot be used (setting DCR[IS]). Thus, external logic must be used if this functionality is desired. External multiplexing is also responsible for putting the command information on the proper address bit. |
| 11 | IS | Initiate self-refresh command.<br>0 Take no action or issue a SELFX command to exit self refresh.<br>1 If DCR[COC] = 0, the DRAM controller sends a SELF command to both SDRAM blocks to put them in low-power, self-refresh state where they remain until IS is cleared, at which point the controller sends a SELFX command for the SDRAMs to exit self-refresh. The refresh counter is suspended while the SDRAMs are in self-refresh; the SDRAM controls the refresh period. |
| 10–9 | RTIM | Refresh timing. Determines the timing operation of auto-refresh in the DRAM controller. Specifically, it determines the number of clocks inserted between a REF command and the next possible ACTV command. This same timing is used for both memory blocks controlled by the DRAM controller. This corresponds to $t_{RC}$ in the SDRAM specifications.<br>00 3 clocks<br>01 6 clocks<br>1x 9 clocks |
| 8–0 | RC | Refresh count. Controls refresh frequency. The number of bus clocks between refresh cycles is (RC + 1) * 16. Refresh can range from 16–8192 bus clocks to accommodate both standard and low-power DRAMs with bus clock operation from less than 2 MHz to greater than 50 MHz.<br>The following example calculates RC for an auto-refresh period for 4096 rows to receive 64 mS of refresh every 15.625 µs for each row (625 bus clocks at 40 MHz). This operation is the same as in asynchronous mode.<br># of bus clocks = 625 = (RC field + 1) * 16<br>RC = (625 bus clocks/16) -1 = 38.06, which rounds to 38; therefore, RC = 0x26. |

### 11.4.3.2 DRAM Address and Control Registers (DACR0/DACR1) in Synchronous Mode

The DRAM address and control registers (DACR0 and DACR1), shown in Figure 11-16, contain the base address compare value and the control bits for both memory blocks 0 and 1 of the DRAM controller. Address and timing are also controlled by bits in DACR*n*.

| | 31 … 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | BA | — | | RE | — | CASL | | — | CBM | | | — | IMRS | PS | | IP | PM | — | |
| Reset | Uninitialized | | | 0 | Uninitialized | | | | | | | | 0 | Uninitialized | | | | | |
| R/W | R/W | | | | | | | | | | | | | | | | | | |
| Addr | MBAR+0x108 (DACR0); 0x110(DACR1) | | | | | | | | | | | | | | | | | | |

**Figure 11-16. DACR0 and DACR1 Registers (Synchronous Mode)**

Table 11-13 describes DACR*n* fields.

**Table 11-13. DACR0/DACR1 Field Descriptions (Synchronous Mode)**

<table>
<tr><th>Bit</th><th>Name</th><th>Description</th></tr>
<tr><td>31–18</td><td>BA</td><td>Base address register. With DCMR[BAM], determines the address range in which the associated DRAM block is located. Each BA bit is compared with the corresponding address of the current bus cycle. If all unmasked bits match, the address hits in the associated DRAM block. BA functions the same as in asynchronous operation.</td></tr>
<tr><td>17–16</td><td>—</td><td>Reserved, should be cleared.</td></tr>
<tr><td>15</td><td>RE</td><td>Refresh enable. Determines when the DRAM controller generates a refresh cycle to the DRAM block.<br>0 Do not refresh associated DRAM block<br>1 Refresh associated DRAM block</td></tr>
<tr><td>14</td><td>—</td><td>Reserved, should be cleared.</td></tr>
<tr><td>13–12</td><td>CASL</td><td>$\overline{\text{CAS}}$ latency. Affects the following SDRAM timing specifications. Timing nomenclature varies with manufacturers. Refer to the SDRAM specification for the appropriate timing nomenclature:
<table>
<tr><th rowspan="2">Parameter</th><th colspan="4">Number of Bus Clocks</th></tr>
<tr><th>CASL= 00</th><th>CASL = 01</th><th>CASL= 10</th><th>CASL= 11</th></tr>
<tr><td>$t_{RCD}$—$\overline{\text{SRAS}}$ assertion to $\overline{\text{SCAS}}$ assertion</td><td>1</td><td>2</td><td>3</td><td>3</td></tr>
<tr><td>$t_{CASL}$—$\overline{\text{SCAS}}$ assertion to data out</td><td>1</td><td>2</td><td>3</td><td>3</td></tr>
<tr><td>$t_{RAS}$—ACTV command to precharge command</td><td>2</td><td>4</td><td>6</td><td>6</td></tr>
<tr><td>$t_{RP}$—Precharge command to ACTV command</td><td>1</td><td>2</td><td>3</td><td>3</td></tr>
<tr><td>$t_{RWL}$,$t_{RDL}$—Last data input to precharge command</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>$t_{EP}$—Last data out to precharge command)</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
</table></td></tr>
<tr><td>11</td><td>—</td><td>Reserved, should be cleared.</td></tr>
<tr><td>10–8</td><td>CBM</td><td>Command and bank MUX [2:0]. Because different SDRAM configurations cause the command and bank select lines to correspond to different addresses, these resources are programmable. CBM determines the addresses onto which these functions are multiplexed.<br>CBM Command Bit Bank Select Bits<br>000 17 18 and up<br>001 18 19 and up<br>010 19 20 and up<br>011 20 21 and up<br>100 21 22 and up<br>101 22 23 and up<br>110 23 24 and up<br>111 24 25 and up<br>This encoding and the address multiplexing scheme handle common SDRAM organizations. Bank select bits include a base bit and all address bits above for SDRAMs with multiple bank select bits.</td></tr>
<tr><td>7</td><td>—</td><td>Reserved, should be cleared.</td></tr>
</table>

**Table 11-13. DACR0/DACR1 Field Descriptions (Synchronous Mode) (Continued)**

| Bit | Name | Description |
|---|---|---|
| 6 | IMRS | Initiate mode register set (MRS) command. Setting IMRS generates a MRS command to the associated SDRAMs. In initialization, IMRS should be set only after all DRAM controller registers are initialized and PALL and REFRESH commands have been issued. After IMRS is set, the next access to an SDRAM block programs the SDRAM's mode register. Thus, the address of the access should be programmed to place the correct mode information on the SDRAM address pins. Because the SDRAM does not register this information, it doesn't matter if the IMRS access is a read or a write or what, if any, data is put onto the data bus. The DRAM controller clears IMRS after the MRS command finishes.<br>0 Take no action<br>1 Initiate MRS command |
| 5–4 | PS | Port size. Indicates the port size of the associated block of SDRAM, which allows for dynamic sizing of associated SDRAM accesses. PS functions the same in asynchronous operation.<br>00 32-bit port<br>01 8-bit port<br>1x 16-bit port |
| 3 | IP | Initiate precharge all (PALL) command. The DRAM controller clears IP after the PALL command is finished. Accesses via IP should be no wider than the port size programmed in PS.<br>0 Take no action.<br>1 A PALL command is sent to the associated SDRAM block. During initialization, this command is executed after all DRAM controller registers are programmed. After IP is set, the next write to an appropriate SDRAM address generates the PALL command to the SDRAM block. |
| 2 | PM | Page mode. Indicates how the associated SDRAM block supports page-mode operation.<br>0 Page mode on bursts only. The DRAM controller dynamically bursts the transfer if it falls within a single page and the transfer size exceeds the port size of the SDRAM block. After the burst, the page closes and a precharge is issued.<br>1 Continuous page mode. The page stays open and only $\overline{\text{SCAS}}$ needs to be asserted for sequential SDRAM accesses that hit in the same page, regardless of whether the access is a burst. |
| 1–0 | — | Reserved, should be cleared. |

### 11.4.3.3 DRAM Controller Mask Registers (DMR0/DMR1)

The DMR*n*, Figure 11-17, include mask bits for the base address and for address attributes. They are the same as in asynchronous operation.

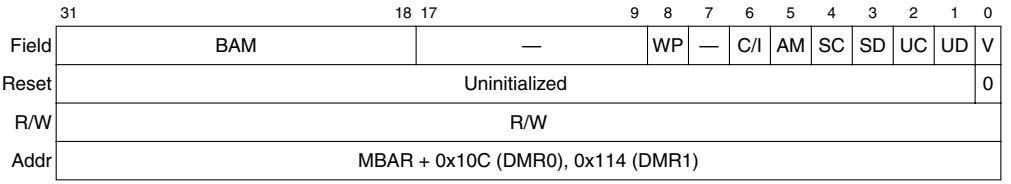



**Figure 11-17. DRAM Controller Mask Registers (DMR0 and DMR1)**

Table 11-14 describes DMR*n* fields.

**Table 11-14. DMR0/DMR1 Field Descriptions**

<table>
<tr><th>Bits</th><th>Name</th><th>Description</th></tr>
<tr><td>31–18</td><td>BAM</td><td>Base address mask. Masks the associated DACRn[BA]. Lets the DRAM controller connect to various DRAM sizes. Mask bits need not be contiguous (see Section 11.5, “SDRAM Example.”)<br>0 The associated address bit is used in decoding the DRAM hit to a memory block.<br>1 The associated address bit is not used in the DRAM hit decode.</td></tr>
<tr><td>17–9</td><td>—</td><td>Reserved, should be cleared.</td></tr>
<tr><td>8</td><td>WP</td><td>Write protect. Determines whether the associated block of DRAM is write protected.<br>0 Allow write accesses<br>1 Ignore write accesses. The DRAM controller ignores write accesses to the memory block and an address exception occurs. Write accesses to a write-protected DRAM region are compared in the chip select module for a hit. If no hit occurs, an external bus cycle is generated. If this external bus cycle is not acknowledged, an access exception occurs.</td></tr>
<tr><td>7</td><td>—</td><td>Reserved, should be cleared.</td></tr>
<tr><td>6–1</td><td>AMx</td><td>Address modifier masks. Determine which accesses can occur in a given DRAM block.<br>0 Allow access type to hit in DRAM<br>1 Do not allow access type to hit in DRAM
<table>
<tr><th>Bit</th><th>Associated Access Type</th><th>Access Definition</th></tr>
<tr><td>C/I</td><td>CPU space/interrupt acknowledge</td><td>MOVEC instruction or interrupt acknowledge cycle</td></tr>
<tr><td>AM</td><td>Alternate master</td><td>External or DMA master</td></tr>
<tr><td>SC</td><td>Supervisor code</td><td>Any supervisor-only instruction access</td></tr>
<tr><td>SD</td><td>Supervisor data</td><td>Any data fetched during the instruction access</td></tr>
<tr><td>UC</td><td>User code</td><td>Any user instruction</td></tr>
<tr><td>UD</td><td>User data</td><td>Any user data</td></tr>
</table></td></tr>
<tr><td>0</td><td>V</td><td>Valid. Cleared at reset to ensure that the DRAM block is not erroneously decoded.<br>0 Do not decode DRAM accesses.<br>1 Registers controlling the DRAM block are initialized; DRAM accesses can be decoded.</td></tr>
</table>

## 11.4.4 General Synchronous Operation Guidelines

To reduce system logic and to support a variety of SDRAM sizes, the DRAM controller provides SDRAM control signals as well as a multiplexed row address and column address to the SDRAM.

When SDRAM blocks are accessed, the DRAM controller can operate in either burst or continuous page mode. The following sections describe the DRAM controller interface to SDRAM, the supported bus transfers, and initialization.

### 11.4.4.1 Address Multiplexing

Table 11-6 shows the generic address multiplexing scheme for SDRAM configurations. All possible address connection configurations can be derived from this table.

The following tables provide a more comprehensive, step-by-step way to determine the correct address line connections for interfacing the MCF5407 to SDRAM. To use the

tables, find the one that corresponds to the number of column address lines on the SDRAM and to the port size as seen by the MCF5407, which is not necessarily the SDRAM port size. For example, if two 1M x 16-bit SDRAMs together form a 2M x 32-bit memory, the port size is 32 bits. Most SDRAMs likely have fewer address lines than are shown in the tables, so follow only the connections shown until all SDRAM address lines are connected.

**Table 11-15. MCF5407 to SDRAM Interface (8-Bit Port, 9-Column Address Lines)**

| MCF5407 Pins | A17 | A16 | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A18 | A19 | A20 | A21 | A22 | A23 | A24 | A25 | A26 | A27 | A28 | A29 | A30 | A31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Row | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Column | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | | | | | | | | | | | | | |
| SDRAM Pins | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 | A14 | A15 | A16 | A17 | A18 | A19 | A20 | A21 | A22 |

**Table 11-16. MCF5407 to SDRAM Interface (8-Bit Port,10-Column Address Lines)**

| MCF5407 Pins | A17 | A16 | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A19 | A20 | A21 | A22 | A23 | A24 | A25 | A26 | A27 | A28 | A29 | A30 | A31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Row | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Column | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 18 | | | | | | | | | | | | |
| SDRAM Pins | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 | A14 | A15 | A16 | A17 | A18 | A19 | A20 | A21 |

**Table 11-17. MCF5407 to SDRAM Interface (8-Bit Port,11-Column Address Lines)**

| MCF5407 Pins | A17 | A16 | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A19 | A21 | A22 | A23 | A24 | A25 | A26 | A27 | A28 | A29 | A30 | A31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Row | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 19 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Column | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 18 | 20 | | | | | | | | | | |
| SDRAM Pins | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 | A14 | A15 | A16 | A17 | A18 | A19 | A20 |

**Table 11-18. MCF5407 to SDRAM Interface (8-Bit Port,12-Column Address Lines)**

| MCF5407 Pins | A17 | A16 | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A19 | A21 | A23 | A24 | A25 | A26 | A27 | A28 | A29 | A30 | A31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Row | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 19 | 21 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Column | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 18 | 20 | 22 | | | | | | | | |
| SDRAM Pins | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 | A14 | A15 | A16 | A17 | A18 | A19 |

### Table 11-19. MCF5407 to SDRAM Interface (8-Bit Port,13-Column Address Lines)

| MCF5407 Pins | A17 | A16 | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A19 | A21 | A23 | A25 | A26 | A27 | A28 | A29 | A30 | A31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Row | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 19 | 21 | 23 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Column | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 18 | 20 | 22 | 24 | | | | | | |
| SDRAM Pins | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 | A14 | A15 | A16 | A17 | A18 |

### Table 11-20. MCF5407 to SDRAM Interface (16-Bit Port, 8-Column Address Lines)

| MCF5407 Pins | A16 | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A17 | A18 | A19 | A20 | A21 | A22 | A23 | A24 | A25 | A26 | A27 | A28 | A29 | A30 | A31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Row | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Column | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | | | | | | | | | | | | | | | |
| SDRAM Pins | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 | A14 | A15 | A16 | A17 | A18 | A19 | A20 | A21 | A22 |

### Table 11-21. MCF5407 to SDRAM Interface (16-Bit Port, 9-Column Address Lines)

| MCF5407 Pins | A16 | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A18 | A19 | A20 | A21 | A22 | A23 | A24 | A25 | A26 | A27 | A28 | A29 | A30 | A31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Row | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Column | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 17 | | | | | | | | | | | | | |
| SDRAM Pins | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 | A14 | A15 | A16 | A17 | A18 | A19 | A20 | A21 |

### Table 11-22. MCF5407 to SDRAM Interface (16-Bit Port, 10-Column Address Lines)

| MCF5407 Pins | A16 | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A18 | A20 | A21 | A22 | A23 | A24 | A25 | A26 | A27 | A28 | A29 | A30 | A31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Row | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 18 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Column | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 17 | 19 | | | | | | | | | | | |
| SDRAM Pins | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 | A14 | A15 | A16 | A17 | A18 | A19 | A20 |

### Table 11-23. MCF5407 to SDRAM Interface (16-Bit Port, 11-Column Address Lines)

| MCF5407 Pins | A16 | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A18 | A20 | A22 | A23 | A24 | A25 | A26 | A27 | A28 | A29 | A30 | A31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Row | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 18 | 20 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Column | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 17 | 19 | 21 | | | | | | | | | |
| SDRAM Pins | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 | A14 | A15 | A16 | A17 | A18 | A19 |

### Table 11-24. MCF5407 to SDRAM Interface (16-Bit Port, 12-Column Address Lines)

| MCF5407 Pins | A16 | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A18 | A20 | A22 | A24 | A25 | A26 | A27 | A28 | A29 | A30 | A31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Row | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 18 | 20 | 22 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Column | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 17 | 19 | 21 | 23 | | | | | | | |
| SDRAM Pins | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 | A14 | A15 | A16 | A17 | A18 |

### Table 11-25. MCF5407 to SDRAM Interface (16-Bit Port, 13-Column-Address Lines)

| MCF5407 Pins | A16 | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A18 | A20 | A22 | A24 | A26 | A27 | A28 | A29 | A30 | A31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Row | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 18 | 20 | 22 | 24 | 26 | 27 | 28 | 29 | 30 | 31 |
| Column | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 17 | 19 | 21 | 23 | 25 | | | | | |
| SDRAM Pins | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 | A14 | A15 | A16 | A17 |

### Table 11-26. MCF5407 to SDRAM Interface (32-Bit Port, 8-Column Address Lines)

| MCF5407 Pins | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A17 | A18 | A19 | A20 | A21 | A22 | A23 | A24 | A25 | A26 | A27 | A28 | A29 | A30 | A31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Row | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Column | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 16 | | | | | | | | | | | | | | |
| SDRAM Pins | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 | A14 | A15 | A16 | A17 | A18 | A19 | A20 | A21 |

### Table 11-27. MCF5407 to SDRAM Interface (32-Bit Port, 9-Column Address Lines)

| MCF5407 Pins | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A17 | A19 | A20 | A21 | A22 | A23 | A24 | A25 | A26 | A27 | A28 | A29 | A30 | A31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Row | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 17 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Column | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 16 | 18 | | | | | | | | | | | | |
| SDRAM Pins | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 | A14 | A15 | A16 | A17 | A18 | A19 | A20 |

### Table 11-28. MCF5407 to SDRAM Interface (32-Bit Port, 10-Column Address Lines)

| MCF5407 Pins | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A17 | A19 | A21 | A22 | A23 | A24 | A25 | A26 | A27 | A28 | A29 | A30 | A31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Row | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 17 | 19 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Column | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 16 | 18 | 20 | | | | | | | | | | |
| SDRAM Pins | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 | A14 | A15 | A16 | A17 | A18 | A19 |

**Table 11-29. MCF5407 to SDRAM Interface (32-Bit Port, 11-Column Address Lines)**

| MCF5407 Pins | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A17 | A19 | A21 | A23 | A24 | A25 | A26 | A27 | A28 | A29 | A30 | A31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Row | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 17 | 19 | 21 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Column | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 16 | 18 | 20 | 22 | | | | | | | | |
| SDRAM Pins | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 | A14 | A15 | A16 | A17 | A18 |

**Table 11-30. MCF5407 to SDRAM Interface (32-Bit Port, 12-Column Address Lines)**

| MCF5407 Pins | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A17 | A19 | A21 | A23 | A25 | A26 | A27 | A28 | A29 | A30 | A31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Row | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 17 | 19 | 21 | 23 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| Column | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 16 | 18 | 20 | 22 | 24 | | | | | | |
| SDRAM Pins | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 | A14 | A15 | A16 | A17 |

### 11.4.4.2 Interfacing Example

The tables in the previous section can be used to configure the interface in the following example. To interface one 2M x 32-bit x 4 bank SDRAM component (8 columns) to the MCF5407, the connections would be as shown in Table 11-31.

**Table 11-31. SDRAM Hardware Connections**

| SDRAM Pins | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 = CMD | BA0 | BA1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MCF5407 Pins | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A17 | A18 | A19 | A20 | A21 | A22 |

### 11.4.4.3 Burst Page Mode

SDRAM can efficiently provide data when an SDRAM page is opened. As soon as $\overline{\text{SCAS}}$ is issued, the SDRAM accepts a new address and asserts $\overline{\text{SCAS}}$ every clock for as long as accesses are in that page. In burst page mode, there are multiple read or write operations for every ACTV command in the SDRAM if the requested transfer size exceeds the port size of the associated SDRAM. The primary cycle of the transfer generates the ACTV and READ or WRITE commands; secondary cycles generate only READ or WRITE commands. As soon as the transfer completes, the PALL command is generated to prepare for the next access.

Note that in synchronous operation, burst mode and address incrementing during burst cycles are controlled by the MCF5407 DRAM controller. Thus, instead of the SDRAM enabling its internal burst incrementing capability, the MCF5407 controls this function. This means that the burst function that is enabled in the mode register of SDRAMs must be disabled when interfacing to the MCF5407.

Figure 11-18 shows a burst read operation. In this example, DACR[CASL] = 01, for an $\overline{\text{SRAS}}$-to-$\overline{\text{SCAS}}$ delay ($t_{RCD}$) of 2 CLKIN cycles. Because $t_{RCD}$ is equal to the read CAS

latency ($\overline{SCAS}$ assertion to data out), this value is also 2 CLKIN cycles. Notice that NOPs are executed until the last data is read. A PALL command is executed one cycle after the last data transfer.

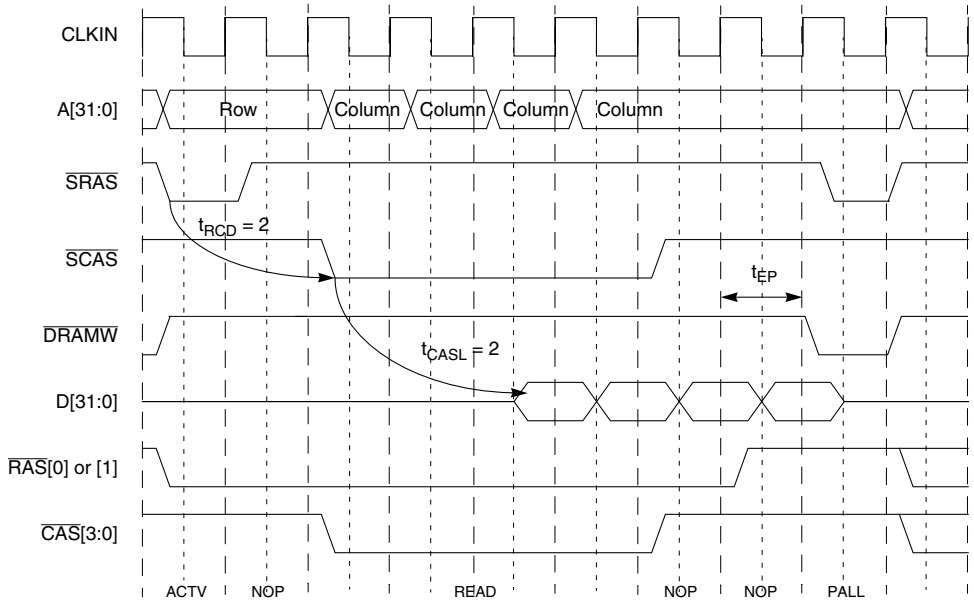



**Figure 11-18. Burst Read SDRAM Access**

Figure 11-19 shows the burst write operation. In this example, DACR[CASL] = 01, which creates an $\overline{SRAS}$-to-$\overline{SCAS}$ delay ($t_{RCD}$) of 2 CLKIN cycles. Note that data is available upon $\overline{SCAS}$ assertion and a burst write cycle completes two cycles sooner than a burst read cycle with the same $t_{RCD}$. The next bus cycle is initiated sooner, but cannot begin an SDRAM cycle until the precharge-to-ACTV delay completes.

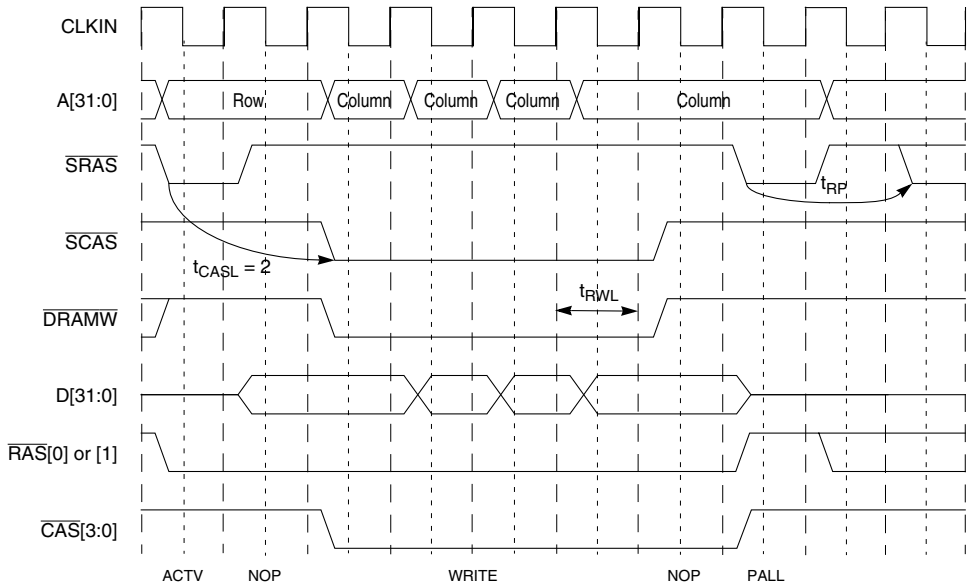



**Figure 11-19. Burst Write SDRAM Access**

Accesses in synchronous burst page mode always cause the following sequence:

1. ACTV command
2. NOP commands to assure $\overline{\text{SRAS}}$-to-$\overline{\text{SCAS}}$ delay (if $\overline{\text{CAS}}$ latency is 1, there are no NOP commands).
3. Required number of READ or WRITE commands to service the transfer size with the given port size.
4. Some transfers need more NOP commands to assure the ACTV-to-precharge delay.
5. PALL command
6. Required number of idle clocks inserted to assure precharge-to-ACTV delay.

### 11.4.4.4 Continuous Page Mode

Continuous page mode is identical to burst page mode, except that it allows the processor core to handle successive bus cycles that hit the same page without having to close the page. When the current bus cycle finishes, the MCF5407 core internal pipelined bus can predict whether the upcoming cycle will hit in the same page.

- If the next bus cycle is not pending or misses in the page, the PALL command is generated to the SDRAM.

- If the next bus cycle is pending and hits in the page, the page is left open, and the next SDRAM access begins with a READ or WRITE command. Because of the nature of the internal CPU pipeline this condition does not occur often, however, the use of continuous page mode is recommended because it can provide a slight performance increase.

Figure 11-20 shows two read accesses in continuous page mode. Note that there is no precharge between the two accesses. Also notice that the second cycle begins with a read operation with no ACTV command.

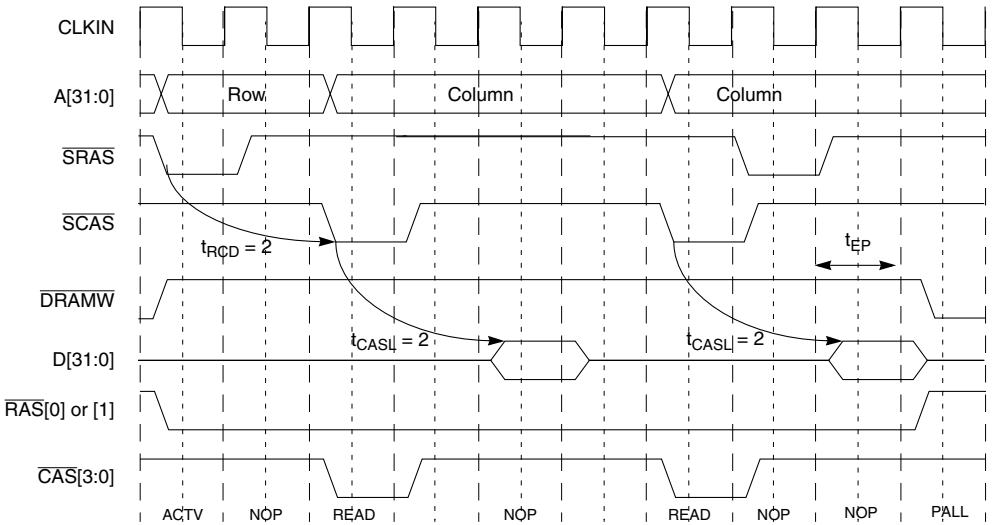



**Figure 11-20. Synchronous, Continuous Page-Mode Access—Consecutive Reads**

Figure 11-21 shows a write followed by a read in continuous page mode. Because the bus cycle is terminated with a WRITE command, the second cycle begins sooner after the write than after the read. A read requires data to be returned before the bus cycle can terminate. Note that in continuous page mode, secondary accesses output the column address only.

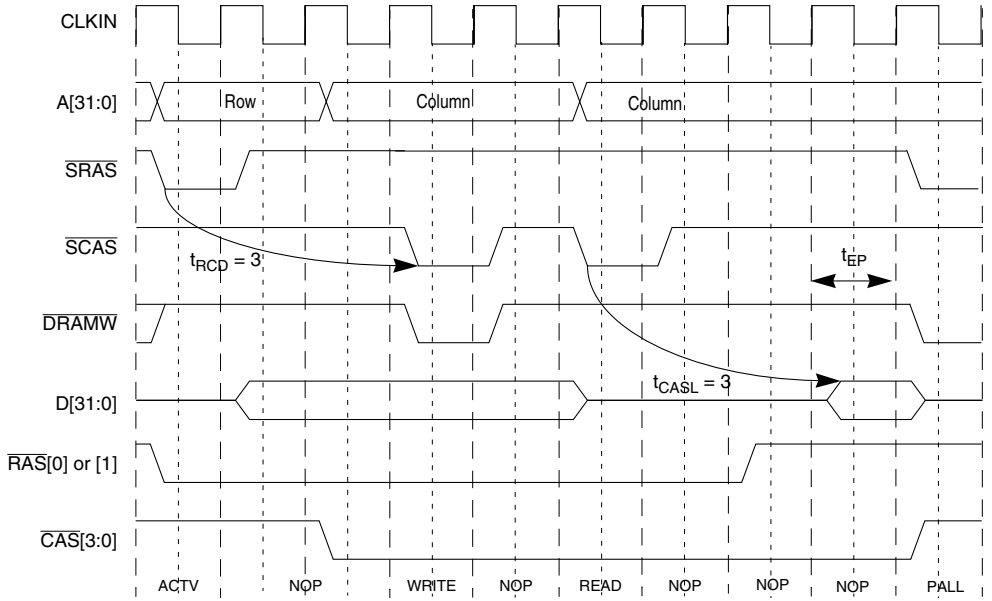



**Figure 11-21. Synchronous, Continuous Page-Mode Access—Read after Write**

### 11.4.4.5 Auto-Refresh Operation

The DRAM controller is equipped with a refresh counter and control. This logic is responsible for providing timing and control to refresh the SDRAM. Once the refresh counter is set, and refresh is enabled, the counter counts to zero. At this time, an internal refresh request flag is set and the counter begins counting down again. The DRAM controller completes any active burst operation and then performs a PALL operation. The DRAM controller then initiates a refresh cycle and clears the refresh request flag. This refresh cycle includes a delay from any precharge to the auto-refresh command, the auto-refresh command, and then a delay until any ACTV command is allowed. Any SDRAM access initiated during the auto-refresh cycle is delayed until the cycle is completed.

Figure 11-22 shows the auto-refresh timing. In this case, there is an SDRAM access when the refresh request becomes active. The request is delayed by the precharge to ACTV delay programmed into the active SDRAM bank by the CAS bits. The REF command is then generated and the delay required by DCR[RTIM] is inserted before the next ACTV command is generated. In this example, the next bus cycle is initiated, but does not generate an SDRAM access until $T_{RC}$ is finished. Because both chip selects are active during the REF command, it is passed to both blocks of external SDRAM.

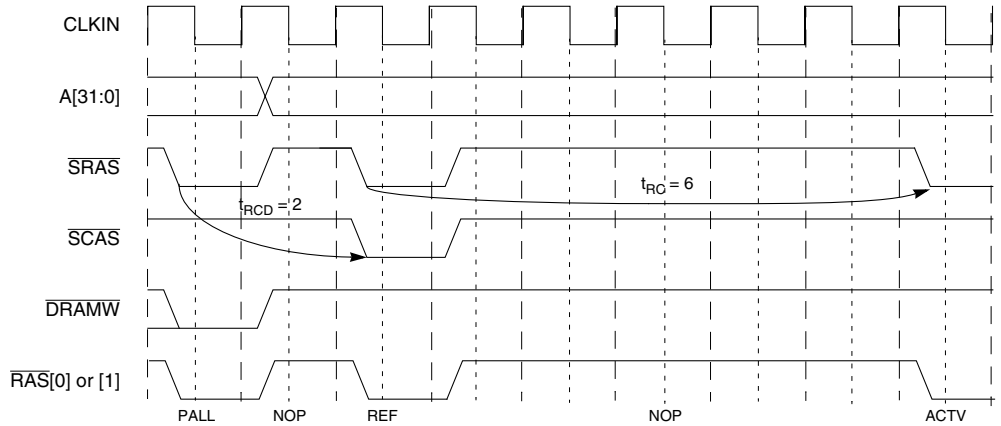



**Figure 11-22. Auto-Refresh Operation**

### 11.4.4.6 Self-Refresh Operation

Self-refresh is a method of allowing the SDRAM to enter into a low-power state, while at the same time to perform an internal refresh operation and to maintain the integrity of the data stored in the SDRAM. The DRAM controller supports self-refresh with DCR[IS]. When IS is set, the SELF command is sent to the SDRAM. When IS is cleared, the SELFX command is sent to the DRAM controller. Figure 11-23 shows the self-refresh operation.

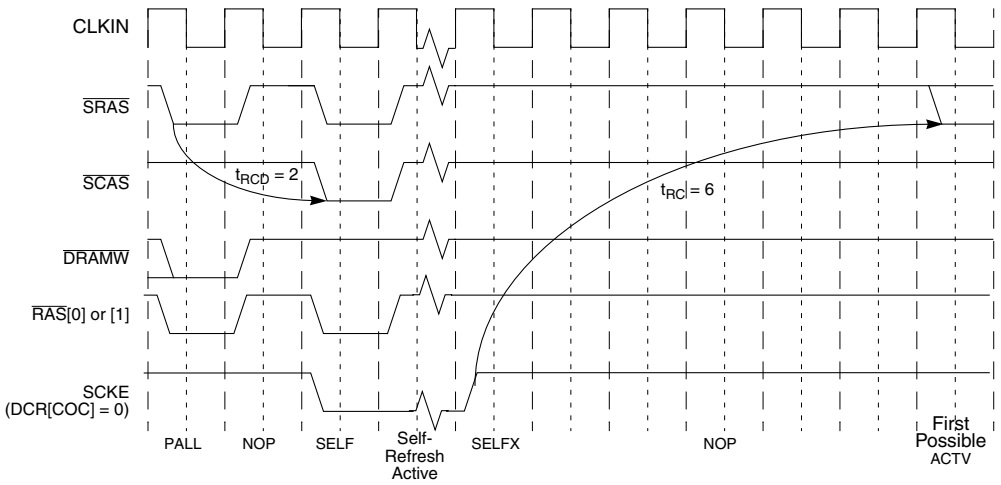



**Figure 11-23. Self-Refresh Operation**

## 11.4.5 Initialization Sequence

Synchronous DRAMs have a prescribed initialization sequence. The DRAM controller

supports this sequence with the following procedure:

1. SDRAM control signals are reset to idle state. Wait the prescribed period after reset before any action is taken on the SDRAMs. This is normally around 100 $\mu$s.
2. Initialize the DCR, DACR, and DMR in their operational configuration. Do not yet enable PALL or REF commands.
3. Issue a PALL command to the SDRAMs by setting DCR[IP] and accessing a SDRAM location. Wait the time (determined by $t_{RP}$) before any other execution.
4. Enable refresh (set DACR[RE]) and wait for at least 8 refreshes to occur.
5. Before issuing the MRS command, determine if the DMR mask bits need to be modified to allow the MRS to execute properly
6. Issue the MRS command by setting DACR[IMRS] and accessing a location in the SDRAM. Note that mode register settings are driven on the SDRAM address bus, so care must be taken to change DMR[BAM] if the mode register configuration does not fall in the address range determined by the address mask bits. After the mode register is set, DMR mask bits can be restored to their desired configuration.

### 11.4.5.1 Mode Register Settings

It is possible to configure the operation of SDRAMs, namely their burst operation and $\overline{\text{CAS}}$ latency, through the SDRAM component's mode register. $\overline{\text{CAS}}$ latency is a function of the speed of the SDRAM and the bus clock of the DRAM controller. The DRAM controller operates at a $\overline{\text{CAS}}$ latency of 1, 2, or 3.

Although the MCF5407 DRAM controller supports bursting operations, it does not use the bursting features of the SDRAMs. Because the MCF5407 can burst operand sizes of 1, 2, 4, or 16 bytes long, the concept of a fixed burst length in the SDRAMs mode register becomes problematic. Therefore, the MCF5407 DRAM controller generates the burst cycles rather than the SDRAM device. Because the MCF5407 generates a new address and a READ or WRITE command for each transfer within the burst, the SDRAM mode register should be set either to a burst length of one or to not burst. This allows bursting to be controlled by the MCF5407 instead.

The SDRAM mode register is written by setting the associated block's DACR[IMRS]. First, the base address and mask registers must be set to the appropriate configuration to allow the mode register to be set. Note that improperly set DMR mask bits may prevent access to the mode register address. Thus, the user should determine the mapping of the mode register address to the MCF5407 address bits to find out if an access is blocked. If the DMR setting prohibits mode register access, the DMR should be reconfigured to enable the access and then set to its necessary configuration after the MRS command executes.

The associated CBM bits should also be initialized. After DACR[IMRS] is set, the next access to the SDRAM address space generates the MRS command to that SDRAM. The address of the access should be selected to place the correct mode information on the SDRAM address pins. The address is not multiplexed for the MRS command. The MRS

access can be a read or write. The important thing is that the address output of that access needs the correct mode programming information on the correct address bits.

Figure 11-24 shows the MRS command, which occurs in the first clock of the bus cycle.

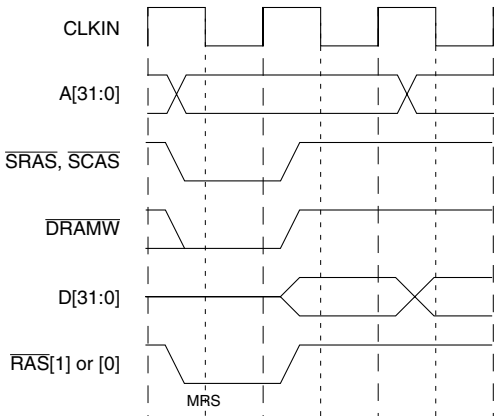



**Figure 11-24. Mode Register Set (MRS) Command**

## 11.5 SDRAM Example

This example interfaces a 2M x 32-bit x 4 bank SDRAM component to a MCF5407 operating at 40 MHz. Table 11-32 lists design specifications for this example.

**Table 11-32. SDRAM Example Specifications**

| Parameter | Specification |
|---|---|
| Speed grade (-8E) | 40 MHz (25-nS period) |
| 10 rows, 8 columns | |
| Two bank-select lines to access four internal banks | |
| ACTV-to-read/write delay ($t_{RCD}$) | 20 nS (min.) |
| Period between auto refresh and ACTV command ($t_{RC}$) | 70 nS |
| ACTV command to precharge command ($t_{RAS}$) | 48 nS (min.) |
| Precharge command to ACTV command ($t_{RP}$) | 20 nS (min.) |
| Last data input to PALL command ($t_{RWL}$) | 1 bus clock (25 nS) |
| Auto refresh period for 4096 rows ($t_{REF}$) | 64 mS |

### 11.5.1 SDRAM Interface Configuration

To interface this component to the MCF5407 DRAM controller, use the connection table that corresponds to a 32-bit port size with 8 columns (Table 11-26). Two pins select one of four banks when the part is functional. Table 11-33 shows the proper hardware hook-up.

**Table 11-33. SDRAM Hardware Connections**

| MCF5407 Pins | A15 | A14 | A13 | A12 | A11 | A10 | A9 | A17 | A18 | A19 | A20 | A21 | A22 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| SDRAM Pins | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 = CMD | BA0 | BA1 |

## 11.5.2 DCR Initialization

At power-up, the DCR has the following configuration if synchronous operation and SDRAM address multiplexing is desired.

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | | | | | | | | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | SO | res | NAM | COC | IS | RTIM | | RC | | | | | | | | |
| Setting | 1 | X | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| (hex) | 8 | | | | 0 | | | 2 | | | | 6 | | | | |

**Figure 11-25. Initialization Values for DCR**

This configuration results in a value of 0x8026 for DCR, as shown in Table 11-34.

**Table 11-34. DCR Initialization Values**

| Bits | Name | Setting | Description |
|---|---|---|---|
| 15 | SO | 1 | Indicating synchronous operation |
| 14 | — | x | Don't care (reserved) |
| 13 | NAM | 0 | Indicating SDRAM controller multiplexes address lines internally |
| 12 | COC | 0 | SCKE is used as clock enable instead of command bit because user is not multiplexing address lines externally and requires external command feed. |
| 11 | IS | 0 | At power-up, allowing power self-refresh state is not appropriate because registers are being set up. |
| 10–9 | RTIM | 00 | Because $t_{RC}$ value is 70 nS, indicating a 3-clock refresh-to-ACTV timing. |
| 8–0 | RC | 0x26 | Specification indicates auto-refresh period for 4096 rows to be 64 mS or refresh every 15.625 µs for each row, or 625 bus clocks at 40 MHz. Because DCR[RC] is incremented by 1 and multiplied by 16, RC = (625 bus clocks/16) -1 = 38.06 = 0x38 |

## 11.5.3 DACR Initialization

As shown in Figure 11-26, in this example the SDRAM is programmed to access only the second 512-Kbyte block of each 1-Mbyte partition in the SDRAM (each 16 Mbytes). The starting address of the SDRAM is 0xFF80_0000. Continuous page mode feature is used.

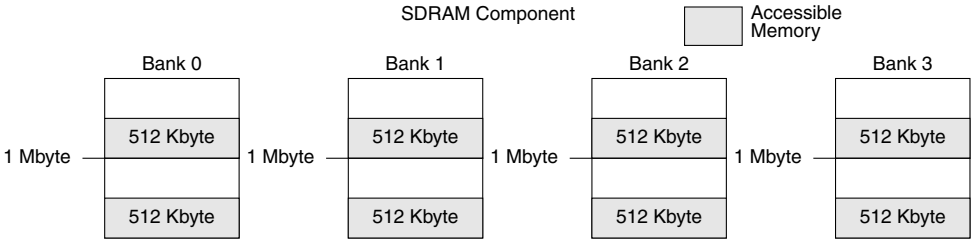



**Figure 11-26. SDRAM Configuration**

The DACRs should be programmed as shown in Figure 11-27.

| | 31 … 18 | 17 | 16 |
|---|---|---|---|
| Field | BA | — | |
| Setting | 1111_1111_1000_10 | xx | |
| (hex) | 15 | 15 | 8 | 8 |

| | 15 | 14 | 13 | 12 | 11 | 10 … 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | RE | — | CASL | | — | CBM | — | IMRS | PS | | IP | PM | — | |
| Setting | 0 | X | 00 | | X | 011 | X | 0 | 00 | | 0 | 1 | xx | |
| (hex) | 0 | | | | 3 | | 0 | | | | 4 | | | |

**Figure 11-27. DACR Register Configuration**

This configuration results in a value of DACR0 = 0xFF88_0304, as described in Table 11-35. DACR1 initialization is not needed because there is only one block. Subsequently, DACR1[RE,IMRS,IP] should be cleared; everything else is a don't care.

**Table 11-35. DACR Initialization Values**

| Bits | Name | Setting | Description |
|---|---|---|---|
| 31–18 | BA | | Base address. So DACR0[31–16] = 0xFF88, which places the starting address of the SDRAM accessible memory at 0xFF88_0000. |
| 17–16 | — | | Reserved. Don't care. |
| 15 | RE | 0 | 0, which keeps auto-refresh disabled because registers are being set up at this time. |
| 14 | — | | Reserved. Don't care. |
| 13–12 | CASL | 00 | Indicates a delay of data 1 cycle after $\overline{\text{CAS}}$ is asserted |
| 11 | — | | Reserved. Don't care. |
| 10–8 | CBM | 011 | Command bit is pin 20 and bank selects are 21 and up. |
| 7 | — | | Reserved. Don't care. |
| 6 | IMRS | 0 | Indicates MRS command has not been initiated. |
| 5–4 | PS | 00 | 32-bit port. |
| 3 | IP | 0 | Indicates precharge has not been initiated. |

**Table 11-35. DACR Initialization Values**

| Bits | Name | Setting | Description |
|---|---|---|---|
| 2 | PM | 1 | Indicates continuous page mode |
| 1–0 | — | | Reserved. Don't care. |

## 11.5.4 DMR Initialization

In this example, again, only the second 512-Kbyte block of each 1-Mbyte space is accessed in each bank. In addition the SDRAM component is mapped only to readable and writable supervisor and user data. The DMRs have the following configuration.

| | 31 | | | | | | | | | | | | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | BAM | | | | | | | | | | | | | — | |
| Setting | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | X | X |
| (hex) | 0 | | | | 0 | | | | 7 | | | | 4 | | | |

| | 15 | | | | | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | — | | | | | | | WP | — | C/I | AM | SC | SD | UC | UD | V |
| Setting | X | X | X | X | X | X | X | 0 | X | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| (hex) | 0 | | | | 0 | | | | 7 | | | | 5 | | | |

**Figure 11-28. DMR0 Register**

With this configuration, the DMR0 = 0x0074_0075, as described in Table 11-36.

**Table 11-36. DMR0 Initialization Values**

| Bits | Name | Setting | Description |
|---|---|---|---|
| 31–16 | BAM | | With bits 17 and 16 as don't cares, BAM = 0x0074, which leaves bank select bits and upper 512K select bits unmasked. Note that bits 22 and 21 are set because they are used as bank selects; bit 20 is set because it controls the 1-Mbyte boundary address. |
| 15–9 | — | | Reserved. Don't care. |
| 8 | WP | 0 | Allow reads and writes |
| 7 | — | | Reserved |
| 6 | C/I | 1 | Disable CPU space access |
| 5 | AM | 1 | Disable alternate master access |
| 4 | SC | 1 | Disable supervisor code accesses |
| 3 | SD | 0 | Enable supervisor data accesses |
| 2 | UC | 1 | Disable user code accesses |
| 1 | UD | 0 | Enable user data accesses |
| 0 | V | 1 | Enable accesses. |

## 11.5.5 Mode Register Initialization

When DACR[IMRS] is set, a bus cycle initializes the mode register. If the mode register setting is read on A[10:0] of the SDRAM on the first bus cycle, the bit settings on the corresponding MCF5407 address pins must be determined while being aware of masking requirements.

Table 11-37 lists the desired initialization setting:

**Table 11-37. Mode Register Initialization**

| MCF5407 Pins | SDRAM Pins | Mode Register Initialization | |
|---|---|---|---|
| A20 | A10 | Reserved | X |
| A19 | A9 | WB | 0 |
| A18 | A8 | Opmode | 0 |
| A17 | A7 | Opmode | 0 |
| A9 | A6 | CASL | 0 |
| A10 | A5 | CASL | 0 |
| A11 | A4 | CASL | 1 |
| A12 | A3 | BT | 0 |
| A13 | A2 | BL | 0 |
| A14 | A1 | BL | 0 |
| A15 | A0 | BL | 0 |

Next, this information is mapped to an address to determine the hexadecimal value.

| | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | | | | | | | | | | | | | | | | |
| Setting | X | X | X | X | X | X | X | X | X | X | X | X | 0 | 0 | 0 | X |
| (hex) | 0 | | | | 0 | | | | 0 | | | | 0 | | | |

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | | | | | | | | | | | | | | | | V |
| Setting | 0 | 0 | 0 | 0 | 1 | 0 | 0 | X | X | X | X | X | X | X | X | X |
| (hex) | 0 | | | | 8 | | | | 0 | | | | 0 | | | |

**Figure 11-29. Mode Register Mapping to MCF5407 A[31:0]**

Although A[31:20] corresponds to the address programmed in DACR0, according to how DACR0 and DMR0 are initialized, bit 19 must be set to hit in the SDRAM. Thus, before the mode register bit is set, DMR0[19] must be set to enable masking.

## 11.5.6 Initialization Code

The following assembly code initializes the SDRAM example.

Power-Up Sequence:

```
move.w  #0x8026, d0             //Initialize DCR
move.w  d0, DCR
move.l  #0xFF880300, d0         //Initialize DACR0
move.l  d0, DACR0
move.l  #0x00740075, d0         //Initialize DMR0
move.l  d0, DMR0
```

Precharge Sequence:

```
move.l  #0xFF880308, d0         //Set DACR0[IP]
move.l  d0, DACR0
move.l  #0xBEADDEED, d0         //Write to memory location to init. precharge
move.l  d0, 0xFF880000
```

Refresh Sequence:

```
move.l  #0xFF888300, d0         //Enable refresh bit in DACR0
move.l  d0, DACR0
```

Mode Register Initialization Sequence:

```
move.l  #0x00600075, d0         //Mask bit 19 of address
move.l  d0, DMR0
move.l  #0xFF888340, d0         //Enable DACR0[IMRS]; DACR0[RE] remains set
move.l  d0, DACR0
move.l  #0x00000000, d0         //Access  SDRAM  address  to  initialize  mode
register
move.l  d0, 0xFF800800
```

# Part III
# Peripheral Module

## Intended Audience

Part III describes the operation and configuration of the MCF5407 DMA, timer, UART, and parallel port modules, and describes how they interface with the system integration unit, described in Part II.

## Contents

It contains the following chapters:

- Chapter 12, “DMA Controller Module,” provides an overview of the DMA controller 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, showing timing diagrams for various operations.
- Chapter 13, “Timer Module,” describes configuration and operation of the two general-purpose timer modules, timer 0 and timer 1. It includes programming examples.
- Chapter 14, “UART Modules,” describes the use of the universal asynchronous/synchronous receiver/transmitters (UARTs) implemented on the MCF5407 and includes programming examples. Particular attention is given to the UART1 implementation of a synchronous interface that provides a controller for an 8- or 16-bit CODEC interface and an audio CODEC ‘97 (AC ’97) digital interface.
- Chapter 15, “Parallel Port (General-Purpose I/O),” describes the operation and programming model of the parallel port pin assignment, direction-control, and data registers. It includes a code example for setting up the parallel port.

## Suggested Reading

The following literature may be helpful with respect to the topics in Part III:

# Acronyms and Abbreviations

Table III-i describes acronyms and abbreviations used in Part III.

**Table III-i. Acronyms and Abbreviated Terms**

| Term | Meaning |
|---|---|
| ADC | Analog-to-digital conversion |
| BIST | Built-in self test |
| CODEC | Code/decode |
| DAC | Digital-to-analog conversion |
| DMA | Direct memory access |
| DSP | Digital signal processing |
| EDO | Extended data output (DRAM) |
| FIFO | First-in, first-out |
| GPIO | General-purpose I/O |
| $I^2C$ | 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 |
| LIFO | Last-in, first-out |
| LRU | Least recently used |
| LSB | Least-significant byte |
| lsb | Least-significant bit |
| MAC | Multiple accumulate unit |
| 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 |

## Table III-i. Acronyms and Abbreviated Terms (Continued)

| Term | Meaning |
|---|---|
| PCLK | Processor clock |
| PLL | Phase-locked loop |
| PLRU | Pseudo least recently used |
| POR | Power-on reset |
| PQFP | Plastic quad flat pack |
| RISC | Reduced instruction set computing |
| Rx | Receive |
| SIM | System integration module |
| SOF | Start of frame |
| TAP | Test access port |
| TTL | Transistor-to-transistor logic |
| Tx | Transmit |
| UART | Universal asynchronous/synchronous receiver transmitter |

# Chapter 12
# DMA Controller Module

This chapter describes the MCF5407 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.

## 12.1 Overview

The direct memory access (DMA) controller module provides an efficient way to move blocks of data with minimal processor interaction. The DMA module, shown in Figure 12-1, provides four channels that allow byte, word, or longword operand transfers. Each channel has a dedicated set of registers that define the source and destination addresses (SAR*n* and DAR*n*), byte count (BCR*n*), and control and status (DCR*n* and DSR*n*). Transfers can be dual or single address to off-chip devices or dual address to on-chip devices, such as UART, SDRAM controller, and parallel port.

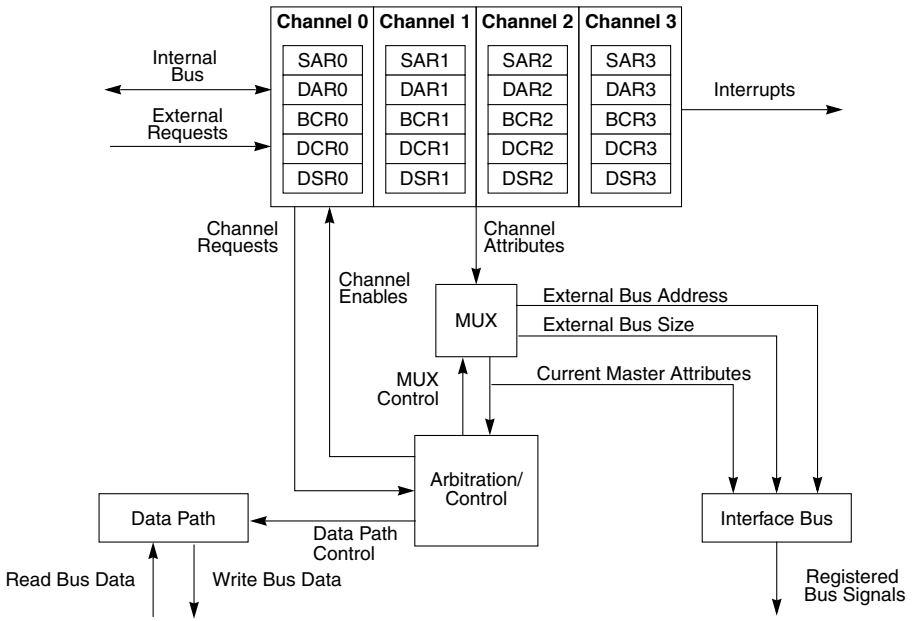



**Figure 12-1. DMA Signal Diagram**

### 12.1.1 DMA Module Features

The DMA controller module features are as follows:

- Four fully independent, programmable DMA controller channels/bus modules
- Auto-alignment feature for source or destination accesses
- Dual- and single-address transfers
- Two external request pins ($\overline{\text{DREQ}}$[1:0]) provided for channels 1 and 0
- Two external acknowledge pins (DACK[1:0]) provided for channels 1 and 0
- Channels 2 and 3 have request signals connected to the interrupt lines of UART0 and UART1, programmable through the channel select field MODCTL[DSL]. See Section 14.3.4, “Modem Control Register (MODCTL).”
- Channel arbitration on transfer boundaries
- Data transfers in 8-, 16-, 32-, or 128-bit blocks using a 16-byte buffer
- Continuous-mode and cycle-steal transfers
- Independent transfer widths for source and destination
- Independent source and destination address registers
- Data transfer can occur in as few as two clocks

## 12.2 DMA Signal Description

Table 12-1 briefly describes the DMA module signals that provide handshake control for either a source or destination external device.

**Table 12-1. DMA Signals**

| Signal | I/O | Description |
|---|---|---|
| $\overline{\text{DREQ}}$[1:0]/ PP[6:5] | I | External DMA request. $\overline{\text{DREQ}}$[1:0] can serve as the DMA request inputs or as two parallel port bits. They are programmable individually through the PAR. A peripheral device asserts these inputs to request an operand transfer between it and memory. $\overline{\text{DREQ}}$ signals are asserted to initiate DMA accesses in the respective channels. The system should drive unused $\overline{\text{DREQ}}$ signals to logic high. Although each channel has an individual $\overline{\text{DREQ}}$ signal, in the MCF5407 only channels 0 and 1 connect to external $\overline{\text{DREQ}}$ pins. $\overline{\text{DREQ2}}$ and $\overline{\text{DREQ3}}$ are programmable for use with UART0 and UART1 through MODCTL[DSL]. See Section 14.3.4, “Modem Control Register (MODCTL).” |
| TT[1:0]/ PP[1:0] | O | Transfer type. A DMA access is indicated by the transfer type pins, TT[1:0] = 01. The transfer modifier, TM[2:0], and DMA acknowledgement, DACK[1:0], configurations shown below are meaningful only if TT[1:0] = 01, indicating an external master or DMA access. |

**Table 12-1. DMA Signals (Continued)**

| Signal | I/O | Description |
|---|---|---|
| TM[2:0]/ DACK[1:0] | O | Transfer modifier/DMA acknowledge. The MCF5407 TM[2:0] encodings are like the MCF5307, with functions shifted slightly, as Figure 12-2 shows. Dedicated DMA acknowledgement pins, DACK[1:0], are added and multiplexed as follows—TM[1:0]/DACK[1:0]/PP[3:2]. TM2 is still multiplexed only with PP4. Chapter 17, “Signal Descriptions,” describes multiplexing.<br>Although on the MCF5407, TM[2:0] can be programmed to be DMA acknowledge signals, bit positions of these encodings differ from the MCF5307. Single-address access indication is now encoded on TM2 when the PAR is set to enable the transfer modifier signal and an external or DMA transfer is occurring. This encoding is driven by TM0 on the MCF5307. In addition, DMA acknowledge signals are multiplexed with TM[1:0] on the MCF5407, as opposed to TM[2:1] providing DMA transfer information on the MCF5307. The MCF5407 encoding for TM[2:0], shown below describes when PAR is set to enable these signals and the IRQPAR is programmed to disable DACK[1:0]. Note that when DACK[1:0] are driven, TM2 is still driven if it is enabled through the PAR.<br>To enable DACK[1:0], first enable TM[1:0] and then program the interrupt assignment register (IRQPAR) to enable bits 0–1. When IRQPAR[ENBDACK1] = 1 and TM1 is enabled, DACK1 for DMA channel 1 is driven in place of TM1 for DMA transfers. Clearing ENBDACK1 disables this function and only the TM1 encoding is driven. Likewise, setting ENBDACK0 enables DACK0 to be driven; clearing ENBDACK0 disables this function and drives the TM0 encoding.<br>TM2 Encoding<br>0 Single-address access negated<br>1 Single-address access<br>TM[1:0] Encoding<br>00 DMA acknowledge information not provided<br>01 DMA transfer, channel 0<br>10 DMA transfer, channel 1<br>11 Reserved<br>The DMA transfer information on TM[1:0] can be provided on every DMA transfer or only on the last transfer by programming DCR[AT]. |
| DACK[1:0] | I/O | DMA acknowledge. These signals provide an acknowledge of a DMA transfer. They can be programmed using DCR[AT] to assert on every transfer or only on the final transfer. |

Table 12-2 shows MCF5407 pin configurations based on PAR and IRQPAR configurations.

**Table 12-2. MCF5407 Signal Configurations for PP[4:2]/TM[2:0]/DACK[1:0]**

| PAR Configuration [1] | IRQPAR Configuration | PP[4:2] | TM[2:0] | DACK[1:0] |
|---|---|---|---|---|
| TM[2:0] disabled, PP[4:2] enabled | ENBDACK[1–0] = 0 or 1 | Driven | Not driven | Not driven |
| TM[2:0] enabled | ENBDACK[1–0] = 0 | Not driven | TM[2:0] driven | Not driven |
| TM[2:0] enabled | ENBDACK[1–0] = 1 | Not driven | TM2 driven only | Driven |

[1] Note that to enable DACK[1:0], PAR must first be programmed to enable TM[1:0].

Designers who used MCF5307 DMA channels should also note that the DMA byte count registers (BCR) for channels 0–3 exclusively support a 24-bit byte count. A 16-bit byte count register and MPARK[BCR24BIT] are no longer supported.

As shown in Figure 12-2, when properly connected, TM[2:0] can be used in MCF5407 designs in the same manner as they were on MCF5307 designs or DACK[1:0] can be used for DMA transfers.

| MCF5307 Function | Pin |
|---|---|
| Single/dual cycle access | TM0 |
| DMA 0 acknowledge configuration | TM1 |
| DMA 1 acknowledge configuration | TM2 |

| Pin | MCF5407 Function |
|---|---|
| TM0 | DMA 0 acknowledge |
| TM1 | DMA 1 acknowledge |
| TM2 | Single/dual cycle access |

**Figure 12-2. MCF5307/MCF5407 TM[2:0] Pin Remapping**

## 12.3 DMA Transfer Overview

The DMA module usually transfers data faster than the ColdFire core can under software control. The term ‘direct memory access’ refers to peripheral device’s ability to access system memory directly, greatly improving overall system performance. The DMA module consists of four independent, functionally equivalent channels, so references to DMA in this chapter apply to any of the channels. It is not possible to implicitly address all four channels at once. The MCF5407 on-chip peripherals do not support single-address transfers.

The processor generates DMA requests internally by setting DCR[START]; a device can generate a DMA request externally by using $\overline{\text{DREQ}}$ pins. The processor can program bus bandwidth for each channel. The channels support cycle-steal and continuous transfer modes; see Section 12.5.1, “Transfer Requests (Cycle-Steal and Continuous Modes).”

The DMA controller supports dual- and single-address transfers as follows. In both, the DMA channel supports 32 address bits and 32 data bits.

- Dual-address transfers—A dual-address transfer consists of a read followed by a write and is initiated by an internal request using the START bit or by an external device using $\overline{\text{DREQ}}$. Two types of transfer can occur, a read from a source device or a write to a destination device; see Figure 12-3.

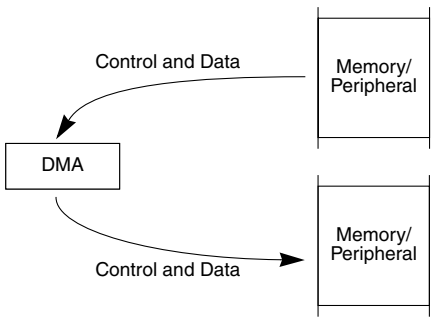



**Figure 12-3. Dual-Address Transfer**

- Single-address transfers—An external device can initiate a single-address transfer by asserting $\overline{\text{DREQ}}$. The MCF5407 provides address and control signals for single-address transfers. The external device reads to or writes from the specified address, as Figure 12-4 shows. External logic is required.

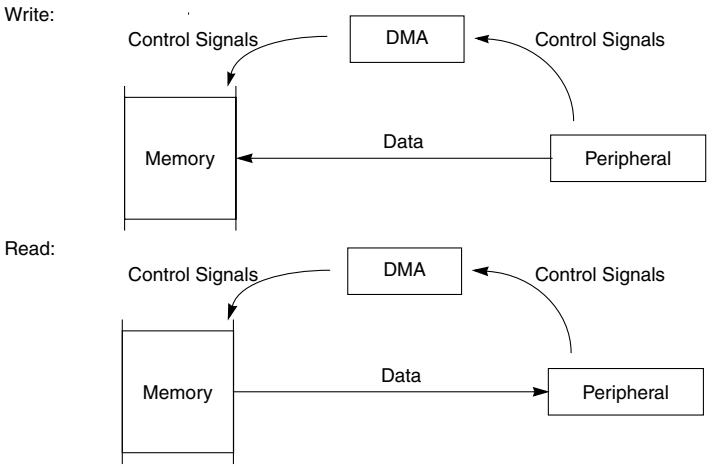



**Figure 12-4. Single-Address Transfers**

Any operation involving the DMA module follows the same three steps:

1. Channel initialization—Channel registers are loaded with control information, address pointers, and a byte-transfer count.
2. Data transfer—The DMA accepts requests for operand transfers and provides addressing and bus control for the transfers.
3. Channel termination—Occurs after the operation is finished, either successfully or due to an error. The channel indicates the operation status in the channel's DSR, described in Section 12.4.5, "DMA Status Registers (DSR0–DSR3)."

## 12.4 DMA Controller Module Programming Model

This section describes each internal register and its bit assignment. Note that there is no way to prevent a write to a control register during a DMA transfer. Table 12-3 shows the mapping of DMA controller registers.

**Table 12-3. Memory Map for DMA Controller Module Registers**

| DMA Channel | MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|---|
| 0 | 0x300 | Source address register 0 (SAR0) [p. 12-7] | | | |
| | 0x304 | Destination address register 0 (DAR0) [p. 12-7] | | | |
| | 0x308 | DMA control register 0 (DCR0) [p. 12-8] | | | |
| | 0x30C | Reserved | Byte count register 0 (BCR0) [p. 12-7] | | |
| | 0x310 | DMA status register 0 (DSR0) [p. 12-10] | Reserved | | |
| | 0x314 | DMA interrupt vector register 0 (DIVR0) [p. 12-11] | Reserved | | |
| 1 | 0x340 | Source address register 1 (SAR1) [p. 12-7] | | | |
| | 0x344 | Destination address register 1 (DAR1) [p. 12-7] | | | |
| | 0x348 | DMA control register 1 (DCR1) [p. 12-8] | | | |
| | 0x34C | Reserved | Byte count register 1 (BCR1) [p. 12-7] | | |
| | 0x350 | DMA status register 1 (DSR1) [p. 12-10] | Reserved | | |
| | 0x354 | DMA interrupt vector register 1 (DIVR1) [p. 12-11] | Reserved | | |
| 2 | 0x380 | Source address register 2 (SAR2) [p. 12-7] | | | |
| | 0x384 | Destination address register 2 (DAR2) [p. 12-7] | | | |
| | 0x388 | DMA control register 2 (DCR2) [p. 12-8] | | | |
| | 0x38C | Reserved | Byte count register 2 (BCR2) [p. 12-7] | | |
| | 0x390 | DMA status register 2 (DSR2) [p. 12-10] | Reserved | | |
| | 0x394 | DMA interrupt vector register 2 (DIVR2) [p. 12-11] | Reserved | | |
| 3 | 0x3C0 | Source address register 3 (SAR3) [p. 12-7] | | | |
| | 0x3C4 | Destination address register 3 (DAR3) [p. 12-7] | | | |
| | 0x3C8 | DMA control register 3 (DCR3) [p. 12-8] | | | |
| | 0x3CC | Reserved | Byte count register 3 (BCR3) [p. 12-7] | | |
| | 0x3D0 | DMA status register 3 (DSR3) [p. 12-10] | Reserved | | |
| | 0x3D4 | DMA interrupt vector register 3 (DIVR3) [p. 12-11] | Reserved | | |

**NOTE:**

External masters cannot access MCF5407 on-chip memories or MBAR, but they can access DMA module registers.

### 12.4.1 Source Address Registers (SAR0–SAR3)

SAR*n*, Figure 12-5, contains the address from which the DMA controller requests data. In single-address mode, SAR*n* provides the address regardless of the direction.

| | 31 | 0 |
|---|---|---|
| Field | SAR | |
| Reset | 0000_0000_0000_0000_0000_0000_0000_0000 | |
| R/W | R/W | |
| Address | MBAR + 0x300, 0x340, 0x380, 0x3C0 | |

**Figure 12-5. Source Address Registers (SARn)**

**NOTE:**

SAR/DAR address ranges cannot be programmed to on-chip SRAM because it cannot be accessed by on-chip DMA.

### 12.4.2 Destination Address Registers (DAR0–DAR3)

For dual-address transfers only, DAR*n*, Figure 12-6, holds the address to which the DMA controller sends data.

| | 31 | 0 |
|---|---|---|
| Field | DAR | |
| Reset | 0000_0000_0000_0000_0000_0000_0000_0000 | |
| R/W | R/W | |
| Address | MBAR + 304, 0x344, 0x384, 0x3C4 | |

**Figure 12-6. Destination Address Registers (DARn)**

**NOTE:**

On-chip DMAs do not maintain coherency with MCF5407 caches and so must not transfer data to cacheable memory.

### 12.4.3 Byte Count Registers (BCR0–BCR3)

BCR*n*, Figure 12-7, holds the number of bytes yet to be transferred for a given block. BCR*n* decrements on the successful completion of the address transfer of either a write transfer in dual-address mode or any transfer in single-address mode. BCR*n* decrements by 1, 2, 4, or 16 for byte, word, longword, or line accesses, respectively.

| | 31 … 24 | 23 … 0 |
|---|---|---|
| Field | — | BCR |
| Reset | — | 0000_0000_0000_0000_0000_0000 |
| R/W | R/W | |
| Address | MBAR + 0x30C, 0x34C, 0x38C, 0x3AC | |

**Figure 12-7. Byte Count Registers (BCRn)**

DSR[DONE], shown in Figure 12-9, is set when the block transfer is complete.

When a transfer sequence is initiated and BCR*n*[BCR] is not divisible by 16, 4, or 2 when the DMA is configured for line, longword, or word transfers, respectively, DSR*n*[CE] is set and no transfer occurs. See Section 12.4.5, “DMA Status Registers (DSR0–DSR3).”

## 12.4.4 DMA Control Registers (DCR0–DCR3)

DCR*n*, Figure 12-8, is used for configuring the DMA controller module.

| | 31 | 30 | 29 | 28 | 27 … 25 | 24 | 23 | 22 | 21 … 20 | 19 | 18 … 17 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | INT | EEXT | CS | AA | BWC | SAA | S_RW | SINC | SSIZE | DINC | DSIZE | START |
| Reset | 0000_0000_0000_0000 | | | | | | | | | | | |
| R/W | R/W | | | | | | | | | | | |

| | 15 | 14 … 0 |
|---|---|---|
| Field | AT | — |
| Reset | 0 | N/A |
| R/W | R/W | |
| Address | MBAR + 0x308, 0x348, 0x388, 0x3A8 | |

**Figure 12-8. DMA Control Registers (DCRn)**

Table 12-4 describes DCR fields.

**Table 12-4. DCRn Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 31 | INT | Interrupt on completion of transfer. Determines whether an interrupt is generated by completing a transfer or by the occurrence of an error condition.<br>0 No interrupt is generated.<br>1 Internal interrupt signal is enabled. |
| 30 | EEXT | Enable external request. Care should be taken because a collision can occur between the START bit and $\overline{\text{DREQ}}$ when EEXT = 1.<br>0 External request is ignored.<br>1 Enables external request to initiate transfer. Internal request is always enabled. It is initiated by writing a 1 to the START bit. |

**Table 12-4. DCRn Field Descriptions (Continued)**

| Bits | Name | Description |
|---|---|---|
| 29 | CS | Cycle steal.<br>0 DMA continuously makes read/write transfers until the BCR decrements to 0.<br>1 Forces a single read/write transfer per request. The request may be internal by setting the START bit, or external by asserting $\overline{\text{DREQ}}$. |
| 28 | AA | Auto-align. AA and SIZE determine whether the source or destination is auto-aligned, that is, transfers are optimized based on the address and size. See Section 12.5.4.2, “Auto-Alignment.”<br>0 Auto-align disabled<br>1 If SSIZE indicates a transfer no smaller than DSIZE, source accesses are auto-aligned; otherwise, destination accesses are auto-aligned. Source alignment takes precedence over destination alignment. If auto-alignment is enabled, the appropriate address register increments, regardless of DINC or SINC. |
| 27–25 | BWC | Bandwidth control. Indicates the number of bytes in a block transfer. When the byte count reaches a multiple of the BWC value, the DMA releases the bus.<br>000 DMA has priority. It does not negate its request until its transfer completes.<br>001 16384<br>010 32768<br>011 65536<br>100 131072<br>101 262144<br>110 524288<br>111 1048576 |
| 24 | SAA | Single-address access. Determines whether the DMA channel is in dual- or single-address mode<br>0 Dual-address mode.<br>1 Single-address mode. The DMA provides an address from the SAR and directional control, bit S_RW, to allow two peripherals (one might be memory) to exchange data within a single access. Data is not stored by the DMA. |
| 23 | S_RW | Single-address access read/write value. Valid only if SAA = 1. Specifies the value of the read signal during single-address accesses. This provides directional control to the bus controller.<br>0 Forces the read signal to 0.<br>1 Forces the read signal to 1. |
| 22 | SINC | Source increment. Controls whether a source address increments after each successful transfer.<br>0 No change to SAR after a successful transfer.<br>1 The SAR increments by 1, 2, 4, or 16, as determined by the transfer size. |
| 21–20 | SSIZE | Source size. Determines the data size of the source bus cycle for the DMA control module.<br>00 Longword<br>01 Byte<br>10 Word<br>11 Line |
| 19 | DINC | Destination increment. Controls whether a destination address increments after each successful transfer.<br>0 No change to the DAR after a successful transfer.<br>1 The DAR increments by 1, 2, 4, or 16, depending upon the size of the transfer. |
| 18–17 | DSIZE | Destination size. Determines the data size of the destination bus cycle for the DMA controller.<br>00 Longword<br>01 Byte<br>10 Word<br>11 Line |
| 16 | START | Start transfer.<br>0 DMA inactive<br>1 The DMA begins the transfer in accordance to the values in the control registers. START is cleared automatically after one clock and is always read as logic 0. |

**Table 12-4. DCRn Field Descriptions (Continued)**

| Bits | Name | Description |
|---|---|---|
| 15 | AT | DMA acknowledge type. Controls whether acknowledge information is provided for the entire transfer or only the final transfer.<br>0 Entire transfer. DMA acknowledge information is displayed anytime the channel is selected as the result of an external request.<br>1 Final transfer (when BCR reaches zero). For dual-address transfer, the acknowledge information is displayed for both the read and write cycles. |
| 14–0 | — | Reserved, should be cleared. |

## 12.4.5 DMA Status Registers (DSR0–DSR3)

In response to an event, the DMA controller writes to the appropriate DSR*n* bit, Figure 12-9. Only a write to DSR*n*[DONE] results in action.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field | — | CE | BES | BED | — | REQ | BSY | DONE |
| Reset | — | 0 | 0 | 0 | — | 0 | 0 | 0 |
| R/W | R/W | | | | | | | |
| Address | MBAR + 0x310, 0x350, 0x390, 0x3D0 | | | | | | | |

**Figure 12-9. DMA Status Registers (DSRn)**

Table 12-5 describes DSR*n* fields.

**Table 12-5. DSRn Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7 | — | Reserved, should be cleared. |
| 6 | CE | Configuration error. Occurs when BCR, SAR, or DAR does not match the requested transfer size, or if BCR = 0 when the DMA receives a start condition. CE is cleared at hardware reset or by writing a 1 to DSR[DONE].<br>0 No configuration error exists.<br>1 A configuration error has occurred. |
| 5 | BES | Bus error on source<br>0 No bus error occurred.<br>1 The DMA channel terminated with a bus error either during the read portion of a transfer or during an access in single-address mode (SAA = 1). |
| 4 | BED | Bus error on destination<br>0 No bus error occurred.<br>1 The DMA channel terminated with a bus error during the write portion of a transfer. |
| 3 | — | Reserved, should be cleared. |
| 2 | REQ | Request<br>0 No request is pending or the channel is currently active. Cleared when the channel is selected.<br>1 The DMA channel has a transfer remaining and the channel is not selected. |

**Table 12-5. DSRn Field Descriptions (Continued)**

| Bits | Name | Description |
|---|---|---|
| 1 | BSY | Busy<br>0 DMA channel is inactive. Cleared when the DMA has finished the last transaction.<br>1 BSY is set the first time the channel is enabled after a transfer is initiated. |
| 0 | DONE | Transactions done. Set when all DMA controller transactions complete normally, as determined by transfer count and error conditions. When BCR reaches zero, DONE is set when the final transfer completes successfully. DONE can also be used to abort a transfer by resetting the status bits. When a transfer completes, software must clear DONE before reprogramming the DMA.<br>0 Writing or reading a 0 has no effect.<br>1 DMA transfer completed. Writing a 1 to this bit clears all DMA status bits and can be used as an interrupt handler to clear the DMA interrupt and error bits. |

### 12.4.6 DMA Interrupt Vector Registers (DIVR0–DIVR3)

The contents of a DMA interrupt vector register (DIVR*n*), Figure 12-10, are driven onto the internal bus in response to an interrupt acknowledge cycle.

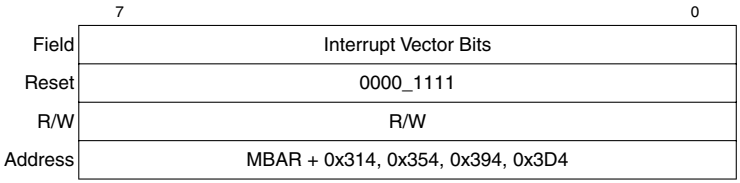



**Figure 12-10. DMA Interrupt Vector Registers (DIVRn)**

## 12.5 DMA Controller Module Functional Description

In the following discussion, the term ‘DMA request’ implies that DCR[START] or DCR[EEXT] is set, followed by assertion of $\overline{\text{DREQ}}$. The START bit is cleared when the channel begins an internal access.

Before initiating a dual-address access, the DMA module verifies that DCR[SSIZE,DSIZE] are consistent with the source and destination addresses. If the source and destination are not the same size, the configuration error bit, DSR[CE], is also set. If misalignment is detected, no transfer occurs, CE is set, and, depending on the DCR configuration, an interrupt event is issued. Note that if the auto-align bit, DCR[AA], is set, error checking is performed on appropriate registers.

A read/write transfer reads bytes from the source address and writes them to the destination address. The number of bytes is the larger of the sizes specified by SSIZE and DSIZE. See Section 12.4.4, “DMA Control Registers (DCR0–DCR3).”

Source and destination address registers (SAR and DAR) can be programmed in the DCR to increment at the completion of a successful transfer. BCR decrements when an address transfer write completes for a single-address access (DCR[SAA] = 0) or when SAA = 1.

### 12.5.1 Transfer Requests (Cycle-Steal and Continuous Modes)

The DMA channel supports internal and external requests. A request is issued by setting DCR[START] or by asserting $\overline{\text{DREQ}}$. Setting DCR[EEXT] enables recognition of external interrupts. Internal interrupts are always recognized. Bus usage is minimized for either internal or external requests by selecting between cycle-steal and continuous modes.

- Cycle-steal mode (DCR[CS] = 1)—Only one complete transfer from source to destination occurs for each request. If DCR[EEXT] is set, a request can be either internal or external. Internal request is selected by setting DCR[START]. An external request is initiated by asserting $\overline{\text{DREQ}}$ while EEXT is set.
- Continuous mode (DCR[CS] = 0)—After an internal or external request, the DMA continuously transfers data until BCR reaches zero or a multiple of DCR[BWC] or DSR[DONE] is set. If BCR is a multiple of BWC, the DMA request signal is negated until the bus cycle terminates to allow the internal arbiter to switch masters. DCR[BWC] = 000 specifies the maximum transfer rate; other values specify a transfer rate limit.

  The DMA performs the specified number of transfers, then relinquishes bus control. The DMA negates its internal bus request on the last transfer before the BCR reaches a multiple of the boundary specified in BWC. On completion, the DMA reasserts its bus request to regain mastership at the earliest opportunity. The minimum time that the DMA loses bus control is one bus cycle.

### 12.5.2 Data Transfer Modes

Each channel supports dual- and single-address transfers, described in the next sections.

#### 12.5.2.1 Dual-Address Transfers

Dual-address transfers consist of a source operand read and a destination operand write. The DMA controller module begins a dual-address transfer sequence when DCR[SAA] is cleared during a DMA request. If no error condition exists, DSR[REQ] is set.

- Dual-address read—The DMA controller drives the SAR value onto the internal address bus. If DCR[SINC] is set, the SAR increments by the appropriate number of bytes upon a successful read cycle. When the appropriate number of read cycles complete (multiple reads if the destination size is wider than the source), the DMA initiates the write portion of the transfer.

  If a termination error occurs, DSR[BES,DONE] are set and DMA transactions stop.
- Dual-address write—The DMA controller drives the DAR value onto the address bus. If DCR[DINC] is set, DAR increments by the appropriate number of bytes at the completion of a successful write cycle. The BCR decrements by the appropriate number of bytes. DSR[DONE] is set when BCR reaches zero. If the BCR is greater

than zero, another read/write transfer is initiated. If the BCR is a multiple of DCR[BWC], the DMA request signal is negated until termination of the bus cycle to allow the internal arbiter to switch masters.

If a termination error occurs, DSR[BES,DONE] are set and DMA transactions stop.

#### 12.5.2.2 Single-Address Transfers

Single-address transfers consist of one DMA bus cycle, allowing either a read or a write cycle to occur. The DMA controller begins a single-address transfer sequence when DCR[SAA] is set during a DMA request. If no error condition exists, DSR[REQ] is set. When the channel is enabled, DSR[BSY] is set and REQ is cleared. SAR contents are then driven onto the address bus and the value of DCR[S_RW] is driven on R/$\overline{W}$. The BCR decrements on each successful address access until it is zero, when DSR[DONE] is set.

If a termination error occurs, DSR[BES,DONE] are set and DMA transactions stop.

### 12.5.3 Channel Initialization and Startup

Before a block transfer starts, channel registers must be initialized with information describing configuration, request-generation method, and the data block.

#### 12.5.3.1 Channel Prioritization

The four DMA channels are prioritized in ascending order (channel 0 having highest priority and channel 3 having the lowest) or as determined by DCR[BWC]. If BWC for a DMA channel is 000, that channel has priority only over the channel immediately preceding it. For example, if DCR3[BWC] = 000, DMA channel 3 has priority over DMA channel 2 (assuming DCR2[BWC] ≠ 000) but not over DMA channel 1.

If DCR1[BWC] = DCR2[BWC] = 000, DMA 1 has priority over DMA 0 and DMA 2. DCR2[BWC] = 000 in this case does not affect prioritization.

Prioritization of simultaneous external requests is either ascending or as determined by each channel's BWC bits as described in the previous paragraphs.

#### 12.5.3.2 Programming the DMA Controller Module

Note the following general guidelines for programming the DMA:

- No mechanism exists to prevent writes to control registers during DMA accesses.
- If the BWC of sequential channels are equal, channel priority is in ascending order.

The SAR is loaded with the source (read) address. If the transfer is from a peripheral device to memory, the source address is the location of the peripheral data register. If the transfer is from memory to either a peripheral device or memory, the source address is the starting address of the data block. This can be any aligned byte address. In single-address mode, this data register is used regardless of transfer direction.

The DAR should contain the destination (write) address. If the transfer is from a peripheral device to memory, or memory to memory, the DAR is loaded with the starting address of the data block to be written. If the transfer is from memory to a peripheral device, DAR is loaded with the address of the peripheral data register. This address can be any aligned byte address. DAR is not used in single-address mode.

SAR and DAR change after each cycle depending on DCR[SSIZE,DSIZE,SINC,DINC] and on the starting address. Increment values can be 1, 2, 4, or 16 for byte, word, longword, or line transfers, respectively. If the address register is programmed to remain unchanged (no count), the register is not incremented after the data transfer.

BCR*n*[BCR] must be loaded with the number of byte transfers to occur. It is decremented by 1, 2, 4, or 16 at the end of each transfer, depending on the transfer size. DSR must be cleared for channel startup.

As soon as the channel has been initialized, it is started by writing a one to DCR[START] or asserting $\overline{\text{DREQ}}$, depending on the status of DCR[EEXT]. Programming the channel for internal request causes the channel to request the bus and start transferring data immediately. If the channel is programmed for external request, $\overline{\text{DREQ}}$ must be asserted before the channel requests the bus.

Changes to DCR are effective immediately while the channel is active. To avoid problems with changing a DMA channel setup, write a one to DSR[DONE] to stop the DMA channel.

## 12.5.4 Data Transfer

This section includes timing diagrams that illustrate the interaction of signals in DMA data transfers. It also describes auto-alignment and bandwidth control.

### 12.5.4.1 External Request and Acknowledge Operation

Channels 0 and 1 initiate transfers to an external module by means of $\overline{\text{DREQ}}$[1:0]. The request for channels 2 and 3 are connected internally to the UART0 and UART1 interrupt signals, respectively. If DCR[EEXT] = 1 and the channel is idle, the DMA initiates a transfer when $\overline{\text{DREQ}}$ is asserted.

Figure 12-11 shows the minimum 4-clock cycle delay from when $\overline{\text{DREQ}}$ is sampled asserted to when a DMA bus cycle begins. This delay may be longer, depending on DMA priority, bus arbitration, DRAM refresh operations, and other factors.

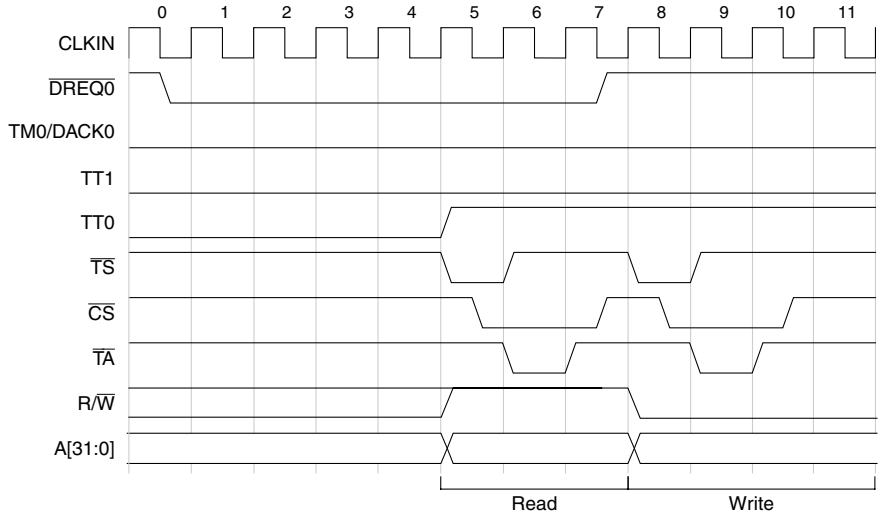



**Figure 12-11. DREQ Timing Constraints, Dual-Address DMA Transfer**

Although Figure 12-11 does not show TM0/DACK0 signaling a DMA acknowledgement, this signal can provide an external request acknowledge response, as shown in subsequent diagrams.

To initiate a request, DREQ need only be asserted long enough to be sampled on one rising clock edge. However, note the following regarding the negation of DREQ:

- In cycle-steal mode (DCR[CS] = 1), the read/write transaction is limited to a single transfer. DREQ must be negated appropriately to avoid generating another request.
  - For dual-address transfers, DREQ must be negated before TS is asserted for the write portion, as shown in Figure 12-11, clock cycle 7.
  - For single-address transfers, DREQ must be negated before TS is asserted for the transfer, as shown in Figure 12-13, clock cycle 4.
- In burst mode, (DCR[CS] = 0), multiple read/write transfers can occur on the bus as programmed. DREQ need not be negated until DSR[DONE] is set, indicating the block transfer is complete. Another transfer cannot be initiated until the DMA registers are reprogrammed.

Figure 12-12 shows a dual-address, peripheral-to-SDRAM DMA transfer. The DMA is not parked on the bus, so the diagram shows how the CPU can generate multiple bus cycles during DMA transfers. It also shows TM0/DACK0 timing. The TT signals indicate whether the CPU (0) or DMA (1) has bus mastership. TM2 indicates dual-address mode.

If DCR[AT] is 1, TM/DACK is asserted during the final transfer. If DCR[AT] is 0, TM/DACK asserts during all DMA accesses.

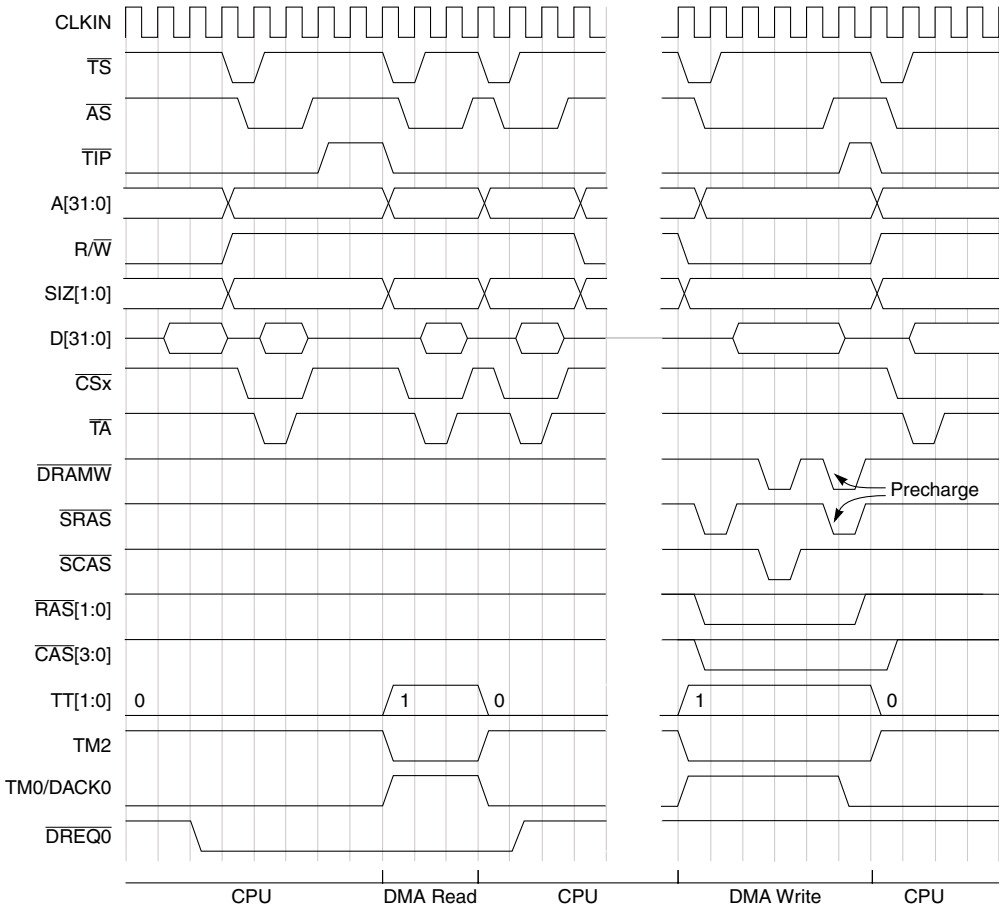



**Figure 12-12. Dual-Address, Peripheral-to-SDRAM, Lower-Priority DMA Transfer**

Figure 12-13 shows a single-address DMA transfer in which the peripheral is reading from memory. Note that TM2 is high, indicating a single-address transfer. Note that $\overline{\text{DREQ}}$ is negated in clock 4, before the assertion of $\overline{\text{TS}}$ in clock 6.

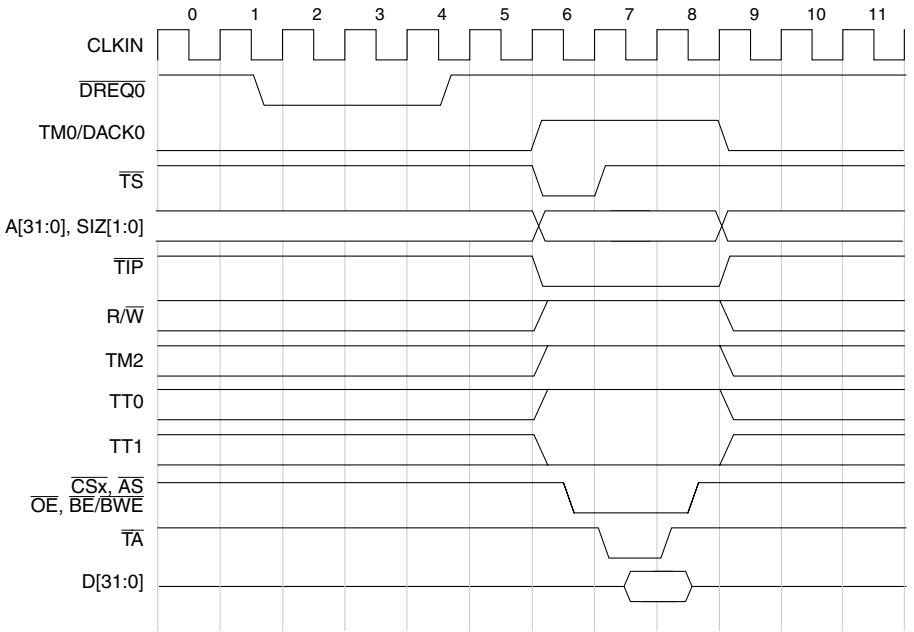



**Figure 12-13. Single-Address DMA Transfer**

### 12.5.4.2 Auto-Alignment

Auto-alignment allows block transfers to occur at the optimal size based on the address, byte count, and programmed size. To use this feature, DCR[AA] must be set. The source is auto-aligned if SSIZE indicates a transfer size larger than DSIZE. Source alignment takes precedence over the destination when the source and destination sizes are equal. Otherwise, the destination is auto-aligned. The address register chosen for alignment increments regardless of the increment value. Configuration error checking is performed on registers not chosen for alignment.

If BCR is greater than 16, the address determines transfer size. Bytes, words, or longwords are transferred until the address is aligned to the programmed size boundary, at which time accesses begin using the programmed size.

If BCR is less than 16 at the start of a transfer, the number of bytes remaining dictates transfer size. For example, AA = 1, SAR = 0x0001, BCR = 0x00F0, SSIZE = 00 (longword), and DSIZE = 01 (byte). Because SSIZE > DSIZE, the source is auto-aligned. Error checking is performed on destination registers. The access sequence is as follows:

1. Read byte from 0x0001—write 1 byte, increment SAR.
2. Read word from 0x0002—write 2 bytes, increment SAR.
3. Read longword from 0x0004—write 4 bytes, increment SAR.
4. Repeat longwords until SAR = 0x00F0.

5. Read byte from 0x00F0—write byte, increment SAR.

If DSIZE is another size, data writes are optimized to write the largest size allowed based on the address, but not exceeding the configured size.

#### 12.5.4.3 Bandwidth Control

Bandwidth control makes it possible to force the DMA off the bus to allow access to another device. DCR[BWC] provides seven levels of block transfer sizes. If the BCR decrements to a multiple of the decode of the BWC, the DMA bus request negates until the bus cycle terminates. If a request is pending, the arbiter may then pass bus mastership to another device. If auto-alignment is enabled, DCR[AA] = 1, the BCR may skip over the programmed boundary, in which case, the DMA bus request is not negated.

If BWC = 000, the request signal remains asserted until BCR reaches zero. DMA has priority over the core. Note that in this scheme, the arbiter can always force the DMA to relinquish the bus. See Section 6.2.10.1, “Default Bus Master Park Register (MPARK).”

### 12.5.5 Termination

An unsuccessful transfer can terminate for one of the following reasons:

- Error conditions—When the MCF5407 encounters a read or write cycle that terminates with an error condition, DSR[BES] is set for a read and DSR[BED] is set for a write before the transfer is halted. If the error occurred in a write cycle, data in the internal holding register is lost.
- Interrupts—If DCR[INT] is set, the DMA drives the appropriate internal interrupt signal. The processor can read DSR to determine whether the transfer terminated successfully or with an error. DSR[DONE] is then written with a one to clear the interrupt and the DONE and error bits.

# Chapter 13
# Timer Module

This chapter describes the configuration and operation of the two general-purpose timer modules (timer 0 and timer 1). It includes programming examples.

## 13.1 Overview

The timer module incorporates two independent, general-purpose 16-bit timers, timer 0 and timer 1. The output of an 8-bit prescaler clocks each timer. There are two sets of registers, one for each timer. The timers can operate from CLKIN or from an external clocking source using one of the TIN signals. If CLKIN is selected, it can be divided by 16 or 1.

Figure 13-1 is a block diagram of one of the two identical ti5mer modules.

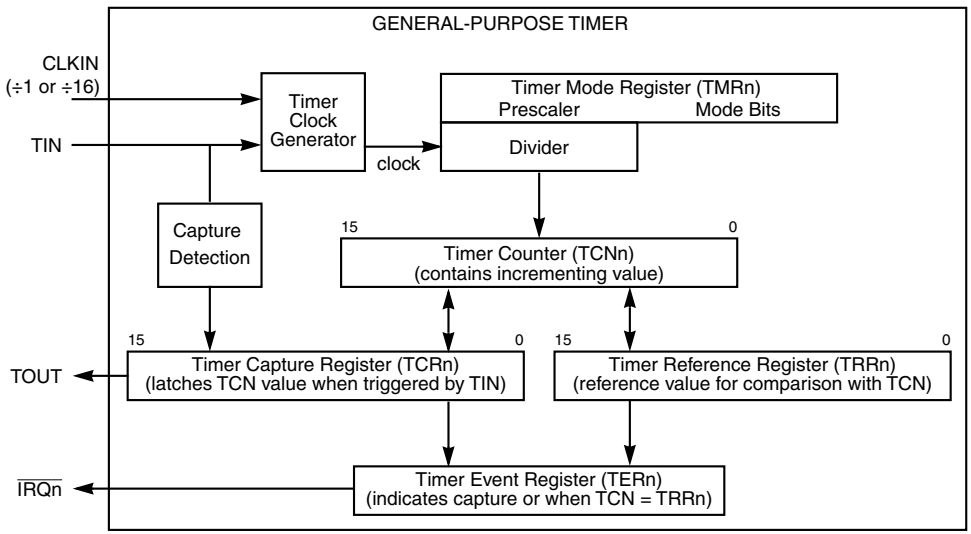



**Figure 13-1. Timer Block Diagram**

### 13.1.1 Key Features

Each general-purpose 16-bit timer unit has the following features:

- Maximum period of 4.97 seconds at 54 MHz
- 18.5-nS resolution at 54 MHz
- Programmable sources for the clock input, including external clock
- 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 on input capture or reference-compare

## 13.2 General-Purpose Timer Units

The general-purpose timer units provide the following features:

- Each timer can be programmed to count and compare to a reference value stored in a register or capture the timer value at an edge detected on TIN.
- System bus clock can be divided by 16 or 1. This clock is input to the prescaler.
- TIN is fed directly into the 8-bit prescaler. The maximum value of TIN is 1/5 of CLKIN, as described in Chapter 20, “Electrical Specifications.”
- The 8-bit prescaler clock divides the clocking source and is user-programmable from 1 to 256.
- Programmed events generate interrupts.
- The timer output signal (TOUT) can be configured to toggle or pulse on an event.

## 13.3 General-Purpose Timer Programming Model

The following features are programmable through the timer registers, shown in Table 13-1:

- Prescaler—The prescaler clock input is selected from CLKIN (divided by 1 or 16) or from the corresponding timer input, TIN. TIN is synchronized to CLKIN. The synchronization delay is between two and three CLKIN clocks. The corresponding TMR*n*[ICLK] selects the clock input source. A programmable prescaler divides the clock input by values from 1 to 256. The prescaler is an input to the 16-bit counter.
- Capture mode—Each timer has a 16-bit timer capture register (TCR0 and TCR1) that latches the counter value when the corresponding input capture edge detector senses a defined TIN transition. The capture edge bits (TMR*n*[CE]) select the type of transition that triggers the capture, sets the timer event register capture event bit, TER*n*[CAP], and issues a maskable interrupt.

- Reference compare—A timer can be configured to count up to a reference value, at which point TER*n*[REF] is set. If TMR*n*[ORI] is one, an interrupt is issued. If the free run/restart bit TMR*n*[FRR] is set, a new count starts. If it is clear, the timer keeps running.
- Output mode—When a timer reaches the reference value selected by TMR*n*[OM], it can send an output signal on TOUT*n*. TOUT*n* can be an active-low pulse or a toggle of the current output under program control.

**NOTE:**

Although external devices cannot access MCF5407 on-chip memories or MBAR, they can access timer module registers.

The timer module registers, shown in Table 13-1, can be modified at any time.

**Table 13-1. General-Purpose Timer Module Memory Map**

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| 0x140 | Timer 0 mode register (TMR0) [p. 13-3] | | Reserved | |
| 0x144 | Timer 0 reference register (TRR0) [p. 13-4] | | Reserved | |
| 0x148 | Timer 0 capture register (TCR0) [p. 13-4] | | Reserved | |
| 0x14C | Timer 0 counter (TCN0) [p. 13-5] | | Reserved | |
| 0x150 | Reserved | Timer 0 event register (TER0) [p. 13-5] | Reserved | |
| 0x180 | Timer 1 mode register (TMR1) [p. 13-3] | | Reserved | |
| 0x184 | Timer 1 reference register (TRR1) [p. 13-4] | | Reserved | |
| 0x188 | Timer 1 capture register (TCR1) [p. 13-4] | | Reserved | |
| 0x18C | Timer 1 counter (TCN1) [p. 13-5] | | Reserved | |
| 0x190 | Reserved | Timer 1 event register (TER1) [p. 13-5] | Reserved | |

## 13.3.1 Timer Mode Registers (TMR0/TMR1)

Timer mode registers (TMR0/TMR1), Figure 13-2, program the prescaler and various timer modes.

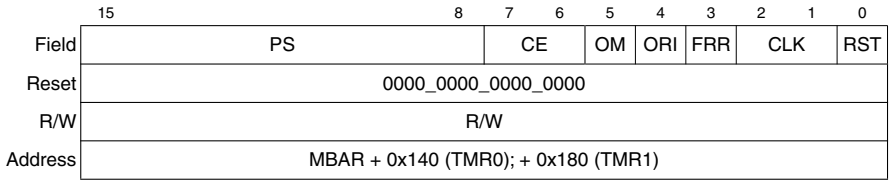



**Figure 13-2. Timer Mode Registers (TMR0/TMR1)**

Table 13-2 describes TMR*n* fields.

**Table 13-2. TMRn Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 15–8 | PS | Prescaler value. The prescaler is programmed to divide the clock input (CLKIN/(16 or 1) or clock on TIN) by values from 1 (PS = 0000_0000) to 256 (PS = 1111_1111). |
| 7–6 | CE | Capture edge and enable interrupt<br>00 Disable interrupt on capture event<br>01 Capture on rising edge only and enable interrupt on capture event<br>10 Capture on falling edge only and enable interrupt on capture event<br>11 Capture on any edge and enable interrupt on capture event |
| 5 | OM | Output mode<br>0 Active-low pulse for one CLKIN cycle (18.5 ns at 54 MHz).<br>1 Toggle output. |
| 4 | ORI | Output reference interrupt enable. If ORI is set when TERn[REF] = 1, an interrupt occurs.<br>0 Disable interrupt for reference reached (does not affect interrupt on capture function).<br>1 Enable interrupt upon reaching the reference value. |
| 3 | FRR | Free run/restart<br>0 Free run. Timer count continues to increment after reaching the reference value.<br>1 Restart. Timer count is reset immediately after reaching the reference value. |
| 2–1 | CLK | Input clock source for the timer<br>00 Stop count<br>01 System bus clock divided by 1<br>10 System bus clock divided by 16. Note that this clock source is not synchronized to the timer; thus successive time-outs may vary slightly.<br>11 TIN 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.<br>0 Reset timer (software reset)<br>1 Enable timer |

## 13.3.2 Timer Reference Registers (TRR0/TRR1)

Each timer reference register (TRR0/TRR1), Figure 13-3, contains the reference value compared with the respective free-running timer counter (TCN0/TCN1) as part of the output-compare function. The reference value is not matched until TCN*n* equals TRR*n*.

| | 15 … 0 |
|---|---|
| Field | REF |
| Reset | 1111_1111_1111_1111 |
| R/W | R/W |
| Address | MBAR + 0x144 (TRR0),+ 0x184 (TRR1) |

**Figure 13-3. Timer Reference Registers (TRR0/TRR1)**

## 13.3.3 Timer Capture Registers (TCR0/TCR1)

Each timer capture register (TCR0/TCR1), Figure 13-4, latches the corresponding TCN*n* value during a capture operation when an edge occurs on TIN, as programmed in TMR*n*. CLKIN is assumed to be the clock source. TIN cannot simultaneously function as a

clocking source and as an input capture pin.

| | 15 … 0 |
|---|---|
| Field | CAP (16-bit capture counter value) |
| Reset | 0000_0000_0000_0000 |
| R/W | Read only |
| Address | MBAR + 0x148 (TCR0); + 0x188 (TCR1) |

**Figure 13-4. Timer Capture Register (TCR0/TCR1)**

## 13.3.4 Timer Counters (TCN0/TCN1)

The current value of the 16-bit, incrementing timer counters (TCN0/TCN1), Figure 13-5, can be read anytime without affecting counting. Writing to TCN*n* clears it. The timer counter decrements on the clock source rising edge (CLKIN ÷ 1, CLKIN ÷ 16, or TIN).

| | 15 … 0 |
|---|---|
| Field | 16-bit timer counter value count |
| Reset | 0000_0000_0000_0000 |
| R/W | R/W (to reset) |
| Address | MBAR + 0x14C (TCN0); + 0x18C (TCN1) |

**Figure 13-5. Timer Counters (TCN0/TCN1)**

## 13.3.5 Timer Event Registers (TER0/TER1)

Each timer event register (TER0/TER1), Figure 13-6, reports capture or reference events events the timer recognizes by setting TER*n*[CAP] or TER*n*[REF], which it does regardless of the corresponding interrupt-enable bit values, TMR*n*[ORI,CE].

Writing a 1 to either REF or CAP clears it (writing a 0 does not affect bit value); both bits can be cleared at the same time. REF and CAP must be cleared early in the exception handler, before the timer negates the $\overline{\text{IRQ}n}$ to the interrupt controller.

| | 7 … 2 | 1 | 0 |
|---|---|---|---|
| Field | — | REF | CAP |
| Reset | 0000_0000 | | |
| R/W | R/W (ones clear/zeros have no effect) | | |
| Address | MBAR + 0x151 (TER0); + 0x191 (TER1) | | |

**Figure 13-6. Timer Event Registers (TER0/TER1)**

Table 13-3 describes TER*n* fields.

**Table 13-3. TERn Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–2 | — | Reserved |
| 1 | REF | Output reference event. The counter has reached the TRRn value. Setting TMRn[ORI] enables the interrupt request caused by this event. Writing a one to REF clears the event condition. |
| 0 | CAP | Capture event. The counter value has been latched into TCRn. Setting TMRn[CE] enables the interrupt request caused by this event. Writing a 1 to CAP clears the event condition. |

## 13.4 Code Example

The following code provides an example of how to initialize timer 0 and how to use the timer for counting time-out periods.

```
MBARx EQU 0x10000 ;Defines the module base address at 0x10000
TMR0 EQU MBARx+0x140;Timer 0 register
TMR1 EQU MBARx+0x180 ;Timer 1 register
TRR0 EQU MBARx+0x144 ;Timer 0 reference register
TRR1 EQU MBARx+0x184 ;Timer 1 reference register
TCR0 EQU MBARx+0x148 ;Timer 0 capture register
TCR1 EQU MBARx+0x188 ;Timer 1 capture register
TCN0 EQU MBARx+0x14C ;Timer 0 counter
TCN1 EQU MBARx+0x18C ;Timer 1 counter
TER0 EQU MBARx+0x151 ;Timer 0 event register
TER1 EQU MBARx+0x191 ;Timer 1 event register

* TMR0 is defined as: *
*[PS]= 0xFF, divide clock by 256
*[CE] = 00disable interrupt
*[OM] = 0 output=active-low pulse
*[ORI] = 0, disable ref.interrupt
*[FRR] = 1, restart mode enabled
*[CLK] = 10, CLKIN/16
*[RST] = 0, timer 0 disabled

     move.w #0xFF0C,D0
     move.w D0,TMR0

     move.w #0x0000,D0;writing to the timer counter with any
     move.w DO,TCN0 ;value resets it to zero

     move.w #AFAF,DO ;set the timer 0 reference to be
     move.w #D0,TRR0 ;defined as 0xAFAF
```

The simple example below uses 0 to count time-out loops. A time-out occurs when the reference value, 0xAFAF, is reached.

```
timer0_ex
     clr.l DO
     clr.l D1
     clt.l D2

     move.w #0x0000,D0
     move,w D0,TCN0;reset the counter to 0x0000

     move.b #0x03,D0 ;writing ones to TER0[REF,CAP]
     move.b D0,TER0 ;clears the event flags
```

```
    move.w TMR0,D0;save the contents of TMR0 while setting
    bset #0,D0 ;the 0 bit. This enables timer 0 and starts counting
    move.w D0, TMR0 ;load the value back into the register, setting TMR0[RST]

T0_LOOP

    move.b TER0,D1 ;load TER0 and see if
    btst #1,D1 ;TER0[REF] has been set
    beq T0_LOOP

    addi.l #1,D2;Increment D2
    cmp.l #5,D2;Did D2 reach 5? (i.e. timer ref has timed)
    beq T0_FINISH;If so, end timer0 example. Otherwise jump back.

    move.b #0x02,D0 ;writing one to TER0[REF] clears the event flag
    move.b D0,TER0
    jmp T0_LOOP

T0_FINISH
    HALT;End processing. Example is finished
```

## 13.5 Calculating Time-Out Values

The formula below determines time-out periods for various reference values:

Time-out period = (1/clock frequency) x (1 or 16) x (TMR*n*[PS] + 1) x (TRR*n*[REF])

When calculating time-out periods, add 1 to the prescaler to simplify calculating, because TMR*n*[PS] = 0x00 yields a prescaler of 1 and TMR*n*[PS] = 0xFF yields a prescaler of 256. For example, if a 54-MHz timer clock is divided by 16, TMR*n*[PS] = 0x7F, and the timer is referenced at 0xABCD (43,981 decimal), the time-out period is as follows:

Time-out period = (1/54,000,000) x (16) x (127 + 1) x (43,981) = 1.67 S

The time-out values in Table 13-4 represent the time it takes the counter value in TCN*n* value to go from 0x0000 to the default reference value, TRR*n*[REF] = 0xFFFF. Time-out values shown for CLKIN are divided by 1 and by 16 (TMR*n*[CLK] is 01 or 10, respectively).

Any clock source (CLKIN ÷ 1, CLKIN ÷ 16, or TIN) can be prescaled using TMR*n*[PS].

**Table 13-4. Time-Out Values (in Seconds)—TRR[REF] = 0xFFFF (162-MHz Processor Clock)**

| TMR[PS] (Dec) | CLK = 10 (÷ 1) | | | CLK = 01 (÷ 16) | | |
|---|---|---|---|---|---|---|
| | CLKIN (MHz) | | | | | |
| | 54 | 40.5 | 32.4 | 54 | 40.5 | 32.4 |
| 0 | 0.019 | 0.026 | 0.032 | 0.001 | 0.002 | 0.002 |
| 1 | 0.039 | 0.052 | 0.065 | 0.002 | 0.003 | 0.004 |
| 2 | 0.058 | 0.078 | 0.097 | 0.004 | 0.005 | 0.006 |
| 3 | 0.078 | 0.104 | 0.129 | 0.005 | 0.006 | 0.008 |
| 4 | 0.097 | 0.129 | 0.162 | 0.006 | 0.008 | 0.010 |
| 5 | 0.117 | 0.155 | 0.194 | 0.007 | 0.010 | 0.012 |

| TMR[PS] (Dec) | CLK = 10 (÷ 1) | | | CLK = 01 (÷ 16) | | |
|---|---|---|---|---|---|---|
| | CLKIN (MHz) | | | | | |
| | 54 | 40.5 | 32.4 | 54 | 40.5 | 32.4 |
| 128 | 2.505 | 3.340 | 4.175 | 0.157 | 0.209 | 0.261 |
| 129 | 2.524 | 3.366 | 4.207 | 0.158 | 0.210 | 0.263 |
| 130 | 2.544 | 3.392 | 4.240 | 0.159 | 0.212 | 0.265 |
| 131 | 2.563 | 3.418 | 4.272 | 0.160 | 0.214 | 0.267 |
| 132 | 2.583 | 3.443 | 4.304 | 0.161 | 0.215 | 0.269 |
| 133 | 2.602 | 3.469 | 4.337 | 0.163 | 0.217 | 0.271 |

## Table 13-4. Time-Out Values (in Seconds)—TRR[REF] = 0xFFFF (162-MHz Processor Clock) (Continued)

| TMR[PS] (Dec) | CLK = 10 (÷ 1) | | | CLK = 01 (÷ 16) | | |
|---|---|---|---|---|---|---|
| | CLKIN (MHz) | | | | | |
| | 54 | 40.5 | 32.4 | 54 | 40.5 | 32.4 |
| 6 | 0.136 | 0.181 | 0.227 | 0.008 | 0.011 | 0.014 |
| 7 | 0.155 | 0.207 | 0.259 | 0.010 | 0.013 | 0.016 |
| 8 | 0.175 | 0.233 | 0.291 | 0.011 | 0.015 | 0.018 |
| 9 | 0.194 | 0.259 | 0.324 | 0.012 | 0.016 | 0.020 |
| 10 | 0.214 | 0.285 | 0.356 | 0.013 | 0.018 | 0.022 |
| 11 | 0.233 | 0.311 | 0.388 | 0.015 | 0.019 | 0.024 |
| 12 | 0.252 | 0.337 | 0.421 | 0.016 | 0.021 | 0.026 |
| 13 | 0.272 | 0.362 | 0.453 | 0.017 | 0.023 | 0.028 |
| 14 | 0.291 | 0.388 | 0.485 | 0.018 | 0.024 | 0.030 |
| 15 | 0.311 | 0.414 | 0.518 | 0.019 | 0.026 | 0.032 |
| 16 | 0.330 | 0.440 | 0.550 | 0.021 | 0.028 | 0.034 |
| 17 | 0.350 | 0.466 | 0.583 | 0.022 | 0.029 | 0.036 |
| 18 | 0.369 | 0.492 | 0.615 | 0.023 | 0.031 | 0.038 |
| 19 | 0.388 | 0.518 | 0.647 | 0.024 | 0.032 | 0.040 |
| 20 | 0.408 | 0.544 | 0.680 | 0.025 | 0.034 | 0.042 |
| 21 | 0.427 | 0.570 | 0.712 | 0.027 | 0.036 | 0.044 |
| 22 | 0.447 | 0.595 | 0.744 | 0.028 | 0.037 | 0.047 |
| 23 | 0.466 | 0.621 | 0.777 | 0.029 | 0.039 | 0.049 |
| 24 | 0.485 | 0.647 | 0.809 | 0.030 | 0.040 | 0.051 |
| 25 | 0.505 | 0.673 | 0.841 | 0.032 | 0.042 | 0.053 |
| 26 | 0.524 | 0.699 | 0.874 | 0.033 | 0.044 | 0.055 |
| 27 | 0.544 | 0.725 | 0.906 | 0.034 | 0.045 | 0.057 |
| 28 | 0.563 | 0.751 | 0.939 | 0.035 | 0.047 | 0.059 |
| 29 | 0.583 | 0.777 | 0.971 | 0.036 | 0.049 | 0.061 |
| 30 | 0.602 | 0.803 | 1.003 | 0.038 | 0.050 | 0.063 |
| 31 | 0.621 | 0.829 | 1.036 | 0.039 | 0.052 | 0.065 |
| 32 | 0.641 | 0.854 | 1.068 | 0.040 | 0.053 | 0.067 |
| 33 | 0.660 | 0.880 | 1.100 | 0.041 | 0.055 | 0.069 |
| 34 | 0.680 | 0.906 | 1.133 | 0.042 | 0.057 | 0.071 |
| 35 | 0.699 | 0.932 | 1.165 | 0.044 | 0.058 | 0.073 |
| 36 | 0.718 | 0.958 | 1.197 | 0.045 | 0.060 | 0.075 |
| 37 | 0.738 | 0.984 | 1.230 | 0.046 | 0.061 | 0.077 |
| 38 | 0.757 | 1.010 | 1.262 | 0.047 | 0.063 | 0.079 |
| 39 | 0.777 | 1.036 | 1.295 | 0.049 | 0.065 | 0.081 |
| 40 | 0.796 | 1.062 | 1.327 | 0.050 | 0.066 | 0.083 |
| 41 | 0.816 | 1.087 | 1.359 | 0.051 | 0.068 | 0.085 |
| 42 | 0.835 | 1.113 | 1.392 | 0.052 | 0.070 | 0.087 |

| TMR[PS] (Dec) | CLK = 10 (÷ 1) | | | CLK = 01 (÷ 16) | | |
|---|---|---|---|---|---|---|
| | CLKIN (MHz) | | | | | |
| | 54 | 40.5 | 32.4 | 54 | 40.5 | 32.4 |
| 134 | 2.621 | 3.495 | 4.369 | 0.164 | 0.218 | 0.273 |
| 135 | 2.641 | 3.521 | 4.401 | 0.165 | 0.220 | 0.275 |
| 136 | 2.660 | 3.547 | 4.434 | 0.166 | 0.222 | 0.277 |
| 137 | 2.680 | 3.573 | 4.466 | 0.167 | 0.223 | 0.279 |
| 138 | 2.699 | 3.599 | 4.499 | 0.169 | 0.225 | 0.281 |
| 139 | 2.719 | 3.625 | 4.531 | 0.170 | 0.227 | 0.283 |
| 140 | 2.738 | 3.651 | 4.563 | 0.171 | 0.228 | 0.285 |
| 141 | 2.757 | 3.676 | 4.596 | 0.172 | 0.230 | 0.287 |
| 142 | 2.777 | 3.702 | 4.628 | 0.174 | 0.231 | 0.289 |
| 143 | 2.796 | 3.728 | 4.660 | 0.175 | 0.233 | 0.291 |
| 144 | 2.816 | 3.754 | 4.693 | 0.176 | 0.235 | 0.293 |
| 145 | 2.835 | 3.780 | 4.725 | 0.177 | 0.236 | 0.295 |
| 146 | 2.854 | 3.806 | 4.757 | 0.178 | 0.238 | 0.297 |
| 147 | 2.874 | 3.832 | 4.790 | 0.180 | 0.239 | 0.299 |
| 148 | 2.893 | 3.858 | 4.822 | 0.181 | 0.241 | 0.301 |
| 149 | 2.913 | 3.884 | 4.855 | 0.182 | 0.243 | 0.303 |
| 150 | 2.932 | 3.910 | 4.887 | 0.183 | 0.244 | 0.305 |
| 151 | 2.952 | 3.935 | 4.919 | 0.184 | 0.246 | 0.307 |
| 152 | 2.971 | 3.961 | 4.952 | 0.186 | 0.248 | 0.309 |
| 153 | 2.990 | 3.987 | 4.984 | 0.187 | 0.249 | 0.311 |
| 154 | 3.010 | 4.013 | 5.016 | 0.188 | 0.251 | 0.314 |
| 155 | 3.029 | 4.039 | 5.049 | 0.189 | 0.252 | 0.316 |
| 156 | 3.049 | 4.065 | 5.081 | 0.191 | 0.254 | 0.318 |
| 157 | 3.068 | 4.091 | 5.113 | 0.192 | 0.256 | 0.320 |
| 158 | 3.087 | 4.117 | 5.146 | 0.193 | 0.257 | 0.322 |
| 159 | 3.107 | 4.143 | 5.178 | 0.194 | 0.259 | 0.324 |
| 160 | 3.126 | 4.168 | 5.211 | 0.195 | 0.261 | 0.326 |
| 161 | 3.146 | 4.194 | 5.243 | 0.197 | 0.262 | 0.328 |
| 162 | 3.165 | 4.220 | 5.275 | 0.198 | 0.264 | 0.330 |
| 163 | 3.185 | 4.246 | 5.308 | 0.199 | 0.265 | 0.332 |
| 164 | 3.204 | 4.272 | 5.340 | 0.200 | 0.267 | 0.334 |
| 165 | 3.223 | 4.298 | 5.372 | 0.201 | 0.269 | 0.336 |
| 166 | 3.243 | 4.324 | 5.405 | 0.203 | 0.270 | 0.338 |
| 167 | 3.262 | 4.350 | 5.437 | 0.204 | 0.272 | 0.340 |
| 168 | 3.282 | 4.376 | 5.469 | 0.205 | 0.273 | 0.342 |
| 169 | 3.301 | 4.401 | 5.502 | 0.206 | 0.275 | 0.344 |
| 170 | 3.320 | 4.427 | 5.534 | 0.208 | 0.277 | 0.346 |

## Table 13-4. Time-Out Values (in Seconds)—TRR[REF] = 0xFFFF (162-MHz Processor Clock) (Continued)

| TMR[PS] (Dec) | CLK = 10 (÷ 1) | | | CLK = 01 (÷ 16) | | |
|---|---|---|---|---|---|---|
| | CLKIN (MHz) | | | | | |
| | 54 | 40.5 | 32.4 | 54 | 40.5 | 32.4 |
| 43 | 0.854 | 1.139 | 1.424 | 0.053 | 0.071 | 0.089 |
| 44 | 0.874 | 1.165 | 1.456 | 0.055 | 0.073 | 0.091 |
| 45 | 0.893 | 1.191 | 1.489 | 0.056 | 0.074 | 0.093 |
| 46 | 0.913 | 1.217 | 1.521 | 0.057 | 0.076 | 0.095 |
| 47 | 0.932 | 1.243 | 1.553 | 0.058 | 0.078 | 0.097 |
| 48 | 0.951 | 1.269 | 1.586 | 0.059 | 0.079 | 0.099 |
| 49 | 0.971 | 1.295 | 1.618 | 0.061 | 0.081 | 0.101 |
| 50 | 0.990 | 1.320 | 1.651 | 0.062 | 0.083 | 0.103 |
| 51 | 1.010 | 1.346 | 1.683 | 0.063 | 0.084 | 0.105 |
| 52 | 1.029 | 1.372 | 1.715 | 0.064 | 0.086 | 0.107 |
| 53 | 1.049 | 1.398 | 1.748 | 0.066 | 0.087 | 0.109 |
| 54 | 1.068 | 1.424 | 1.780 | 0.067 | 0.089 | 0.111 |
| 55 | 1.087 | 1.450 | 1.812 | 0.068 | 0.091 | 0.113 |
| 56 | 1.107 | 1.476 | 1.845 | 0.069 | 0.092 | 0.115 |
| 57 | 1.126 | 1.502 | 1.877 | 0.070 | 0.094 | 0.117 |
| 58 | 1.146 | 1.528 | 1.909 | 0.072 | 0.095 | 0.119 |
| 59 | 1.165 | 1.553 | 1.942 | 0.073 | 0.097 | 0.121 |
| 60 | 1.185 | 1.579 | 1.974 | 0.074 | 0.099 | 0.123 |
| 61 | 1.204 | 1.605 | 2.007 | 0.075 | 0.100 | 0.125 |
| 62 | 1.223 | 1.631 | 2.039 | 0.076 | 0.102 | 0.127 |
| 63 | 1.243 | 1.657 | 2.071 | 0.078 | 0.104 | 0.129 |
| 64 | 1.262 | 1.683 | 2.104 | 0.079 | 0.105 | 0.131 |
| 65 | 1.282 | 1.709 | 2.136 | 0.080 | 0.107 | 0.133 |
| 66 | 1.301 | 1.735 | 2.168 | 0.081 | 0.108 | 0.136 |
| 67 | 1.320 | 1.761 | 2.201 | 0.083 | 0.110 | 0.138 |
| 68 | 1.340 | 1.786 | 2.233 | 0.084 | 0.112 | 0.140 |
| 69 | 1.359 | 1.812 | 2.265 | 0.085 | 0.113 | 0.142 |
| 70 | 1.379 | 1.838 | 2.298 | 0.086 | 0.115 | 0.144 |
| 71 | 1.398 | 1.864 | 2.330 | 0.087 | 0.117 | 0.146 |
| 72 | 1.418 | 1.890 | 2.363 | 0.089 | 0.118 | 0.148 |
| 73 | 1.437 | 1.916 | 2.395 | 0.090 | 0.120 | 0.150 |
| 74 | 1.456 | 1.942 | 2.427 | 0.091 | 0.121 | 0.152 |
| 75 | 1.476 | 1.968 | 2.460 | 0.092 | 0.123 | 0.154 |
| 76 | 1.495 | 1.994 | 2.492 | 0.093 | 0.125 | 0.156 |
| 77 | 1.515 | 2.019 | 2.524 | 0.095 | 0.126 | 0.158 |
| 78 | 1.534 | 2.045 | 2.557 | 0.096 | 0.128 | 0.160 |
| 79 | 1.553 | 2.071 | 2.589 | 0.097 | 0.129 | 0.162 |

| TMR[PS] (Dec) | CLK = 10 (÷ 1) | | | CLK = 01 (÷ 16) | | |
|---|---|---|---|---|---|---|
| | CLKIN (MHz) | | | | | |
| | 54 | 40.5 | 32.4 | 54 | 40.5 | 32.4 |
| 171 | 3.340 | 4.453 | 5.567 | 0.209 | 0.278 | 0.348 |
| 172 | 3.359 | 4.479 | 5.599 | 0.210 | 0.280 | 0.350 |
| 173 | 3.379 | 4.505 | 5.631 | 0.211 | 0.282 | 0.352 |
| 174 | 3.398 | 4.531 | 5.664 | 0.212 | 0.283 | 0.354 |
| 175 | 3.418 | 4.557 | 5.696 | 0.214 | 0.285 | 0.356 |
| 176 | 3.437 | 4.583 | 5.728 | 0.215 | 0.286 | 0.358 |
| 177 | 3.456 | 4.609 | 5.761 | 0.216 | 0.288 | 0.360 |
| 178 | 3.476 | 4.634 | 5.793 | 0.217 | 0.290 | 0.362 |
| 179 | 3.495 | 4.660 | 5.825 | 0.218 | 0.291 | 0.364 |
| 180 | 3.515 | 4.686 | 5.858 | 0.220 | 0.293 | 0.366 |
| 181 | 3.534 | 4.712 | 5.890 | 0.221 | 0.295 | 0.368 |
| 182 | 3.554 | 4.738 | 5.923 | 0.222 | 0.296 | 0.370 |
| 183 | 3.573 | 4.764 | 5.955 | 0.223 | 0.298 | 0.372 |
| 184 | 3.592 | 4.790 | 5.987 | 0.225 | 0.299 | 0.374 |
| 185 | 3.612 | 4.816 | 6.020 | 0.226 | 0.301 | 0.376 |
| 186 | 3.631 | 4.842 | 6.052 | 0.227 | 0.303 | 0.378 |
| 187 | 3.651 | 4.867 | 6.084 | 0.228 | 0.304 | 0.380 |
| 188 | 3.670 | 4.893 | 6.117 | 0.229 | 0.306 | 0.382 |
| 189 | 3.689 | 4.919 | 6.149 | 0.231 | 0.307 | 0.384 |
| 190 | 3.709 | 4.945 | 6.181 | 0.232 | 0.309 | 0.386 |
| 191 | 3.728 | 4.971 | 6.214 | 0.233 | 0.311 | 0.388 |
| 192 | 3.748 | 4.997 | 6.246 | 0.234 | 0.312 | 0.390 |
| 193 | 3.767 | 5.023 | 6.279 | 0.235 | 0.314 | 0.392 |
| 194 | 3.787 | 5.049 | 6.311 | 0.237 | 0.316 | 0.394 |
| 195 | 3.806 | 5.075 | 6.343 | 0.238 | 0.317 | 0.396 |
| 196 | 3.825 | 5.100 | 6.376 | 0.239 | 0.319 | 0.398 |
| 197 | 3.845 | 5.126 | 6.408 | 0.240 | 0.320 | 0.400 |
| 198 | 3.864 | 5.152 | 6.440 | 0.242 | 0.322 | 0.403 |
| 199 | 3.884 | 5.178 | 6.473 | 0.243 | 0.324 | 0.405 |
| 200 | 3.903 | 5.204 | 6.505 | 0.244 | 0.325 | 0.407 |
| 201 | 3.922 | 5.230 | 6.537 | 0.245 | 0.327 | 0.409 |
| 202 | 3.942 | 5.256 | 6.570 | 0.246 | 0.328 | 0.411 |
| 203 | 3.961 | 5.282 | 6.602 | 0.248 | 0.330 | 0.413 |
| 204 | 3.981 | 5.308 | 6.635 | 0.249 | 0.332 | 0.415 |
| 205 | 4.000 | 5.333 | 6.667 | 0.250 | 0.333 | 0.417 |
| 206 | 4.020 | 5.359 | 6.699 | 0.251 | 0.335 | 0.419 |
| 207 | 4.039 | 5.385 | 6.732 | 0.252 | 0.337 | 0.421 |

## Table 13-4. Time-Out Values (in Seconds)—TRR[REF] = 0xFFFF (162-MHz Processor Clock) (Continued)

| TMR[PS] (Dec) | CLK = 10 (÷ 1) | | | CLK = 01 (÷ 16) | | |
|---|---|---|---|---|---|---|
| | CLKIN (MHz) | | | | | |
| | 54 | 40.5 | 32.4 | 54 | 40.5 | 32.4 |
| 80 | 1.573 | 2.097 | 2.621 | 0.098 | 0.131 | 0.164 |
| 81 | 1.592 | 2.123 | 2.654 | 0.100 | 0.133 | 0.166 |
| 82 | 1.612 | 2.149 | 2.686 | 0.101 | 0.134 | 0.168 |
| 83 | 1.631 | 2.175 | 2.719 | 0.102 | 0.136 | 0.170 |
| 84 | 1.651 | 2.201 | 2.751 | 0.103 | 0.138 | 0.172 |
| 85 | 1.670 | 2.227 | 2.783 | 0.104 | 0.139 | 0.174 |
| 86 | 1.689 | 2.252 | 2.816 | 0.106 | 0.141 | 0.176 |
| 87 | 1.709 | 2.278 | 2.848 | 0.107 | 0.142 | 0.178 |
| 88 | 1.728 | 2.304 | 2.880 | 0.108 | 0.144 | 0.180 |
| 89 | 1.748 | 2.330 | 2.913 | 0.109 | 0.146 | 0.182 |
| 90 | 1.767 | 2.356 | 2.945 | 0.110 | 0.147 | 0.184 |
| 91 | 1.786 | 2.382 | 2.977 | 0.112 | 0.149 | 0.186 |
| 92 | 1.806 | 2.408 | 3.010 | 0.113 | 0.150 | 0.188 |
| 93 | 1.825 | 2.434 | 3.042 | 0.114 | 0.152 | 0.190 |
| 94 | 1.845 | 2.460 | 3.075 | 0.115 | 0.154 | 0.192 |
| 95 | 1.864 | 2.486 | 3.107 | 0.117 | 0.155 | 0.194 |
| 96 | 1.884 | 2.511 | 3.139 | 0.118 | 0.157 | 0.196 |
| 97 | 1.903 | 2.537 | 3.172 | 0.119 | 0.159 | 0.198 |
| 98 | 1.922 | 2.563 | 3.204 | 0.120 | 0.160 | 0.200 |
| 99 | 1.942 | 2.589 | 3.236 | 0.121 | 0.162 | 0.202 |
| 100 | 1.961 | 2.615 | 3.269 | 0.123 | 0.163 | 0.204 |
| 101 | 1.981 | 2.641 | 3.301 | 0.124 | 0.165 | 0.206 |
| 102 | 2.000 | 2.667 | 3.333 | 0.125 | 0.167 | 0.208 |
| 103 | 2.019 | 2.693 | 3.366 | 0.126 | 0.168 | 0.210 |
| 104 | 2.039 | 2.719 | 3.398 | 0.127 | 0.170 | 0.212 |
| 105 | 2.058 | 2.744 | 3.431 | 0.129 | 0.172 | 0.214 |
| 106 | 2.078 | 2.770 | 3.463 | 0.130 | 0.173 | 0.216 |
| 107 | 2.097 | 2.796 | 3.495 | 0.131 | 0.175 | 0.218 |
| 108 | 2.117 | 2.822 | 3.528 | 0.132 | 0.176 | 0.220 |
| 109 | 2.136 | 2.848 | 3.560 | 0.133 | 0.178 | 0.222 |
| 110 | 2.155 | 2.874 | 3.592 | 0.135 | 0.180 | 0.225 |
| 111 | 2.175 | 2.900 | 3.625 | 0.136 | 0.181 | 0.227 |
| 112 | 2.194 | 2.926 | 3.657 | 0.137 | 0.183 | 0.229 |
| 113 | 2.214 | 2.952 | 3.689 | 0.138 | 0.184 | 0.231 |
| 114 | 2.233 | 2.977 | 3.722 | 0.140 | 0.186 | 0.233 |
| 115 | 2.252 | 3.003 | 3.754 | 0.141 | 0.188 | 0.235 |
| 116 | 2.272 | 3.029 | 3.787 | 0.142 | 0.189 | 0.237 |

| TMR[PS] (Dec) | CLK = 10 (÷ 1) | | | CLK = 01 (÷ 16) | | |
|---|---|---|---|---|---|---|
| | CLKIN (MHz) | | | | | |
| | 54 | 40.5 | 32.4 | 54 | 40.5 | 32.4 |
| 208 | 4.058 | 5.411 | 6.764 | 0.254 | 0.338 | 0.423 |
| 209 | 4.078 | 5.437 | 6.796 | 0.255 | 0.340 | 0.425 |
| 210 | 4.097 | 5.463 | 6.829 | 0.256 | 0.341 | 0.427 |
| 211 | 4.117 | 5.489 | 6.861 | 0.257 | 0.343 | 0.429 |
| 212 | 4.136 | 5.515 | 6.893 | 0.259 | 0.345 | 0.431 |
| 213 | 4.155 | 5.541 | 6.926 | 0.260 | 0.346 | 0.433 |
| 214 | 4.175 | 5.567 | 6.958 | 0.261 | 0.348 | 0.435 |
| 215 | 4.194 | 5.592 | 6.991 | 0.262 | 0.350 | 0.437 |
| 216 | 4.214 | 5.618 | 7.023 | 0.263 | 0.351 | 0.439 |
| 217 | 4.233 | 5.644 | 7.055 | 0.265 | 0.353 | 0.441 |
| 218 | 4.253 | 5.670 | 7.088 | 0.266 | 0.354 | 0.443 |
| 219 | 4.272 | 5.696 | 7.120 | 0.267 | 0.356 | 0.445 |
| 220 | 4.291 | 5.722 | 7.152 | 0.268 | 0.358 | 0.447 |
| 221 | 4.311 | 5.748 | 7.185 | 0.269 | 0.359 | 0.449 |
| 222 | 4.330 | 5.774 | 7.217 | 0.271 | 0.361 | 0.451 |
| 223 | 4.350 | 5.800 | 7.249 | 0.272 | 0.362 | 0.453 |
| 224 | 4.369 | 5.825 | 7.282 | 0.273 | 0.364 | 0.455 |
| 225 | 4.388 | 5.851 | 7.314 | 0.274 | 0.366 | 0.457 |
| 226 | 4.408 | 5.877 | 7.347 | 0.275 | 0.367 | 0.459 |
| 227 | 4.427 | 5.903 | 7.379 | 0.277 | 0.369 | 0.461 |
| 228 | 4.447 | 5.929 | 7.411 | 0.278 | 0.371 | 0.463 |
| 229 | 4.466 | 5.955 | 7.444 | 0.279 | 0.372 | 0.465 |
| 230 | 4.486 | 5.981 | 7.476 | 0.280 | 0.374 | 0.467 |
| 231 | 4.505 | 6.007 | 7.508 | 0.282 | 0.375 | 0.469 |
| 232 | 4.524 | 6.033 | 7.541 | 0.283 | 0.377 | 0.471 |
| 233 | 4.544 | 6.058 | 7.573 | 0.284 | 0.379 | 0.473 |
| 234 | 4.563 | 6.084 | 7.605 | 0.285 | 0.380 | 0.475 |
| 235 | 4.583 | 6.110 | 7.638 | 0.286 | 0.382 | 0.477 |
| 236 | 4.602 | 6.136 | 7.670 | 0.288 | 0.384 | 0.479 |
| 237 | 4.622 | 6.162 | 7.703 | 0.289 | 0.385 | 0.481 |
| 238 | 4.641 | 6.188 | 7.735 | 0.290 | 0.387 | 0.483 |
| 239 | 4.660 | 6.214 | 7.767 | 0.291 | 0.388 | 0.485 |
| 240 | 4.680 | 6.240 | 7.800 | 0.292 | 0.390 | 0.487 |
| 241 | 4.699 | 6.266 | 7.832 | 0.294 | 0.392 | 0.489 |
| 242 | 4.719 | 6.291 | 7.864 | 0.295 | 0.393 | 0.492 |
| 243 | 4.738 | 6.317 | 7.897 | 0.296 | 0.395 | 0.494 |
| 244 | 4.757 | 6.343 | 7.929 | 0.297 | 0.396 | 0.496 |

**Table 13-4. Time-Out Values (in Seconds)—TRR[REF] = 0xFFFF (162-MHz Processor Clock) (Continued)**

| TMR[PS] (Dec) | CLK = 10 (÷ 1) | | | CLK = 01 (÷ 16) | | |
|---|---|---|---|---|---|---|
| | CLKIN (MHz) | | | | | |
| | 54 | 40.5 | 32.4 | 54 | 40.5 | 32.4 |
| 117 | 2.291 | 3.055 | 3.819 | 0.143 | 0.191 | 0.239 |
| 118 | 2.311 | 3.081 | 3.851 | 0.144 | 0.193 | 0.241 |
| 119 | 2.330 | 3.107 | 3.884 | 0.146 | 0.194 | 0.243 |
| 120 | 2.350 | 3.133 | 3.916 | 0.147 | 0.196 | 0.245 |
| 121 | 2.369 | 3.159 | 3.948 | 0.148 | 0.197 | 0.247 |
| 122 | 2.388 | 3.185 | 3.981 | 0.149 | 0.199 | 0.249 |
| 123 | 2.408 | 3.210 | 4.013 | 0.150 | 0.201 | 0.251 |
| 124 | 2.427 | 3.236 | 4.045 | 0.152 | 0.202 | 0.253 |
| 125 | 2.447 | 3.262 | 4.078 | 0.153 | 0.204 | 0.255 |
| 126 | 2.466 | 3.288 | 4.110 | 0.154 | 0.206 | 0.257 |
| 127 | 2.486 | 3.314 | 4.143 | 0.155 | 0.207 | 0.259 |

| TMR[PS] (Dec) | CLK = 10 (÷ 1) | | | CLK = 01 (÷ 16) | | |
|---|---|---|---|---|---|---|
| | CLKIN (MHz) | | | | | |
| | 54 | 40.5 | 32.4 | 54 | 40.5 | 32.4 |
| 245 | 4.777 | 6.369 | 7.961 | 0.299 | 0.398 | 0.498 |
| 246 | 4.796 | 6.395 | 7.994 | 0.300 | 0.400 | 0.500 |
| 247 | 4.816 | 6.421 | 8.026 | 0.301 | 0.401 | 0.502 |
| 248 | 4.835 | 6.447 | 8.059 | 0.302 | 0.403 | 0.504 |
| 249 | 4.855 | 6.473 | 8.091 | 0.303 | 0.405 | 0.506 |
| 250 | 4.874 | 6.499 | 8.123 | 0.305 | 0.406 | 0.508 |
| 251 | 4.893 | 6.524 | 8.156 | 0.306 | 0.408 | 0.510 |
| 252 | 4.913 | 6.550 | 8.188 | 0.307 | 0.409 | 0.512 |
| 253 | 4.932 | 6.576 | 8.220 | 0.308 | 0.411 | 0.514 |
| 254 | 4.952 | 6.602 | 8.253 | 0.309 | 0.413 | 0.516 |
| 255 | 4.971 | 6.628 | 8.285 | 0.311 | 0.414 | 0.518 |

# Chapter 14
# UART Modules

This chapter describes the use of the universal asynchronous/synchronous receiver/transmitters (UARTs) implemented on the MCF5407 and includes programming examples. All references to UART refer to one of these modules when in UART mode as opposed to modem mode. Particular attention is given to the UART1 implementation of a synchronous interface that provides a controller for an 8- or 16-bit CODEC interface and an audio CODEC ‘97 (AC ’97) digital interface.

## 14.1 Overview

The MCF5407 contains two independent UARTs. UART1 on the MCF5407 provides synchronous operation and a CODEC interface for soft modem support. Each UART can be clocked by CLKIN, eliminating the need for an external crystal. As Figure 14-1 shows, each UART module interfaces directly to the CPU and consists of the following:

- Serial communication channel
- Programmable transmitter and receiver clock generation
- Internal channel control logic
- Interrupt control logic

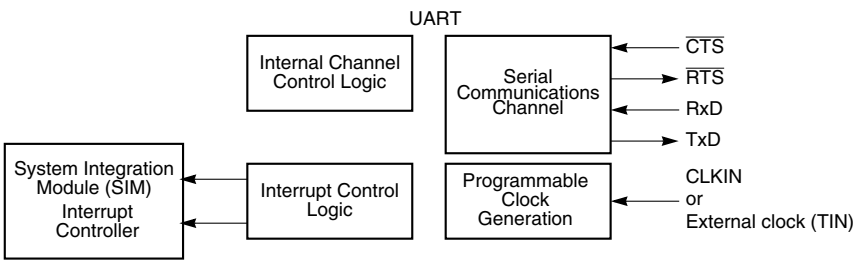



**Figure 14-1. Simplified Block Diagram**

The serial communication channel provides a full-duplex asynchronous/synchronous receiver and transmitter deriving an operating frequency from CLKIN 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 channel transmitter serial data output (TxD). See Section 14.5.2.1, “Transmitting in UART Mode.”

The receiver converts serial data from the channel receiver serial data input (RxD) 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- or interrupt-driven. See Section 14.5.2.3, “Receiver.”

UART1 can be programmed to function like original UART (identical to UART0) or in one of the following three modem modes:

- An 8-bit CODEC interface
- A 16-bit CODEC interface
- An audio CODEC ‘97 (AC ’97) digital interface controller

A CODEC (code/decode) chip provides a data conversion interface for high-speed modem designs meeting a high range of standards, such as ITU-T V.34 and PCM. UART1 interfaces to the CODEC through a serial port consisting of Tx and Rx serial data and serial bit clock and frame inputs from the CODEC. UART1 transfers digital sample data to and from the CODEC through the serial port.

AC ’97 defines an architecture for audio-intensive personal computer applications such as gaming, authoring, and high-resolution music and video playback. An external AC ’97 analog device performs mixing, analog processing, and sample-rate DAC and ADC. UART1 interfaces to the AC ’97 device through a serial port consisting of Tx and Rx serial data, a serial bit clock, and a frame sync output generated by UART1 from the serial bit clock. An MCF5407 general-purpose I/O (GPIO) is used as a reset to the AC ‘97 device. UART1 transfers digital sample data as well as control/status information to and from the AC ‘97 device through the serial port.

Unless otherwise specified, descriptions in this chapter refer to UART mode.

## 14.2 Serial Module Overview

The MCF5407 contains two independent UART modules, whose features are as follows:

- Each can be clocked by CLKIN, eliminating a need for an external crystal
- Full-duplex asynchronous/synchronous receiver/transmitter channel
- 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 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
- UART0 and UART1 have interrupt capability to DMA channels 2 and 3, respectively, when either the RxRDY or FFULL bit is set in the USR.
- 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

UART1 has the following additional features:

- Programmable to interface to an 8- or 16-bit CODEC for soft modem support
- Programmable to function as a digital AC ’97 controller
- Tx and Rx FIFOs can hold the following:
  - 32 1-byte samples when programmed as a UART or as an 8-bit CODEC interface
  - 16 2-byte samples when programmed as a 16-bit CODEC interface
  - 16 20-bit samples when programmed as a digital AC ’97 controller
- Both DMA channels associated with the UARTs can be programmed to service UART1 (one for the Tx channel and one for the Rx channel)
- No parity error, framing error, or line break detection in modem mode

## 14.3 Register Descriptions

This section contains a detailed description of each register and its specific function. Flowcharts in Section 14.5.6, “Programming,” describe basic UART module programming. The operation of the UART module is controlled by writing control bytes into the appropriate registers. Table 14-1 is a memory map for UART module registers.

## Table 14-1. UART Module Programming Model

| MBAR Offset | | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|---|
| UART0 | UART1 | | | | |
| 0x1C0 | 0x200 | UART mode registers[1]—(UMR1n) [p. 14-5], (UMR2n) [p. 14-7] | Rx FIFO threshold register—(RXLVL) [p. 14-8] (UART1 only) | Modem control register—(MODCTL) [p. 14-9] (UART1 only) | Tx FIFO threshold register—(TXLVL) [p. 14-10] (UART1 only) |
| 0x1C4 | 0x204 | (Read) UART status registers—(USRn) [p. 14-10] | — | (Read) Rx samples available register—(RSMP) [p. 14-12] (UART1 only) | (Read) Tx space available register—(TSPC) [p. 14-12] (UART1 only) |
| | | (Write) UART clock-select register[1]—(UCSRn) [p. 14-12] | | | |
| 0x1C8 | 0x208 | (Read) Do not access[2] | — | | |
| | | (Write) UART command registers—(UCRn) [p. 14-13] | — | | |
| 0x1CC | 0x20C | (UART0/Read) UART receiver buffers—(URBn) [p. 14-15] | — | | |
| | | (UART1/Read) UART receiver buffers—(URBn) [p. 14-15] | | | |
| | | (UART0/Write) UART transmitter buffers—(UTBn) [p. 14-16] | — | | |
| | | (UART1/Write) UART transmitter buffers—(UTBn) [p. 14-16] | | | |
| 0x1D0 | 0x210 | (Read) UART input port change registers—(UIPCRn) [p. 14-17] | — | | |
| | | (Write) UART auxiliary control registers[1]—(UACRn) [p. 14-17] | — | | |
| 0x1D4 | 0x214 | (Read) UART interrupt status registers—(UISRn) [p. 14-18] | — | | |
| | | (Write) UART interrupt mask registers—(UIMRn) [p. 14-18] | — | | |
| 0x1D8 | 0x218 | UART divider upper registers—(UDUn) [p. 14-19] | — | | |

**Table 14-1. UART Module Programming Model (Continued)**

| MBAR Offset | | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|---|
| UART0 | UART1 | | | | |
| 0x1DC | 0x21C | UART divider lower registers—(UDLn) [p. 14-19] | — | | |
| 0x1E0–0x1EC | 0x220–0x22C | Do not access[2] | — | | |
| 0x1F0 | 0x230 | UART interrupt vector register—(UIVRn) [p. 14-20] | — | | |
| 0x1F4 | 0x234 | (Read) UART input port registers—(UIPn) [p. 14-20] | — | | |
| | | (Write) Do not access[2] | — | | |
| 0x1F8 | 0x238 | (Read) Do not access[2] | — | | |
| | | (Write) UART output port bit set command registers—(UOP1n[3]) [p. 14-21] | — | | |
| 0x1FC | 0x23C | (Read) Do not access[2] | — | | |
| | | (Write) UART output port bit reset command registers—(UOP0n[3]) [p. 14-21] | — | | |

1 UMR1n, UMR2n, and UCSRn should be changed only after the receiver/transmitter is issued a software reset command. That is, if channel operation is not disabled, undesirable results may occur.

2 This address is for factory testing. Reading this location results in undesired effects and possible incorrect transmission or reception of characters. Register contents may also be changed.

3 Address-triggered commands

**NOTE:**

UART registers are accessible only as bytes. Although external masters cannot access on-chip memories or MBAR, they can access any UART registers.

## 14.3.1 UART Mode Registers 1 (UMR1n)

The UART mode registers 1 (UMR1*n*) control configuration. UMR1*n* can be read or written when the mode register pointer points to it, at RESET or after a RESET MODE REGISTER POINTER command using UCR*n*[MISC]. After UMR1*n* is read or written, the pointer points to UMR2*n*.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field | RxRTS | RxIRQ/FFULL | ERR | PM | | PT | B/C | |
| Reset | 0000_0000 | | | | | | | |
| R/W | R/W | | | | | | | |
| Address | MBAR + 0x1C0 (UART0), 0x200 (UART1). After UMR1n is read or written, the pointer points to UMR2n. | | | | | | | |

**Figure 14-2. UART Mode Registers 1 (UMR1n)**

Table 14-2 describes UMR1*n* fields.

**Table 14-2. UMR1n Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7 | RxRTS | Receiver request-to-send. Allows the $\overline{\text{RTS}}$ output to control the $\overline{\text{CTS}}$ input of the transmitting device to prevent receiver overrun. If both the receiver and transmitter are incorrectly programmed for $\overline{\text{RTS}}$ control, $\overline{\text{RTS}}$ control is disabled for both. Transmitter RTS control is configured in UMR2n[TxRTS]. Not used in modem mode.<br>0 The receiver has no effect on $\overline{\text{RTS}}$.<br>1 When a valid start bit is received, $\overline{\text{RTS}}$ is negated if the UART's FIFO is full. $\overline{\text{RTS}}$ is reasserted when the FIFO has an empty position available. |
| 6 | RxIRQ/ FFULL | Receiver interrupt select. This bit is used in UART and modem modes.<br>0 RxRDY is the source that generates IRQ.<br>1 FFULL is the source that generates IRQ. |
| 5 | ERR | Error mode. Configures the FIFO status bits, USRn[RB,FE,PE]. This bit is not used in modem mode.<br>0 Character mode. The USRn 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.<br>1 Block mode. The USRn values are the logical OR of the status for all characters reaching the top of the FIFO because the last RESET ERROR STATUS command for the channel was issued. See Section 14.3.10, “UART Command Registers (UCRn).” |
| 4–3 | PM | Parity mode. Selects the parity or multidrop mode for the channel. 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. PM is not used in modem mode. |
| 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). PT is not used in modem mode. |
| 1–0 | B/C | Bits per character. Select the number of data bits per character to be sent. The values shown do not include start, parity, or stop bits. B/C is not used in modem mode.<br>00 5 bits<br>01 6 bits<br>10 7 bits<br>11 8 bits |

| 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 |

## 14.3.2 UART Mode Register 2 (UMR2n)

UART mode registers 2 (UMR2*n*) 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.

| | 7 | 6 | 5 | 4 | 3 | 0 |
|---|---|---|---|---|---|---|
| Field | CM | | TxRTS | TxCTS | SB | |
| Reset | 0000_0000 | | | | | |
| R/W | R/W | | | | | |
| Address | MBAR + 0x1C0, 0x200. After UMR1n is read or written, the pointer points to UMR2n. | | | | | |

**Figure 14-3. UART Mode Register 2 (UMR2n)**

Table 14-3 describes UMR2*n* fields.

**Table 14-3. UMR2n Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–6 | CM | Channel mode. Selects a channel mode. Section 14.5.3, “Looping Modes,” describes individual modes. CM is used in both UART and modem modes.<br>00 Normal<br>01 Automatic echo<br>10 Local loop-back<br>11 Remote loop-back |
| 5 | TxRTS | Transmitter ready-to-send. Controls negation of $\overline{\text{RTS}}$ to automatically terminate a message transmission. Attempting to program a receiver and transmitter in the same channel for $\overline{\text{RTS}}$ control is not permitted and disables $\overline{\text{RTS}}$ control for both. TxRTS is not used in modem mode.<br>0 The transmitter has no effect on $\overline{\text{RTS}}$.<br>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 channel 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 enabled, TxCTS controls the operation of the transmitter. TxCTS is not used in modem mode.<br>0 $\overline{\text{CTS}}$ has no effect on the transmitter.<br>1 Enables clear-to-send operation. The transmitter checks the state of $\overline{\text{CTS}}$ each time it is ready to send a character. If $\overline{\text{CTS}}$ is asserted, the character is sent; if it is negated, the channel TxD remains in the high state and transmission is delayed until $\overline{\text{CTS}}$ is asserted. Changes in $\overline{\text{CTS}}$ as a character is being sent do not affect its transmission. |

**Table 14-3. UMR2n Field Descriptions (Continued)**

| Bits | Name | Description |
|---|---|---|
| 3–0 | SB | Stop-bit length control. Selects the length of the stop bit appended to the transmitted character. Stop-bit lengths of 9/16th to 2 bits are programmable for 6–8 bit characters. Lengths of 1 1/16th 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 2 stop bits for transmission. Not used in modem mode. |

| SB | 5 Bits | 6–8 Bits | SB | 5 Bits | 6–8 Bits | SB | 5–8 Bits | SB | 5–8 Bits |
|---|---|---|---|---|---|---|---|---|---|
| 0000 | 1.063 | 0.563 | 0100 | 1.313 | 0.813 | 1000 | 1.563 | 1100 | 1.813 |
| 0001 | 1.125 | 0.625 | 0101 | 1.375 | 0.875 | 1001 | 1.625 | 1101 | 1.875 |
| 0010 | 1.188 | 0.688 | 0110 | 1.438 | 0.938 | 1010 | 1.688 | 1110 | 1.938 |
| 0011 | 1.250 | 0.750 | 0111 | 1.500 | 1.000 | 1011 | 1.750 | 1111 | 2.000 |

## 14.3.3 Rx FIFO Threshold Register (RXLVL)

The Rx FIFO threshold register (RXLVL) supports UART1 only and is used in both UART and modem modes. The threshold is one less than the value at which the Rx FIFO is considered to be full for purposes of alerting the CPU that the Rx FIFO needs to be read.

| | 7 … 5 | 4 … 0 |
|---|---|---|
| Field | — | RXLVL |
| Reset | 000 | 0_0000 |
| R/W | R/W | |
| Address | MBAR + 0x201 | |

**Figure 14-4. Rx FIFO Threshold Register (RXLVL)**

Table 14-4 describes RXLVL fields.

**Table 14-4. RXLVL Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–5 | — | Reserved, should be cleared. |
| 4–0 | RXLVL | Rx FIFO full threshold level. Values of 0000–1111 specify 1–32 bytes. The Rx FIFO is full when the number of bytes in the FIFO equals or exceeds the Rx threshold. Although FIFO thresholds are in bytes, data is written into and read out of the FIFOs in numbers of samples, as follows:<br>• 1 sample = 1 byte for 8-bit CODEC and UART modes,<br>• 1 sample = 2 bytes for 16-bit CODEC and AC ‘97 modes<br>For choosing threshold values, AC ‘97 samples should be thought of as 2-byte entities.<br>The Rx threshold is RXLVL + 1, so RXLVL = [(# samples) * (# bytes per sample)] - 1. For example, for Rx FIFO to indicate full when 13 or more samples have arrived, calculate RXLVL as follows:<br>• For 8-bit CODEC or UART mode, RXLVL = [(13 samples) * (1 byte per sample)] - 1 = 12 bytes<br>• For 16-bit CODEC or AC ‘97 modes, RXLVL = [(13 samples) * (2 bytes per sample)] - 1 = 25 bytes |

## 14.3.4 Modem Control Register (MODCTL)

The modem control register (MODCTL), Figure 14-5, controls whether UART1 is in UART mode or in one of three modem modes.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field | ACRB | AWR | DSL | | DTS1 | SHDIR | MODE | |
| Reset | 1000_0000 | | | | | | | |
| R/W | R/W | | | | | | | |
| Address | MBAR + 0x202 | | | | | | | |

**Figure 14-5. Modem Control Register (MODCTL)**

Table 14-5 describes MODCTL fields.

**Table 14-5. Modem Control Register (MODCTL) Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7 | ACRB | AC ‘97 cold reset (active low).<br>0 The genera-purpose I/O used as the AC ‘97 cold reset output pin is active<br>1 The genera-purpose I/O used as the AC ‘97 cold reset output pin is inactive |
| 6 | AWR | AC ‘97 warm reset (active high)<br>0 Warm reset is inactive, letting UART1’s $\overline{\text{RTS}}$ output to function normally as the AC ‘97 frame sync.<br>1 Forces a 1 on UART1’s $\overline{\text{RTS}}$ output, which is used as the AC ‘97 frame sync. |
| 5–4 | DSL | Channel select for DMA channels 2 and 3. The sources for the interrupt request lines that drive DMA $\overline{\text{DREQ}}$[3:2] are selected by multiplexers in UART1, which are controlled by DSL.<br>00 DMA $\overline{\text{DREQ2}}$ is driven by UART0 combined Tx/Rx interrupt<br>DMA $\overline{\text{DREQ3}}$ is driven by UART1 combined Tx/Rx interrupt<br>01 DMA $\overline{\text{DREQ2}}$ is driven by UART1 Rx interrupt<br>DMA $\overline{\text{DREQ3}}$ is driven by UART1 Tx interrupt<br>10 same as 00<br>11 DMA $\overline{\text{DREQ2}}$ is driven by UART1 Rx interrupt<br>DMA $\overline{\text{DREQ3}}$ is driven by UART1 combined Tx/Rx interrupt. This combination is a by-product of implementation and may not be useful.<br>When UART1 uses both $\overline{\text{DREQ}}$ lines, $\overline{\text{DREQ3}}$ and $\overline{\text{DREQ2}}$ are driven by the Tx FIFO empty and Rx FIFO full conditions, respectively. The Rx FIFO not-empty condition can be used instead of Rx FIFO full by clearing UMR1n[6]. UART0 and UART1 have separate request lines to the interrupt controller. Each is sourced from the combined Tx/Rx interrupt from the associated UART. |
| 3 | DTS1 | Delay of time slot 1. Determines the starting point of the first bit of the first time slot of a new frame.<br>0 The rising edge of frame sync<br>1 One bit-clock cycle after the rising edge of frame sync |
| 2 | SHDIR | Shift direction. For AC ‘97 this bit must be 0.<br>0 Samples/time slots are transferred msb first<br>1 Samples/time slots are transferred lsb first |
| 1–0 | MODE | Mode select for UART1.<br>00 UART mode (default mode after hard reset). Changing from modem mode back to UART mode by writing 00 to this field has the same effect on UART1 as a hard reset—all registers and control logic are reset and the Tx and Rx FIFO pointers are reinitialized, effectively emptying the FIFOs.<br>01 8-bit CODEC interface mode<br>10 16-bit CODEC interface mode<br>11 AC ‘97 mode |

## 14.3.5 Tx FIFO Threshold Register (TXLVL)

Tx FIFO threshold register (TXLVL) supports only UART1 in both UART and modem modes. TXLVL holds the Tx FIFO threshold, the value at which the Tx FIFO is considered to be empty for purposes of alerting the CPU that the Tx FIFO needs more data/samples.

| | 7 | 5 | 4 | 0 |
|---|---|---|---|---|
| Field | — | | TXLVL | |
| Reset | — | | 0_0000 | |
| R/W | R/W | | | |
| Address | MBAR + 0x203 | | | |

**Figure 14-6. Tx FIFO Threshold Register (TXLVL)**

Table 14-6 describes TXLVL fields.

**Table 14-6. TXLVL Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–5 | — | Reserved, should be cleared. |
| 4–0 | TXLVL | Tx FIFO empty threshold level. 0000–1111 selects values of 0–31 bytes, respectively. The Tx FIFO is empty when the number of bytes in the FIFO is less than or equal to the Tx threshold value. Although the FIFO thresholds are in numbers of bytes, data is written into and read out of the FIFOs in numbers of samples, as follows:<br>1 sample = 1 byte for 8-bit CODEC and UART modes<br>1 sample = 2 bytes for 16-bit CODEC and AC ‘97 modes<br>For choosing threshold values, AC ‘97 samples should be thought of as 2-byte entities.<br>Choose the Tx threshold register value using the following formula:<br>TXLVL = (# samples) * (# bytes per sample)<br>For example, to indicate empty when three or fewer samples remain, calculate TXLVL as follows:<br>For 8-bit CODEC or UART mode, TXLVL = (3 samples) * (1 byte per sample) = 3 bytes<br>For 16-bit CODEC or AC ‘97 modes, TXLVL = (3 samples) * (2 bytes per sample) = 6 bytes |

## 14.3.6 UART Status Registers (USRn)

The USR*n*, Figure 14-7, shows status of the transmitter, the receiver, and the FIFO.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field | RB | FE | PE | OE | TxEMP | TxRDY | FFULL | RxRDY |
| Reset | 0000_0000 | | | | | | | |
| R/W | Read only | | | | | | | |
| Address | MBAR + 0x1C4 (USR0), 0x204 (USR1) | | | | | | | |

**Figure 14-7. UART Status Register (USRn)**

Table 14-7 describes USR*n* fields.

## Table 14-7. USRn Field Descriptions

| Bits | Name | 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. RB is not used (and is always 0) in modem mode.<br>0 No break was received.<br>1 An all-zero character of the programmed length was received without a stop bit. RB is valid only when RxRDY = 1. Only a single FIFO position is occupied when a break is received. Further entries to the FIFO are inhibited until RxD returns to the high state for at least one-half bit time, which is equal to two successive edges of the UART clock. |
| 6 | FE | Framing error. FE is not used (and is always 0) in modem mode.<br>0 No framing error occurred.<br>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. PE is not used (and is always 0) in modem mode.<br>0 No parity error occurred.<br>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 A/D bit. |
| 4 | OE | Overrun error. Indicates whether an overrun occurs. OE also functions this way for UART1 in modem mode. (For purposes of overrun, FIFO full means all space in the FIFO is occupied; the Rx FIFO threshold is irrelevant to overrun.)<br>0 No overrun occurred.<br>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 | TxEMP | Transmitter empty. For UART1, the function of TxEMP depends on which mode is used.<br>UART mode:<br>0 The transmitter 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].<br>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.<br>Modem mode:<br>0 The transmitter does not have underrun as described above.<br>1 The transmitter has underrun, which means the number of bytes in the Tx FIFO is zero, the Tx shift register is empty, and a frame sync occurs. In other words, the time has come to transmit a new sample but no sample is available in the Tx shift register. Unlike UART mode, TxEMP high indicates an error condition similar to the overrun condition (OE = 1), and as such it is now cleared the same way as OE, by a RESET ERROR STATUS command in the UCRn and not by a RESET TRANSMITTER command in the UCRn. |
| 2 | TxRDY | Transmitter ready.<br>UART0:<br>0 The CPU loaded the transmitter holding register or the transmitter is disabled.<br>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 and when the transmitter is first enabled. If the transmitter is disabled, characters loaded into the transmitter holding register are not sent.<br>UART1 (in UART or modem modes):<br>0 The transmitter FIFO is not empty, or the transmitter is disabled.<br>1 The transmitter FIFO is empty, as defined by TXLVL. TxRDY is set when the number of bytes in the Tx FIFO falls to, or below, the TXLVL value, due to the transfer of a sample (1 or 2 bytes) from the Tx FIFO to the Tx shift register. |

**Table 14-7. USRn Field Descriptions (Continued)**

| Bits | Name | Description |
|---|---|---|
| 1 | FFULL | FIFO full.<br>UART0:<br>0 The FIFO is not full but may hold up to two unread characters.<br>1 A character was received and is waiting in the receiver buffer FIFO.<br>UART1 (in UART or modem modes):<br>1 Rx FIFO is full, as defined by the RXLVL. FFULL is set as soon as the number of bytes in the Rx FIFO exceeds the RXLVL value, due to the transfer of a sample (1 or 2 bytes) from the Rx shift register to the Rx FIFO. |
| 0 | RxRDY | Receiver ready (in UART or modem modes).<br>0 The CPU has read the receiver buffer and no characters remain in the FIFO after this read.<br>1 One or more characters were received and are waiting in the receiver buffer FIFO. |

## 14.3.7 UART Clock-Select Registers (UCSRn)

The UART clock-select registers (UCSR*n*) select an external clock on the TIN input (divided by 1 or 16) or a prescaled CLKIN as the clocking source for the transmitter and receiver. See Section 14.5.1, “Transmitter/Receiver Clock Source.” UCSR1 is used in UART mode only. The transmitter and receiver can use different clock sources. To use CLKIN for both, set UCSR*n* to 0xDD.

| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Field | RCS | | | | TCS | | | |
| Reset | 0000_0000 | | | | | | | |
| R/W | Write only | | | | | | | |
| Address | MBAR + 0x1C4 (UCSR0), 0x204 (UCSR1) | | | | | | | |

**Figure 14-8. UART Clock-Select Register (UCSRn)**

Table 14-8 describes UCSR*n* fields.

**Table 14-8. UCSRn Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–4 | RCS | Receiver clock select. Selects the clock source for the receiver channel.<br>1101 Prescaled CLKIN<br>1110 TIN divided by 16<br>1111 TIN |
| 3–0 | TCS | Transmitter clock select. Selects the clock source for the transmitter channel.<br>1101 Prescaled CLKIN<br>1110 TIN divided by 16<br>1111 TIN |

## 14.3.8 Receive Samples Available Register (RSMP)

The receive samples available register (RSMP), Figure 14-9, shows the current byte count for the Rx FIFO. It is in UART1 only and can be used in both UART and modem modes.

| | 7 | 5 | 4 | 0 |
|---|---|---|---|---|
| Field | — | | RSMP | |
| Reset | — | | 0_0000 | |
| R/W | Read only | | | |
| Address | MBAR + 0x206 | | | |

**Figure 14-9. Receive Samples Available Register (RSMP)**

Table 14-9 describes RSMP fields.

**Table 14-9. RSMP Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–5 | — | Reserved, should be cleared. |
| 4–0 | RSMP | Number of bytes in the Rx FIFO. |

## 14.3.9 Transmit Space Available Register (TSPC)

The transmit space available register (TSPC), Figure 14-10, shows available bytes in Tx FIFO. TSPC supports UART1 only and can be used in UART and modem modes.

| | 7 | 5 | 4 | 0 |
|---|---|---|---|---|
| Field | — | | TSPC | |
| Reset | — | | 0_0000 | |
| R/W | Read only | | | |
| Address | MBAR + 0x207 | | | |

**Figure 14-10. Tx Space Available Register (TSPC)**

Table 14-10 describes TSPC fields.

**Table 14-10. TSPC Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–5 | — | Reserved, should be cleared. |
| 4–0 | TSPC | Number of empty bytes in the Tx FIFO. |

## 14.3.10 UART Command Registers (UCRn)

The UART command registers (UCR*n*), Figure 14-11, supply commands to the UART in both UART and modem modes. 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.

| | 7 | 6 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Field | — | MISC | | TC | | RC | |
| Reset | 0000_0000 | | | | | | |
| R/W | Write only | | | | | | |
| Address | MBAR + 0x1C8, 0x208 | | | | | | |

**Figure 14-11. UART Command Register (UCRn)**

Table 14-11 describes UCR*n* fields and commands. Examples in Section 14.5.2, "Transmitter and Receiver Operating Modes," show how these commands are used.

**Table 14-11. UCRn Field Descriptions**

| Bits | Value | Command | Description |
|---|---|---|---|
| 7 | — | — | Reserved, should be cleared. |
| 6–4 | **MISC Field** (This field selects a single command.) | | |
| | 000 | NO COMMAND | — |
| | 001 | RESET MODE REGISTER POINTER | Causes the mode register pointer to point to UMR1n. |
| | 010 | RESET RECEIVER | Immediately disables the receiver, clears USRn[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 | In UART mode, immediately disables the transmitter and clears USRn[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. When UART1 is in modem mode, TxEMP is not cleared by this soft reset. It is cleared the same way as the Rx overflow bit, by a RESET ERROR STATUS command. |
| | 100 | RESET ERROR STATUS | In UART mode, clears USRn[RB,FE,PE,OE]. Also used in block mode to clear all error bits after a data block is received. When UART1 is in modem mode, this command also clears TxEMP. |
| | 101 | RESET BREAK–CHANGE INTERRUPT | Clears the delta break bit, UISRn[DB]. This command has no effect in modem mode. |
| | 110 | START BREAK | Forces TxD 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 $\overline{\text{CTS}}$ and has no effect in modem mode. |
| | 111 | STOP BREAK | Causes TxD to go high (mark) within two bit times. Any characters in the transmitter buffer are sent. |

**Table 14-11. UCRn Field Descriptions (Continued)**

| Bits | Value | Command | Description |
|---|---|---|---|
| 3–2 | **TC Field** (This field selects a single command) | | |
| | 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 channel's transmitter. USRn[TxEMP,TxRDY] are set. If the transmitter is already enabled, this command has no effect.<br>For UART1 in modem mode, Tx FIFO can be loaded while the transmitter is disabled, unlike in UART mode. Therefore, this command does not affect the behavior of TxRDY. It does not automatically set TxRDY and TxEMP; however, if no data is written to the Tx FIFO, TxEMP is set at the first frame sync after the transmitter is enabled.<br>In AC '97 mode, TxEMP is set if Tx FIFO is empty, the transmitter is enabled, the receiver detects a coded ready condition, and a frame sync occurs before samples are written to the Tx FIFO. |
| | 10 | TRANSMITTER DISABLE | Terminates transmitter operation and clears USRn[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.<br>In modem mode, the transmitter does not clear USRn[TxRDY] unless UART1 is in remote loop-back or auto-echo mode. This is because in modem mode, unlike in UART mode, the Tx FIFO may be loaded while the Tx is disabled. |
| | 11 | — | Reserved, do not use. |
| 1–0 | **RC** (This field selects a single command) | | |
| | 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 (UMR1n[PM] ≠ 11), RECEIVER ENABLE enables the channel'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.<br>When UART1 is in modem mode, if the receiver is disabled while a character is being received, reception completes before the receiver becomes inactive. |
| | 11 | — | Reserved, do not use. |

## 14.3.11 UART Receiver Buffers (URBn)

The receiver buffer for UART0 contains one serial shift register and three receiver holding registers, which act as a FIFO. RxD is connected to the serial shift register. The CPU reads from the top of the stack while the receiver shifts and updates from the bottom when the shift register is full (see Figure 14-29). RB contains the character in the receiver.

| | 7 | 0 |
|---|---|---|
| Field | RB | |
| Reset | 0000_0000 | |
| R/W | Read only | |
| Address | MBAR + 0x1CC | |

**Figure 14-12. UART Receiver Buffer for UART0 (URB0)**

Figure 14-13 shows the configuration of URB1.

| | 31 24 | 23 16 | 15 8 | 7 0 |
|---|---|---|---|---|
| Field | RB[31:24] | RB[23:16] | RB[15:8] | RB[7:0] |
| Reset | 0000_0000_0000_0000_0000_0000_0000_0000 | | | |
| R/W | Read only | | | |
| Address | MBAR + 0x20C | | | |

**Figure 14-13. UART Receiver Buffer for UART1 (URB1)**

## 14.3.12 UART Transmitter Buffers (UTBn)

The transmitter buffer for UART0 consists of the transmitter holding register and the transmitter shift register. The holding register accepts characters from the bus master if channel's USR*n*[TxRDY] is set. A write to the transmitter buffer clears 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 transmitter buffer when the channel's TxRDY = 0 and when the transmitter is disabled have no effect on the transmitter buffer.

Figure 14-14 shows UTB0. TB contains the character in the transmitter buffer.

| | 7 | 0 |
|---|---|---|
| Field | TB | |
| Reset | 0000_0000 | |
| R/W | Write only | |
| Address | MBAR + 0x1CC | |

**Figure 14-14. UART Transmitter Buffer for UART0 (UTB0)**

The transmitter buffer in UART1 consists of the transmitter shift register and the Tx FIFO, as described in Section 14.5.2.6, "FIFOs in UART1." The Tx FIFO in UART1 accepts characters/samples from the bus master if there is room for them in the FIFO. A write to the transmitter buffer clears TxRDY if the number of bytes in the FIFO exceeds the threshold level in TXLVL. When the shift register is empty, it checks if the FIFO has a valid character/sample to be sent. Valid characters are loaded into the shift register. Unlike UART

mode, in modem mode the Tx FIFO in UART1 can be loaded while the Tx is disabled. For UART1, FIFOs can be accessed as longwords.

Figure 14-15 shows the configuration of the UTB1. These bits contain the samples in the transmitter buffer for UART1.

| | 31 24 | 23 16 | 15 8 | 7 0 |
|---|---|---|---|---|
| Field | TB[31:24] | TB[23:16] | TB[15:8] | TB[7:0] |
| Reset | 0000_0000_0000_0000_0000_0000_0000_0000 | | | |
| R/W | Write only | | | |
| Address | MBAR + 0x20C | | | |

**Figure 14-15. UART Transmitter Buffer for UART1 (UTB1)**

## 14.3.13 UART Input Port Change Registers (UIPCRn)

The input port change registers (UIPCR*n*), Figure 14-16, hold the current state and the change-of-state for $\overline{\text{CTS}}$.

| | 7 5 | 4 | 3 1 | 0 |
|---|---|---|---|---|
| Field | — | COS | 111 | CTS |
| Reset | 0000 | | 0 11 | $\overline{\text{CTS}}$ |
| R/W | Read only | | | |
| Address | MBAR + 0x1D0 (UIPCR0), 0x210 (UIPCR1) | | | |

**Figure 14-16. UART Input Port Change Register (UIPCRn)**

Table 14-12 describes UIPCR*n* fields.

**Table 14-12. UIPCRn Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–5 | — | Reserved, should be cleared. |
| 4 | COS | Change of state (high-to-low or low-to-high transition). Not used in modem mode.<br>0 No change-of-state since the CPU last read UIPCRn. Reading UIPCRn clears UISRn[COS].<br>1 A change-of-state longer than 25–50 μs occurred on the $\overline{\text{CTS}}$ input. UACRn can be programmed to generate an interrupt to the CPU when a change of state is detected. |
| 3–1 | — | Reserved, should be cleared. |
| 0 | CTS | Current state. Starting two serial clock periods after reset, CTS reflects the state of $\overline{\text{CTS}}$. If $\overline{\text{CTS}}$ is detected asserted at that time, COS is set, which initiates an interrupt if UACRn[IEC] is enabled. This bit is not used in modem mode.<br>0 The current state of the $\overline{\text{CTS}}$ input is asserted.<br>1 The current state of the $\overline{\text{CTS}}$ input is negated. |

## 14.3.14 UART Auxiliary Control Register (UACRn)

The UART auxiliary control registers (UACR*n*), Figure 14-12, control the input enable.

| | 7 | 1 | 0 |
|---|---|---|---|
| Field | — | | IEC |
| Reset | 0000_0000 | | |
| R/W | Write only | | |
| Address | MBAR + 0x1D0 (UACR0), 0x210 (UACR1) | | |

**Figure 14-17. UART Auxiliary Control Register (UACRn)**

Table 14-13 describes UACR*n* fields.

**Table 14-13. UACRn Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–1 | — | Reserved, should be cleared. |
| 0 | IEC | Input enable control. This bit is not used in modem mode.<br>0 Setting the corresponding UIPCRn bit has no effect on UISRn[COS].<br>1 UISRn[COS] is set and an interrupt is generated when the UIPCRn[COS] is set by an external transition on the $\overline{\text{CTS}}$ input (if UIMRn[COS] = 1). |

## 14.3.15 UART Interrupt Status/Mask Registers (UISRn/UIMRn)

The UART interrupt status registers (UISR*n*), Figure 14-18, 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.

**NOTE:**

True status is provided in the UISR*n* regardless of UIMR*n* settings. UISR*n* is cleared when the UART module is reset.

| | 7 | 6 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|
| Field | COS | — | | DB | FFULL/RxRDY | TxRDY |
| Reset | 0000_0000 | | | | | |
| R/W | Read only | | | | | |
| Address | MBAR + 0x1D4 (UISR0), 0x214 (UISR1); MBAR + 0x1D4 (UIMR0), 0x214 (UIMR1) | | | | | |

**Figure 14-18. UART Interrupt Status/Mask Registers (UISRn/UIMRn)**

Table 14-14 describes UISR*n* and UIMR*n* fields.

**Table 14-14. UISRn/UIMRn Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7 | COS | Change-of-state. Not used by UART1 in modem mode.<br>0 UIPCRn[COS] is not selected.<br>1 Change-of-state occurred on $\overline{\text{CTS}}$ and was programmed in UACRn[IEC] to cause an interrupt. |
| 6–3 | — | Reserved, should be cleared. |
| 2 | DB | Delta break. Not used by UART1 in modem mode.<br>0 No new break-change condition to report. Section 14.3.10, “UART Command Registers (UCRn),” describes the RESET BREAK-CHANGE INTERRUPT command.<br>1 The receiver detected the beginning or end of a received break. |
| 1 | FFULL/ RxRDY | RxRDY (receiver ready) if UMR1n[FFULL/RxRDY] = 0; FIFO full (FFULL) if UMR1n[FFULL/RxRDY] = 1. Duplicate of USRn[FFULL/RxRDY]. Used by UART1 in modem mode. If FFULL is enabled for UART0 or UART1, DMA channels 2 or 3 are respectively interrupted when the FIFO is full. |
| 0 | TxRDY | Transmitter ready. This bit is the duplication of USRn[TxRDY]. Used by UART1 in modem mode.<br>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.<br>1 The transmitter holding register is empty and ready to be loaded with a character. |

## 14.3.16 UART Divider Upper/Lower Registers (UDUn/UDLn)

The UDU*n* registers (formerly called UBG1*n*) holds the MSB, and the UDL*n* registers (formerly UBG2*n*) hold the LSB of the preload value. UDU*n* and UDL*n* concatenate to provide a divider to CLKIN for transmitter/receiver operation, as described in Section 14.5.1.2.1, “CLKIN Baud Rates.”

| | 7 | 0 |
|---|---|---|
| Field | Divider MSB | |
| Reset | 0000_0000 | |
| R/W | R/W | |
| Address | MBAR + 0x1D8 (UDU0), 0x218 (UDU1) | |

**Figure 14-19. UART Divider Upper Register (UDUn)**

| | 7 | 0 |
|---|---|---|
| Field | Divider LSB | |
| Reset | 0000_0000 | |
| R/W | R/W | |
| Address | MBAR + 0x1DC (UDL0), 0x21C (UDL1) | |

**Figure 14-20. UART Divider Lower Register (UDLn)**

**NOTE:**

The minimum value that can be loaded on the concatenation of UDU*n* with UDL*n* is 0x0002. Both UDU*n* and UDL*n* are write-only and cannot be read by the CPU.

## 14.3.17 UART Interrupt Vector Register (UIVRn)

The UIVR*n*, Figure 14-21, contain the 8-bit internal interrupt vector number (IVR).

| | 7 | 0 |
|---|---|---|
| Field | IVR | |
| Reset | 0000_1111 | |
| R/W | R/W | |
| Address | MBAR + 0x1F0 (UIVR0), 0x230 (UIVR1) | |

**Figure 14-21. UART Interrupt Vector Register (UIVRn)**

Table 14-15 describes UIVR*n* fields.

**Table 14-15. UIVRn Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–0 | IVR | Interrupt vector. Indicates the vector number where the address of the exception handler for the specified interrupt is located. UIVRn is reset to 0x0F, indicating an uninitialized interrupt condition. |

## 14.3.18 UART Input Port Register (UIPn)

The UART input port registers (UIP*n*), Figure 14-22, show the current state of the $\overline{\text{CTS}}$ input when the processor is in UART mode.

| | 7 | 1 | 0 |
|---|---|---|---|
| Field | — | | CTS |
| Reset | 1111_1111 | | |
| R/W | Read only | | |
| Address | MBAR + 0x1F4 (UIP0), 0x234 (UIP1) | | |

**Figure 14-22. UART Input Port Register (UIPn)**

Table 14-16 describes UIP*n* fields.

**Table 14-16. UIPn Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–1 | — | Reserved, should be cleared. |
| 0 | CTS | Current state. The $\overline{\text{CTS}}$ value is latched and reflects the state of the input pin when UIPn is read.<br>Note: This bit has the same function and value as UIPCRn[RTS].<br>0 The current state of the $\overline{\text{CTS}}$ input is logic 0.<br>1 The current state of the $\overline{\text{CTS}}$ input is logic 1.<br>When UART1 is in modem mode, $\overline{\text{CTS}}$ toggles when a frame sync occurs. It is used during testing to synchronize test code running on the CPU with frames transferred on the serial interface. |

### 14.3.19 UART Output Port Data Registers (UOP1n/UOP0n)

In UART mode, the $\overline{\text{RTS}}$ output can be asserted by writing a 1 to UOP1*n*[RTS] and negated by writing a 1 to UOP0*n*[RTS]. UOP registers have no effect in modem mode. See Figure 14-23.

| | 7 … 1 | 0 |
|---|---|---|
| Field | — | RTS |
| Reset | 0000_0000 | |
| R/W | Write only | |
| Addr | UART0: MBAR + 0x1F8 (UOP1), 0x1FC (UOP0); UART1 0x238 (UOP1), 0x23C (UOP0) | |

**Figure 14-23. UART Output Port Data 1 Register (UOP1/UOP0)**

Table 14-17 describes UOP1 fields.

**Table 14-17. UOP1/UOP0 Field Descriptions**

| Bits | Name | Description |
|---|---|---|
| 7–1 | — | Reserved, should be cleared. |
| 0 | RTS | Output port parallel output. Controls assertion (UOP1)/negation (UOP0) of $\overline{\text{RTS}}$ output.<br>0 Not affected.<br>1 Asserts $\overline{\text{RTS}}$ (UOP1). Negates $\overline{\text{RTS}}$ (UOP0). |

## 14.4 UART Module Signal Definitions

Figure 14-24 shows both the external and internal signal groups.

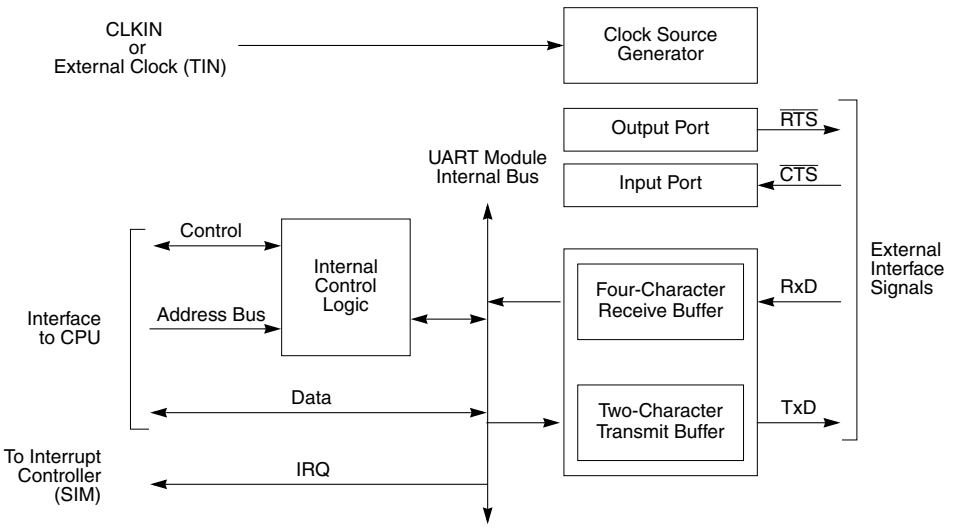



**Figure 14-24. UART Block Diagram Showing External and Internal Interface Signals**

An internal interrupt request signal ($\overline{IRQ}$) 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 of a UART module is programmed in the interrupt controller in the system integration module (SIM). The UART can use the autovector for the programmed interrupt level or supply the vector from the UIVR*n* when the UART interrupt is acknowledged.

The interrupt level, priority, and auto-vectoring capability is programmed in SIM register ICR4 for UART0 and ICR5 for UART1. See Section 9.2.1, “Interrupt Control Registers (ICR0–ICR9).”

Note that the UARTs can also automatically transfer data by using the DMA rather than interrupting the core. When UIMR[FFULL] is 1 and a receiver’s FIFO is full, it can send an interrupt to a DMA channel so the FIFO data can be transferred to memory. Note also that UART0 and UART1’s interrupt requests are connected to DMA channel 2 and channel 3, respectively.

Table 14-18 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 14-18. UART Module Signals**

| Signal | Description |
|---|---|
| Transmitter Serial Data Output (TxD) | In UART mode, TxD is held high (mark condition) when the transmitter is disabled, idle, or operating in the local loop-back mode. Data is shifted out on TxD on the falling edge of the clock source, with the least significant bit (lsb) sent first. For UART1 in modem mode, TxD is held low when the transmitter is disabled or idle. Data is shifted out on TxD on the rising edge of the clock signal driving UART1’s $\overline{CTS}$ input. UART1 transfers can be specified as either lsb or msb first. |
| Receiver Serial Data Input (RxD) | Data received on RxD is sampled on the rising edge of the clock source, with the lsb received first. For UART1 in modem mode, data received on RxD is sampled on the falling edge of the clock signal driving UART1’s $\overline{CTS}$ input. UART1 transfers can be specified as either lsb or msb first. |
| Clear-to-Send ($\overline{CTS}$) | This input can generate an interrupt on a change of state. For UART1 in modem mode, $\overline{CTS}$ must be driven by the serial bit clock from the external CODEC or AC ‘97 controller. |
| Request-to-Send ($\overline{RTS}$) | This output can be programmed to be negated or asserted automatically by either the receiver or the transmitter. When connected to a transmitter’s $\overline{CTS}$, $\overline{RTS}$ can control serial data flow. For UART1 in AC ‘97 mode, $\overline{RTS}$ serves as the frame sync or start-of-frame (SOF) output to the external AC ‘97 controller. When this mode is used, the AC ‘97 BIT_CLK, which is input on $\overline{CTS}$, is divided by 256. |
| Timer Input (TIN1) | When UART1 in modem mode is used as an 8- or 16-bit CODEC interface, TIN1 must be driven by the frame sync or SOF output from the external CODEC. SOF is sampled on the falling edge of the bit clock driving $\overline{CTS}$. TIN1 can still be used in all timer modes except capture mode when UART1 is being used as an 8- or 16-bit CODEC interface. See Table 14-26. |

Figure 14-25 shows a signal configuration for a UART/RS-232 interface.

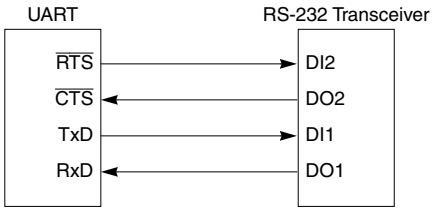



**Figure 14-25. UART/RS-232 Interface**

Figure 14-26 shows a signal configuration for a UART1/CODEC interface.

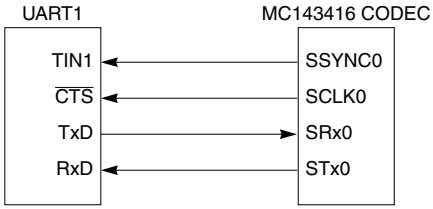



**Figure 14-26. UART1/CODEC Interface**

Figure 14-27 shows a signal configuration for a UART1/CODEC interface. An MCF5407 general-purpose I/O (GPIO) is used as a reset to the AC ‘97 device.

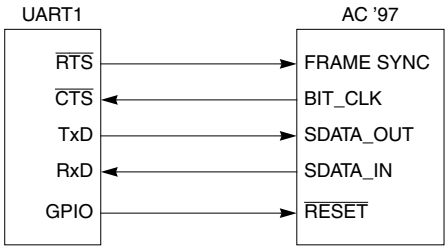



**Figure 14-27. UART1/AC ’97 Interface**

## 14.5 Operation

This section describes operation of the clock source generator, transmitter, and receiver.

### 14.5.1 Transmitter/Receiver Clock Source

CLKIN 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 the UART. The clock generator cannot produce standard baud rates if CLKIN is used, so the 16-bit divider should be used.

#### 14.5.1.1 Programmable Divider

As Figure 14-28 shows, the UART transmitter and receiver can use the following clock sources:

- An external clock signal on the TIN pin that can be divided by 16. When not divided, TIN provides a synchronous clock mode; when divided by 16, it is asynchronous.
- CLKIN supplies an asynchronous clock source that is divided by 32 and then divided by the 16-bit value programmed in UDU*n* and UDL*n*. See Section 14.3.16, "UART Divider Upper/Lower Registers (UDUn/UDLn)."

The choice of TIN or CLKIN is programmed in the UCSR.

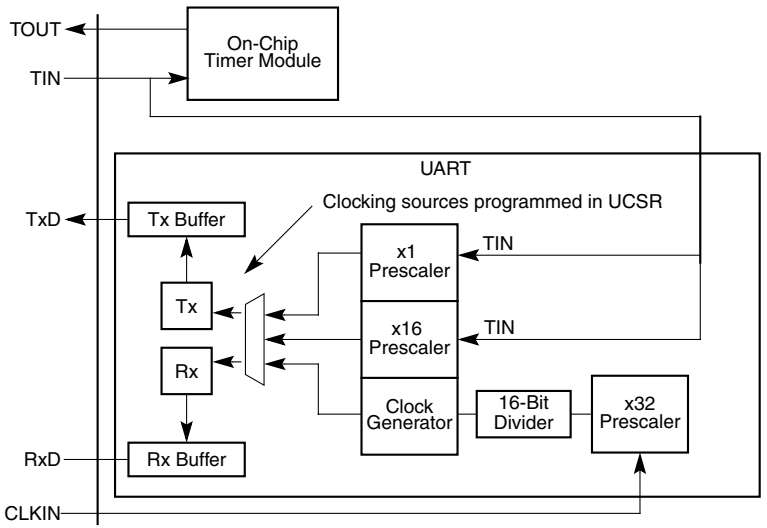



**Figure 14-28. Clocking Source Diagram**

**NOTE:**

If TIN is a clocking source for either the timer or UART, as is the case for UART1 used as an 8- or 16-bit CODEC interface, the timer module cannot use TIN for timer capture.

#### 14.5.1.2 Calculating Baud Rates

The following sections describe how to calculate baud rates.

##### 14.5.1.2.1 CLKIN Baud Rates

When CLKIN is the UART clocking source, it goes through a divide-by-32 prescaler and then passes through the 16-bit divider of the concatenated UDU*n* and UDL*n* registers. Using a 54-MHz CLKIN, the baud-rate calculation is as follows:

$$\text{Baudrate} = \frac{54\text{MHz}}{[32 \times \text{divider}]}$$

Let baud rate = 9600; the divider can be calculated as follows:

$$\text{Divider} = \frac{54\text{MHz}}{[32 \times 9600]} = 176(\text{decimal}) = 0\text{x}00\text{B}0$$

Therefore UDU*n* = 0x00 and UDL*n* = 0xB0.

#### 14.5.1.2.2 External Clock

An external source clock (TIN) can be used as is or divided by 16.

$$\text{Baudrate} = \frac{\text{Externalclockfrequency}}{16\text{or}1}$$

## 14.5.2 Transmitter and Receiver Operating Modes

Figure 14-29 is a functional block diagram of the transmitter and receiver showing the command and operating registers, which are described generally in the following sections and described in detail in Section 14.3, “Register Descriptions.”

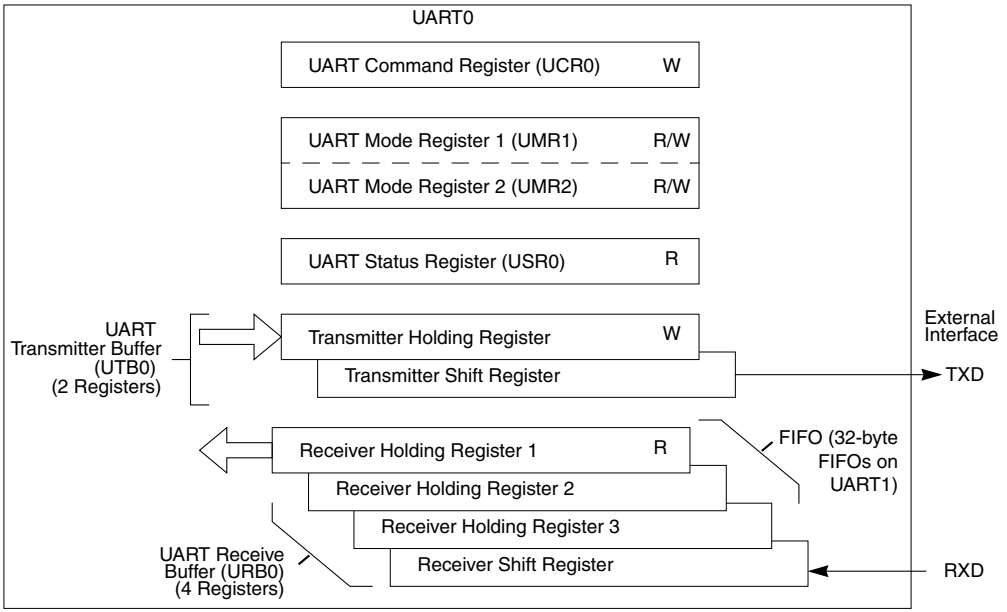



**Figure 14-29. Transmitter and Receiver Functional Diagram**

### 14.5.2.1 Transmitting in UART Mode

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 TxD. 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.

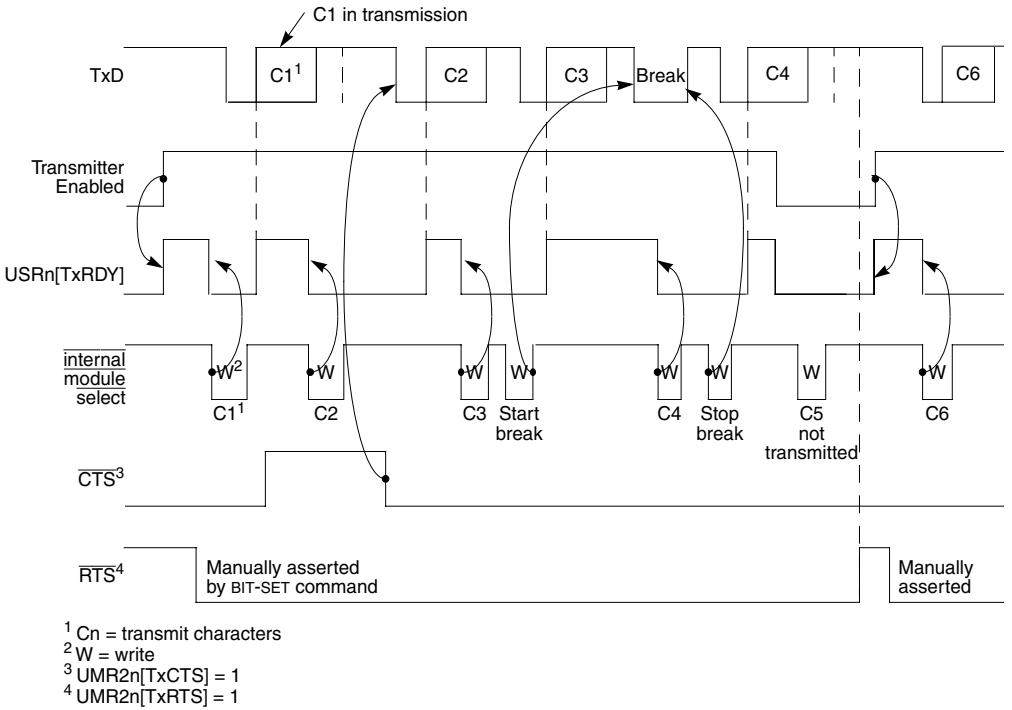
After the stop bits are sent, if no new character is in the transmitter holding register, the TxD output remains high (mark condition) and the transmitter empty bit, USR*n*[TxEMP], is set. Transmission resumes and TxEMP is cleared when the CPU loads a new character into the UART transmitter buffer (UTB*n*). 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 14.3.10, “UART Command Registers (UCRn)”). 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, $\overline{\text{CTS}}$ must be asserted for the character to be transmitted. If $\overline{\text{CTS}}$ is negated in the middle of a transmission, the character in the shift register is sent and TxD remains in mark state until $\overline{\text{CTS}}$ 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 $\overline{\text{CTS}}$.

If the transmitter is programmed to automatically negate $\overline{\text{RTS}}$ when a message transmission completes, $\overline{\text{RTS}}$ must be asserted manually before a message is sent. In applications in which the transmitter is disabled after transmission is complete and $\overline{\text{RTS}}$ is appropriately programmed, $\overline{\text{RTS}}$ is negated one bit time after the character in the shift register is completely transmitted. The transmitter must be manually reenabled by reasserting $\overline{\text{RTS}}$ before the next message is to be sent.

Figure 14-30 shows the functional timing information for the transmitter.




[1] Cn = transmit characters
[2] W = write
[3] UMR2n[TxCTS] = 1
[4] UMR2n[TxRTS] = 1

**Figure 14-30. Transmitter Timing Diagram**

### 14.5.2.2 Transmitter in Modem Mode (UART1)

After a hardware reset, UART1 is in UART mode. UART1 can be put in one of the modem modes by writing the appropriate value for MODCTL[MODE], as described in Section 14.3.4, "Modem Control Register (MODCTL)." The other MODCTL fields should be initialized at the same time. Set the Tx FIFO threshold as described in Section 14.3.5, "Tx FIFO Threshold Register (TXLVL)."

The serial bit clock is always an input to UART1 in modem mode (on $\overline{\text{CTS}}$). For an 8- or 16-bit CODEC, the frame sync is also an input to UART1 (on TIN1). However for an AC '97 controller, UART1 provides the frame sync (on $\overline{\text{RTS}}$). Figure 14-31 shows an example timing for UART1-CODEC interfaces (lsb first).

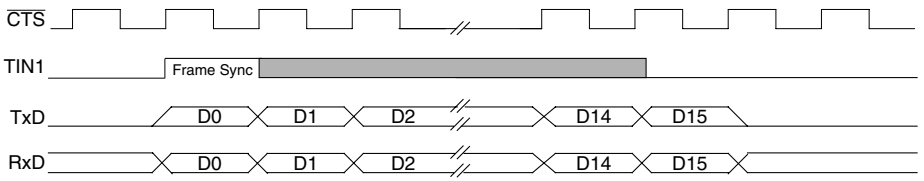



**Figure 14-31. 16-Bit CODEC Interface Timing (lsb First)**

Figure 14-32 is an example timing diagram for the UART1-CODEC interface (msb first).

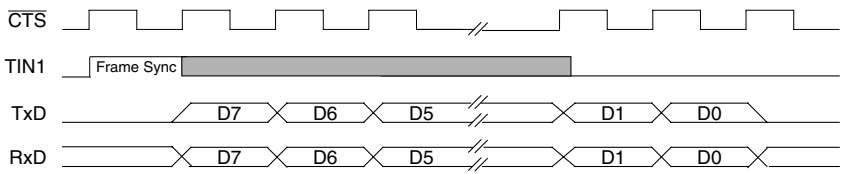



**Figure 14-32. 8-Bit CODEC Interface Timing (msb First)**

Figure 14-33 shows an example timing diagram for the UART1-AC '97 interface.

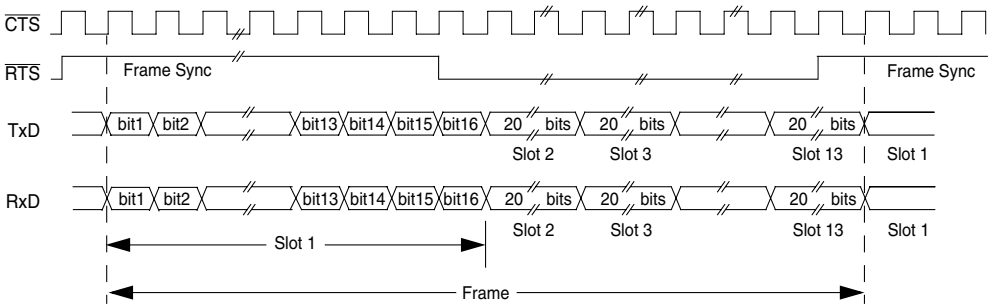



**Figure 14-33. AC '97 Interface Timing**

For more information about interfacing to an AC '97 controller, refer to the *Audio CODEC '97 Component Specification*.

When interfaced to an 8- or 16-bit CODEC (MODCTL[MODE] = 01 or 10), UART1 starts to send a sample either during the 1-bit clock cycle after the rising edge of frame sync, according to the value of MODCTL[DTS1]. The width of the frame sync pulse makes no difference. MODCTL[SHDIR] controls whether bits are shifted out msb or lsb first. After the 8- or 16-bit sample is sent, zeros are sent until the next frame sync.

When interfacing to an AC '97 controller (MODCTL[MODE] = 11), UART1 starts to transmit time slot 1 data one bit-clock cycle after the rising edge of frame sync, regardless of the value of MODCTL[DTS1]. However, MODCTL[SHDIR] must be 0, because the shift order must be msb first. UART1 divides the bit clock by 256 to generate a frame sync pulse that is high for 16-bit clock cycles. The transmitter sends zeros until the receiver detects the CODEC-ready condition (a 1 in the first bit of a new frame).

Because Rx data is sampled on the falling edge of the bit clock, for transmit purposes, the frame has already started when the receiver detects a CODEC-ready condition. For this reason, transmission starts at the next frame sync after the CODEC-ready condition is detected. UART1 stops transmission at the end of the frame in which the first bit of the received frame is detected low (CODEC not ready). During transmission, UART1 fills each of the 13 time slots of the AC '97 frame with samples from the Tx FIFO.

#### 14.5.2.2.1 AC ‘97 Low-Power Mode

A general-purpose I/O (GPIO) must be used as an AC ‘97 reset output pin. UART1 monitors the first three time slots of each Tx frame to detect the power-down condition for the AC ‘97 digital interface. The power-down condition is detected as follows:

1. The first 3 bits of slot 1 must be set, indicating that the Tx frame and slots 1 and 2 are valid.
2. Slot 2 holds the address of the power-down register (0x26) in the external AC ‘97 device.
3. Slot 3 contains a 1 in the fourth bit (bit 12/PR4 in power-down register 1), as defined in the AC ’97 specification.

Low-power mode can be left through either a warm or cold reset. The CPU performs a warm reset by setting MODCTL[AWR] for at least 1 $\mu$s. This negates $\overline{\text{RTS}}$, which is used as the frame sync output in AC ‘97 mode. The CPU performs a cold reset in two steps:

1. Writes a 0 to whichever GPIO is being used as the active low AC ‘97 reset pin for the minimum time specified in the AC ‘97 specification.
2. Writes a 0 to UART1’s MODCTL[ACRB] (bit 7). The CPU sets this bit after writing a 1 to the GPIO used for the AC ‘97 reset pin.

Step 2 is required so UART1 knows when an AC ‘97 cold reset is occurring.

### 14.5.2.3 Receiver

The receiver is enabled through its UCR*n*, as described in Section 14.3.10, “UART Command Registers (UCRn).” Figure 14-34 shows receiver functional timing.

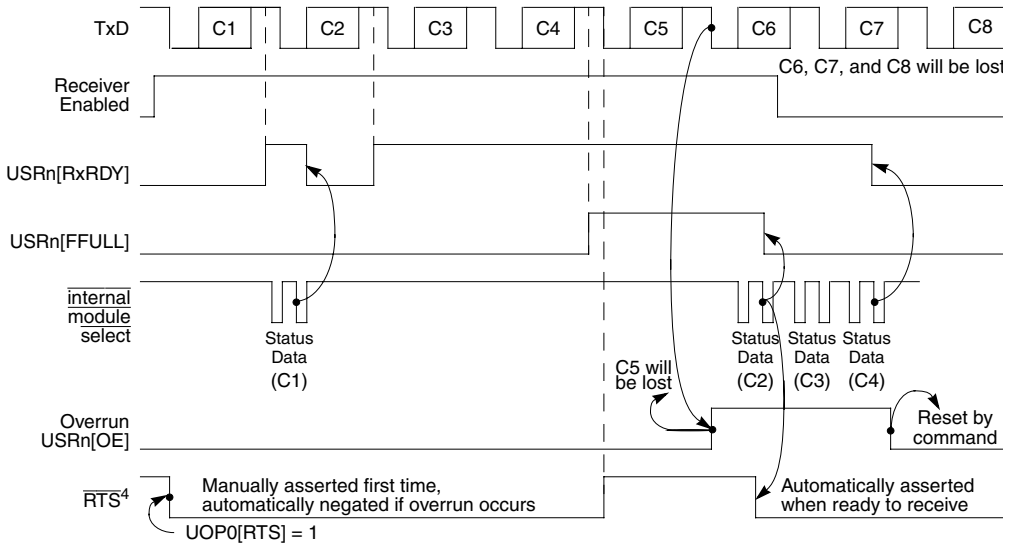



**Figure 14-34. Receiver Timing**

When the receiver detects a high-to-low (mark-to-space) transition of the start bit on RxD, the state of RxD is sampled each 16× clock for eight clocks, starting one-half clock after the transition (asynchronous operation) or at the next rising edge of the bit time clock (synchronous operation). If RxD is sampled high, the start bit is invalid and the search for the valid start bit begins again.

If RxD 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 RxD 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 USR*n*[RxRDY] is set. If the character is less than eight 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 RxD 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, RB error and break flags in the USR*n* at the received character boundary and are valid only if USR*n*[RxRDY] is set.

If a break condition is detected (RxD is low for the entire character including the stop bit), a character of all zeros is loaded into the receiver holding register (RHR) and USR*n*[RB,RxRDY] are set. RxD 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 stack and sets the corresponding USR*n* error bits and USR*n*[RxRDY]. Then, if the break lasts until the next character time, the receiver places an all-zero character into the Rx FIFO and sets USR*n*[RB,RxRDY].

#### 14.5.2.4 UART1 in UART Mode

After a hardware reset, UART1 is in UART mode and differs from UART0 only with respect to receiver overrun, which is a function of Rx FIFO depth. UART0 has an effective depth of 4 bytes counting the Rx shift register, whereas UART1 has an effective depth of 33 bytes counting the Rx shift register. As a result, an overrun error won't occur in UART1 until 34 bytes are received without a CPU read from the Rx FIFO. In UART0, an overrun error would occur after 5 bytes are received without a CPU read from the Rx FIFO. In all other respects, UART1 in UART mode operates identically to UART0.

##### 14.5.2.4.1 Receiver in Modem Mode (UART1)

After a hardware reset, UART1 is in UART mode. Modem modes are chosen by setting MODCTL[MODE]. Other MODCTL fields should be initialized at the same time, as described in Section 14.3.4, "Modem Control Register (MODCTL)." Set the Rx FIFO threshold as described in Section 14.3.3, "Rx FIFO Threshold Register (RXLVL)."

The serial bit clock is always an input to UART1 in modem mode (on $\overline{\text{CTS}}$). When interfacing to an 8- or 16-bit CODEC, the frame sync is also an input to UART1 (on TIN1). However when an AC '97 controller is used, UART1 provides the frame sync (on $\overline{\text{RTS}}$).

Figure 14-31 on page 14-27 and Figure 14-32 on page 14-28 show timing diagrams for the UART1-CODEC interfaces. Figure 14-33 shows an example timing diagram for the UART1-AC '97 interface.

When an 8- or 16-bit CODEC is specified (MODCTL[MODE] = 01 or 10), UART1 starts to receive a sample either at the rising edge of frame sync or 1 bit clock cycle after the rising edge of frame sync, according to the value of MODCTL[DTS1]. The width of the frame sync pulse makes no difference. MODCTL[SHDIR] controls whether the sample is shifted in msb or lsb first. After the 8- or 16-bit sample is received, the receiver shift register shuts off until the next frame sync occurs.

When an AC '97 controller is specified (MODCTL[MODE] = 11), UART1 starts to receive time slot 1 data one bit-clock cycle after the rising edge of frame sync, regardless of the MODCTL[DTS1] value. However, MODCTL[SHDIR] must be 0 because the shift order must be msb first. Until the receiver detects the CODEC ready condition (a 1 in the first bit of a new frame), no data is put into the Rx FIFO for that frame. When a CODEC ready condition is detected, the receiver starts loading the Rx FIFO with the received time slot samples and continues to do so until a 0 is received in the first bit of a new frame.

### 14.5.2.5 FIFO Stack in UART0

The FIFO stack is used in the UART's receiver buffer logic. The stack consists of three receiver holding registers. The receive buffer consists of the FIFO and a receiver shift register connected to the RxD (see Figure 14-29). 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; OE (overrun error) is not appended. By programming the ERR bit in the channel's mode register (UMR1*n*), status is provided in character or block modes.

USR*n*[RxRDY] is set when at least one character is available to be read by the CPU. A read of the receiver buffer produces an output of data from the top of the FIFO stack. After the read cycle, the data at the top of the FIFO stack and its associated status bits are popped and the receiver shift register can add new data at the bottom of the stack. The FIFO-full status bit (FFULL) is set if all three stack positions are filled with data. Either the RxRDY or FFULL bit can be selected to cause an interrupt.

The two error modes are selected by UMR1*n*[ERR] as follows:

- In character mode (UMR1*n*[ERR] = 0, status is given in the USR*n* for the character at the top of the FIFO.
- In block mode, the USR*n* shows a logical OR of all characters reaching the top of the FIFO stack since the last RESET ERROR STATUS command. Status is updated as characters reach the top of the FIFO stack. Block mode offers a data-reception speed advantage where the software overhead of error-checking each character cannot be tolerated. However, errors are not detected until the check is performed at the end of an entire message—the faulting character is not identified.

In either mode, reading the USR*n* does not affect the FIFO. The FIFO is popped only when the receive buffer is read. The USR*n* should be read before reading the receive buffer. If all three receiver holding registers are full, a new character is held in the receiver shift register until space is available. However, if a second new character is received, the contents of the the character in the receiver shift register is lost, the FIFOs are unaffected, and USR*n*[OE] is set when the receiver detects the start bit of the new overrunning character.

To support flow control, the receiver can be programmed to automatically negate and assert $\overline{\text{RTS}}$, in which case the receiver automatically negates $\overline{\text{RTS}}$ when a valid start bit is detected and the FIFO stack is full. The receiver asserts $\overline{\text{RTS}}$ when a FIFO position becomes available; therefore, overrun errors can be prevented by connecting $\overline{\text{RTS}}$ to the $\overline{\text{CTS}}$ input of the transmitting device.

**NOTE:**

The receiver can still read characters in the FIFO stack if the receiver is disabled. If the receiver is reset, the FIFO stack, $\overline{\text{RTS}}$ control, all receiver status bits, and interrupt requests are reset. No more characters are received until the receiver is reenabled.

### 14.5.2.6 FIFOs in UART1

For UART1, FIFOs can be accessed as longwords. Other properties are as follows:

- 8-bit CODEC mode (MODCTL[MODE] = 01):
  - Can access FIFOs either one, two, or four 1-byte samples at a time.
  - For one-sample accesses, the sample occupies internal data bus bits 31–24.
- For two-sample accesses, the samples occupy internal data bus bits 31–16.16-bit CODEC mode (MODCTL[MODE] = 10):
  - Can access FIFOs one or two 2-byte samples at a time.
  - For one-sample accesses, the sample occupies internal data bus bits 31–16.
- AC ‘97 mode (MODCTL[MODE] = 11):
  - Must access FIFOs one sample at a time
  - Because time slot 1 has 16 bits, compared to 20 for all other time slots in a frame, time slot 1 data occupies internal data bus bits 31–16.
  - All 20-bit time slots occupy internal data bus bits 31–12 for Tx and Rx FIFOs.
  - In addition, when the Rx FIFO is being read, a 1 in internal data bus bit 11 marks this sample as the first time slot of a new frame.

The Tx FIFO functions as follows:

- AC ‘97 mode—Tx FIFO is effectively a 16 x 20 dual-port RAM to hold sixteen 20-bit AC ‘97 time slots. One sample/time slot is written to Tx FIFO per internal bus cycle.
- For all other modes the Tx FIFO is effectively 8 x 32.
  - 8-bit CODEC or as a UART—Tx FIFO can hold thirty-two 8-bit samples. One, two, or four bytes/samples can be written to Tx FIFO per internal bus cycle.
  - 16-bit CODEC—Tx FIFO can hold sixteen 16-bit samples. Either one or two 16-bit samples can be written to Tx FIFO per internal bus cycle.

The Rx FIFO functions as follows:

- AC ‘97 mode—Rx FIFO is effectively a 16 x 21 dual-port RAM to hold sixteen 20-bit AC ‘97 time slots. The extra flag bit is set to indicate the first time slot of a new AC ‘97 frame. One sample/time slot is read from Rx FIFO per internal bus cycle.

- For all other modes, the Rx FIFO is effectively 8 x 32.
  - 8-bit CODEC or as a UART—Rx FIFO holds thirty-two 8-bit samples. One, two, or four bytes/samples can be read from the Rx FIFO per internal bus cycle.
  - 16-bit CODEC—Rx FIFO holds sixteen 16-bit samples. Either one or two 16-bit samples can be read from the Rx FIFO per internal bus cycle.

## 14.5.3 Looping Modes

The UART can be configured to operate in various looping modes as shown in Figure 14-34 on page 14-30. These modes are useful for local and remote system diagnostic functions and can be used by UART1 in modem mode as well as UART mode. The modes are described in the following paragraphs and in Section 14.3, “Register Descriptions.”

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.

### 14.5.3.1 Automatic Echo Mode

In automatic echo mode, shown in Figure 14-35, 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 resent on TxD. The receiver must be enabled, but the transmitter need not be.

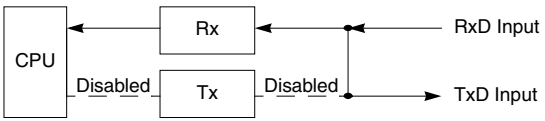



**Figure 14-35. Automatic Echo**

Because the transmitter is inactive, USR*n*[TxEMP,TxRDY] are inactive and data is sent as it is received. Received parity is checked but is 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.

### 14.5.3.2 Local Loop-Back Mode

Figure 14-36 shows how TxD and RxD are internally connected in local loop-back mode. This mode is for testing the operation of a local UART module channel by sending data to the transmitter and checking data assembled by the receiver to ensure proper operations.

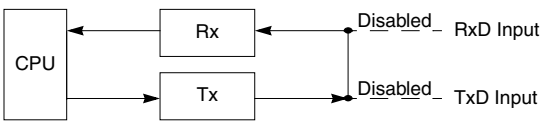



**Figure 14-36. Local Loop-Back**

Features of this local loop-back mode are as follows:

- Transmitter and CPU-to-receiver communications continue normally in this mode.
- RxD input data is ignored
- TxD is held marking
- The receiver is clocked by the transmitter clock. The transmitter must be enabled, but the receiver need not be.

#### 14.5.3.3 Remote Loop-Back Mode

In remote loop-back mode, shown in Figure 14-37, the channel automatically transmits received data bit by bit on the TxD output. The local CPU-to-transmitter link is disabled. This mode is useful in testing receiver and transmitter operation of a remote channel. For this mode, the transmitter uses the receiver clock.

Because the receiver is not active, received data cannot be read by the CPU and error 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 14-37. Remote Loop-Back**

### 14.5.4 Multidrop Mode

Setting UMR1*n*[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 channel receivers disabled, they continuously monitor the master's data stream. When the master sends an address character, the slave receiver channel notifies its respective CPU by setting USR*n*[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. Slave stations not addressed 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 14-38.

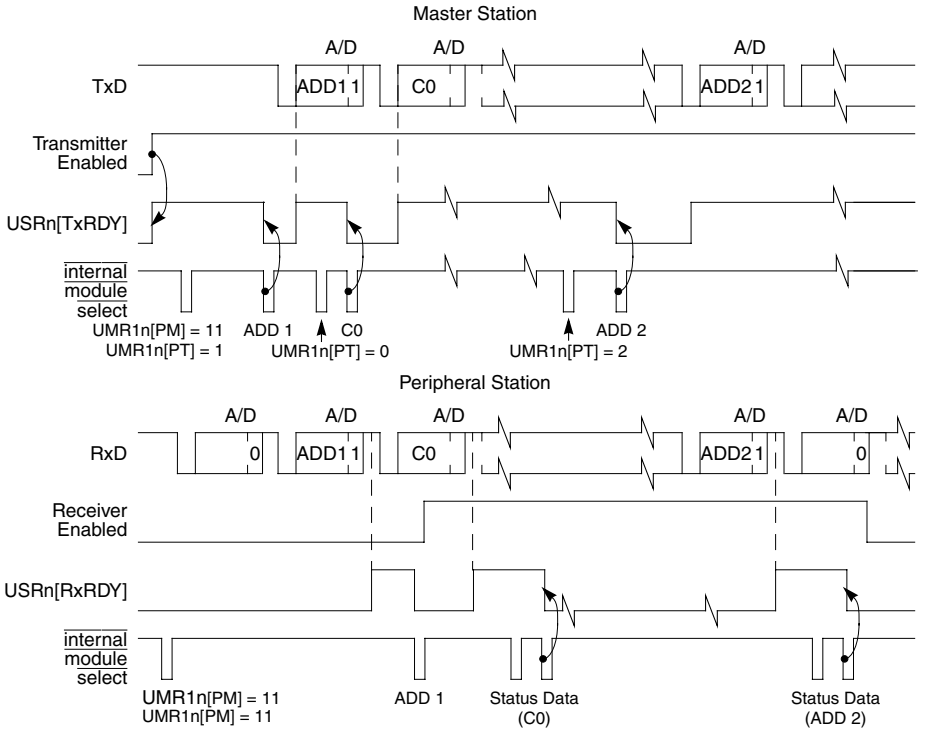



**Figure 14-38. Multidrop Mode Timing Diagram**

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 UMR1*n*[PT]. UMR1*n* 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 stack 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 stack during read operations.

In either case, the data bits are loaded into the data portion of the stack while the A/D bit is loaded into the status portion of the stack normally used for a parity error (USR*n*[PE]).

Framing error, overrun error, and break detection operate normally. The A/D bit takes the place of the parity bit; therefore, parity is neither calculated nor checked. Messages in this mode may still contain error detection and correction information. One way to provide error

detection, if 8-bit characters are not required, is to use software to calculate parity and append it to the 5-, 6-, or 7-bit character.

## 14.5.5 Bus Operation

This section describes bus operation during read, write, and interrupt acknowledge cycles to the UART module.

### 14.5.5.1 Read Cycles

The UART module responds to reads with byte data. Reserved registers return zeros.

### 14.5.5.2 Write Cycles

The UART module accepts write data as bytes. Write cycles to read-only or reserved registers complete normally without exception processing, but data is ignored.

**NOTE:**

The UART module is accessed by the CPU with zero wait states, as CLKIN is used for the UART module.

### 14.5.5.3 Interrupt Acknowledge Cycles

The UART module supplies the interrupt vector in response to a UART IACK cycle. If UIVR*n* is not initialized to provide a vector number, a spurious exception is taken if an interrupt is generated. This works in conjunction with the interrupt controller, which allows a programmable priority level.

## 14.5.6 Programming

The software flowchart, Figure 14-39, consists of the following:

- UART module initialization—These routines consist of SINIT and CHCHK (sheets 1 and 2). Before SINIT is called at system initialization, the calling routine allocates 2 words on the system stack. On return to the calling routine, SINIT passes UART status data on the stack. 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 (sheets 4 and 5) 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—Consists of SIRQ (sheet 4), 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.

### 14.5.6.1 UART Module Initialization Sequence

**NOTE:**

UART module registers can be accessed by word or byte operations, but only data byte D[7:0] is valid.

Table 14-19 shows the UART module initialization sequence.

**Table 14-19. UART Module Initialization Sequence**

| Register | Setting |
|---|---|
| UCRn | Reset the receiver and transmitter—UART or modem modes.<br>Reset the mode pointer (MISC[2–0] = 0b001)—UART or modem modes. |
| UIVRn | Program the vector number for a UART module interrupt—UART or modem modes. |
| UIMRn | Enable the preferred interrupt sources—UART or modem modes. |
| UACRn | Initialize the input enable control (IEC bit)—UART mode only. |
| UCSRn | Select the receiver and transmitter clock. Use timer as source if required—UART mode only. |
| UMR1n | If preferred, program operation of receiver ready-to-send (RxRTS bit)—UART mode only.<br>Select receiver-ready or FIFO-full notification (RxRDY/FFULL bit)—UART or modem modes.<br>Select character or block error mode (ERR bit)—UART mode only.<br>Select parity mode and type (PM and PT bits)—UART mode only.<br>Select number of bits per character (B/Cx bits)—UART mode only. |
| UMR2n | Select the mode of operation (CMx bits)—UART or modem modes.<br>If preferred, program operation of transmitter ready-to-send (TxRTS)—UART mode only.<br>If preferred, program operation of clear-to-send (TxCTS bit)—UART mode only.<br>Select stop-bit length (SBx bits)—UART mode only. |
| RXLVL | UART1 only, UART or modem modes—Choose Rx FIFO full threshold level. |
| TXLVL | UART1 only, UART or modem modes—Choose Tx FIFO empty threshold level. |
| MODCTL | Modem mode only<br>Choose the desired modem mode.<br>Choose whether msb or lsb is to be transferred first.<br>Choose 0- or 1-bit delay between rising edge of frame sync and first bit of time slot 1.<br>Choose 1 or 2 DMA channels to service UART1.<br>Activate/deactivate AC ‘97 warm and cold resets. |
| UCRn | Enable the receiver and transmitter—UART or modem modes. |

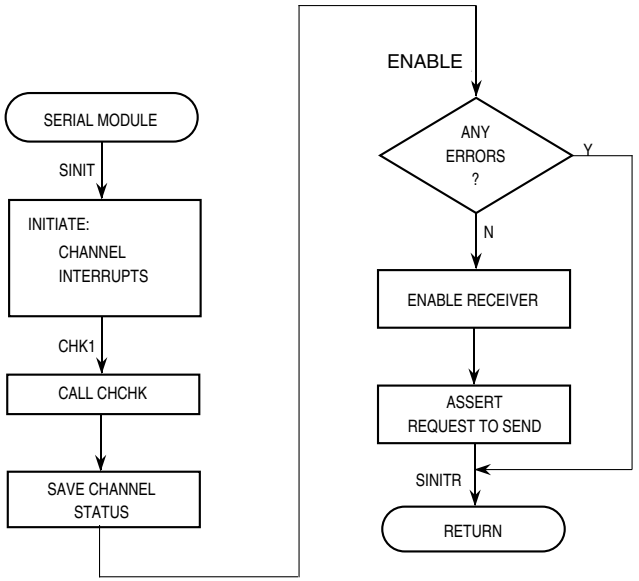



**Figure 14-39. UART Mode Programming Flowchart (Sheet 1 of 5)**

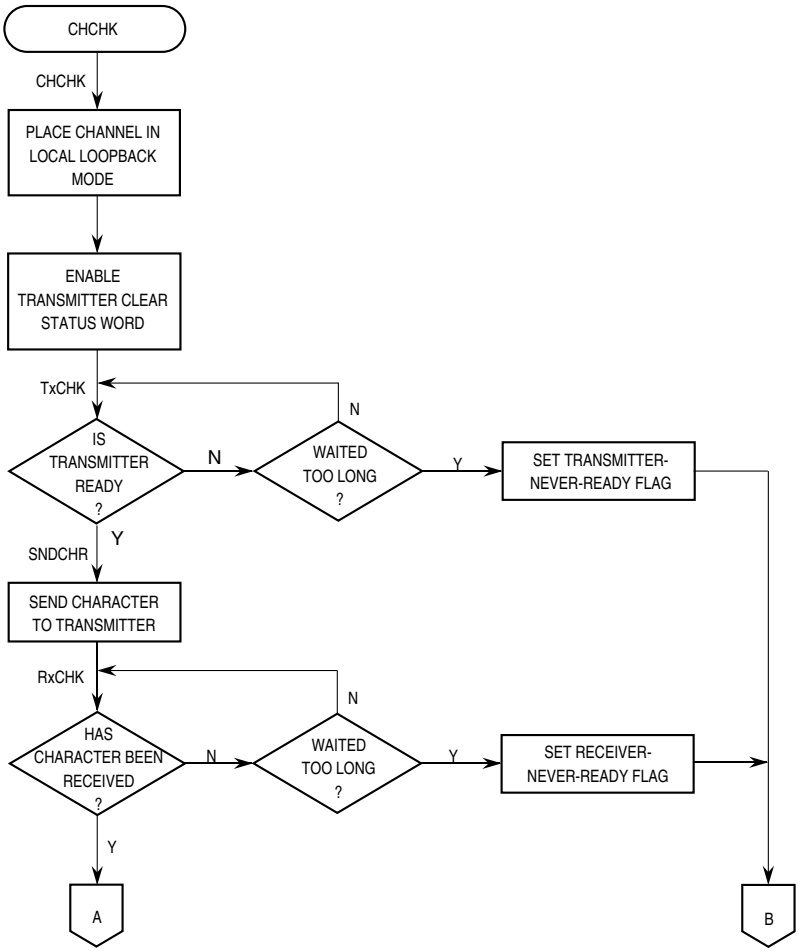



**Figure 14-39. UART Mode Programming Flowchart (Sheet 2 of 5)**

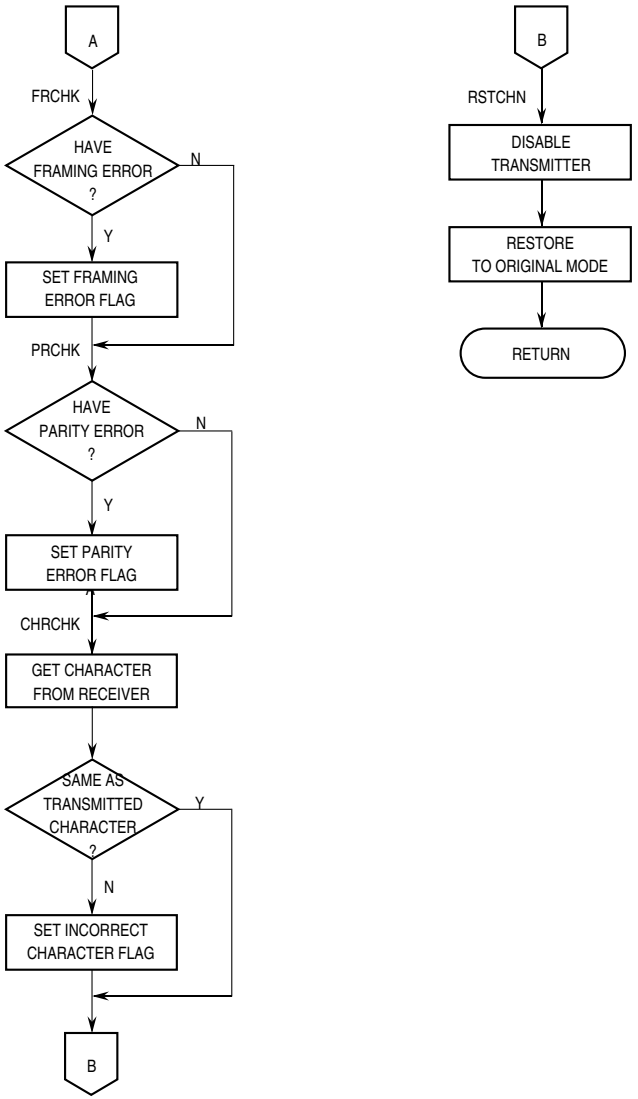



**Figure 14-39. UART Mode Programming Flowchart (Sheet 3 of 5)**

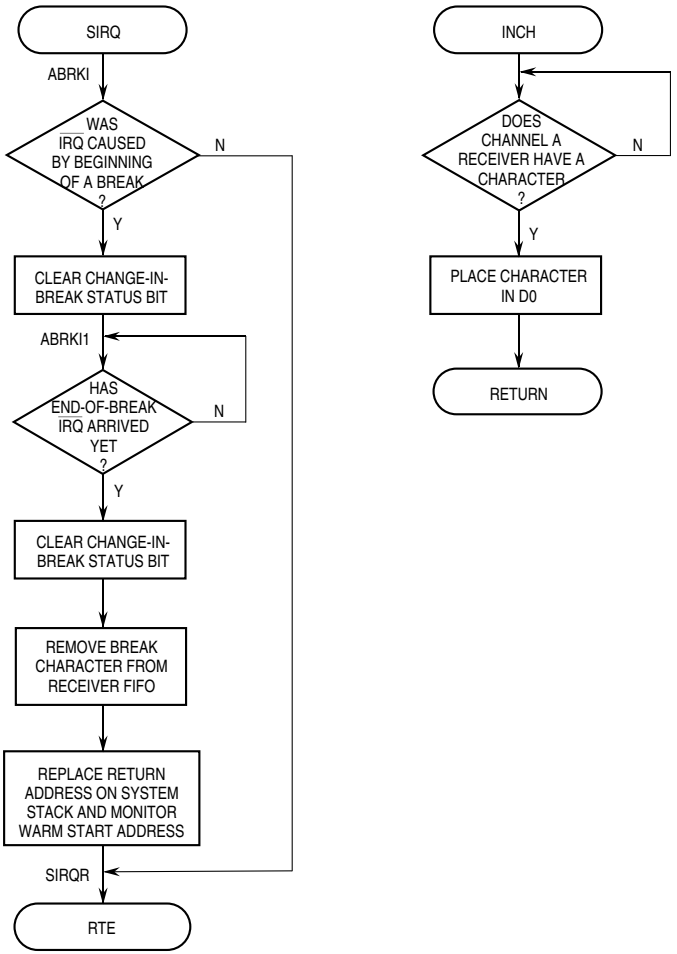



**Figure 14-39. UART Mode Programming Flowchart (Sheet 4 of 5)**

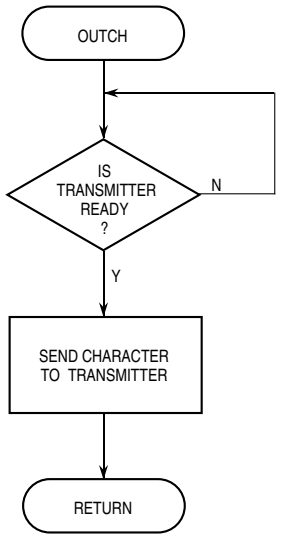



**Figure 14-39. UART Mode Programming Flowchart (Sheet 5 of 5)**

# Chapter 15
# Parallel Port (General-Purpose I/O)

This chapter describes the operation and programming model of the parallel port pin assignment, direction-control, and data registers. It includes a code example for setting up the parallel port.

## 15.1 Parallel Port Operation

The MCF5407 parallel port module has 16 signals, which are programmed as follows:

- The pin assignment register (PAR) selects the function of the 16 multiplexed pins.
- Port A data direction register (PADDR) determines whether pins configured as parallel port signals are inputs or outputs.
- The Port A data register (PADAT) shows the status of the parallel port signals.

The operations of the PAR, PADDR, and PADAT are described in the following sections.

### 15.1.1 Pin Assignment Register (PAR)

The pin assignment register (PAR), which is part of the system integration module (SIM), defines how each PAR bit determines each pin function, as shown in Figure 15-1.

| | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Field | PAR15 | PAR14 | PAR13 | PAR12 | PAR11 | PAR10 | PAR9 | PAR8 | PAR7 | PAR6 | PAR5 | PAR4 | PAR3 | PAR2 | PAR1 | PAR0 |
| PAR[n] = 0 | PP15 | PP14 | PP13 | PP12 | PP11 | PP10 | PP9 | PP8 | PP7 | PP6 | PP5 | PP4 | PP3 | PP2 | PP1 | PP0 |
| PAR[n] = 1 | A31 | A30 | A29 | A28 | A27 | A26 | A25 | A24 | $\overline{\text{TIP}}$ | $\overline{\text{DREQ0}}$ | $\overline{\text{DREQ1}}$ | TM2 | TM1/ DACK1 | TM0/ DACK0 | TT1 | TT0 |
| Reset | Determined by driving D4/ADDR_CONFIG with a 1 or 0 when $\overline{\text{RSTI}}$ negates. The system is configured as PP[15:0] if D4 is low; otherwise alternate pin functions selected by PAR[n] = 1 are used. | | | | | | | | | | | | | | | |
| R/W | R/W | | | | | | | | | | | | | | | |
| Address | Address MBAR + 0x004 | | | | | | | | | | | | | | | |

**Figure 15-1. Parallel Port Pin Assignment Register (PAR)**

If PP[9:8]/A[25:24] are unavailable because A[25:0] are needed for external addressing, PP[15:10]/A[31:26] can be configured as general-purpose I/O. Table 15-1 summarizes MCF5407 parallel port pins, described in detail in Chapter 17, “Signal Descriptions.”

**Table 15-1. Parallel Port Pin Descriptions**

| Pin | Description |
|---|---|
| PP[15:8]/ A[31:24] | MSB of the address bus/parallel port. Programmed through PAR[15–8]. If a PAR bit is 0, the associated pin functions as a parallel port signal. If a bit is 1, the pin functions as an address bus signal. If all pins are address signals, as much as 4 Gbytes of memory space are available. |
| $\overline{\text{TIP}}$/PP7 | Transfer-in-progress output/parallel port bit 7. Programmed through PAR[7]. Assertion indicates a bus transfer is in progress; negation indicates an idle bus cycle if the bus is still granted to the processor. Note that $\overline{\text{TIP}}$ is held asserted on back-to-back bus cycles. |
| $\overline{\text{DREQ}}$[1:0]/ PP[6:5] | DMA request inputs/two bits of the parallel port. Programmed through PAR[6–5]. These inputs are asserted by a peripheral device to request a DMA transfer. |
| TM[2:0]/ PP[4:2]/ DACK[1:0] | Transfer type outputs/parallel port bits 4–2. Programmed through PAR[4–2]. For DMA transfers, these signals provide acknowledge information or can be programmed to function as DMA acknowledge signals. For emulation transfers, TM[2:0] indicate user or data transfer types. For CPU space transfers, TM[2:0] are low. For interrupt acknowledge transfers, TM[2:0] carry the interrupt level being acknowledged. |
| TT[1:0]/ PP[1:0] | Transfer type outputs/parallel port bits 1–0. Programmed through PAR[1–0]. When the MCF5407 is bus master, it outputs these signals. They indicate the current bus access type. |

## 15.1.2 Port A Data Direction Register (PADDR)

The PADDR determines the signal direction of each parallel port pin programmed as a general-purpose I/O port in the PAR.

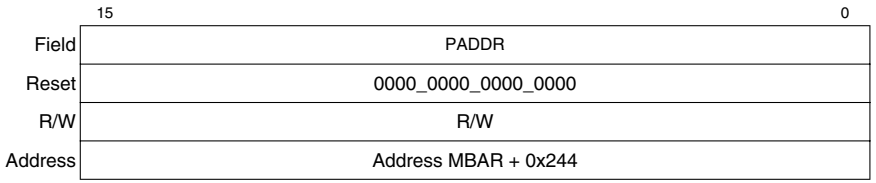



**Figure 15-2. Port A Data Direction Register (PADDR)**

Table 15-2 describes PADDR fields.

**Table 15-2. PADDR Field Description**

| Bits | Name | Description |
|---|---|---|
| 15–0 | PADDR | Data direction bits. Each data direction bit selects the direction of the signal as follows:<br>0 Signal is defined as an input.<br>1 Signal is defined as an output. |

## 15.1.3 Port A Data Register (PADAT)

The PADAT value for inputs corresponds to the logic level at the pin; for outputs, the value corresponds to the logic level driven onto the pin. Note the following:

- PADAT has no effect on pins not configured for general-purpose I/O.

- PADAT settings do not affect inputs. PADAT bit values determine the corresponding logic levels of pins configured as outputs.
- PADAT can be written to anytime. A read from PADAT returns values of corresponding pins configured as general-purpose I/O in the PAR and designated as inputs by the PADDR.

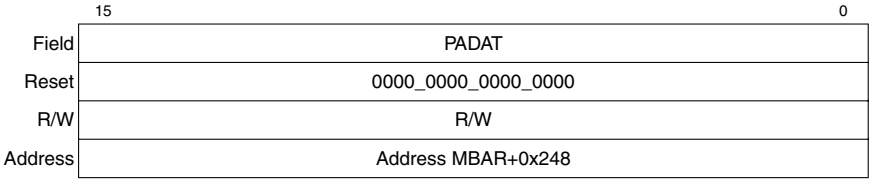



**Figure 15-3. Port A Data Register (PADAT)**

Table 15-3 shows relationships between PADAT bits and parallel port pins when PADAT is accessed. The effect differs when the parallel port pin is an input or output.

The following results occur when a parallel port pin is configured as an input:

- When the PADAT is read, the value returned is the logic value on the pin.
- When the PADAT is written, the register contents are updated without affecting the logic value on the pin.

The following results occur when a parallel port pin is configured as an output:

- When the PADAT is read, the register contents are returned and the pin is the logic value of the register.
- When the PADAT is written, the register contents are updated and the pin is the logic value of the register.

These relationships are also described in Table 15-3.

**Table 15-3. Relationship between PADAT Register and Parallel Port Pin (PP)**

| PP Status | PADAT R/W | Effect on PADAT | Effect on PP |
|---|---|---|---|
| Input | Read | Register bit value is the pin's logic value | No effect. Source of logic value |
| | Write | Register contents updated | No effect on the logic value at the pin |
| Output | Read | Register contents are returned | Pin is the logic value of the register bit |
| | Write | Register contents updated | Pin is the logic value of the register bit |

**NOTE:**

Although external devices cannot access the MCF5407's on-chip memories or MBAR, they can access any parallel port module registers in the SIM.

## 15.1.4 Code Example

The following code example shows how to set up the parallel port. Here, PP[7:0] are general-purpose I/O, PP[3:0] are inputs, and PP[7:4] are outputs.

```
MBARx    EQU   0x00010000
PAR      EQU   MBARx+0x004
PADDR    EQU   MBARx+0x244
PADAT    EQU   MBARx+0x248

move.l   #MBARx,D0       ;because MBAR is an internal register, MBARx is used as
movec    D0, MBAR        ;label for the memory map address
move.w   #0x00FF,D0
move.w   D0,PAR          ;set up the PAR. PP[7:0] set up as I/O
move.w   #0x00F0,D0
move.w   D0,PADDR        ;set PP[7:4] as outputs; PP[3:0] as inputs
move.b   #0xA0,D0
move.b   D0,PADAT        ;0xA0 written into PADAT; PP[7:4] being outputs,
                         ;PP[7:4] becomes 1010; i.e. PP7, PP5 = 1 and
                         ;PP6, PP4 = 0
```

# Part IV
# Hardware Interface

## Intended Audience

Part IV is intended for hardware designers who need to know the functions and electrical characteristics of the MCF5407 interface. It includes a pinout, and both electrical and functional descriptions of the MCF5407 signals. It also describes how these signals interact to support the variety of bus operations shown in timing diagrams.

## Contents

Part IV contains the following chapters:

- Chapter 16, “Mechanical Data,” provides a functional pin listing and package diagram for the MCF5407.
- Chapter 17, “Signal Descriptions,” provides an alphabetical listing of MCF5407 signals. This chapter describes the MCF5407 signals. In particular, it shows which are inputs or outputs, how they are multiplexed, which signals require pull-up resistors, and the state of each signal at reset.
- Chapter 18, “Bus Operation,” describes data transfers, error conditions, bus arbitration, and reset operations. It describes transfers initiated by the MCF5407 and by an external bus master, and includes detailed timing diagrams showing the interaction of signals in supported bus operations. Note that Chapter 11, “Synchronous/Asynchronous DRAM Controller Module,” describes DRAM cycles.
- Chapter 19, “IEEE 1149.1 Test Access Port (JTAG),” describes configuration and operation of the MCF5407 JTAG test implementation. It describes the use of JTAG instructions and how to disable JTAG functionality.
- Chapter 20, “Electrical Specifications,” describes AC and DC electrical specifications and thermal characteristics for the MCF5407. Because additional speeds may have become available since the publication of this book, consult Motorola’s ColdFire web page, http://www.motorola.com/coldfire, to confirm that this is the latest information.

## Suggested Reading

The following literature may be helpful with respect to the topics in Part IV:

- *IEEE Standard Test Access Port and Boundary-Scan Architecture*
- *IEEE Supplement to Standard Test Access Port and Boundary-Scan Architecture (1149.1)*

## Acronyms and Abbreviations

Table IV-i describes acronyms and abbreviations used in Part IV.

**Table IV-i. Acronyms and Abbreviated Terms**

| Term | Meaning |
|---|---|
| BDM | Background debug mode |
| BIST | Built-in self test |
| BSDL | Boundary-scan description language |
| DMA | Direct memory access |
| DSP | Digital signal processing |
| EDO | Extended data output (DRAM) |
| GPIO | General-purpose I/O |
| $I^2C$ | Inter-integrated circuit |
| IEEE | Institute for Electrical and Electronics Engineers |
| IPL | Interrupt priority level |
| JEDEC | Joint Electron Device Engineering Council |
| JTAG | Joint Test Action Group |
| LSB | Least-significant byte |
| lsb | Least-significant bit |
| MAC | Multiple accumulate unit |
| MBAR | Memory base address register |
| MSB | Most-significant byte |
| msb | Most-significant bit |
| Mux | Multiplex |
| PCLK | Processor clock |
| PLL | Phase-locked loop |
| POR | Power-on reset |
| PQFP | Plastic quad flat pack |

## Table IV-i. Acronyms and Abbreviated Terms (Continued)

| Term | Meaning |
|---|---|
| RISC | Reduced instruction set computing |
| Rx | Receive |
| SIM | System integration module |
| TAP | Test access port |
| TTL | Transistor-to-transistor logic |
| Tx | Transmit |

# Chapter 16
# Mechanical Data

This chapter provides a function pin listing and package diagram for the MCF5407. See the website [http://www.motorola.com/coldfire] for any updated information.

## 16.1 Package

The MCF5407 is assembled in a 208-pin, thermally enhanced plastic QFP package.

## 16.2 Pinout

The MCF5407 pinout is detailed in the following tables, including the primary and secondary functions of multiplexed signals. Additional columns indicate the output drive capability of each pin, whether it is internally synchronized, and if the signal can change on a negative clock transition. Note that the pin names IVCC and EVCC indicate which pins require 1.8- and 3.3-V power inputs, respectively.

These tables show MCF5407 pin numbers, including signal multiplexing. Additional columns indicate the direction, description, and output drive capability of each pin.

**Table 16-1. Pins 1–52 (Left, Top-to-Bottom)**

| Pin | | Alternate Function | I/O | Description | Drive (mA) |
|---|---|---|---|---|---|
| No | Name | | | | |
| 1 | IVCC | — | — | 1.8-V power input | — |
| 2 | A0 | — | I/O | Address bus bit | 8 |
| 3 | A1 | — | I/O | Address bus bit | 8 |
| 4 | GND | — | — | Ground pin | — |
| 5 | A2 | — | I/O | Address bus bit | 8 |
| 6 | A3 | — | I/O | Address bus bit | 8 |
| 7 | EVCC | — | — | 3.3-V power input | — |
| 8 | A4 | — | I/O | Address bus bit | 8 |
| 9 | A5 | — | I/O | Address bus bit | 8 |
| 10 | GND | — | — | Ground pin | — |
| 11 | A6 | — | I/O | Address bus bit | 8 |

## Table 16-1. Pins 1–52 (Left, Top-to-Bottom) (Continued)

| Pin | | Alternate Function | I/O | Description | Drive (mA) |
|---|---|---|---|---|---|
| No | Name | | | | |
| 12 | A7 | — | I/O | Address bus bit | 8 |
| 13 | EVCC | — | — | 3.3-V power input | — |
| 14 | A8 | — | I/O | Address bus bit | 8 |
| 15 | A9 | — | I/O | Address bus bit | 8 |
| 16 | A10 | — | I/O | Address bus bit | 8 |
| 17 | GND | — | — | Ground pin | — |
| 18 | A11 | — | I/O | Address bus bit | 8 |
| 19 | A12 | — | I/O | Address bus bit | 8 |
| 20 | A13 | — | I/O | Address bus bit | 8 |
| 21 | EVCC | — | — | 3.3-V power input | — |
| 22 | A14 | — | I/O | Address bus bit | 8 |
| 23 | A15 | — | I/O | Address bus bit | 8 |
| 24 | A16 | — | I/O | Address bus bit | 8 |
| 25 | GND | — | — | Ground pin | — |
| 26 | A17 | — | I/O | Address bus bit | 8 |
| 27 | A18 | — | I/O | Address bus bit | 8 |
| 28 | A19 | — | I/O | Address bus bit | 8 |
| 29 | EVCC | — | — | 3.3-V power input | — |
| 30 | A20 | — | I/O | Address bus bit | 8 |
| 31 | A21 | — | I/O | Address bus bit | 8 |
| 32 | A22 | — | I/O | Address bus bit | 8 |
| 33 | GND | — | — | Ground pin | — |
| 34 | A23 | — | I/O | Address bus bit | 8 |
| 35 | PP8 | A24 | I/O | Parallel port bit/Address bus bit | 8 |
| 36 | PP9 | A25 | I/O | Parallel port bit/Address bus bit | 8 |
| 37 | EVCC | — | — | 3.3-V power input | — |
| 38 | PP10 | A26 | I/O | Parallel port bit/Address bus bit | 8 |
| 39 | PP11 | A27 | I/O | Parallel port bit/Address bus bit | 8 |
| 40 | PP12 | A28 | I/O | Parallel port bit/Address bus bit | 8 |
| 41 | GND | — | — | Ground pin | — |
| 42 | PP13 | A29 | I/O | Parallel port bit/Address bus bit | 8 |
| 43 | PP14 | A30 | I/O | Parallel port bit/Address bus bit | 8 |
| 44 | PP15 | A31 | I/O | Parallel port bit/Address bus bit | 8 |
| 45 | EVCC | — | — | 3.3-V power input | — |
| 46 | SIZ0 | — | I/O | Size attribute | 8 |

**Table 16-1. Pins 1–52 (Left, Top-to-Bottom) (Continued)**

| Pin | | Alternate Function | I/O | Description | Drive (mA) |
|---|---|---|---|---|---|
| No | Name | | | | |
| 47 | SIZ1 | — | I/O | Size attribute | 8 |
| 48 | GND | — | — | Ground pin | — |
| 49 | $\overline{OE}$ | — | O | Output enable for chip selects | 8 |
| 50 | $\overline{CS0}$ | — | O | Chip select | 8 |
| 51 | $\overline{CS1}$ | — | O | Chip select | 8 |
| 52 | EVCC | — | — | 3.3-V power input | — |

**Table 16-2. Pins 53–104 (Bottom, Left-to-Right)**

| Pin | | Alternate Function | I/O | Description | Drive (mA) |
|---|---|---|---|---|---|
| No | Name | | | | |
| 53 | GND | — | — | Ground pin | — |
| 54 | $\overline{CS2}$ | — | O | Chip select | 8 |
| 55 | $\overline{CS3}$ | — | O | Chip select | 8 |
| 56 | $\overline{CS4}$ | — | O | Chip select | 8 |
| 57 | IVCC | — | — | 1.8-V power input | — |
| 58 | $\overline{CS5}$ | — | O | Chip select | 8 |
| 59 | $\overline{CS6}$ | — | O | Chip select | 8 |
| 60 | $\overline{CS7}$ | — | O | Chip select | 8 |
| 61 | GND | — | — | Ground pin | — |
| 62 | $\overline{AS}$ | — | I/O | Address strobe | 8 |
| 63 | R/$\overline{W}$ | — | I/O | Read/$\overline{Write}$ | 8 |
| 64 | $\overline{TA}$ | — | I/O | Transfer acknowledge | 8 |
| 65 | EVCC | — | — | 3.3-V power input | — |
| 66 | $\overline{TS}$ | — | I/O | Transfer start | 8 |
| 67 | $\overline{RSTI}$ | — | I | Reset | — |
| 68 | $\overline{IRQ7}$ | — | I | Interrupt request | — |
| 69 | GND | — | — | Ground pin | — |
| 70 | $\overline{IRQ5}$ | $\overline{IRQ4}$ | I | Interrupt request | — |
| 71 | $\overline{IRQ3}$ | $\overline{IRQ6}$ | I | Interrupt request | — |
| 72 | $\overline{IRQ1}$ | $\overline{IRQ2}$ | I | Interrupt request | — |
| 73 | IVCC | — | — | 1.8-V power input | — |
| 74 | $\overline{BR}$ | — | O | Bus request | 8 |
| 75 | $\overline{BD}$ | — | O | Bus driven | 8 |
| 76 | $\overline{BG}$ | — | I | Bus grant | — |

## Table 16-2. Pins 53–104 (Bottom, Left-to-Right) (Continued)

| Pin | | Alternate Function | I/O | Description | Drive (mA) |
|---|---|---|---|---|---|
| No | Name | | | | |
| 77 | GND | — | — | Ground pin | — |
| 78 | TOUT1 | — | O | Timer output | 8 |
| 79 | TOUT0 | — | O | Timer output | 8 |
| 80 | TIN0 | — | I | Timer input | — |
| 81 | EVCC | — | — | 3.3-V power input | — |
| 82 | TIN1 | — | I | Timer input | — |
| 83 | $\overline{\text{RAS0}}$ | — | O | DRAM row address strobe | 16 |
| 84 | $\overline{\text{RAS1}}$ | — | O | DRAM row address strobe | 16 |
| 85 | GND | — | — | Ground pin | — |
| 86 | $\overline{\text{CAS0}}$ | — | O | DRAM column address strobe | 16 |
| 87 | $\overline{\text{CAS1}}$ | — | O | DRAM column address strobe | 16 |
| 88 | $\overline{\text{CAS2}}$ | — | O | DRAM column address strobe | 16 |
| 89 | EVCC | — | — | 3.3-V power input | — |
| 90 | $\overline{\text{CAS3}}$ | — | O | DRAM column address strobe | 16 |
| 91 | $\overline{\text{DRAMW}}$ | — | O | DRAM write | 16 |
| 92 | $\overline{\text{SRAS}}$ | — | O | SDRAM row address strobe | 16 |
| 93 | GND | — | — | Ground pin | — |
| 94 | $\overline{\text{SCAS}}$ | — | O | SDRAM column address strobe | 16 |
| 95 | SCKE | — | O | SDRAM clock enable | 16 |
| 96 | $\overline{\text{BE0}}$ | $\overline{\text{BWE0}}$ | O | Byte enable/byte write enable | 8 |
| 97 | EVCC | — | — | 3.3-V power input | — |
| 98 | $\overline{\text{BE1}}$ | $\overline{\text{BWE1}}$ | O | Byte enable/byte write enable | 8 |
| 99 | $\overline{\text{BE2}}$ | $\overline{\text{BWE2}}$ | O | Byte enable/byte write enable | 8 |
| 100 | $\overline{\text{BE3}}$ | $\overline{\text{BWE3}}$ | O | Byte enable/byte write enable | 8 |
| 101 | GND | — | — | Ground pin | — |
| 102 | SCL | — | I/OD [1] | Serial clock line | 8 |
| 103 | SDA | — | I/OD [1] | Serial data line | 8 |
| 104 | GND | — | — | Ground pin | — |

[1] OD: Open-drain output

## Table 16-3. Pins 105–156 (Right, Bottom-to-Top)

| Pin | | Alternate Function | I/O | Description | Drive (mA) |
|---|---|---|---|---|---|
| No | Name | | | | |
| 105 | IVCC | — | — | 1.8-V power input | — |
| 106 | D31 | — | I/O | Data bus | 8 |
| 107 | D30 | — | I/O | Data bus | 8 |
| 108 | D29 | — | I/O | Data bus | 8 |
| 109 | GND | — | — | Ground pin | — |
| 110 | D28 | — | I/O | Data bus | 8 |
| 111 | D27 | — | I/O | Data bus | 8 |
| 112 | D26 | — | I/O | Data bus | 8 |
| 113 | EVCC | — | — | 3.3-V power input | — |
| 114 | D25 | — | I/O | Data bus | 8 |
| 115 | D24 | — | I/O | Data bus | 8 |
| 116 | D23 | — | I/O | Data bus | 8 |
| 117 | GND | — | — | Ground pin | — |
| 118 | D22 | — | I/O | Data bus | 8 |
| 119 | D21 | — | I/O | Data bus | 8 |
| 120 | D20 | — | I/O | Data bus | 8 |
| 121 | EVCC | — | — | 3.3-V power input | — |
| 122 | D19 | — | I/O | Data bus | 8 |
| 123 | D18 | — | I/O | Data bus | 8 |
| 124 | D17 | — | I/O | Data bus | 8 |
| 125 | GND | — | — | Ground pin | — |
| 126 | D16 | — | I/O | Data bus | 8 |
| 127 | D15 | — | I/O | Data bus | 8 |
| 128 | D14 | — | I/O | Data bus | 8 |
| 129 | EVCC | — | — | 3.3-V power input | — |
| 130 | D13 | — | I/O | Data bus | 8 |
| 131 | D12 | — | I/O | Data bus | 8 |
| 132 | D11 | — | I/O | Data bus | 8 |
| 133 | GND | — | — | Ground pin | — |
| 134 | D10 | — | I/O | Data bus | 8 |
| 135 | D9 | — | I/O | Data bus | 8 |
| 136 | D8 | — | I/O | Data bus | 8 |
| 137 | EVCC | — | — | 3.3-V power input | — |
| 138 | D7 | CS_CONF2 | I/O | Data bus/Chip select configuration | 8 |

**Table 16-3. Pins 105–156 (Right, Bottom-to-Top) (Continued)**

| Pin | | Alternate Function | I/O | Description | Drive (mA) |
|---|---|---|---|---|---|
| No | Name | | | | |
| 139 | D6 | CS_CONF1 | I/O | Data bus/Chip select configuration | 8 |
| 140 | D5 | CS_CONF0 | I/O | Data bus/Chip select configuration | 8 |
| 141 | GND | — | — | Ground pin | — |
| 142 | D4 | ADDR_CONF | I/O | Data bus/Address configuration | 8 |
| 143 | D3 | BE_CONFIG0 | I/O | Data bus/Byte enable configuration | 8 |
| 144 | D2 | DIVIDE2 | I/O | Data bus/Divide control PCLK:CLKIN | 8 |
| 145 | EVCC | — | — | 3.3-V power input | — |
| 146 | D1 | DIVIDE1 | I/O | Data bus/Divide control PCLK:CLKIN | 8 |
| 147 | D0 | DIVIDE0 | I/O | Data bus/Divide control PCLK:CLKIN | 8 |
| 148 | GND | — | — | Ground pin | — |
| 149 | DSCLK | $\overline{\text{TRST}}$ | I | Debug serial clock/JTAG Reset | — |
| 150 | TCK | TCK | I | JTAG clock | — |
| 151 | DSO | TDO | O | Debug serial out/JTAG data out | 8 |
| 152 | IVCC | — | — | 1.8-V power input | — |
| 153 | DSI | TDI | I | Debug serial input/JTAG data in | — |
| 154 | $\overline{\text{BKPT}}$ | TMS | I | Debug breakpoint/JTAG mode select | — |
| 155 | $\overline{\text{HIZ}}$ | — | I | High impedance override | — |
| 156 | GND | — | — | Ground pin | — |

**Table 16-4. Pins 157–208 (Top, Right-to-Left)**

| Pin | | Alternate Function | I/O | Description | Drive (mA) |
|---|---|---|---|---|---|
| No | Name | | | | |
| 157 | IVCC | — | — | 1.8-V power input | — |
| 158 | $\overline{\text{CTS1}}$ | — | I | UART1 clear-to-send | — |
| 159 | $\overline{\text{RTS1}}$ | — | O | UART1 request-to-send | 8 |
| 160 | RXD1 | — | I | UART1 receive data | — |
| 161 | TXD1 | — | O | UART1 transmit data | 8 |
| 162 | GND | — | — | Ground pin | — |
| 163 | $\overline{\text{CTS0}}$ | — | I | UART0 clear-to-send | — |
| 164 | $\overline{\text{RTS0}}$ | — | O | UART0 request-to-send | 8 |
| 165 | RXD0 | — | I | UART0 receive data | — |
| 166 | TXD0 | — | O | UART0 transmit data | 8 |
| 167 | EVCC | — | — | 3.3-V power input | — |

**Table 16-4. Pins 157–208 (Top, Right-to-Left) (Continued)**

| Pin | | Alternate Function | I/O | Description | Drive (mA) |
|---|---|---|---|---|---|
| No | Name | | | | |
| 168 | EDGESEL | — | I | SDRAM bus clock edge select | — |
| 169 | GND | — | — | Ground pin | — |
| 170 | BCLKO | — | O | Bus clock output | 16 |
| 171 | IVCC | — | — | 1.8-V power input | — |
| 172 | $\overline{\text{RSTO}}$ | — | O | Processor reset output | 8 |
| 173 | GND | — | — | Ground pin | — |
| 174 | CLKIN | — | I | Clock input | — |
| 175 | IVCC | — | — | 1.8-V power input | — |
| 176 | MTMOD0 | — | I | JTAG/BDM select (Tie high or low) | — |
| 177 | MTMOD1 | — | I | Tie high or low | — |
| 178 | PGND | — | — | PLL ground pin | — |
| 179 | NC | — | O | | — |
| 180 | PVCC | — | — | 1.8-V filter supply for PLL | — |
| 181 | MTMOD2 | — | I | Tie high or low | — |
| 182 | MTMOD3 | — | I | Tie high or low | — |
| 183 | GND | — | — | Ground pin | — |
| 184 | PSTCLK | — | O | Processor status clock | 8 |
| 185 | IVCC | — | — | 1.8-V power input | — |
| 186 | PSTDDATA0 | — | O | Processor status/debug data | 8 |
| 187 | PSTDDATA1 | — | O | Processor status/debug data | 8 |
| 188 | GND | — | — | Ground pin | — |
| 189 | PSTDDATA2 | — | O | Processor status/debug data | 8 |
| 190 | PSTDDATA3 | — | O | Processor status/debug data | 8 |
| 191 | EVCC | — | — | 3.3-V power input | — |
| 192 | PSTDDATA4 | — | O | Processor status/debug data | 8 |
| 193 | PSTDDATA5 | — | O | Processor status/debug data | 8 |
| 194 | GND | — | — | Ground pin | — |
| 195 | PSTDDATA6 | — | O | Processor status/debug data | 8 |
| 196 | PSTDDATA7 | — | O | Processor status/debug data | 8 |
| 197 | IVCC | — | — | 1.8-V power input | — |
| 198 | PP7 | $\overline{\text{TIP}}$ | I/O | Parallel port bit/transfer in progress | 8 |
| 199 | PP6 | $\overline{\text{DREQ0}}$ | I/O | Parallel port bit/DMA request | 8 |
| 200 | PP5 | $\overline{\text{DREQ1}}$ | I/O | Parallel port bit/DMA request | 8 |
| 201 | GND | — | — | Ground pin | — |
| 202 | PP4 | TM2 | I/O | Parallel port bit/Transfer modifier | 8 |

**Table 16-4. Pins 157–208 (Top, Right-to-Left) (Continued)**

| Pin | | Alternate Function | I/O | Description | Drive (mA) |
|---|---|---|---|---|---|
| No | Name | | | | |
| 203 | PP3 | TM1/DACK1[1] | I/O | Parallel port bit/Transfer modifier/DMA acknowledge | 8 |
| 204 | PP2 | TM0/DACK0[1] | I/O | Parallel port bit/Transfer modifier/DMA acknowledge | 8 |
| 205 | EVCC | — | — | 3.3-V power input | — |
| 206 | PP1 | TT1 | I/O | Parallel port bit/Transfer type | 8 |
| 207 | PP0 | TT0 | I/O | Parallel port bit/Transfer type | 8 |
| 208 | GND | — | — | Ground pin | — |

1 When the internal DMA is used, PP3 and PP2 (PP[3:2]/TM[1:0]) can be programmed to a third function, (DACK[1:0]), which indicates DMA acknowledge.

## 16.3 Mechanical Diagram

Figure 16-1 is a mechanical diagram of the 208-pin QFP MCF5407.

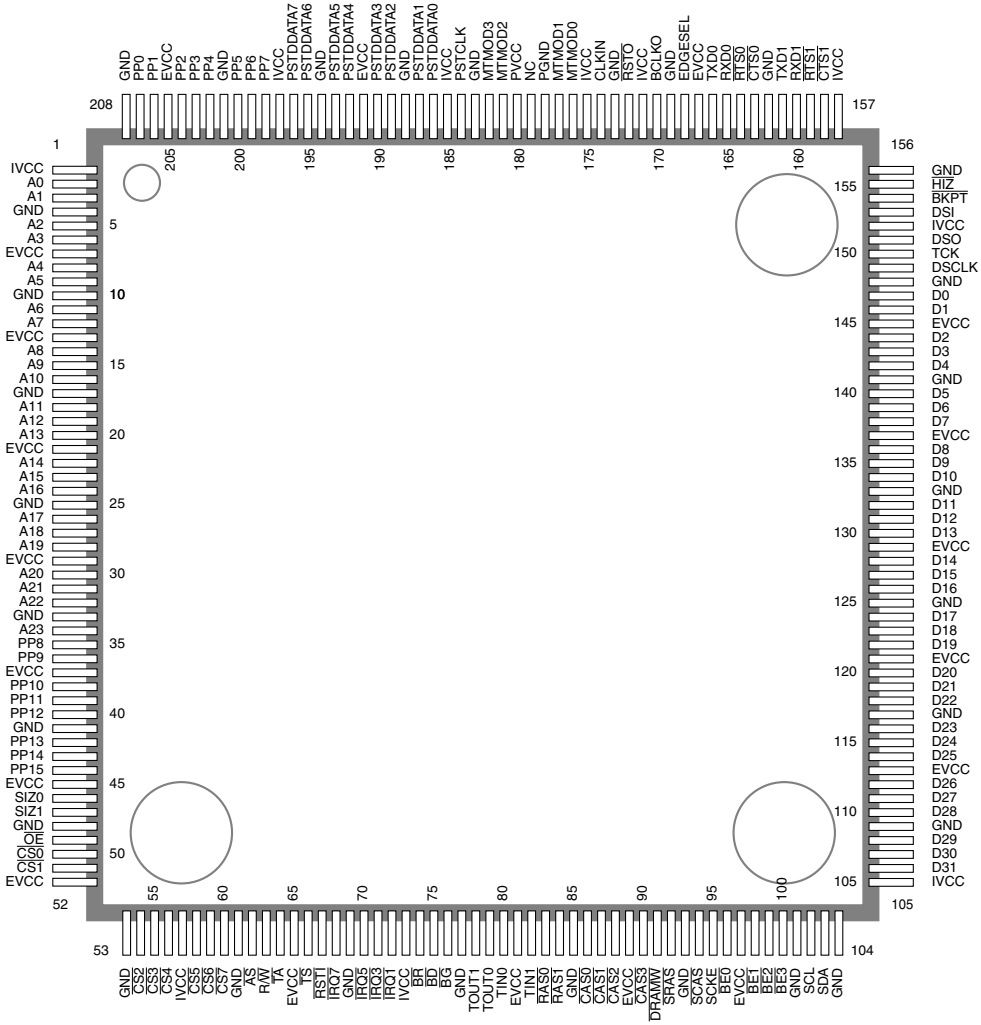



**Figure 16-1. Mechanical Diagram**

## 16.4 Case Drawing

Figure 16-2 and Figure 16-3 show the MCF5407 case drawings.

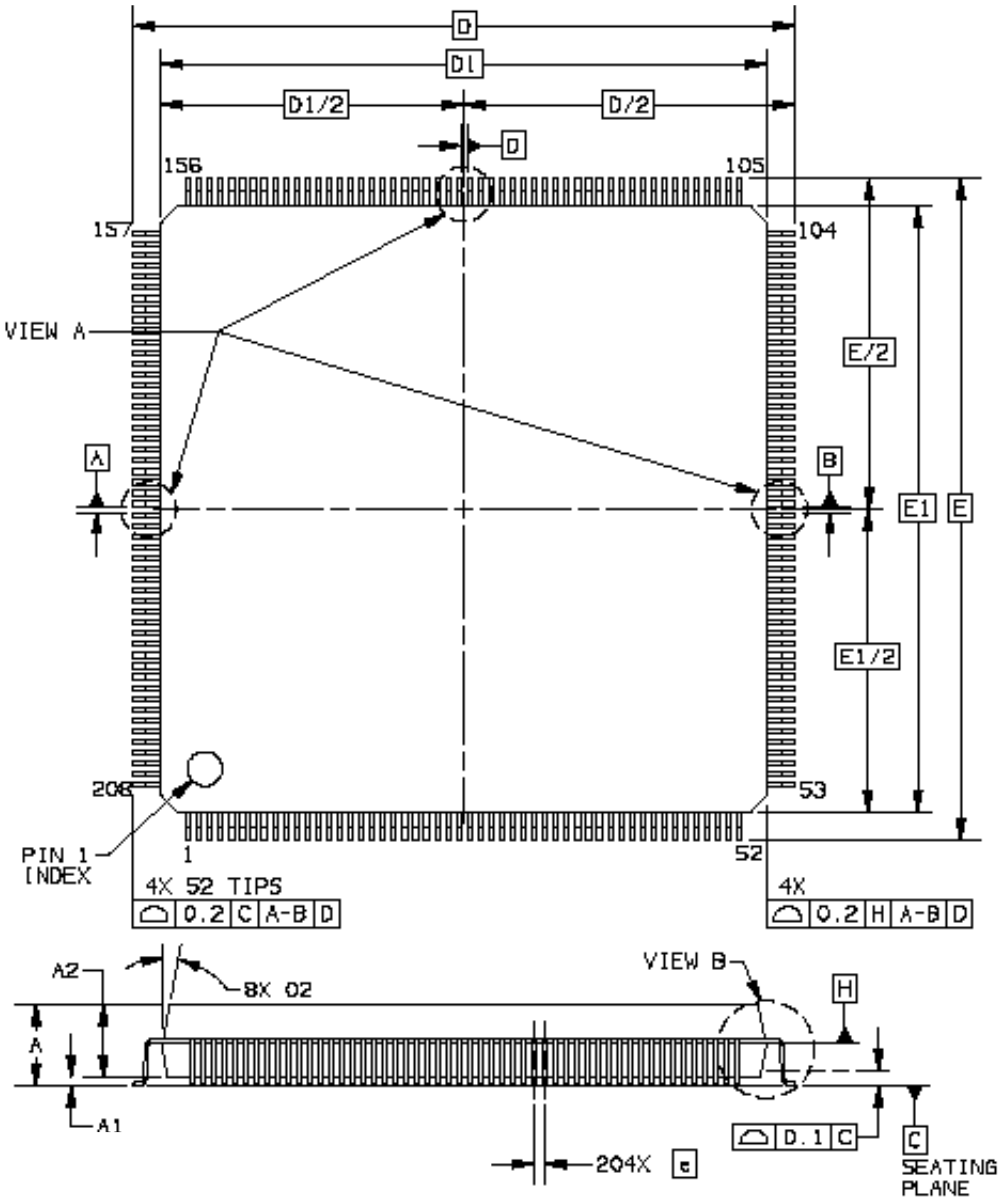
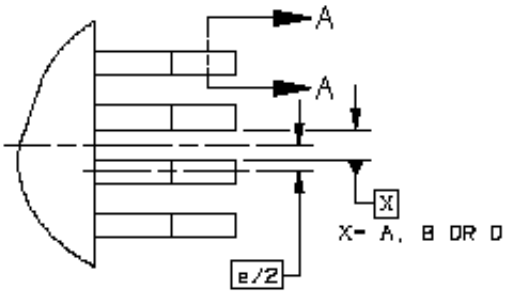
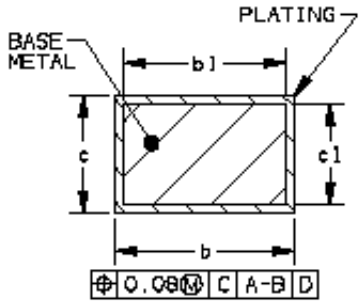


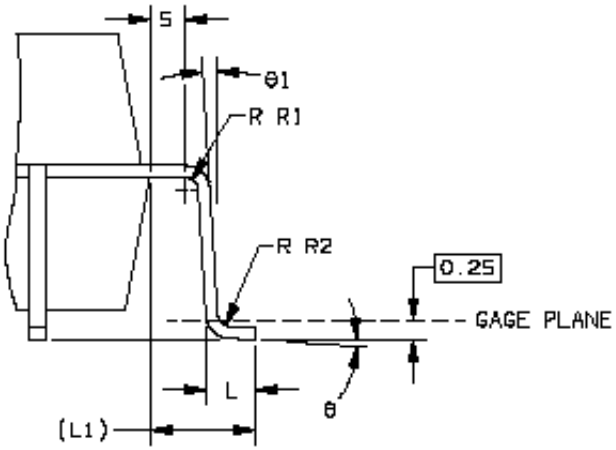

**Figure 16-2. MCF5407 Case Drawing (General View)**




View A: Three Places




Section A-A: 160 Places Rotated 90° CW




View B

**Figure 16-3. Case Drawing (Details)**

The dimensions in Figure 16-2 and Figure 16-3 are referenced in Table 16-5.

**Table 16-5. Dimensions**

| Reference | Dimension (Millimeters) | |
|---|---|---|
| | Minimum | Maximum |
| A | — | 4.10 |
| A1 | 0.25 | 0.50 |
| A2 | 3.20 | 3.60 |
| b | 0.17 | 0.27 |
| b1 | 0.17 | 0.23 |
| c | 0.09 | 0.20 |
| c1 | 0.09 | 0.16 |
| D | 30.60 BSC | |

**Table 16-5. Dimensions (Continued)**

| Reference | Dimension (Millimeters) | |
|---|---|---|
| | Minimum | Maximum |
| D1 | 28.00 BSC | |
| e | 0.50 BSC | |
| E | 30.60 BSC | |
| E1 | 28.00 BSC | |
| L | 0.45 | 0.75 |
| L1 | 1.30 REF | |
| R1 | 0.08 | — |
| R2 | 0.08 | 0.25 |
| S | 0.20 | — |
| ϑ | 0* | 8* |
| ϑ1 | 0* | — |
| ϑ2 | 5* | 16* |

# Chapter 17
# Signal Descriptions

This chapter describes MCF5407 signals. It includes an alphabetical listing of signals, showing multiplexing, whether it is an input or output to the MCF5407, the state at reset, and whether a pull-up resistor should be used. The following chapter, Chapter 18, “Bus Operation,” describes how these signals interact.

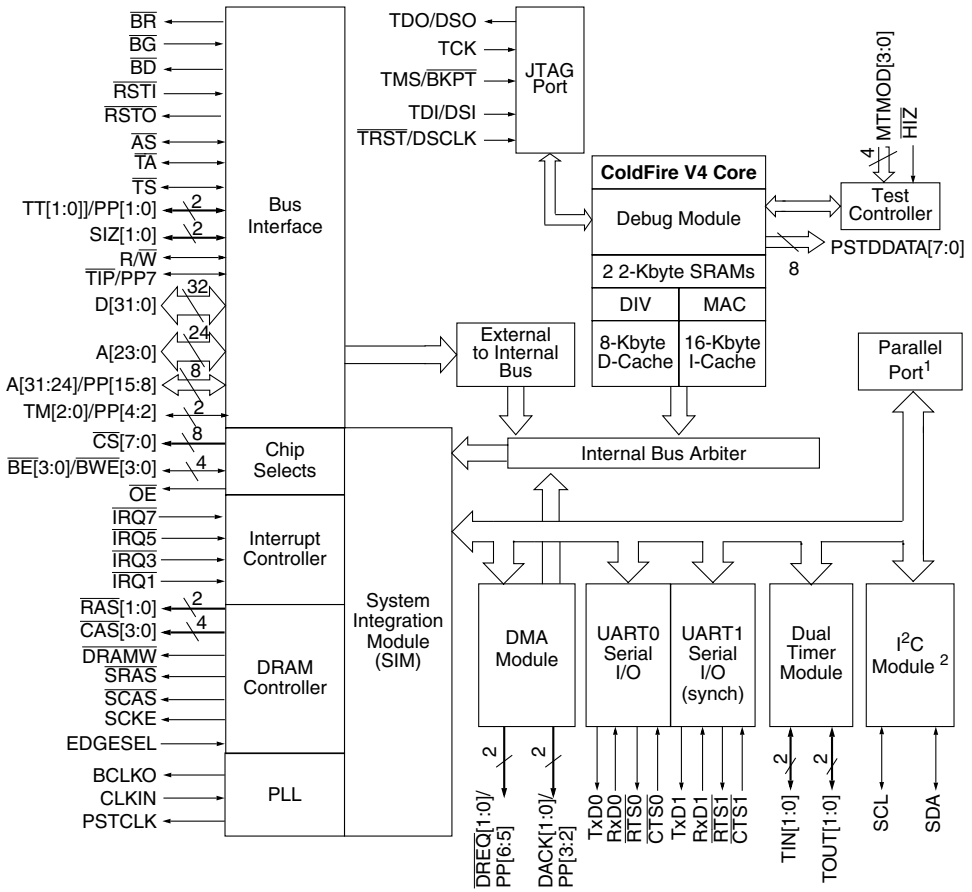
**NOTE:**

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

Active-low signals, such as $\overline{\text{SRAS}}$ and $\overline{\text{TA}}$, are indicated with an overbar.

## 17.1 Overview

Figure 17-1 shows the block diagram of the MCF5407 with the signal interface.




[1] Parallel port pins (PPn) are multiplexed with other bus functions as shown.
[2] I²C is a Philips proprietary interface.

**Figure 17-1. MCF5407 Block Diagram with Signal Interfaces**

Table 17-1 lists the MCF5407 signals grouped by functionality.

## Table 17-1. MCF5407 Signal Index

| Signal Name | Abbreviation | Function | I/O | Reset | Pull-Up | Page |
|---|---|---|---|---|---|---|
| **Section 17.2, "MCF5407 Bus Signals"** | | | | | | 17-7 |
| Address | A[31:0] | 32-bit address bus. A[4:2] indicate the interrupt level for external interrupts. | I/O | Three state | | 17-7 |
| Data | D[31:0] | Data bus. D[7:0] are loaded at reset for bus configuration. | I/O | Three state | | 17-8 |
| Read/Write | R/$\overline{W}$ | Identifies read and write transfers | I/O | Three state | Up | 17-8 |
| Size | SIZ[1:0] | Indicates the data transfer size | I/O | Three state | | 17-8 |
| Transfer start | $\overline{TS}$ | Indicates the start of a bus transfer | I/O | Three state | | 17-9 |
| Address strobe | $\overline{AS}$ | Indicates a bus cycle has been initiated and address is stable | I/O | Three state | Up | 17-9 |
| Transfer acknowledge | $\overline{TA}$ | Assertion terminates transfer synchronously | I/O | Three state | Up | 17-9 |
| Transfer in progress | $\overline{TIP}$/PP7 | Indicates a bus cycle is in progress; multiplexed with PP7 | O | Parallel port | | 17-10 |
| Transfer type | TT[1:0] | Indicates transfer type: normal, CPU space, emulator mode, or DMA; multiplexed with PP[1:0] | O | Parallel port | | 17-10 |
| Transfer modifier | TM[2:0] | Provides transfer modifier information; multiplexed with TM2/PP4 and TM[1:0]/PP[3:2]/DACK[1:0] | O | Parallel port | | 17-10 |
| **Section 17.3, "Interrupt Control Signals"** | | | | | | 17-12 |
| Interrupt request | $\overline{IRQ7}$, $\overline{IRQ5}$, $\overline{IRQ3}$, $\overline{IRQ1}$ | Four external interrupts are set to default levels 1,3,5,7; user-alterable. | I | — | Up | 17-12 |
| **Section 17.4, "Bus Arbitration Signals"** | | | | | | 17-12 |
| Bus request | $\overline{BR}$ | Indicates processor needs bus | O | High | | 17-12 |
| Bus grant | $\overline{BG}$ | Arbiter asserts to grant mastership. | I | — | Note [1] | 17-12 |
| Bus driven | $\overline{BD}$ | Indicates processor is driving bus | O | High | | 17-13 |
| **Section 17.5, "Clock and Reset Signals"** | | | | | | 17-13 |
| Reset in | $\overline{RSTI}$ | Processor reset input | I | — | Up | 17-13 |
| Clock input | CLKIN | Input used to clock internal logic | I | — | | 17-13 |
| Bus clock out | BCLKO | Bus clock reference output | O | — | | 17-13 |
| Reset out | $\overline{RSTO}$ | Processor reset output | O | Low | | 17-13 |
| Auto-acknowledge configuration [2] | AA_CONFIG | Controls auto acknowledge timing for $\overline{CS0}$ at reset | I | — | | 17-14 |
| Port size configuration [2] | PS_CONFIG[1:0] | Controls port size for $\overline{CS0}$ at reset | I | — | User cfg | 17-14 |

## Table 17-1. MCF5407 Signal Index (Continued)

| Signal Name | Abbreviation | Function | I/O | Reset | Pull-Up | Page |
|---|---|---|---|---|---|---|
| Address configuration [2] | ADDR_CONFIG | Programs parallel I/O ports | I | — | User cfg | 17-15 |
| BE[3:0] configuration | BE_CONFIG | Programs byte enable pins | I | — | User cfg | 17-15 |
| Divide control PCLK to CLKIN [2] | DIVIDE[2:0] | Selects CLKIN/PCLK ratio | I | — | User cfg | 17-15 |
| **Section 17.6, "Chip-Select Module Signals"** | | | | | | 17-15 |
| Chip selects[7:0] | $\overline{CS}$[7:0] | Enables peripherals at programmed addresses; $\overline{CS0}$ provides boot ROM selection. | O | High | | 17-15 |
| Byte enable[3:0]/ Byte write enable[3:0] | $\overline{BE}$[3:0]/ $\overline{BWE}$[3:0] | $\overline{BE}$[3:0] select bytes in memory. Programmed at reset for $\overline{CS0}$ | O | High | | 17-16 |
| Output enable | $\overline{OE}$ | Output enable for chip select read cycles | O | High | | 17-16 |
| **Section 17.7, "DRAM Controller Signals"** | | | | | | 17-16 |
| Row address strobe | $\overline{RAS}$[1:0] | DRAM row address strobe | O | High | | 17-16 |
| Column address strobe | $\overline{CAS}$[3:0] | DRAM column address strobe | O | High | | 17-16 |
| DRAM write | $\overline{DRAMW}$ | Asserted for DRAM write; negated for DRAM read | O | High | | 17-16 |
| Synchronous column address strobe | $\overline{SCAS}$ | SDRAM column address strobe | O | High | | 17-17 |
| Synchronous row address strobe | $\overline{SRAS}$ | SDRAM row address strobe | O | High | | 17-17 |
| Synchronous clock enable | SCKE | Clock enable for external SDRAM | O | Low | | 17-17 |
| Synchronous edge select | EDGESEL | Timing select for external SDRAM | I | — | User cfg | 17-17 |
| **Section 17.8, "DMA Controller Module Signals"** | | | | | | 17-17 |
| DMA request | $\overline{DREQ}$[1:0] | External DMA transfer request; multiplexed with PP[6:5] | I | — | | 17-17 |
| DMA acknowledge | DACK[1:0] | Indicates DMA transfer terminated; multiplexed with TM[1:0]/PP[3:2] | O | Parallel port | | 17-18 |
| **Section 17.9, "Serial Module Signals"** | | | | | | 17-18 |
| Receive data | RxD[1:0] | Receive serial data input for UART | I | — | | 17-19 |
| Transmit data | TxD[1:0] | Transmit serial data output for UART | O | High | | 17-18 |
| Request-to-send | $\overline{RTS}$[1:0] | UART asserts when ready to receive data query. | O | High | | 17-19 |
| Clear-to-send | $\overline{CTS}$[1:0] | Signals UART that data can be sent to peripheral | I | — | | 17-19 |
| **Section 17.10, "Timer Module Signals"** | | | | | | 17-19 |
| Timer input | TIN[1:0] | Clock input to timer or trigger to timer value capture logic | I | — | | 17-19 |

**Table 17-1. MCF5407 Signal Index (Continued)**

| Signal Name | Abbreviation | Function | I/O | Reset | Pull-Up | Page |
|---|---|---|---|---|---|---|
| Timer outputs | TOUT[1:0] | Outputs waveform or pulse. | O | High | | 17-19 |
| **Section 17.11, "Parallel I/O Port (PP[15:0])"** | | | | | | 17-19 |
| Parallel port | PP[15:0] | Interfaces with I/O; multiplexed with bus address and attribute signals. | I/O | Input | | 17-19 |
| **Section 17.12, "$I^2C$ Module Signals"** | | | | | | 17-20 |
| Serial clock line | SCL | Clock signal for $I^2C$ operation | I/O | Open drain | Up | 17-20 |
| Serial data line | SDA | Serial data port for $I^2C$ operation | I/O | Open drain | Up | 17-20 |
| **Section 17.13, "Debug and Test Signals"** | | | | | | 17-20 |
| Motorola test mode | MTMOD0 | Puts processor in functional or emulator mode | I | — | User cfg | 17-20 |
| Motorola test mode | MTMOD[3:1] | Reserved | I | — | Down | 17-20 |
| High impedance | $\overline{HIZ}$ | Assertion three-states all outputs | I | — | Up | 17-20 |
| Processor clock out | PSTCLK | Output clock used for PSTDDATA | O | — | | 17-21 |
| Processor status/debug data | PSTDDATA[7:0] | Displays captured processor data and breakpoint status | O | Driven | | 17-21 |
| **Section 17.14, "Debug Module/JTAG Signals"** | | | | | | 17-21 |
| Test clock | TCK | Clock signal for IEEE 1149.1 JTAG | I | — | Low | 17-22 |
| Test reset/ Development serial clock | $\overline{TRST}$/DSCLK | Asynchronous reset for JTAG; debug module clock input | I | — | Up | 17-21 |
| Test mode select/ Breakpoint | TMS/$\overline{BKPT}$ | TMS (JTAG)/hardware breakpoint (debug) | I | — | Up | 17-21 |
| Test data input/ Development serial input | TDI/DSI | Multiplexed serial input for the JTAG or background debug module | I | — | Up | 17-22 |
| Test data output/ Development serial output | TDO/DSO | Multiplexed serial output for the JTAG or background debug module | O | Driven | | 17-22 |

1 If there is no arbiter, $\overline{BG}$ should be tied low; otherwise, it should be negated.

2 These data pins are sampled at reset for configuration.

Table 17-2 lists signals in alphabetical order by abbreviated name.

**Table 17-2. MCF5407 Alphabetical Signal Index**

| Abbreviation | Signal Name | Function | I/O | Page |
|---|---|---|---|---|
| AA_CONFIG | Auto-acknowledge configuration | Clock/reset | I | 17-14 |
| ADDR_CONFIG | Address configuration | Clock/reset | I | 17-15 |
| $\overline{AS}$ | Address strobe | Bus | I/O | 17-9 |

## Table 17-2. MCF5407 Alphabetical Signal Index (Continued)

| Abbreviation | Signal Name | Function | I/O | Page |
|---|---|---|---|---|
| A[31:0] | Address | Bus | I/O | 17-7 |
| BCLKO | Bus clock out | Clock/reset | O | 17-13 |
| $\overline{BD}$ | Bus driven | Bus arbitration | O | 17-13 |
| $\overline{BE}$[3:0]/$\overline{BWE}$[3:0] | Byte enable[3:0]/Byte write enable[3:0] | Chip select | O | 17-16 |
| $\overline{BG}$ | Bus grant | Bus arbitration | I | 17-12 |
| $\overline{BR}$ | Bus request | Bus arbitration | O | 17-12 |
| $\overline{CAS}$[3:0] | Column address strobe | DRAM | O | 17-16 |
| CLKIN | Clock input | Clock/reset | I | 17-13 |
| $\overline{CS}$[7:0] | Chip selects[7:0] | UART | O | 17-15 |
| $\overline{CTS}$[1:0] | Clear-to-send | Serial module | I | 17-19 |
| DACK[1:0] | DMA acknowledge | DMA | O | 17-18 |
| DIVIDE[2:0] | Divide control PCLK to CLKIN | Clock/reset | I | 17-15 |
| $\overline{DRAMW}$ | DRAM write | DRAM | O | 17-16 |
| $\overline{DREQ}$[1:0] | DMA request | DMA | I | 17-17 |
| D[31:0] | Data | Bus | I/O | 17-8 |
| EDGESEL | Sync edge select | DRAM | I | 17-17 |
| $\overline{HIZ}$ | High impedance | Debug | I | 17-20 |
| $\overline{IRQ7}$, $\overline{IRQ5}$, $\overline{IRQ3}$, $\overline{IRQ1}$ | Interrupt request | Interrupt control | I | 17-12 |
| MTMOD[3:0] | Motorola test mode | Debug | I | 17-20 |
| $\overline{OE}$ | Output enable | Chip select | O | 17-16 |
| PP[15:0] | Parallel port | Parallel port | I/O | 17-19 |
| PSTCLK | Processor clock out | Debug | O | 17-20 |
| PSTDDATA[7:0] | Processor status/debug data | Debug | O | 17-20 |
| PS_CONFIG[1:0] | Port size configuration | Clock/reset | I | 17-14 |
| R/$\overline{W}$ | Read/Write | Bus | I/O | 17-8 |
| $\overline{RAS}$[1:0] | Row address strobe | DRAM | O | 17-16 |
| $\overline{RSTI}$ | Reset In | Clock/reset | I | 17-13 |
| $\overline{RSTO}$ | Reset Out | Clock/reset | O | 17-13 |
| $\overline{RTS}$[1:0] | Request-to-send | Serial module | O | 17-19 |
| RxD[1:0] | Receive data | Serial module | I | 17-19 |
| $\overline{SCAS}$ | Synchronous column address strobe | DRAM | O | 17-17 |
| SCKE | Synchronous clock enable | DRAM | O | 17-17 |
| SCL | Serial clock line | $I^2C$ | I/O | 17-20 |
| SDA | Serial data line | $I^2C$ | I/O | 17-20 |
| SIZ[1:0] | Size | Bus | I/O | 17-8 |

**Table 17-2. MCF5407 Alphabetical Signal Index (Continued)**

| Abbreviation | Signal Name | Function | I/O | Page |
|---|---|---|---|---|
| $\overline{\text{SRAS}}$ | Synchronous row address strobe | DRAM | O | 17-17 |
| $\overline{\text{TA}}$ | Transfer acknowledge | Bus | I/O | 17-9 |
| TCK | Test clock | JTAG | I | 17-22 |
| TDI/DSI | Test data input/Development serial input | JTAG | I | 17-22 |
| TDO/DSO | Test data output/Development serial output | JTAG | O | 17-22 |
| TIN[1:0] | Timer input | Timer | I | 17-19 |
| $\overline{\text{TIP}}$ | Transfer in progress | Bus | O | 17-10 |
| TMS/$\overline{\text{BKPT}}$ | Test mode select/Breakpoint | JTAG | I | 17-21 |
| TM[2:0] | Transfer modifier | Bus | O | 17-10 |
| TOUT[1:0] | Timer outputs | Timer | O | 17-19 |
| $\overline{\text{TRST}}$/DSCLK | Test reset/Development serial clock | JTAG | I | 17-21 |
| $\overline{\text{TS}}$ | Transfer start | Bus | I/O | 17-9 |
| TT[1:0] | Transfer type | Bus | O | 17-10 |
| TxD[1:0] | Transmit data | Serial module | O | 17-18 |

# 17.2 MCF5407 Bus Signals

The bus signals provide the external bus interface to the MCF5407.

## 17.2.1 Address Bus

The address bus provides the address of the byte or most-significant byte (MSB) of the word or longword being transferred. The address lines also serve as the DRAM addressing, providing multiplexed row and column address signals. When an external device has ownership of the MCF5407 bus, the device must drive the address bus and assert $\overline{\text{TS}}$ or $\overline{\text{AS}}$ to indicate the start of a bus cycle. During an interrupt acknowledge access, A[4:2] indicate the interrupt level being acknowledged.

### 17.2.1.1 Address Bus (A[23:0])

The lower 24 bits of the address bus become valid when $\overline{\text{TS}}$ is asserted. A[4:2] indicate the interrupt level during interrupt acknowledge cycles.

### 17.2.1.2 Address Bus (A[31:24]/PP[15:8])

These multiplexed pins can serve as the most-significant byte of the address bus, or as the most-significant byte of the parallel port. Programming the PAR in the system integration module (SIM) determines the function of each of these eight multiplexed pins. These pins are programmable on a bit-by-bit basis.

- A[31:24]—Pins are configured as address bits by setting corresponding PAR bits; they represent the most-significant address bus bits. As much as 4 Gbytes of memory are available when all of these pins are programmed as address signals.
- PP[15:8]—Pins are configured as parallel port signals by clearing corresponding PAR bits; these represent the most-significant parallel port bits.

## 17.2.2 Data Bus (D[31:0])

The data bus is bidirectional and non-multiplexed. Data is sampled by the MCF5407 on the rising CLKIN edge. The data bus port width, wait states, and internal termination are initially defined for the boot chip select by D[7:0] during reset. The port width for each chip select and DRAM bank are programmable. The data bus uses a default configuration if none of the chip selects or DRAM bank match the address decode. The default configuration is a 32-bit port with external termination and burst-inhibited transfers. The data bus can transfer byte, word, or longword data widths. All 32 data bus signals are driven during writes, regardless of port width and operand size.

D[7:0] are used during reset initialization as inputs to configure the functions as described in Table 17-3. They are defined in Section 17.5.5, “Data/Configuration Pins (D[7:0]).”

**Table 17-3. Data Pin Configuration**

| Pin | Function | Section |
|---|---|---|
| D7 | Auto-acknowledge configuration (AA_CONFIG) | Section 17.5.5.2, “D7—Auto Acknowledge Configuration (AA_CONFIG)” |
| D[6:5] | Port size configuration (PS_CONFIG[1:0]) | Section 17.5.5.3, “D[6:5]—Port Size Configuration (PS_CONFIG[1:0])” |
| D4 | Address configuration (ADDR_CONFIG/D4) | Section 17.5.6, “D4—Address Configuration (ADDR_CONFIG)” |
| D3 | Byte enable configuration (BE_CONFIG) | Section 17.5.5.4, “D3—Byte-Enable Configuration (BE_CONFIG)” |
| D[2:0] | Divide control (DIVIDE[2:0]) | Section 17.5.6.1, “D[2:0]—Divide Control (DIVIDE[2:0])” |

## 17.2.3 Read/Write (R/$\overline{W}$)

When the MCF5407 is the bus master, it drives the R/$\overline{W}$ signal to indicate the direction of subsequent data transfers. It is driven high during read bus cycles and driven low during write bus cycles. This signal is an input during an external master access.

## 17.2.4 Size (SIZ[1:0])

When it is the bus master, the MCF5407 outputs these signals to indicate the requested data transfer size. Table 17-4 shows the definition of the bus request size encodings. When the MCF5407 device is not the bus master, these signals function as inputs.

Note that for misaligned transfers, SIZ[1:0] indicate the size of each transfer. For example,

if a longword access occurs at a misaligned offset of 0x1, a byte is transferred first (SIZ[1:0] = 01), a word is next transferred at offset 0x2 (SIZ[1:0] = 10), then the final byte is transferred at offset 0x4 (SIZ[1:0] = 01).

For aligned transfers larger than the port size, SIZ[1:0] behaves as follows:

- If bursting is used, SIZ[1:0] stays at the size of transfer.
- If bursting is inhibited, SIZ[1:0] first shows the size of the transfer and then shows the port size.

**Table 17-4. Bus Cycle Size Encoding**

| SIZ[1:0] | Port Size |
|---|---|
| 00 | Longword |
| 01 | Byte |
| 10 | Word |
| 11 | Line |

For burst-inhibited transfers, SIZ[1:0] changes with each $\overline{TS}$ assertion to reflect the next transfer size. For transfers to port sizes smaller than the transfer size, SIZ[1:0] indicates the size of the entire transfer on the first access and the size of the current port transfer on subsequent transfers. For example, for a longword write to an 8-bit port, SIZ[1:0] = 00 for the first byte transfer and 01 for the next three.

## 17.2.5 Transfer Start ($\overline{TS}$)

The MCF5407 asserts $\overline{TS}$ during the first clock cycle when address and attributes (TM, TT, $\overline{TIP}$, R/$\overline{W}$, and SIZ) are valid. $\overline{TS}$ is negated in the following clock cycle. When the MCF5407 is not the bus master, $\overline{TS}$ is an input.

## 17.2.6 Address Strobe ($\overline{AS}$)

Address strobe ($\overline{AS}$) is asserted to indicate when the address is stable at the start of a bus cycle. The address and attributes are guaranteed to be valid during the entire period that $\overline{AS}$ is asserted. This signal is asserted and negated on the falling edge of the clock. When the MCF5407 is not the bus master, $\overline{AS}$ is an input.

## 17.2.7 Transfer Acknowledge ($\overline{TA}$)

When the MCF5407 is bus master, the external system drives this input to terminate the bus transfer. The bus continues to be driven until this synchronous signal is asserted. For write cycles, the processor continues to drive data one clock after $\overline{TA}$ is asserted. During read cycles, the peripheral must continue to drive data until $\overline{TA}$ is recognized.

If all bus cycles support fast termination, $\overline{TA}$ can be tied low. However, note that $\overline{TA}$ cannot be tied low if potential external bus masters are present. The MCF5407 drives $\overline{TA}$ for an

external master access. This condition is indicated by the AM bit in the chip-select mask register (CSMR) being cleared. See Chapter 10, “Chip-Select Module.”

## 17.2.8 Transfer In Progress ($\overline{\text{TIP}}$/PP7)

The $\overline{\text{TIP}}$/PP7 pin is programmed in the PAR to serve as the transfer-in-progress output or as a parallel port bits. The $\overline{\text{TIP}}$ output is asserted indicating a bus transfer is in progress. It is negated during idle bus cycles if the bus is still granted to the processor. It is three-stated for external master accesses. Note that $\overline{\text{TIP}}$ is held asserted on back-to-back bus cycles.

## 17.2.9 Transfer Type (TT[1:0]/PP[1:0])

The TT[1:0]/PP[1:0] pins are programmed in the PAR to serve as the transfer type outputs or as two parallel port bits. When the MCF5407 is bus master and TT[1:0] are enabled, these signals are driven as outputs only. If an external master owns the bus and TT[1:0] are enabled, these pins are three-stated by the MCF5407 and can be driven by the external master. Table 17-5 shows the definition of the encodings.

**Table 17-5. Bus Cycle Transfer Type Encoding**

| TT[1:0] | Transfer Type |
|---|---|
| 00 | Normal access |
| 01 | DMA access |
| 10 | Emulator access |
| 11 | CPU space or interrupt acknowledge |

## 17.2.10 Transfer Modifier (TM[2:0]/PP[4:2]/DACK[1:0])

The TM[2:0]/PP[4:2] pins are programmed in the PAR to serve as the transfer modifier outputs or as three parallel port bits. These outputs provide supplemental information for each transfer type; see Table 17-6 through Table 17-10.

When the MCF5407 is the bus master and TM[2:0] are enabled, these signals are driven as outputs only. If an external device is bus master and TM[2:0] are enabled, these pins are three-stated by the MCF5407 and can be driven by the external master.

**Table 17-6. TM[2:0] Encodings for TT = 00 (Normal Access)**

| TM[2:0] | Transfer Modifier |
|---|---|
| 000 | Cache push access |
| 001 | User data access |
| 010 | User code access |
| 011–100 | Reserved |
| 101 | Supervisor data access |

**Table 17-6. TM[2:0] Encodings for TT = 00 (Normal Access) (Continued)**

| TM[2:0] | Transfer Modifier |
|---|---|
| 110 | Supervisor code access |
| 111 | Reserved |

As shown in Table 17-7, if the DMA is bus master (TT = 01), TM[2:0] indicate the type of DMA access and provide the DMA acknowledgement information for channels 0 and 1. In addition, TM[1:0] are multiplexed with DMA acknowledge signals for channels 0 and 1.

**NOTE:**

When TT= 01, the TM2 encoding is independent from TM[1:0] encoding.

**Table 17-7. TM2 Encoding for DMA as Master (TT = 01)**

| TM2 | Transfer Modifier Encoding |
|---|---|
| 0 | Single-address access negated |
| 1 | Single-address access |

**Table 17-8. TM[1:0] Encoding for DMA as Master (TT = 01)**

| TM[1:0] | Transfer Modifier Encoding |
|---|---|
| 00 | DMA acknowledges negated |
| 01 | DMA acknowledge, channel 0 |
| 10 | DMA acknowledge, channel 1 |
| 11 | Reserved |

Table 17-9 shows TM[2:0] encodings for emulator mode accesses.

**Table 17-9. TM[2:0] Encodings for TT = 10 (Emulator Access)**

| TM[2:0] | Transfer Modifier |
|---|---|
| 000–100 | Reserved |
| 101 | Emulator mode data access |
| 110 | Emulator mode code access |
| 111 | Reserved |

The TM signals indicate user or data transfer types during emulation transfers, while for interrupt acknowledge transfers, the TM signals carry the interrupt level being acknowledged; see Table 17-10.

**Table 17-10. TM[2:0] Encodings for TT = 11 (Interrupt Level)**

| TM[2:0] | Transfer Modifier |
|---|---|
| 000 | CPU Space |
| 001 | Interrupt level 1 acknowledge |
| 010 | Interrupt level 2 acknowledge |
| 011 | Interrupt level 3 acknowledge |
| 100 | Interrupt level 4 acknowledge |
| 101 | Interrupt level 5 acknowledge |
| 110 | Interrupt level 6 acknowledge |
| 111 | Interrupt level 7 acknowledge |

## 17.3 Interrupt Control Signals

The interrupt control signals supply the external interrupt level to the MCF5407 device.

### 17.3.1 Interrupt Request ($\overline{IRQ1}$/$\overline{IRQ2}$, $\overline{IRQ3}$/$\overline{IRQ6}$, $\overline{IRQ5}$/$\overline{IRQ4}$, and $\overline{IRQ7}$)

The $\overline{IRQ1}$, $\overline{IRQ3}$, $\overline{IRQ5}$, and $\overline{IRQ7}$ signals are the default interrupt request signals ($\overline{IRQn}$). However, by setting the appropriate bit in the interrupt port assignment register (IRQPAR), $\overline{IRQ1}$, $\overline{IRQ3}$, and $\overline{IRQ5}$ can be changed to function as $\overline{IRQ2}$, $\overline{IRQ6}$, and $\overline{IRQ4}$, respectively. See Section 9.2.4, “Interrupt Port Assignment Register (IRQPAR).”

## 17.4 Bus Arbitration Signals

The bus arbitration signals provide the external bus arbitration control for the MCF5407.

### 17.4.1 Bus Request ($\overline{BR}$)

The $\overline{BR}$ output indicates to an external arbiter that the processor is requesting to be bus master for one or more bus cycles. $\overline{BR}$ is negated when the MCF5407 begins an access to the external bus with no other internal accesses pending. $\overline{BR}$ remains negated until another internal request occurs.

### 17.4.2 Bus Grant ($\overline{BG}$)

An external arbiter asserts the $\overline{BG}$ input to indicate that the MCF5407 can take control of the bus on the next rising edge of CLKIN. When the arbiter negates $\overline{BG}$, the MCF5407 will release the bus as soon as the current transfer completes. The external arbiter must not grant the bus to any other master until both $\overline{BD}$ and $\overline{BG}$ are negated.

### 17.4.3 Bus Driven ($\overline{BD}$)

The MCF5407 asserts $\overline{BD}$ to indicate that it is the current master and is driving the bus. The MCF5407 behaves as follows:

- If the MCF5407 is the bus master but is not using the bus, $\overline{BD}$ is asserted.
- If the MCF5407 loses mastership during a transfer, it completes the last transfer of the access, negates $\overline{BD}$, and three-states all bus signals on the rising edge of CLKIN.
- If the MCF5407 loses bus mastership during an idle clock cycle, it three-states all bus signals on the rising edge of CLKIN.
- $\overline{BD}$ cannot be negated unless $\overline{BG}$ is negated.

## 17.5 Clock and Reset Signals

The clock and reset signals configure the MCF5407 and provide interface signals to the external system.

### 17.5.1 Reset In ($\overline{RSTI}$)

Asserting $\overline{RSTI}$ causes the MCF5407 to enter reset exception processing. When $\overline{RSTI}$ is recognized, $\overline{BR}$ and $\overline{BD}$ are negated and the address bus, data bus, TT, SIZ, R/$\overline{W}$, $\overline{AS}$, and $\overline{TS}$ are three-stated. $\overline{RSTO}$ is asserted automatically when $\overline{RSTI}$ is asserted.

### 17.5.2 Clock Input (CLKIN)

CLKIN is the MCF5407 input clock frequency to the on-board phase-locked-loop (PLL) clock generator. CLKIN is used to internally clock or sequence the MCF5407 internal bus interface at a selected multiple of the input frequency used for internal module logic. CLKIN should be used as the bus timing reference.

### 17.5.3 Bus Clock Output (BCLKO)

The internal PLL generates BCLKO. It has the same frequency as CLKIN, which is used as the bus timing reference by the external devices. BCLKO is provided for compatibility with earlier devices.

### 17.5.4 Reset Out ($\overline{RSTO}$)

After $\overline{RSTI}$ is asserted, the PLL temporarily loses its lock, during which time $\overline{RSTO}$ is asserted. When the PLL regains its lock, $\overline{RSTO}$ negates again. This signal can be used to reset external devices.

## 17.5.5 Data/Configuration Pins (D[7:0])

This section describes data pins, D[7:0], that are read at reset for configuration. Table 17-11 shows pin assignments.

**Table 17-11. Data Pin Configuration**

| Pin | Function |
|---|---|
| D7 | Auto-acknowledge configuration (AA_CONFIG) |
| D[6:5] | Port size configuration (PS_CONFIG[1:0]) |
| D4 | Address configuration (ADDR_CONFIG/D4) |
| D3 | Byte enable configuration (BE_CONFIG) |
| D[2:0] | Divide control (DIVIDE[2:0]) |

### 17.5.5.1 D[7:5,3]—Boot Chip-Select ($\overline{CS0}$) Configuration

D[7:5,3] determine defaults for the global chip select ($\overline{CS0}$), the only chip select valid at reset. These signals correspond to bits in chip-select configuration register 0 (CSCR0).

### 17.5.5.2 D7—Auto Acknowledge Configuration (AA_CONFIG)

At reset, the enabling and disabling of auto acknowledge for boot $\overline{CS0}$ is determined by the logic level driven on D7 at the rising edge of $\overline{RSTI}$. AA_CONFIG is multiplexed with D7 and sampled only at reset. The D7 logic level is reflected as the reset value of CSCR[AA]. Table 17-12 shows how the D7 logic level corresponds to the auto acknowledge timing for $\overline{CS0}$ at reset. Note that auto acknowledge can be disabled by driving a logic 0 on D7 at reset.

**Table 17-12. D7 Selection of $\overline{CS0}$ Automatic Acknowledge**

| D7 (CSCR0[AA]) | Boot $\overline{CS0}$ AA |
|---|---|
| 0 | Disabled |
| 1 | Enabled with 15 wait states |

### 17.5.5.3 D[6:5]—Port Size Configuration (PS_CONFIG[1:0])

The default port size value of the boot $\overline{CS0}$ is determined by the logic levels driven on D[6:5] at the rising edge of $\overline{RSTI}$, which are reflected as the reset value of CSCR[PS]. Table 17-13 shows how the logic levels of D[6:5] correspond to the $\overline{CS0}$ port size at reset.

**Table 17-13. D6 and D5 Selection of $\overline{CS0}$ Port Size**

| D[6:5] (CSCR0[PS]) | Boot $\overline{CS0}$ Port Size |
|---|---|
| 00 | 32-bit port |
| 01 | 8-bit port |
| 1x | 16-bit port |

#### 17.5.5.4 D3—Byte-Enable Configuration (BE_CONFIG)

The default byte-enable mode of the boot $\overline{CS0}$ is determined by the logic level driven on D3 at the rising edge of $\overline{RSTI}$. This logic level is reflected as the reset value of CSCR0[BEM]. Table 17-13 shows how the logic levels of D[6:5] correspond to the port size for $\overline{CS0}$ at reset.

**Table 17-14. D3/BE_CONFIG, BE[3:0] Boot Configuration**

| D3 (CSCR0[BEM]) | Boot $\overline{CS0}$ Byte Enable Configuration |
|---|---|
| 0 | Neither $\overline{BE}$ nor $\overline{BWE}$ is asserted for read. $\overline{BWE}$ is generated for data write only. |
| 1 | $\overline{BE}$ is asserted for read; $\overline{BWE}$ is asserted for write. |

### 17.5.6 D4—Address Configuration (ADDR_CONFIG)

The address configuration signal (ADDR_CONFIG) programs the PAR of the parallel I/O port to be either parallel I/O or to be the upper address bus bits along with various attribute and control signals at reset to give the user the option to access a broader addressing range of memory if desired. ADDR_CONFIG is multiplexed with D4 and its configuration is sampled at reset as shown in Table 17-15.

**Table 17-15. D4/ADDR_CONFIG, Address Pin Assignment**

| D4/ADDR_CONFIG | PAR Configuration at Reset |
|---|---|
| 0 | PP[15:0], defaulted to inputs upon reset |
| 1 | A[31:24]/$\overline{TIP}$/$\overline{DREQ}$[1:0]/TM[2:0]/TT[1:0] |

#### 17.5.6.1 D[2:0]—Divide Control (DIVIDE[2:0])

The divide control input bus, DIVIDE[2:0], indicates the CLKIN/PCLK ratio. These signals are sampled on the rising edge of $\overline{RSTI}$ to indicate the ratios described in Chapter 20, “Electrical Specifications.”

## 17.6 Chip-Select Module Signals

The MCF5407 device provides eight programmable chip-select signals that can directly interface with SRAM, EPROM, EEPROM, and peripherals. These signals are asserted and negated on the falling edge of the clock.

### 17.6.1 Chip-Select ($\overline{CS}$[7:0])

Each chip select can be programmed for a base address location and for masking addresses, port size and burst-capability indication, wait-state generation, and internal/external termination.

Reset clears all chip select programming; $\overline{CS0}$ is the only chip select initialized out of reset. $\overline{CS0}$ is also unique because it can function at reset as a global chip select that allows boot

ROM to be selected at any defined address space. Port size and termination (internal vs. external) for boot $\overline{CS0}$ are configured by the levels on D[7:5,3] on the rising edge of $\overline{RSTI}$, as described in Section 17.5.5.1, “D[7:5,3]—Boot Chip-Select (CS0) Configuration.”

The chip-select implementation is described in Chapter 10, “Chip-Select Module.”

### 17.6.2 Byte Enables/Byte Write Enables ($\overline{BE}$[3:0]/$\overline{BWE}$[3:0])

The four byte enables are multiplexed with the MCF5407 byte-write-enable signals. Each pin can be individually programmed through the chip-select control registers (CSCRs). For each chip select, assertion of byte enables for reads and byte-write enables for write cycles can be programmed. Alternatively, users can program byte-write enables to assert on writes and no byte enable assertion for read transfers.

### 17.6.3 Output Enable ($\overline{OE}$)

The output enable ($\overline{OE}$) signal 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.

## 17.7 DRAM Controller Signals

The DRAM signals in the following sections interface to external DRAM. DRAM with widths of 8, 16, and 32 bits are supported and can access as much as 512 Mbytes of DRAM.

### 17.7.1 Row Address Strobes ($\overline{RAS}$[1:0])

The row address strobes ($\overline{RAS}$[1:0]) interface to $\overline{RAS}$ inputs on industry-standard ADRAMs. When SDRAMs are used, these signals interface to the chip-select lines of the SDRAMs within a memory block. Thus, there is one $\overline{RAS}$ line for each memory block (because the MCF5407 supports only two memory blocks).

### 17.7.2 Column Address Strobes ($\overline{CAS}$[3:0])

The column address strobes ($\overline{CAS}$[3:0]) interface to $\overline{CAS}$ inputs on industry-standard DRAMs. These provide $\overline{CAS}$ for a given ADRAM block. When SDRAMs are used, $\overline{CAS}$ signals control the byte enables for standard SDRAMs (referred to as DQM*x*). $\overline{CAS3}$ accesses the LSB and $\overline{CAS0}$ accesses the MSB of data.

### 17.7.3 DRAM Write ($\overline{DRAMW}$)

The DRAM write signal ($\overline{DRAMW}$) is asserted to signify that a DRAM write cycle is underway. A read bus cycle is indicated by the negation of $\overline{DRAMW}$.

### 17.7.4 Synchronous DRAM Column Address Strobe ($\overline{\text{SCAS}}$)

The synchronous DRAM column address strobe ($\overline{\text{SCAS}}$) is registered during synchronous mode to route directly to the $\overline{\text{SCAS}}$ signal of SDRAMs.

### 17.7.5 Synchronous DRAM Row Address Strobe ($\overline{\text{SRAS}}$)

The synchronous DRAM row address strobe output ($\overline{\text{SRAS}}$) is registered during synchronous mode to route directly to the $\overline{\text{SRAS}}$ signal of external SDRAMs.

### 17.7.6 Synchronous DRAM Clock Enable (SCKE)

The synchronous DRAM clock enable output (SCKE) is registered during synchronous mode to route directly to the SCKE signal of external SDRAMs. This signal provides the clock enable to the SDRAM.

### 17.7.7 Synchronous Edge Select (EDGESEL)

The synchronous edge select input (EDGESEL) helps select additional output hold times for signals that interface to external SDRAMs. It provides the following three modes of operation for SDRAM control signals:

- When EDGESEL is tied high, SDRAM control signals change on the rising edge of CLKIN.
- When EDGESEL is tied low, SDRAM control signals change on the falling edge of CLKIN.
- When EDGESEL is tied to the external clock (normally buffered CLKIN), which drives the SDRAM and other devices, SDRAM signals are generated within the MCF5407 make a transition on the rising edge of the SDRAM clock. See Figure 11-14 on page 11-19. This loop-back configuration provides additional output hold time for MCF5407 interface signals provided to the SDRAM. In this case, the SDRAM clock operates at the CLKIN frequency, with a possible slight phase delay.

## 17.8 DMA Controller Module Signals

The DMA controller module uses the signals in the following subsections to provide external request for either a source or destination.

### 17.8.1 DMA Request ($\overline{\text{DREQ}}$[1:0]/PP[6:5])

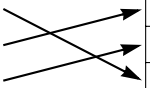
The DMA request pins ($\overline{\text{DREQ}}$[1:0]/PP[6:5]) can serve as the DMA request inputs or as two bits of the parallel port, as determined by individually programmable bits in the PAR.

These inputs are asserted by a peripheral device to request an operand transfer between that peripheral and memory by either channel 0 or 1 of the on-chip DMA.

### 17.8.2 Transfer Modifier/DMA Acknowledge (TM[2:0]/DACK[1:0])

Although the MCF5407 provides similar encodings on TM[2:0], DMA acknowledgement pins (DACK[1:0]) are now combined with PP[3:2]/TM[1:0], resulting in three-to-one multiplexed signals, PP[3:2]/TM[1:0]/DACK[1:0]. TM2 is still multiplexed only with PP4. When properly connected, TM[2:0] can be used in MCF5407 designs as on MCF5307 designs, or DACK[1:0] can be used for DMA transfers, as shown in Figure 17-2.

| MCF5307 Function | Pin |
|---|---|
| Single/dual cycle access | TM0 |
| DMA 0 acknowledge configuration | TM1 |
| DMA 1 acknowledge configuration | TM2 |



| Pin | MCF5407 Function |
|---|---|
| TM0 | DMA 0 acknowledge |
| TM1 | DMA 1 acknowledge |
| TM2 | Single/dual cycle access |

**Figure 17-2. MCF5307 to MCF5407 TM[2:0] Pin Remapping**

To enable DACK[1:0], first enable TM[1:0] through the PAR and then program the interrupt assignment register (IRQPAR) in the MCF5407 SIM module to enable bits 0–1.

When IRQPAR[ENBDACK1] = 1 and PAR is programmed to enable TM1, DACK1 for DMA channel 1 is driven in place of TM1 for DMA transfers. Clearing ENBDACK1 disables this function and only the TM1 encoding is driven. Likewise, setting ENBDACK0 enables DACK0 to be driven; clearing ENBDACK0 disables this function and drives the TM0 encoding.

Although the MCF5407 TM[2:0] signals can drive DMA access encoding, the bit positions of these encodings differ from the MCF5307. Single-address access indication is now encoded on TM2 when the PAR is set to enable the transfer modifier signal and an external master or DMA transfer is occurring. This encoding is driven by TM0 on the MCF5307. In addition, DMA acknowledge encodings are driven on TM[1:0] on the MCF5407, as opposed to TM[2:1] on the MCF5307.

## 17.9 Serial Module Signals

The signals in the following sections are used to transfer serial data between the two UART modules and external peripherals.

### 17.9.1 Transmitter Serial Data Output (TxD)

In UART mode, TxD is held high (mark condition) when the transmitter is disabled, idle, or operating in the local loop-back mode. Data is shifted out least-significant bit (lsb) first on TxD on the falling edge of the clock source. For UART1 in modem mode, TxD is held low when the transmitter is disabled or idle. Data is shifted out on TxD on the rising edge of the clock signal driving UART1's $\overline{\text{CTS}}$ input. UART1 transfers can be specified as either lsb or msb first.

### 17.9.2 Receiver Serial Data Input (RxD)

Data received on RxD is sampled on the rising edge of the clock source, with the lsb received first. For UART1 in modem mode, data received on RxD is sampled on the falling edge of the clock signal driving UART1's $\overline{\text{CTS}}$ input. UART1 transfers can be specified as either lsb or msb first.

### 17.9.3 Clear to Send ($\overline{\text{CTS}}$)

This input can generate an interrupt on a change of state. For UART1 in modem mode, $\overline{\text{CTS}}$ must be driven by the serial bit clock from the external CODEC or AC97 controller.

### 17.9.4 Request to Send ($\overline{\text{RTS}}$)

This output can be programmed to be negated or asserted automatically by either the receiver or the transmitter. When connected to a transmitter's $\overline{\text{CTS}}$, $\overline{\text{RTS}}$ can control serial data flow. For UART1 in AC97 mode, $\overline{\text{RTS}}$ serves as the frame sync or start of frame (SOF), output to the external AC97 controller. When this mode is used, the AC97 BIT_CLK, which is input on $\overline{\text{CTS}}$, is divided by 256.

## 17.10 Timer Module Signals

The signals in the following sections are external interfaces to the two general-purpose MCF5407 timers. These 16-bit timers can capture timer values, trigger external events or internal interrupts, or count external events.

### 17.10.1 Timer Inputs (TIN[1:0])

TIN[1:0] can be programmed as clocks that cause events in the counter and prescalers. They can also cause captures on the rising edge, falling edge, or both edges.

### 17.10.2 Timer Outputs (TOUT1, TOUT0)

The programmable timer outputs (TOUT1 and TOUT0) pulse or toggle on various timer events.

## 17.11 Parallel I/O Port (PP[15:0])

This 16-bit bus is dedicated for general-purpose I/O. The parallel port is multiplexed with the A[31:24], TT[1:0], TM[2:0], $\overline{\text{TIP}}$, and $\overline{\text{DREQ}}$[1:0]. These 16 bits are programmed for functionality with the PAR in the SIM.

The system designer controls the reset value of this register by driving D4 with a 1 or 0 on the rising edge of $\overline{\text{RSTI}}$ (reset input to MCF5407 device). At reset, the system is configured as PP[15:0] if D4 is 0; otherwise alternate pin functions selected by PAR = 1 are used.

Motorola recommends that D4 be driven during reset to a logic level.

## 17.12 I²C Module Signals

The I²C module acts as a two-wire, bidirectional serial interface between the MCF5407 and peripherals with an I²C interface (such as LED controller, A-to-D converter, or D-to-A converter). Devices connected to the I²C must have open-drain or open-collector outputs.

### 17.12.1 I²C Serial Clock (SCL)

The bidirectional, open-drain I²C serial clock signal (SCL) is the clock signal for I²C module operation. The I²C module controls this signal when the bus is in master mode; all I²C devices drive this signal to synchronize I²C timing.

### 17.12.2 I²C Serial Data (SDA)

The bidirectional, open-drain I²C serial data signal (SDA) is the data input/output for the serial I²C interface.

## 17.13 Debug and Test Signals

The signals in this section interface with external I/O to provide processor status signals.

### 17.13.1 Test Mode (MTMOD[3:0])

The test mode signals choose between multiplexed debug module and JTAG signals. If MTMOD0 is low, the part is in normal and background debug mode (BDM); if it is high, it is in normal and JTAG mode. All other MTMOD values are reserved; MTMOD[3:1] should be tied to ground and MTMOD[3:0] should not be changed while $\overline{\text{RSTI}}$ is negated.

### 17.13.2 High Impedance ($\overline{\text{HIZ}}$)

The assertion of $\overline{\text{HIZ}}$ forces all output drivers to high-impedance state. The timing on $\overline{\text{HIZ}}$ is independent of the clock. Note that $\overline{\text{HIZ}}$ does not override the JTAG operation; TDO/DSO can be forced to high impedance by asserting $\overline{\text{TRST}}$.

### 17.13.3 Processor Clock Output (PSTCLK)

The internal PLL generates this output signal, and is the processor clock output that is used as the timing reference for the debug bus timing (PSTDDATA[7:0]). PSTCLK is at the same frequency as the core processor and cache memory.

### 17.13.4 Processor Status Debug Data (PSTDDATA[7:0])

Processor status data outputs indicate both processor status and captured address and data values. They operate at half the processor's frequency, using PSTCLK. Given that real-time trace information appears as a sequence of 4-bit data values, there are no alignment restrictions; that is, PST values and operands may appear on either PSTDDATA[7:0] nibble. The upper nibble, PSTDDATA[7:4], is most significant. See Chapter 5, "Debug Support."

## 17.14 Debug Module/JTAG Signals

The MCF5407 complies with the IEEE 1149.1a JTAG testing standard. JTAG test pins are multiplexed with background debug pins. Except for TCK, these signals are selected by the value of MTMOD0. If MTMOD0 is high, JTAG signals are chosen; if it is low, debug module signals are chosen. MTMOD0 should be changed only while $\overline{\text{RSTI}}$ is asserted.

### 17.14.1 Test Reset/Development Serial Clock ($\overline{\text{TRST}}$/DSCLK)

If MTMOD0 is high, $\overline{\text{TRST}}$ is selected. $\overline{\text{TRST}}$ asynchronously resets the internal JTAG controller to the test logic reset state, causing the JTAG instruction register to choose the bypass instruction. When this occurs, JTAG logic is benign and does not interfere with normal MCF5407 functionality.

Although $\overline{\text{TRST}}$ is asynchronous, Motorola recommends that it makes an asserted-to-negated transition only while TMS is held high. $\overline{\text{TRST}}$ has an internal pull-up resistor so if it is not driven low, it defaults to a logic level of 1. If $\overline{\text{TRST}}$ is not used, it can be tied to ground or, if TCK is clocked, to $V_{DD}$. Tying $\overline{\text{TRST}}$ to ground places the JTAG controller in test logic reset state immediately. Tying it to $V_{DD}$ causes the JTAG controller (if TMS is a logic level of 1) to eventually enter test logic reset state after 5 TCK clocks.

If MTMOD0 is low, DSCLK is selected. DSCLK is the development serial clock for the serial interface to the debug module. The maximum DSCLK frequency is 1/5 CLKIN. See Chapter 5, "Debug Support."

### 17.14.2 Test Mode Select/Breakpoint (TMS/$\overline{\text{BKPT}}$)

If MTMOD0 is high, TMS is selected. The TMS input provides information to determine the JTAG test operation mode. The state of TMS and the internal 16-state JTAG controller state machine at the rising edge of TCK determine whether the JTAG controller holds its current state or advances to the next state. This directly controls whether JTAG data or instruction operations occur. TMS has an internal pull-up resistor so that if it is not driven low, it defaults to a logic level of 1. But if TMS is not used, it should be tied to $V_{DD}$.

If MTMOD0 is low, $\overline{\text{BKPT}}$ is selected. $\overline{\text{BKPT}}$ signals a hardware breakpoint to the

processor in debug mode. See Chapter 5, “Debug Support.”

### 17.14.3 Test Data Input/Development Serial Input (TDI/DSI)

If MTMOD0 is high, TDI is selected. TDI provides the serial data port for loading the various JTAG boundary scan, bypass, and instruction registers. Shifting in data depends on the state of the JTAG controller state machine and the instruction in the instruction register. Shifts occur on the TCK rising edge. TDI has an internal pull-up resistor, so when not driven low it defaults to high. But if TDI is not used, it should be tied to $V_{DD}$.

If MTMOD0 is low, DSI is selected. DSI provides the single-bit communication for debug module commands. See Chapter 5, “Debug Support.”

### 17.14.4 Test Data Output/Development Serial Output (TDO/DSO)

If MTMOD0 is high, TDO is selected. The TDO output provides the serial data port for outputting data from JTAG logic. Shifting out data depends on the JTAG controller state machine and the instruction in the instruction register. Data shifting occurs on the falling edge of TCK. When TDO is not outputting test data, it is three-stated. TDO can be three-stated to allow bused or parallel connections to other devices having JTAG.

If MTMOD0 is low, DSO is selected. DSO provides single-bit communication for debug module responses. See Chapter 5, “Debug Support.”

### 17.14.5 Test Clock (TCK)

TCK is the dedicated JTAG test logic clock independent of the MCF5407 processor clock. Various JTAG operations occur on the rising or falling edge of TCK. Holding TCK high or low for an indefinite period does not cause JTAG test logic to lose state information. If TCK is not used, it must be tied to ground.

# Chapter 18
# Bus Operation

This chapter describes data-transfer operations, error conditions, bus arbitration, and reset operations. It describes transfers initiated by the MCF5407 and by an external bus master, and includes detailed timing diagrams showing the interaction of signals in supported bus operations. Chapter 11, "Synchronous/Asynchronous DRAM Controller Module," describes DRAM cycles.

## 18.1 Features

The following list summarizes bus operation features:

- Up to 32 bits of address and data
- 8-, 16-, and 32-bit port sizes
- Byte, word, longword, and line size transfers
- Bus arbitration for external devices
- Burst and burst-inhibited transfer support
- Internal termination for core and DMA bus cycles
- External termination of bus cycles controlled by an external bus master

Note that, throughout this manual, an overbar indicates an active-low signal.

## 18.2 Bus and Control Signals

Table 18-1 summarizes MCF5407 bus signals described in Chapter 17, "Signal Descriptions."

**Table 18-1. ColdFire Bus Signal Summary**

| Signal Name | Description | MCF5407 Master | External Master | Edge |
|---|---|---|---|---|
| $\overline{AS}$ | Address strobe | O | I | Falling |
| A[31:0] | Address bus | O | I | Rising |
| $\overline{BE}$/$\overline{BWE}$ [1] | Byte enable/Byte write enable | O | O | Falling |
| $\overline{CS}$[7:0] [1] | Chip selects | O | O | Falling |
| D[31:0] | Data bus | I/O | I/O | Rising |

**Table 18-1. ColdFire Bus Signal Summary (Continued)**

| Signal Name | Description | MCF5407 Master | External Master | Edge |
|---|---|---|---|---|
| $\overline{IRQ}$[7,5,3,1] | Interrupt request | I | I | Rising |
| $\overline{OE}$ [1] | Output enable | O | I | Falling |
| R/$\overline{W}$ | Read/write | O | I | Rising |
| SIZ[1:0] | Transfer size | O | I | Rising |
| $\overline{TA}$ | Transfer acknowledge | I | O | Rising |
| $\overline{TIP}$ | Transfer in progress | O | Three-state | Rising |
| TM[2:0] | Transfer modifier | O | Three-state | Rising |
| $\overline{TS}$ | Transfer start | O | I | Rising |
| TT[1:0] | Transfer type | O | Three-state | Rising |

1 These signals change after the falling edge. In Chapter 20, “Electrical Specifications,” these signals are specified off the rising edge because CLKIN is squared up internally.

## 18.3 Bus Characteristics

The MCF5407 uses an input clock signal (CLKIN) to generate its internal clock and outputs BCLKO, which is provided for backwards compatibility for MCF5307 designs. CLKIN is the bus clock rate, where all bus operations are synchronous to the rising edge of CLKIN. Some of the bus control signals ($\overline{BE}$/$\overline{BWE}$, $\overline{OE}$, $\overline{CS}x$, and $\overline{AS}$) are synchronous to the falling edge, shown in Figure 18-1. Bus characteristics may differ somewhat for interfacing with external DRAM.

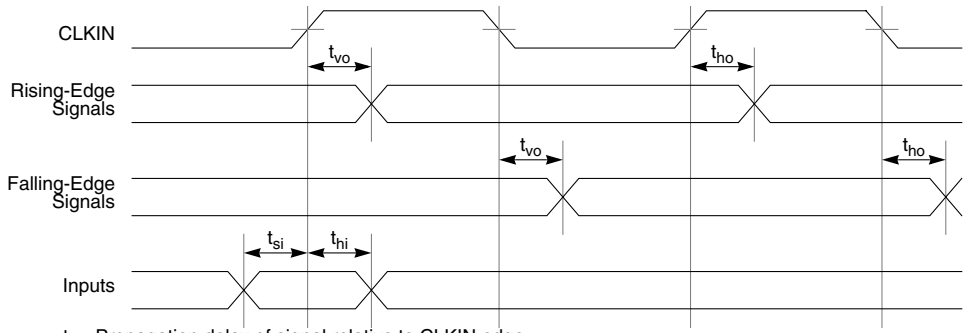



**Figure 18-1. Signal Relationship to CLKIN for Non-DRAM Access**

## 18.4 Data Transfer Operation

Data transfers between the MCF5407 and other devices involve the following signals:

- Address bus (A[31:0])
- Data bus (D[31:0])
- Control signals ($\overline{TS}$ and $\overline{TA}$)
- $\overline{AS}$, $\overline{CSx}$, $\overline{OE}$, $\overline{BE}$/$\overline{BWE}$
- Attribute signals (R/$\overline{W}$, SIZ, TT, TM, and $\overline{TIP}$)

The address bus, write data, $\overline{TS}$, and all attribute signals change on the rising edge of CLKIN. Read data is latched into the MCF5407 on the rising edge of CLKIN. $\overline{AS}$, $\overline{CSx}$, $\overline{OE}$, and $\overline{BE}$/$\overline{BWE}$ change on the falling edge.

The MCF5407 bus supports byte, word, and longword operand transfers and allows accesses to 8-, 16-, and 32-bit data ports. Transfer parameters such as port size, the number of wait states for the external slave being accessed, and whether internal transfer termination is enabled, can be programmed in the chip-select control registers (CSCRs) and DRAM control registers (DACRs).

For aligned transfers larger than the port size, SIZ[1:0] behaves as follows:

- If bursting is used, SIZ[1:0] stays at the size of transfer.
- If bursting is inhibited, SIZ[1:0] first shows the size of the transfer and then shows the port size.

Table 18-2 shows encoding for SIZ[1:0].

**Table 18-2. Bus Cycle Size Encoding**

| SIZ[1:0] | Port Size |
|---|---|
| 00 | Longword |
| 01 | Byte |
| 10 | Word |
| 11 | Line |

Figure 18-2 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 D[31:24] ($\overline{BE0}$). A longword transfer takes four transfers on D[31:24], starting with the MSB and going to the LSB.

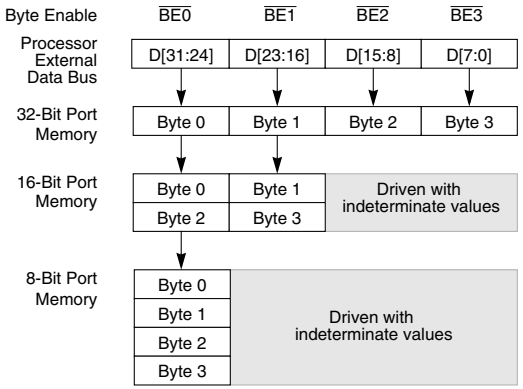



**Figure 18-2. Connections for External Memory Port Sizes**

The timing relationships between CLKIN and chip select ($\overline{CS}$[7:0]), byte enable/byte write enables ($\overline{BE}$/$\overline{BWE}$[3:0]), and output enable ($\overline{OE}$) are similar to their relationships with address strobe ($\overline{AS}$) in that all transitions occur during the low phase of CLKIN. However, as shown in Figure 18-3, differences in on-chip signal routing and external loading may prevent signals from asserting simultaneously.

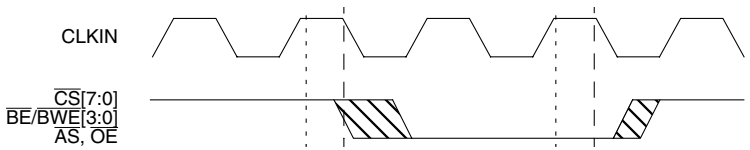



**Figure 18-3. Chip-Select Module Output Timing Diagram**

## 18.4.1 Bus Cycle Execution

When a bus cycle is initiated, the MCF5407 first compares its address with the base address and mask configurations programmed for chip selects 0–7 (CSCR0–CSCR7) and for DRAM blocks 0 and 1 address and control registers (DACR0 and DACR1). If the driven address matches a programmed chip select or DRAM block, the appropriate chip select is asserted or the DRAM block is selected using the specifications programmed in the respective configuration register. Otherwise, the following occurs:

- If the address and attributes do not match in CSCR or DACR, the MCF5407 runs an external burst-inhibited bus cycle with a default of external termination on a 32-bit port.
- If an address and attribute match in multiple CSCRs, the matching chip-select signals are driven; however, the MCF5407 runs an external burst-inhibited bus cycle with external termination on a 32-bit port.
- If an address and attribute match both DACRs or a DACR and a CSCR, the operation is undefined.

Table 18-3 shows the type of access as a function of match in the CSCRs and DACRs.

**Table 18-3. Accesses by Matches in CSCRs and DACRs**

| Number of CSCR Matches | Number of DACR Matches | Type of Access |
|---|---|---|
| 0 | 0 | External |
| 1 | 0 | Defined by CSCRs |
| Multiple | 0 | External, burst-inhibited, 32-bit |
| 0 | 1 | Defined by DACRs |
| 1 | 1 | Undefined |
| Multiple | 1 | Undefined |
| 0 | Multiple | Undefined |
| 1 | Multiple | Undefined |
| Multiple | Multiple | Undefined |

Basic bus operations occur in three clocks, as follows:

1. During the first clock, the address, attributes, and $\overline{TS}$ are driven. $\overline{AS}$ is asserted at the falling edge of the clock to indicate that address and attributes are valid and stable.
2. Data and $\overline{TA}$ are sampled during the second clock of a bus-read cycle. During a read, the external device provides data and is sampled at the rising edge at the end of the second bus clock. This data is concurrent with $\overline{TA}$, which is also sampled at the rising clock edge.

   During a write, the MCF5407 drives data from the rising clock edge at the end of the first clock to the rising clock edge at the end of the bus cycle. Wait states can be added between the first and second clocks by delaying the assertion of $\overline{TA}$. $\overline{TA}$ can be configured to be generated internally through the DACRs and CSCRs. If $\overline{TA}$ is not generated internally, the system must provide it externally.
3. The 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. Figure 18-6 and Figure 18-8 show the basic read and write operations.

## 18.4.2 Data Transfer Cycle States

The data transfer operation in the MCF5407 is controlled by an on-chip state machine. Each bus clock cycle is divided into two states. Even states occur when CLKIN is high and odd states occur when CLKIN is low. The state transition diagram for basic and fast-termination read and write cycles is shown in Figure 18-4.

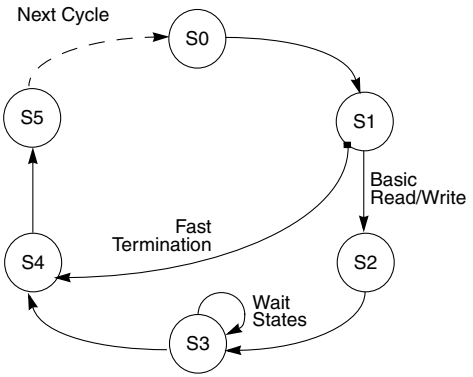



**Figure 18-4. Data Transfer State Transition Diagram**

Table 18-4 describes the states as they appear in subsequent timing diagrams. Note that the TT[1:0], TM[2:0], and $\overline{\text{TIP}}$ functions are chosen in the PAR, as described in Section 15.1.1, "Pin Assignment Register (PAR)."

**Table 18-4. Bus Cycle States**

| State | Cycle | CLKIN | Description |
|---|---|---|---|
| S0 | All | High | The read or write cycle is initiated. On the rising edge of CLKIN, the MCF5407 places a valid address on the address bus, asserts $\overline{\text{TIP}}$, and drives R/$\overline{\text{W}}$ high for a read and low for a write, if these signals are not already in the appropriate state. The MCF5407 asserts TT[1:0], TM[2:0], SIZ[1:0], and $\overline{\text{TS}}$ on the rising edge of CLKIN. |
| S1 | All | Low | $\overline{\text{AS}}$ asserts on the falling edge of CLKIN, indicating that the address and attributes are stable. The appropriate $\overline{\text{CS}}$x, $\overline{\text{BE}}$/$\overline{\text{BWE}}$, and $\overline{\text{OE}}$ signals assert on the CLKIN falling edge. |
| | Fast termination | | $\overline{\text{TA}}$ must be asserted during S1. Data is made available by the external device and is sampled on the rising edge of CLKIN with $\overline{\text{TA}}$ asserted. |
| S2 | Read/write (skipped for fast termination) | High | $\overline{\text{TS}}$ is negated on the rising edge of CLKIN. |
| | Write | | The data bus is driven out of high impedance as data is placed on the bus on the rising edge of CLKIN. |
| S3 | Read/write (skipped for fast termination) | Low | The MCF5407 waits for $\overline{\text{TA}}$ assertion. If $\overline{\text{TA}}$ is not sampled as asserted before the rising edge of CLKIN at the end of the first clock cycle, the MCF5407 inserts wait states (full clock cycles) until $\overline{\text{TA}}$ is sampled as asserted. |
| | Read | | Data is made available by the external device on the falling edge of CLKIN and is sampled on the rising edge of CLKIN with $\overline{\text{TA}}$ asserted. |
| S4 | All | High | The external device should negate $\overline{\text{TA}}$. |
| | Read (including fast termination) | | The external device can stop driving data after the rising edge of CLKIN. However, data could be driven up to S5. |

**Table 18-4. Bus Cycle States (Continued)**

| State | Cycle | CLKIN | Description |
|---|---|---|---|
| S5 | S5 | Low | $\overline{AS}$, $\overline{CS}$, $\overline{BE}$/$\overline{BWE}$, and $\overline{OE}$ are negated on the CLKIN falling edge. The MCF5407 stops driving address lines and R/$\overline{W}$ on the rising edge of CLKIN, terminating the read or write cycle. At the same time, the MCF5407 negates TT[1:0], TM[2:0], $\overline{TIP}$, and SIZ[1:0] on the rising edge of CLKIN. Note that the rising edge of CLKIN may be the start of S0 for the next access cycle; in this case, $\overline{TIP}$ remains asserted and R/$\overline{W}$ may not transition, depending on the nature of the back-to-back cycles. |
| | Read | | The external device stops driving data between S4 and S5. |
| | Write | | The data bus returns to high impedance on the rising edge of CLKIN. The rising edge of CLKIN may be the start of S0 for the next access. |

**NOTE:**

An external device has at most two CLKIN cycles after the start of S4 to three-state the data bus after data is sampled in S3. This applies to basic read cycles, fast-termination cycles, and the last transfer of a burst.

## 18.4.3 Read Cycle

During a read cycle, the MCF5407 receives data from memory or from a peripheral device. Figure 18-5 is a read cycle flowchart.

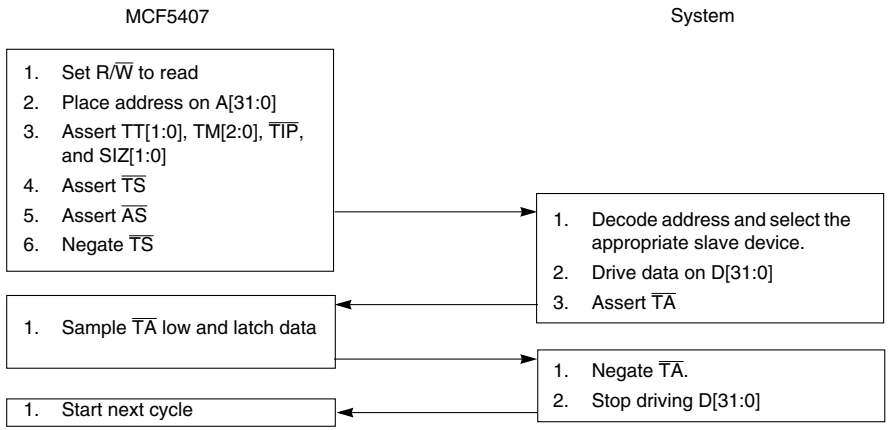



**Figure 18-5. Read Cycle Flowchart**

The read cycle timing diagram is shown in Figure 18-6.

**NOTE:**

In the following timing diagrams, $\overline{TA}$ waveforms apply for chip selects programmed to enable either internal or external

termination. $\overline{TA}$ assertion should look the same in either case.

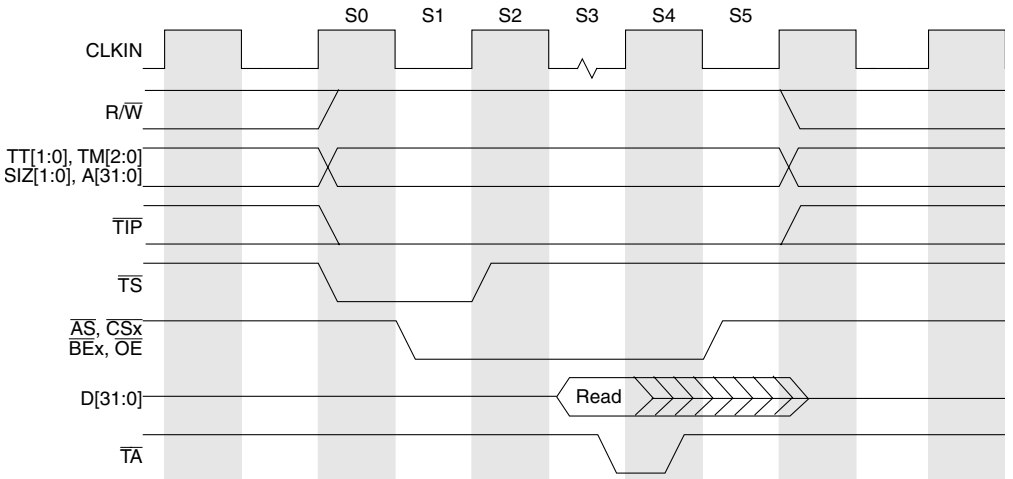



**Figure 18-6. Basic Read Bus Cycle**

Note the following characteristics of a basic read:

- In S3, data is made available by the external device on the falling edge of CLKIN and is sampled on the rising edge of CLKIN with $\overline{TA}$ asserted.
- In S4, the external device can stop driving data after the rising edge of CLKIN. However, data could be driven up to S5.
- For a read cycle, the external device stops driving data between S4 and S5.

States are described in Table 18-4.

## 18.4.4 Write Cycle

During a write cycle, the MCF5407 sends data to memory or to a peripheral device. The write cycle flowchart is shown in Figure 18-7.

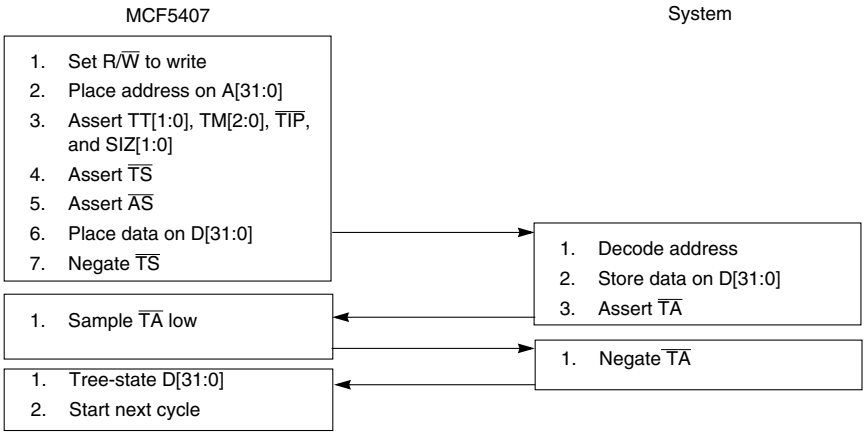



**Figure 18-7. Write Cycle Flowchart**

The write cycle timing diagram is shown in Figure 18-8.

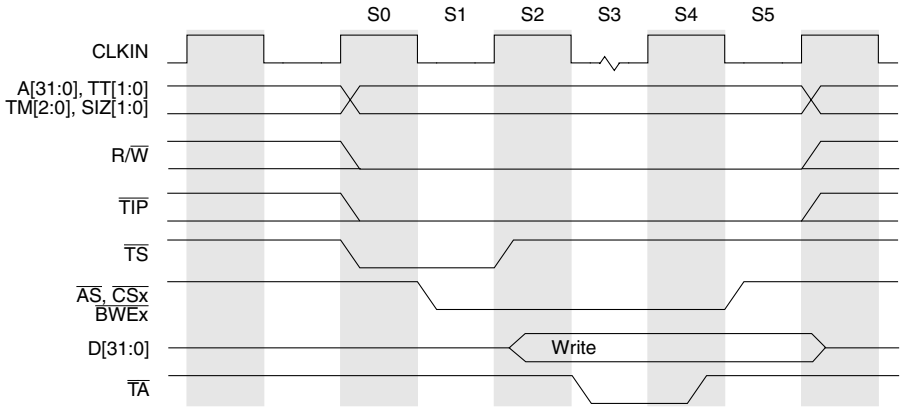



**Figure 18-8. Basic Write Bus Cycle**

Table 18-4 describes the six states of a basic write cycle.

## 18.4.5 Fast-Termination Cycles

Two clock-cycle transfers are supported on the MCF5407 bus. In most cases, this is impractical to use in a system because the termination must take place in the same half clock during which AS is asserted. Because this is atypical, it is not referred to as the zero-wait-state case but is called the fast-termination case. A fast-termination cycle is one in which an external device or memory asserts TA as soon as TS is detected. This means that the MCF5407 samples TA on the rising edge of the second cycle of the bus transfer. Figure 18-9 shows a read cycle with fast termination. Note that fast termination cannot be used with internal termination.

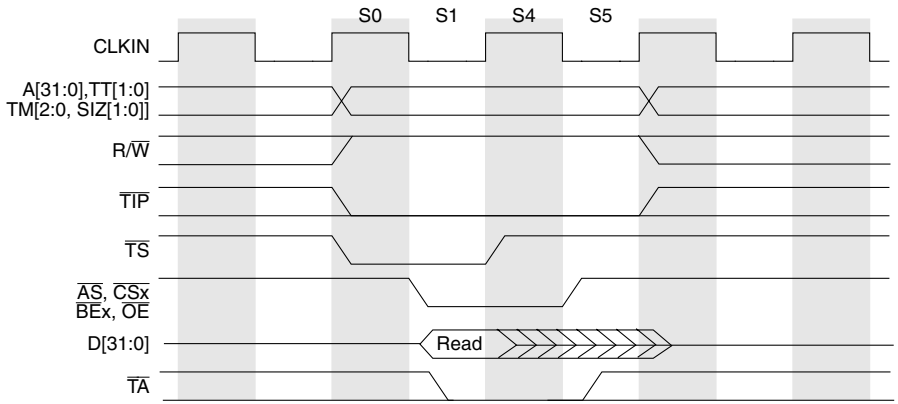



**Figure 18-9. Read Cycle with Fast Termination**

Figure 18-10 shows a write cycle with fast termination.

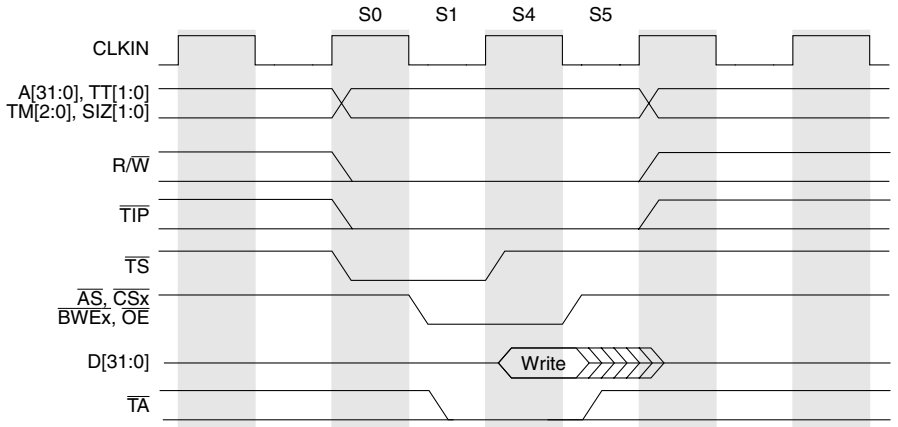



**Figure 18-10. Write Cycle with Fast Termination**

## 18.4.6 Back-to-Back Bus Cycles

The MCF5407 runs back-to-back bus cycles whenever possible. For example, when a longword read is started on a word-size bus, the processor performs two back-to-back word read accesses. Back-to-back accesses are distinguished by the continuous assertion of TIP throughout the cycle. Figure 18-11 shows a read back-to-back with a write.

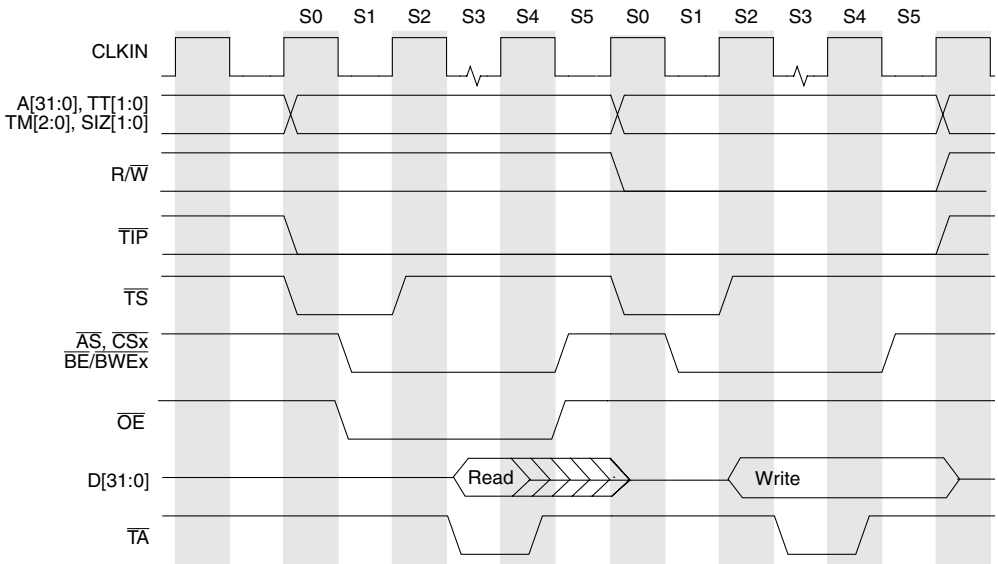



**Figure 18-11. Back-to-Back Bus Cycles**

Basic read and write cycles are used to show a back-to-back cycle, but there is no restriction as to the type of operations to be placed back to back. The initiation of a back-to-back cycle is not user definable.

## 18.4.7 Burst Cycles

The MCF5407 can be programmed to initiate burst cycles if its transfer size exceeds the size of the port it is transferring to. For example, with bursting enabled, a word transfer to an 8-bit port would take a 2-byte burst cycle for which SIZ[1:0] = 10 throughout. A line transfer to a 32-bit port would take a 4-longword burst cycle, for which SIZ[1:0] = 11 throughout.

The MCF5407 bus can support 2-1-1-1 burst cycles to maximize cache performance and optimize DMA transfers. A user can add wait states by delaying termination of the cycle. The initiation of a burst cycle is encoded on the size pins. For burst transfers to smaller port sizes, SIZ[1:0] indicates the size of the entire transfer. For example, if the MCF5407 writes a longword to an 8-bit port, SIZ[1:0] = 00 for the first byte transfer and does not change.

CSCRs are used to enable bursting for reads, writes, or both. MCF5407 memory space can be declared burst-inhibited for reads and writes by clearing the appropriate CSCR*x*[BSTR,BSTW]. A line access to a burst-inhibited region is broken into separate port-width accesses. Unlike a burst access, SIZ[1:0] = 11 only for the first port-width access; for the remaining accesses, SIZ[1:0] reflects the port width, with individual accesses separated by $\overline{AS}$ negations. The address changes if internal termination is used but does not change if external termination is used, as shown in Figure 18-12 and Figure 18-14.

### 18.4.7.1 Line Transfers

A line is a 16-byte-aligned, 16-byte value. Despite the alignment, a line access may not begin on the aligned address; therefore, the bus interface supports line transfers on multiple address boundaries. Table 18-5 shows allowable patterns for line accesses.

**Table 18-5. Allowable Line Access Patterns**

| A[3:2] | Longword Accesses |
|---|---|
| 00 | 0–4–8–C |
| 01 | 4–8–C–0 |
| 10 | 8–C–0–4 |
| 11 | C–0–4–8 |

### 18.4.7.2 Line Read Bus Cycles

Figure 18-12 shows line read with zero wait states. The access starts like a basic read bus cycle with the first data transfer sampled on the rising edge of S4, but the next pipelined burst data is sampled a cycle later on the rising edge of S6. Each subsequent pipelined data burst is single cycle until the last one, which can be held for up to 2 CLKIN cycles after $\overline{TA}$ is asserted. Note that $\overline{AS}$ and $\overline{CSx}$ are asserted throughout the burst transfer. This example shows the timing for external termination, which differs only from the internal termination example in Figure 18-13 in that the address lines change only at the beginning (assertion of $\overline{TS}$ and $\overline{TIP}$) and end (negation of $\overline{TIP}$) of the transfer.

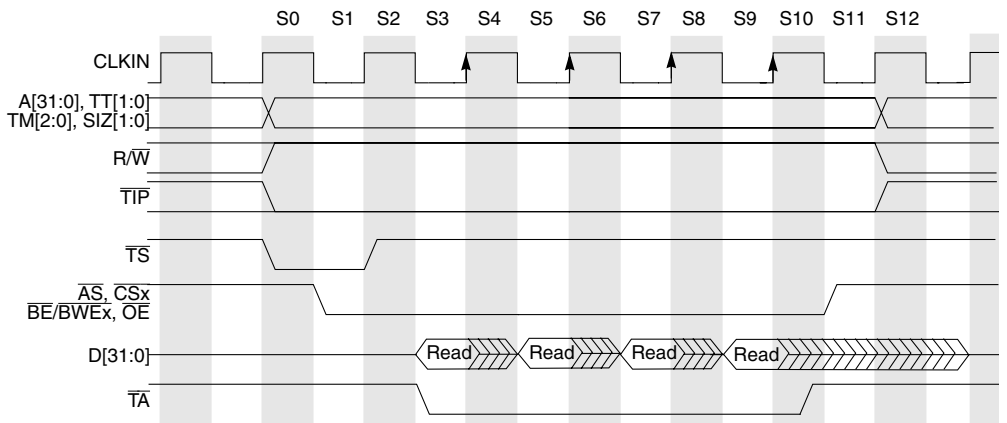



**Figure 18-12. Line Read Burst (2-1-1-1), External Termination**

Figure 18-13 shows timing when internal termination is used.

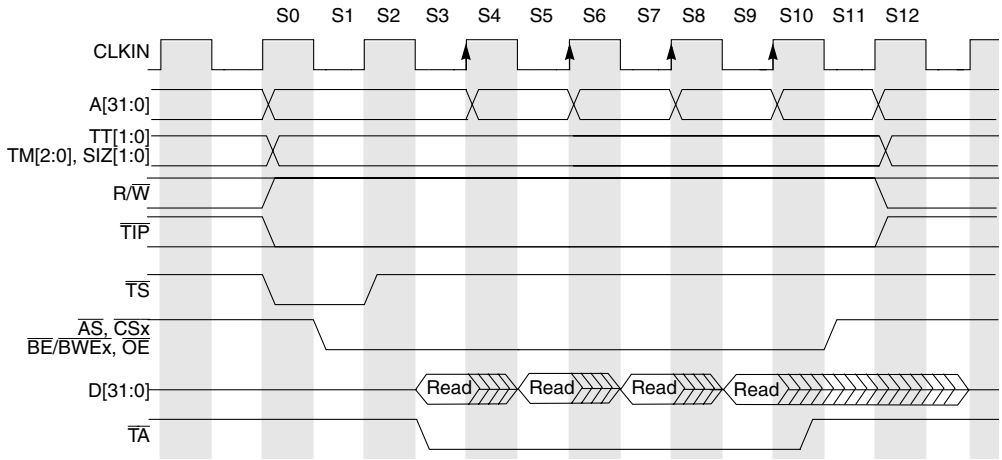



**Figure 18-13. Line Read Burst (2-1-1-1), Internal Termination**

Figure 18-14 shows a line access read with one wait state programmed in CSCR*x* to give the peripheral or memory more time to return read data. This figure follows the same execution as a zero-wait state read burst with the exception of an added wait state.

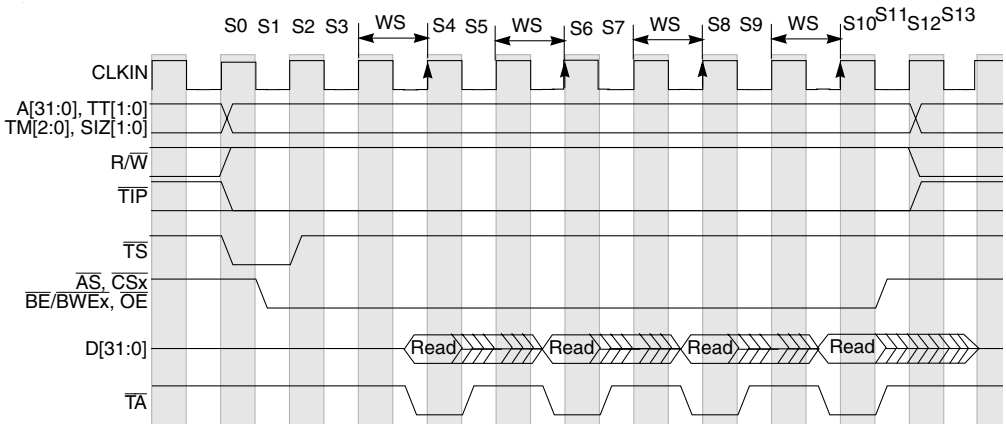



**Figure 18-14. Line Read Burst (3-2-2-2), External Termination**

Figure 18-15 shows a burst-inhibited line read access with fast termination. The external device executes a basic read cycle while determining that a line is being transferred. The external device uses fast termination for subsequent transfers.

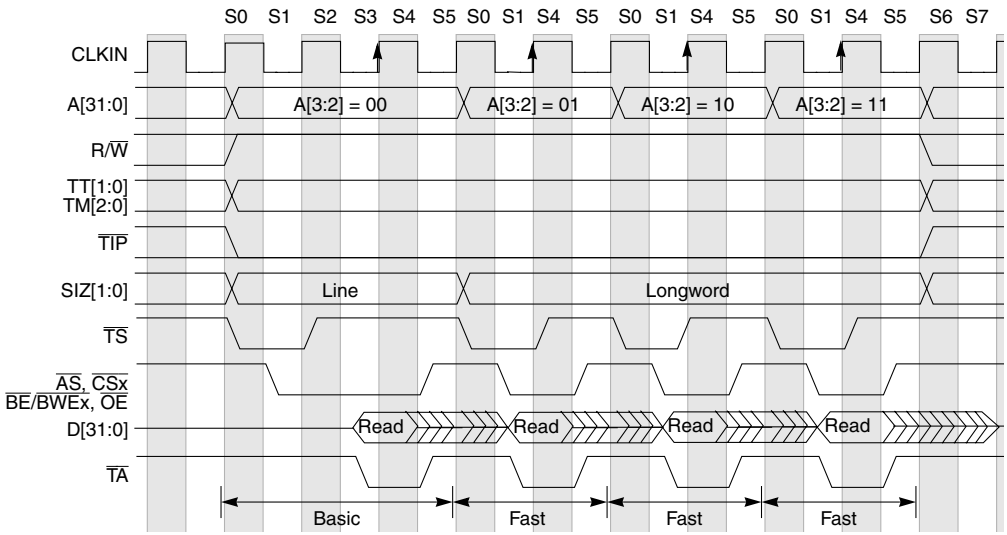



**Figure 18-15. Line Read Burst-Inhibited, Fast, External Termination**

### 18.4.7.3 Line Write Bus Cycles

Figure 18-16 shows a line access write with zero wait states. It begins like a basic write bus cycle with data driven one clock after $\overline{TS}$. The next pipelined burst data is driven a cycle after the write data is registered (on the rising edge of S6). Each subsequent burst takes a single cycle. Note that as with the line read example in Figure 18-12, $\overline{AS}$ and $\overline{CS}x$ remain asserted throughout the burst transfer. This example shows the behavior of the address lines for both internal and external termination. Note that with external termination, address lines, like SIZ, TT, and TM, hold the same value for the entire transfer.

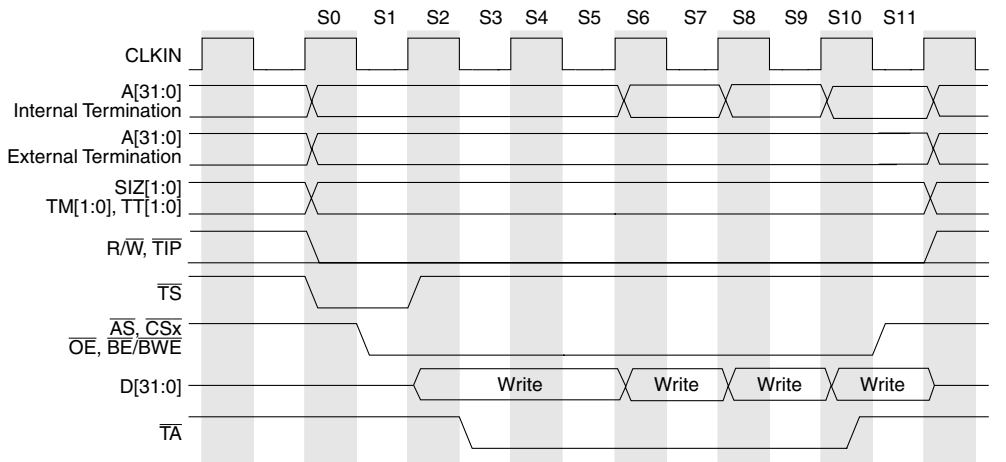



**Figure 18-16. Line Write Burst (2-1-1-1), Internal/External Termination**

Figure 18-17 shows a line burst write with one wait-state insertion.

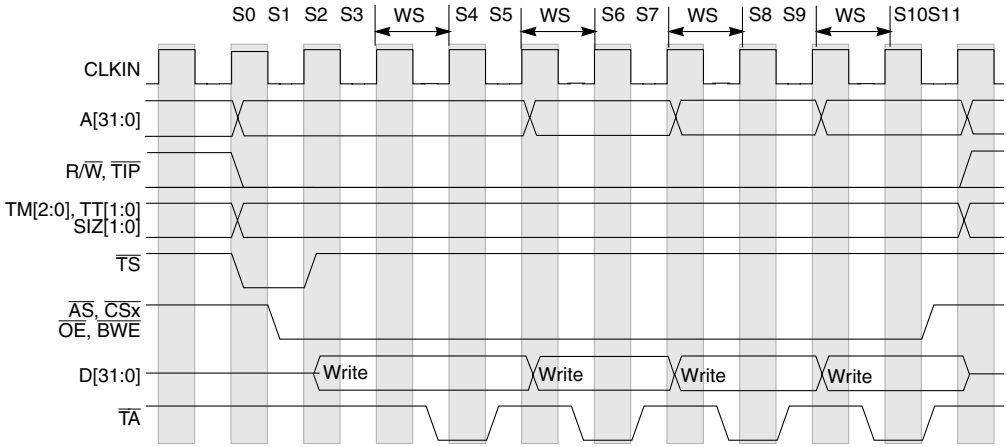



**Figure 18-17. Line Write Burst (3-2-2-2) with One Wait State, Internal Termination**

Figure 18-18 shows a burst-inhibited line write. The external device executes a basic write cycle while determining that a line is being transferred. The external device uses fast termination to end each subsequent transfer.

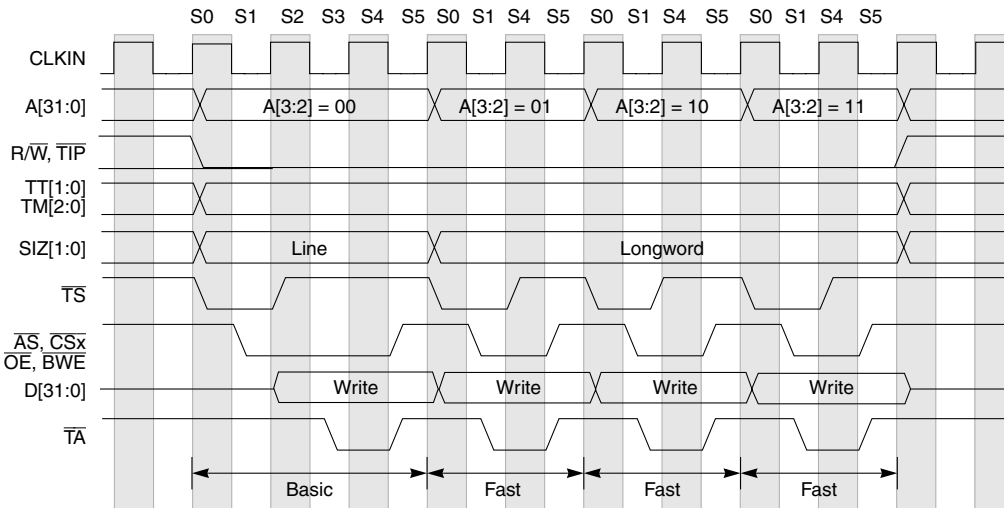



**Figure 18-18. Line Write Burst-Inhibited, Internal Termination**

### 18.4.7.4 Transfers Using Mixed Port Sizes

Figure 18-19 shows timing for a longword read from an 8-bit port using external termination. Figure 18-20 shows the same transfer with internal termination. For both, SIZ[1:0] change only at the start of a new transfer because this burst is implemented as one

transfer.

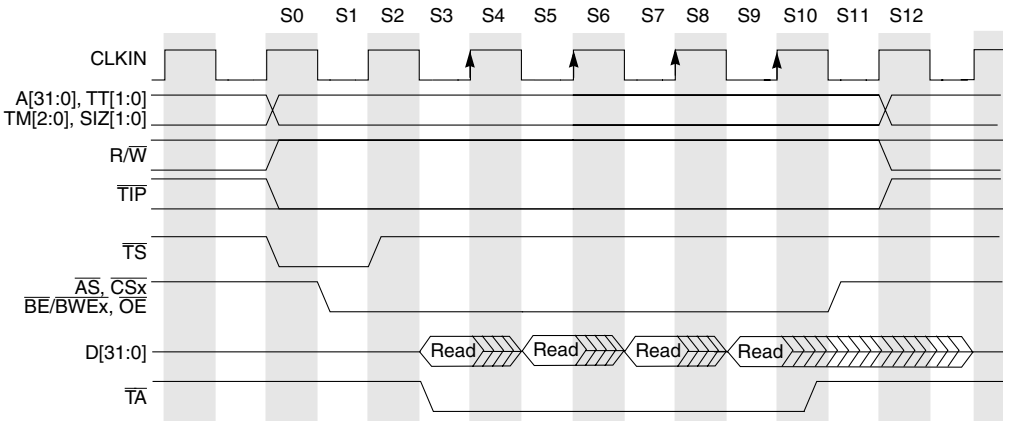



**Figure 18-19. Longword Read from an 8-Bit Port, External Termination**

Note that with external termination, address signals do not change. With internal termination, Figure 18-20, A[1:0] increment for the same longword transfer.

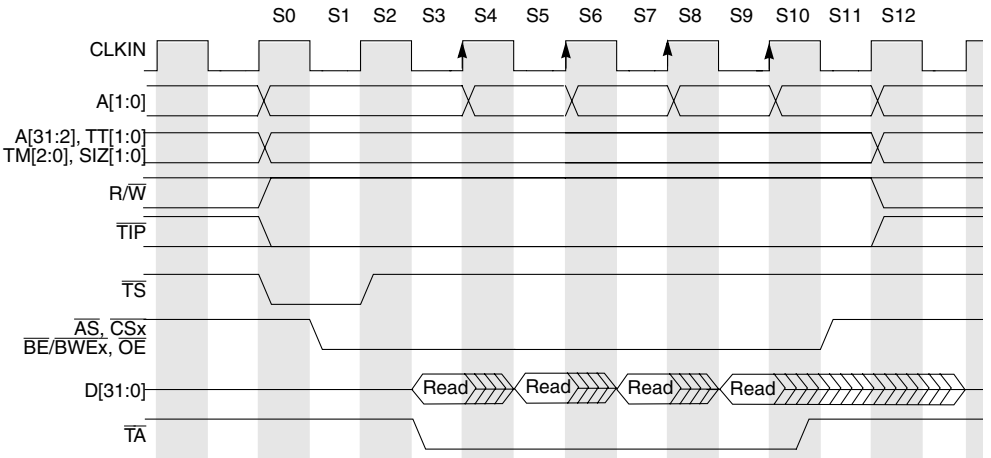



**Figure 18-20. Longword Read from an 8-Bit Port, Internal Termination**

## 18.5 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 MCF5407 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 MCF5407 converts misaligned, cache-inhibited operand accesses to multiple aligned accesses. Figure 18-21 shows the transfer of a longword operand from a byte address to a 32-bit port. In this example, SIZ[1:0] specify a byte transfer and a byte offset of 0x1. The slave device supplies the byte and acknowledges the data transfer. When the MCF5407 starts the second cycle, SIZ[1:0] specify a word transfer 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 18-21. 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 18-22 differs from the one in Figure 18-21 in that 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 18-22. Example of a Misaligned Word Transfer (32-Bit Port)**

**NOTE:**

External masters using internal MCF5407 chip selects and default memory control signals must initiate aligned transfers.

## 18.6 Bus Errors

The MCF5407 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 MCF5407 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.

## 18.7 Interrupt Exceptions

A peripheral device uses the interrupt-request signals ($\overline{IRQx}$) to signal the core to take an interrupt exception when it needs the MCF5407 or is ready to send information to it. The interrupt transfers control to an appropriate routine.

The MCF5407 has the following two levels of interrupt masking:

- Interrupt mask registers in the SIM compare interrupt inputs with programmable interrupt mask levels. The SIM outputs only unmasked interrupts.
- The status register uses a 3-bit interrupt priority mask. The core recognizes only interrupt requests of higher priority than the value in the mask. See Section 2.2.2.1, “Status Register (SR).”

**NOTE:**

To mask a level 1–6 interrupt source, write a higher-level SR interrupt mask before setting IMR. Then restore the mask to its previous value. Do not mask a level 7 interrupt source.

The MCF5407 continuously samples and synchronizes external interrupt inputs. An interrupt request must be held for at least two consecutive CLKIN periods to be considered valid. To guarantee that the interrupt is recognized, the request level must be maintained until the MCF5407 acknowledges the interrupt with an interrupt-acknowledge cycle.

**NOTE:**

Interrupt levels 1–7 are level-sensitive. Level 7 is also edge-triggered. See Section 18.7.1, “Level 7 Interrupts.”

The MCF5407 takes an interrupt exception for a pending interrupt within one instruction boundary after processing any higher-priority pending exception. Thus, the MCF5407 executes at least one instruction in an interrupt exception handler before recognizing another interrupt request.

If autovector generation is used for internal interrupts (ICR*n*[AVEC] = 1), the interrupt acknowledge vector is generated internally and no interrupt acknowledge cycle is generated on the external bus.

If autovector generation is used for external interrupts, no interrupt acknowledge cycle is shown on the external bus ($\overline{AS}$ is not asserted) unless AVR[BLK] is 0. Consequently, the external interrupt must be cleared in the interrupt service routine. See Section 9.2.2, “Autovector Register (AVR).”

## 18.7.1 Level 7 Interrupts

Level 7 interrupts are nonmaskable and are handled differently than other interrupts. Level 7 interrupts are edge triggered by a transition from a lower priority request to the level 7 request. Interrupts at all other levels are level sensitive. Therefore, if $\overline{IRQ7}$ remains asserted, the MCF5407 recognizes only one level 7 interrupt because only one transition from a lower level request to a level 7evel 7 request occurred. For the processor to recognize two consecutive level 7 interrupts, one of the following must occur:

- The interrupt request on the interrupt control pins is raised to level 7 and stays there until an interrupt-acknowledge cycle begins. The level later drops but then returns to level 7, causing a second transition on the interrupt control lines.
- The interrupt request on the interrupt control pins is raised to level 7 and stays there. If the level 7 interrupt routine lowers the mask level, a second level 7 interrupt is recognized without a transition of the interrupt control pins. After the level 7 routine completes, the MCF5407 compares the mask level to the request level on the $\overline{\text{IRQ}}x$ signals. Because the mask level is lower than the requested level, the interrupt mask is set back to level 7. To ensure it is recognized, the level 7 request on $\overline{\text{IRQ7}}$ must be held until the second interrupt-acknowledge bus cycle begins.

## 18.7.2 Interrupt-Acknowledge Cycle

When the MCF5407 processes an interrupt exception, it performs an interrupt-acknowledge bus cycle to obtain the vector number that contains the starting location of the interrupt exception handler. The interrupt-acknowledge bus cycle is a read transfer that differs from normal read cycles in the following respects:

- TT[1:0] = 0x3 to indicate a CPU space or acknowledge bus cycle.
- TM[2:0] = the level of interrupt being acknowledged.
- A[31:5] = 0x7F_FFFF.
- A[4:2] = the interrupt request level being acknowledged (same as TM[2:0]).
- A[1:0] = 00.

During the interrupt-acknowledge bus cycle (a read cycle), the responding device places the vector number on D[31:24] and the cycle is terminated normally with $\overline{\text{TA}}$. Figure 18-23 is a flow diagram for an interrupt-acknowledge cycle terminated with $\overline{\text{TA}}$.

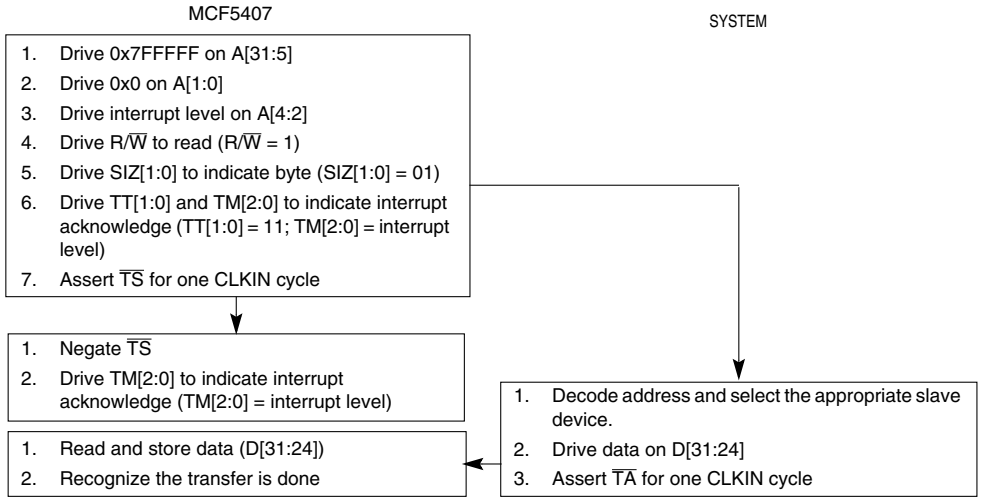



**Figure 18-23. Interrupt-Acknowledge Cycle Flowchart**

## 18.8 Bus Arbitration

The MCF5407 bus protocol gives either the MCF5407 or an external device access to the external bus. If more than one external device uses the bus, an external arbiter can prioritize requests and determine which device is bus master. When the MCF5407 is bus master, it uses the bus to fetch instructions and transfer data to and from external memory. When an external device is bus master, the MCF5407 can monitor the external master's transfers and interact through its chip-select, DRAM control, and transfer termination signals. See Section 10.4.1.3, "Chip-Select Control Registers (CSCR0–CSCR7)," and Chapter 11, "Synchronous/Asynchronous DRAM Controller Module."

Two-wire bus arbitration is used where the MCF5407 shares the bus with a single external device. This mode uses $\overline{BG}$ and $\overline{BD}$. The external device can ignore $\overline{BR}$. Three-wire mode is used where the MCF5407 shares the bus with multiple external devices. This requires an external bus arbiter and uses $\overline{BG}$, $\overline{BD}$, and $\overline{BR}$. In either mode, the MCF5407 bus arbiter operates synchronously and transitions between states on the rising edge of CLKIN.

Table 18-6 shows the four arbitration states the MCF5407 can be in during bus operation.

**Table 18-6. MCF5407 Arbitration Protocol States**

| State | Master | Bus | $\overline{BD}$ | Description |
|---|---|---|---|---|
| Reset | None | Not driven | Negated | The MCF5407 enters reset state from any other state when $\overline{RSTI}$ or software watchdog reset is asserted. If both are negated, the MCF5407 enters implicit or external device mastership state, depending on $\overline{BG}$. |
| Implicit master | MCF5407 | Not driven | Negated | The MCF5407 is bus master ($\overline{BG}$ input is asserted) but is not ready to begin a bus cycle. It continues to three-state the bus until an internal bus request. |

**Table 18-6. MCF5407 Arbitration Protocol States (Continued)**

| State | Master | Bus | $\overline{BD}$ | Description |
|---|---|---|---|---|
| Explicit master | MCF5407 | Driven | Asserted | The MCF5407 is explicit bus master when $\overline{BG}$ is asserted and at least one bus cycle has been initiated. It asserts $\overline{BD}$ and retains explicit mastership until $\overline{BG}$ is negated even if no active bus cycles are executed. It releases the bus at the end of the current bus cycle, then negates $\overline{BD}$ and three-states the bus signals. |
| External master | External | Not driven | Negated | An external device is bus master ($\overline{BG}$ negated to MCF5407). The MCF5407 can assert $\overline{OE}$, $\overline{CS}$[7:0], $\overline{BE}$/$\overline{BWE}$[3:0], $\overline{TA}$, and all DRAM controller signals ($\overline{RAS}$[1:0], $\overline{CAS}$[3:0], $\overline{SRAS}$, $\overline{SCAS}$, $\overline{DRAMW}$, SCKE). |

If the MCF5407 is the only possible master, $\overline{BG}$ can be tied to GND—no arbiter is needed.

## 18.8.1 Bus Arbitration Signals

Bus arbitration signal timings in Table 18-7 are referenced to the system clock, which is not considered a bus signal. Clock routing is expected to meet application requirements.

**Table 18-7. ColdFire Bus Arbitration Signal Summary**

| Signal | I/O | Description |
|---|---|---|
| $\overline{BR}$ | O | Bus request. Indicates to an external arbiter that the processor needs to become bus master. $\overline{BR}$ is negated when the MCF5407 begins an access to the external bus with no other internal accesses pending. $\overline{BR}$ remains negated until another internal request occurs. |
| $\overline{BG}$ | I | Bus grant. An external arbiter asserts $\overline{BG}$ to indicate that the MCF5407 can control the bus at the next rising edge of CLKIN. When the arbiter negates $\overline{BG}$, the MCF5407 must release the bus as soon as the current transfer completes. The external arbiter must not grant the bus to any other device until both $\overline{BD}$ and $\overline{BG}$ are negated. |
| $\overline{BD}$ | O | Bus driven. The MCF5407 asserts $\overline{BD}$ to indicate it is current master and is driving the bus. If it loses bus mastership during a transfer, it completes the last transfer of the current access, negates $\overline{BD}$, and three-states all bus signals on the rising edge of CLKIN. If it loses mastership during an idle clock cycle, it three-states all bus signals on the rising edge of CLKIN. |

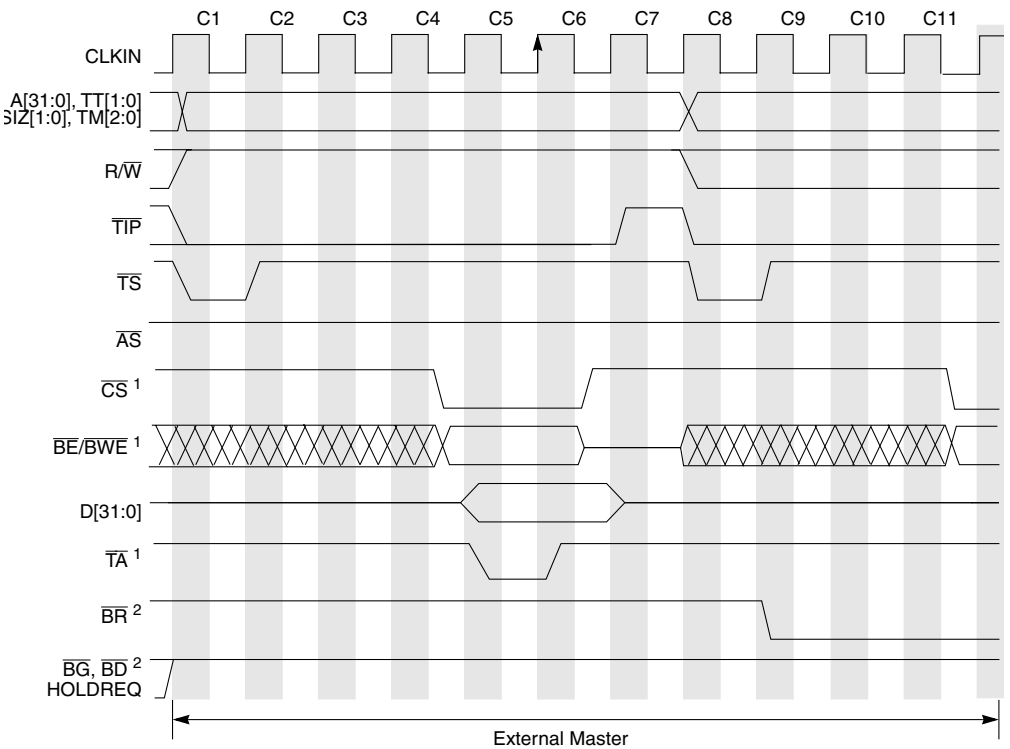
# 18.9 General Operation of External Master Transfers

An external master asserts its hold signal (such as HOLDREQ) when it executes a bus cycle, driving $\overline{BG}$ high and forcing the MCF5407 to hold all bus requests. During an external master cycle, the MCF5407 can provide memory control signals ($\overline{OE}$, $\overline{CS}$[7:0], $\overline{BE}$/$\overline{BWE}$[3:0], $\overline{RAS}$[1:0], $\overline{CAS}$[3:0]) and $\overline{TA}$ while the external master drives the address and data bus and other required bus control signals. When the external master asserts $\overline{TS}$ or $\overline{AS}$ to the MCF5407, the beginning of a bus cycle is identified and the MCF5407 starts decoding the address driven.

Note the following regarding external master accesses:

- For the MCF5407 to assert a $\overline{CSx}$ during external master accesses, CSMR*n*[AM] must be set. External master hits use the corresponding CSCR*n* settings for auto-acknowledge, byte enables, and wait states. See Section 10.4.1.3, “Chip-Select Control Registers (CSCR0–CSCR7).”
- To enable DRAM control signals during external master accesses, DCMR*n*[AM] must be set.
- During external master bus cycles, either $\overline{TS}$ or $\overline{AS}$ (but not both) should be driven to the MCF5407. Driving both during a bus cycle causes indeterminate results.

External master transfers that use the MCF5407 to drive memory control signals and $\overline{TA}$ are like normal MCF5407 transfers. Figure 18-24 shows timing for basic back-to-back bus cycles during an external master transfer.




1 Depending on programming, these signals may or may not be driven by the processor.
2 This signal is driven by the processor for an external master transfer.

**Figure 18-24. Basic No-Wait-State External Master Access**

R/$\overline{W}$ is asserted high for reads and low for writes; otherwise, the transfers are the same. In Figure 18-24, the MCF5407 chip select’s internal transfer acknowledge is enabled and the MCF5407 drives $\overline{TA}$ as an output after a programmed number of wait states.

**NOTE:**

Bus timing diagrams for external master transfers are not valid for on-chip internal four-channel DMA accesses on the MCF5407.

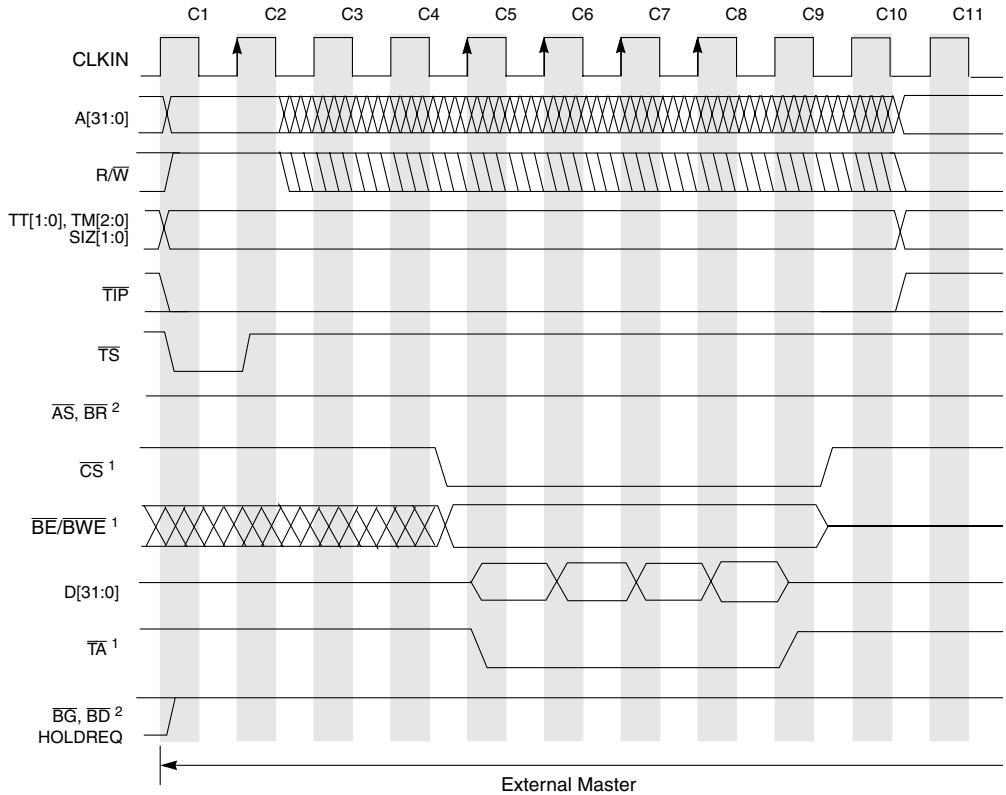
Timing diagrams describe transactions in general terms of bus cycles (C*n*) rather than the states (S*n*) used in the bus diagrams.

Table 18-8 defines the cycles for Figure 18-24.

**Table 18-8. Cycles for Basic No-Wait-State External Master Access**

| Cycle | Definition |
|---|---|
| C1 | The external master asserts HOLDREQ, signaling the MCF5407 to hold bus requests. $\overline{BD}$ should not be asserted. The external master drives address, $\overline{TS}$, R/$\overline{W}$, TT[1:0], TM[2:0], $\overline{TIP}$, and SIZ[1:0] as MCF5407 inputs. |
| C2–C3 | The MCF5407 decodes the external master's address and control signals to identify the proper chip select and byte enable assertion. The external master negates $\overline{TS}$ in C2. |
| C4 | On the falling edge of CLKIN, the MCF5407 asserts the appropriate chip select for the external master access along with the appropriate byte enables. |
| C5 | On the rising edge of CLKIN, data is driven onto the bus by the device selected by $\overline{CS}$. On the rising edge, the MCF5407 asserts $\overline{TA}$ to indicate the cycle is complete. |
| C6 | $\overline{TA}$ negates on the rising edge of CLKIN. On the falling edge, the MCF5407 negates the chip select and byte enables and the next cycle can begin. |
| C7 | The external master negates $\overline{TIP}$ on the rising edge of CLKIN. |
| C8 | The external device retains bus mastership and drives the address bus, $\overline{TS}$, R/$\overline{W}$, TT[1:0], TM[2:0], $\overline{TIP}$, and SIZ[1:0] as inputs to the MCF5407. |
| C9 | The MCF5407 decodes the external master's address and control signals to identify the proper chip select and byte enable assertion. The external master negates $\overline{TS}$. The MCF5407 asserts $\overline{BR}$ on the rising edge of CLKIN, signalling that it wants to arbitrate for the bus when the current cycle completes. |
| C10 | The MCF5407 continues to decode the external device's address and control signals to identify the proper chip select and byte enable assertion. |
| C11 | On the falling edge of CLKIN, the MCF5407 asserts the appropriate chip select for the external master access along with the appropriate byte enables. |

Figure 18-25 shows a burst line access for an external master transfer with the chip select set to no-wait states and with internal transfer-acknowledge assertion enabled.




1 Depending on programming, these signals may or may not be driven by the processor.
2 These signals are driven by the processor for an external master transfer.

**Figure 18-25. External Master Burst Line Access to 32-Bit Port**

Table 18-9 defines the cycles for Figure 18-25.

**Table 18-9. Cycles for External Master Burst Line Access to 32-Bit Port**

| Cycle | Definition |
|---|---|
| C1 | The external device is bus master and asserts HOLDREQ, indicating to the MCF5407 to hold all bus requests. In other words, BD should not be asserted. The external master drives address, TS, R/W, TT[1:0], TM[2:0], TIP, and SIZ[1:0] as inputs to the MCF5407. SIZ[1:0] inputs indicate a line transfer. The MCF5407 is not asserting BR. |
| C2–C3 | The MCF5407 decodes the external device's address and control signals to identify the proper chip-select and byte-enable assertion. The external device negates TS in C2. Address and R/W are latched in the MCF5407 on the rising edge of CLKIN in C2. After C2, the address and R/W are ignored for the rest of the burst transfer. |
| C4 | On the falling edge of CLKIN, the MCF5407 asserts the appropriate chip select for the external device access along with the appropriate byte enables. |
| C5 | On the rising edge of CLKIN, data is driven onto the bus by the device selected by CS. The MCF5407 asserts TA on the rising edge of CLKIN, indicating the first data transfer is complete. |

**Table 18-9. Cycles for External Master Burst Line Access to 32-Bit Port (Continued)**

| Cycle | Definition |
|---|---|
| C6–C8 | No-wait state data transfers 2–4 occur on the rising edges of CLKIN. $\overline{TA}$ continues to be asserted indicating completion of each transfer. $\overline{TIP}$, $\overline{CSx}$, and $\overline{BE}/\overline{BWE}$[3:0] are driven. |
| C9 | $\overline{TA}$ negates on the rising edge of CLKIN along with external device's negation of $\overline{TIP}$. On the falling edge, the MCF5407 negates chip select and byte enables, creating an opportunity for another cycle to begin. |

## 18.9.1 Two-Device Bus Arbitration Protocol (Two-Wire Mode)

Two-wire mode bus arbitration lets the MCF5407 share the external bus with a single external bus device without requiring an external bus arbiter. Figure 18-26 shows the MCF5407 connecting to an external device using the two-wire mode. The MCF5407 $\overline{BG}$ input is connected to the HOLDREQ output of the external device; the MCF5407 $\overline{BD}$ output is connected to the HOLDACK input of the external device. Because the external device controls the state of HOLDREQ, it controls when the MCF5407 is granted the bus, giving the MCF5407 lower priority.

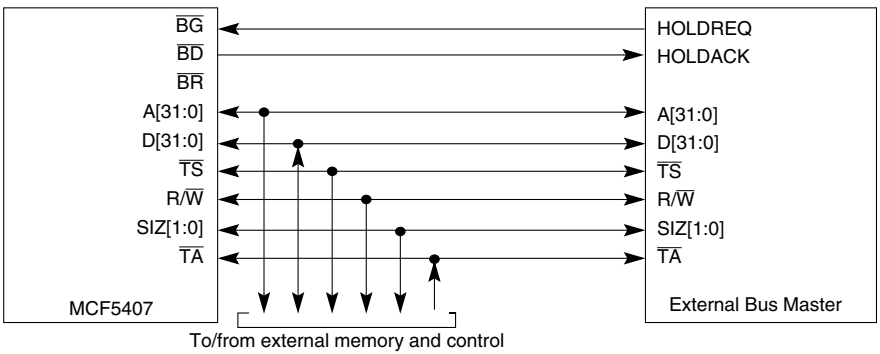



**Figure 18-26. MCF5407 Two-Wire Mode Bus Arbitration Interface**

When the external device is not using the bus, it negates HOLDREQ, driving $\overline{BG}$ low and granting the bus to the MCF5407. When the MCF5407 has an internal bus request pending and $\overline{BG}$ is low, the MCF5407 drives $\overline{BD}$ low, negating HOLDACK to the external device. When the external bus device needs the external bus, it asserts HOLDREQ, driving $\overline{BG}$ high, requesting the MCF5407 to release the bus. If $\overline{BG}$ is negated while a bus cycle is in progress, the MCF5407 releases the bus at the completion of the bus cycle. Note that the MCF5407 considers the individual transfers of a burst or burst-inhibited access to be a single bus cycle and does not release the bus until the last transfer of the series completes.

When the bus has been granted to the MCF5407, one of two situations can occur. In the first case, if the MCF5407 has an internal bus request pending, the MCF5407 asserts $\overline{BD}$ to indicate explicit bus mastership and begins the pending bus cycle by asserting $\overline{TS}$. As

shown in Figure 18-25, the MCF5407 continues to assert $\overline{BD}$ until the completion of the bus cycle. If $\overline{BG}$ is negated by the end of the bus cycle, the MCF5407 negates $\overline{BD}$. While $\overline{BG}$ is asserted, $\overline{BD}$ remains asserted to indicate the MCF5407 is master, and it continuously drives the address bus, attributes, and control signals.

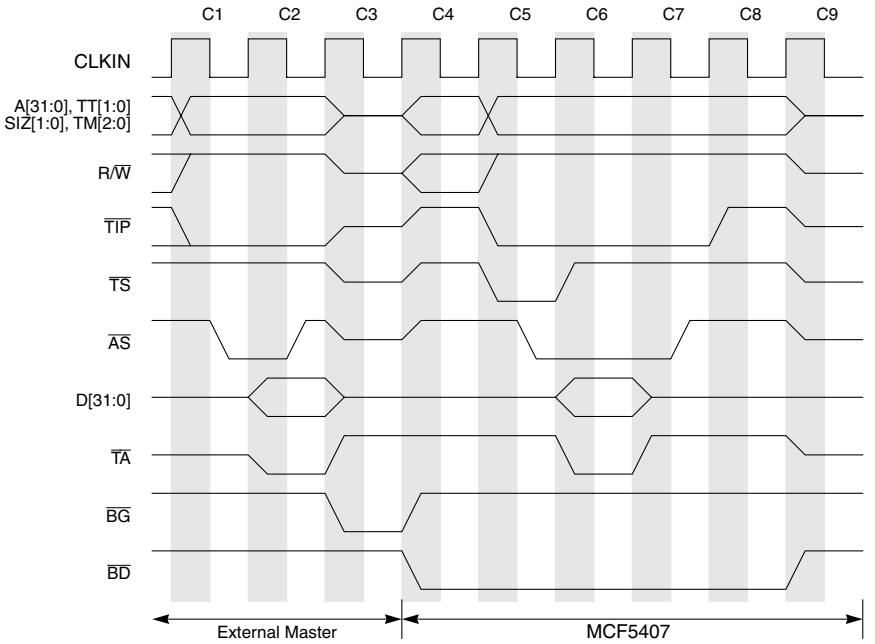



**Figure 18-27. Two-Wire Bus Arbitration with Bus Request Asserted**

In the second situation, the bus is granted to the MCF5407, but it does not have an internal bus request pending, so it takes implicit bus mastership. The MCF5407 does not drive the bus and does not assert $\overline{BD}$ if the bus has an implicit master. If an internal bus request is generated, the MCF5407 assumes explicit bus mastership. If explicit mastership was assumed because an internal request was generated, the MCF5407 immediately begins an access and asserts $\overline{BD}$.

In Figure 18-28, the external device is bus master during C1 and C2. During C3 the external device releases control of the bus by asserting $\overline{BG}$ to the MCF5407. At this point, there is an internal access pending so the MCF5407 asserts $\overline{BD}$ during C4 and begins the access. Thus, the MCF5407 becomes the explicit external bus master. Also during C4, the external device removes the grant from the MCF5407 by negating $\overline{BG}$. As the current bus master, the MCF5407 continues to assert $\overline{BD}$ until the current transfer completes. Because $\overline{BG}$ is negated, the MCF5407 negates $\overline{BD}$ during C9 and three-states the external bus, thereby returning external bus mastership to the external device.

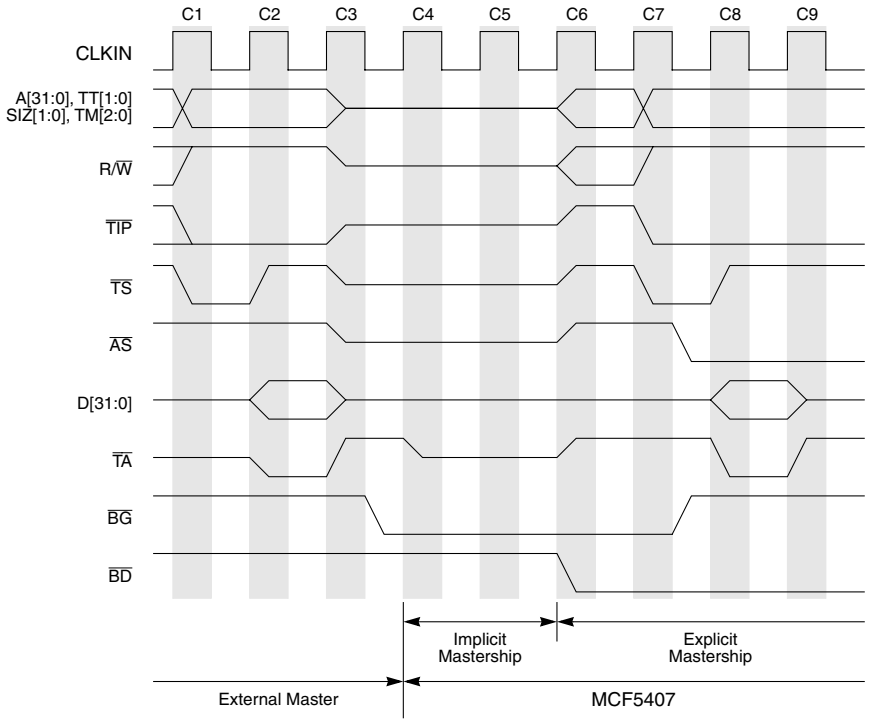



**Figure 18-28. Two-Wire Implicit and Explicit Bus Mastership**

In Figure 18-28, the external device is master during C1 and C2. It releases bus control in C3 by asserting $\overline{BG}$ to the MCF5407. During C4 and C5, the MCF5407 is implicit master because no internal access is pending. In C5, an internal bus request becomes pending, causing the MCF5407 to become explicit bus master in C6 by asserting $\overline{BD}$. In C7, the external device removes the bus grant to the MCF5407. The MCF5407 does not release the bus (the MCF5407 continues to assert $\overline{BD}$) until the transfer ends.

**NOTE:**

The MCF5407 can start a transfer in the clock cycle after $\overline{BG}$ is asserted. The external master must not assert $\overline{BG}$ to the MCF5407 while driving the bus or the part may be damaged.

Figure 18-29 is a MCF5407 bus arbitration state diagram. States are described in Table 18-6.

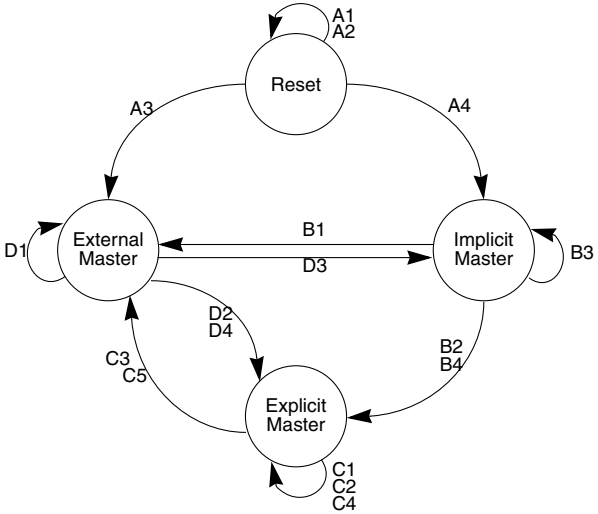



**Figure 18-29. MCF5407 Two-Wire Bus Arbitration Protocol State Diagram**

Table 18-10 describes the two-wire bus arbitration protocol transition conditions.

**Table 18-10. MCF5407 Two-Wire Bus Arbitration Protocol Transition Conditions**

| Present State | Condition Label | RSTI | Software Watchdog Reset | BG | Bus Request | Transfer in Progress | End of Cycle[1] | Next State |
|---|---|---|---|---|---|---|---|---|
| Reset | A1 | A[2] | — | — | — | — | — | Reset |
| | A2 | N[3] | A | — | — | — | — | Reset |
| | A3 | N | N | N | — | — | — | EM [4] |
| | A4 | N | N | A | — | — | — | Implicit mas |
| Implicit Master | B1 | N | N | N | — | — | — | EM |
| | B2 | N | N | A | — | — | — | Explicit mas |
| | B3 | N | N | A | N | — | — | Implicit mas |
| | B4 | N | N | A | A | — | — | Explicit mas |
| Explicit Master | C1 | N | N | A | — | — | — | Explicit mas |
| | C2 | N | N | N | — | — | — | Explicit mas |
| | C3 | N | N | N | — | N | — | EM |
| | C4 | N | N | N | — | A | N | Explicit mas |
| | C5 | N | N | N | — | A | A | EM |
| External Master | D1 | N | N | N | — | — | — | EM mas |
| | D2 | N | N | A | — | — | _ | Explicit mas |
| | D3 | N | N | A | N | — | — | Implicit mas |
| | D4 | N | N | A | A | — | — | Explicit mas |

1 Both normal terminations and terminations due to bus errors generate an end of cycle. Bus cycles resulting from a burst-inhibited transfer are considered part of that original transfer.
2 A means asserted.
3 N means negated.
4 EM means external master.

## 18.9.2 Multiple External Bus Device Arbitration Protocol (Three-Wire Mode)

Three-wire mode lets the MCF5407 share the external bus with multiple external devices. This mode requires an external arbiter to assign priorities to each potential master and to determine which device accesses the external bus. The arbiter uses the MCF5407 bus arbitration signals, $\overline{BR}$, $\overline{BD}$, and $\overline{BG}$, to control use of the external bus by the MCF5407.

The MCF5407 requests the bus from the external bus arbiter by asserting $\overline{BR}$ when the core requests an access. It continues to assert $\overline{BR}$ until after the transfer starts. It can negate $\overline{BR}$ at any time regardless of the $\overline{BG}$ status. If the MCF5407 is granted the bus when an internal bus request is generated, it asserts $\overline{BD}$ and the access begins immediately. The MCF5407 always drives $\overline{BR}$ and $\overline{BD}$, which cannot be directly wire-ORed with other devices.

The external arbiter asserts $\overline{BG}$ to grant the bus to MCF5407, which can begin a bus cycle after the next rising edge of CLKIN. If $\overline{BG}$ is negated during a bus cycle, the MCF5407 releases the bus when the cycle completes. To guarantee that the bus is released, $\overline{BG}$ must be negated before the rising edge of the CLKIN in which the last $\overline{TA}$ is asserted. Note that the MCF5407 treats any series of burst or a burst-inhibited transfers as a single bus cycle and does not release the bus until the last transfer of the series completes.

When the MCF5407 is granted the bus after it asserts $\overline{BR}$, one of two things can occur. If the MCF5407 has an internal bus request pending, it asserts $\overline{BD}$, indicating explicit bus mastership, and begins the pending bus cycle by asserting $\overline{TS}$. The MCF5407 continues to assert $\overline{BD}$ until the external bus arbiter negates $\overline{BG}$, after which $\overline{BD}$ is negated at the completion of the bus cycle. As long as $\overline{BG}$ is asserted, $\overline{BD}$ remains asserted to indicate that the MCF5407 is bus master, and the MCF5407 continuously drives the address bus, attributes, and control signals.

If no internal request is pending, the MCF5407 takes implicit bus mastership. It does not drive the bus and does not assert $\overline{BD}$ if the bus has an implicit master. If an internal bus request is generated, the MCF5407 assumes explicit bus mastership and immediately begins an access and asserts $\overline{BD}$. Figure 18-30 shows implicit and explicit bus mastership due to generation of an internal bus request.

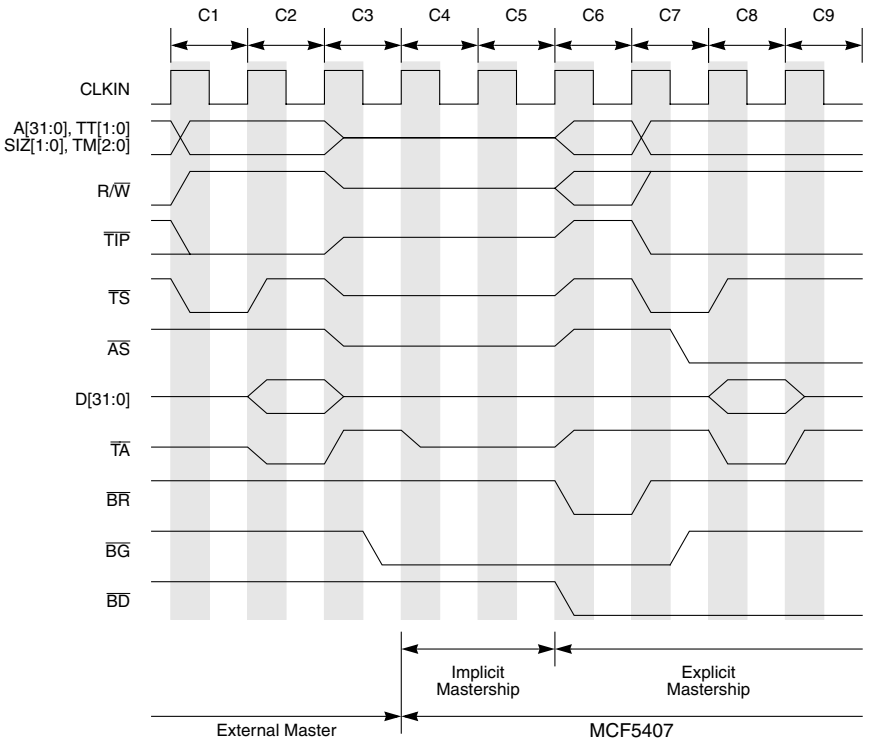



**Figure 18-30. Three-Wire Implicit and Explicit Bus Mastership**

In Figure 18-30, the external device is bus master during C1 and C2, releasing control in C3, at which time the external arbiter asserts $\overline{BG}$ to the MCF5407. During C4 and C5, the MCF5407 is implicit master because no internal access is pending. In C5, an internal bus request becomes pending, causing the MCF5407 to take explicit bus mastership in C6 by asserting $\overline{BR}$ and $\overline{BD}$. In C7, the external device removes the bus grant to the MCF5407. The MCF5407 does not release the bus (the MCF5407 asserts $\overline{BD}$) until the transfer ends.

**NOTE:**

The MCF5407 can start a transfer in the CLKIN cycle after $\overline{BG}$ is asserted. The external arbiter should not assert $\overline{BG}$ to the MCF5407 until the previous external master stops driving the bus. Asserting $\overline{BG}$ during another external master's transfer may damage the part.

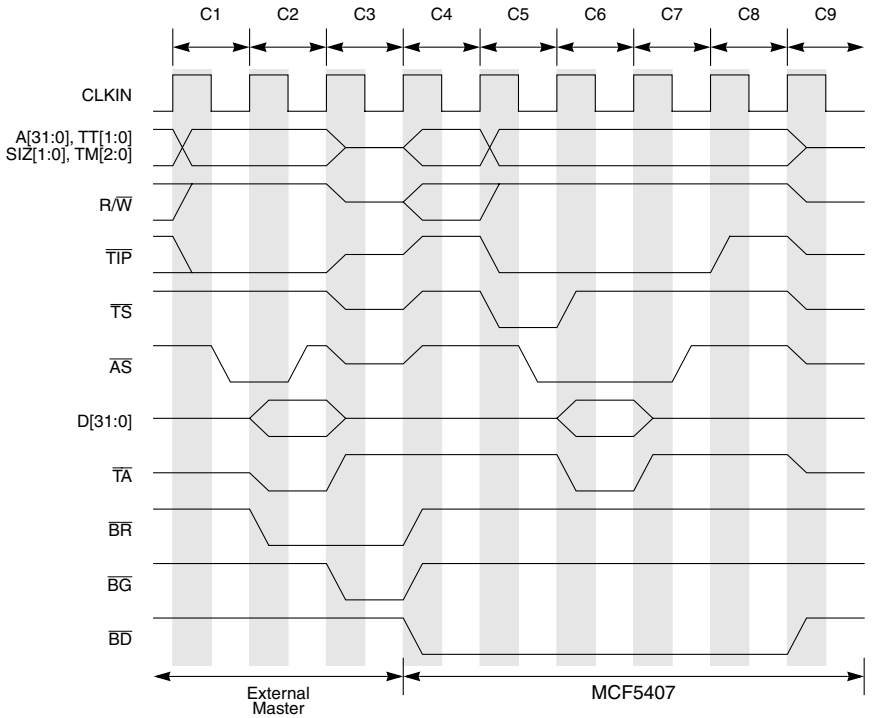



**Figure 18-31. Three-Wire Bus Arbitration**

In Figure 18-31, the external device is bus master during C1 and C2. During C2, the MCF5407 requests the external bus because of a pending internal transfer. On C3, the external releases mastership and the external arbiter grants the bus to the MCF5407 by asserting $\overline{BG}$. At this point, an internal is access pending so the MCF5407 asserts $\overline{BD}$ during C4 and begins the access. Thus, the MCF5407 becomes the explicit bus master. Also during C4, the external arbiter removes the grant from the MCF5407 by negating $\overline{BG}$. Because the MCF5407 is bus master, it continues to assert $\overline{BD}$ until the current transfer completes. Because $\overline{BG}$ is negated, the MCF5407 negates $\overline{BD}$ during C9 and three-states the external bus, thereby passing mastership to an external device.

The MCF5407 can assert $\overline{BR}$ to signal the external arbiter that it needs the bus. However, there is no guarantee that when the bus is granted to the MCF5407 that a bus cycle will be performed. At best, $\overline{BR}$ must be used as a status output that indicates when the MCF5407 needs the bus, but not as an indication that the MCF5407 is in a certain bus arbitration state.

Figure 18-32 is a high-level state diagram for MCF5407 bus arbitration protocol. Table 18-6 describes the four states shown in Figure 18-32.

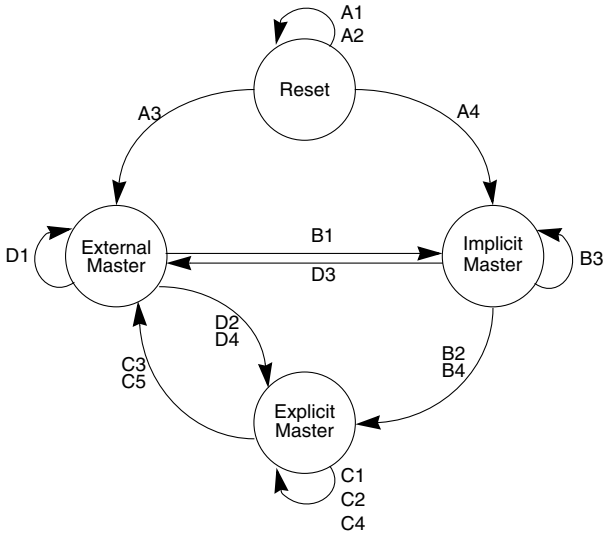



**Figure 18-32. Three-Wire Bus Arbitration Protocol State Diagram**

Table 18-11 lists conditions that cause state transitions.

**Table 18-11. Three-Wire Bus Arbitration Protocol Transition Conditions**

| Current State | Condition Label | $\overline{\text{RSTI}}$ | Software Watchdog Reset | $\overline{\text{BG}}$ | Bus Request | Transfer in Progress | End of Cycle [1] | Next State |
|---|---|---|---|---|---|---|---|---|
| Reset | A1 | Asserted | — | — | — | — | — | Reset |
| | A2 | Negated | Asserted | — | — | — | — | Reset |
| | A3 | Negated | Negated | Negated | — | — | — | EM |
| | A4 | Negated | Negated | Asserted | — | — | — | Implicit master |
| Implicit master | B1 | Negated | Negated | Negated | — | — | — | External device master |
| | B2 | Negated | Negated | Asserted | — | — | — | Explicit master |
| | B3 | Negated | Negated | Asserted | Negated | — | — | Implicit master |
| | B4 | Negated | Negated | Asserted | Asserted | — | — | Explicit master |

**Table 18-11. Three-Wire Bus Arbitration Protocol Transition Conditions (Continued)**

| Current State | Condition Label | $\overline{\text{RSTI}}$ | Software Watchdog Reset | $\overline{\text{BG}}$ | Bus Request | Transfer in Progress | End of Cycle [1] | Next State |
|---|---|---|---|---|---|---|---|---|
| Explicit master | C1 | Negated | Negated | Asserted | — | — | — | Explicit master |
| | C2 | Negated | Negated | Negated | — | — | — | Explicit master |
| | C3 | Negated | Negated | Negated | — | Negated | — | External device master |
| | C4 | Negated | Negated | Negated | — | Yes | Negated | Explicit master |
| | C5 | Negated | Negated | Negated | — | Yes | Yes | External device master |
| External master | D1 | Negated | Negated | Negated[1] | — | — | — | External device master |
| | D2 | Negated | Negated | Asserted | — | — | — | Explicit master |
| | D3 | Negated | Negated | Asserted | Negated | — | — | Implicit master |
| | D4 | Negated | Negated | Asserted | Asserted | — | — | Explicit master |

[1] Both normal terminations and terminations due to bus errors generate an end of cycle. Bus cycles resulting from a burst-inhibited transfer are considered part of that original transfer.

The bus arbitration state diagram can be used for the MCF5407 three-wire bus arbitration protocol to approximate the high-level behavior of the MCF5407. It is assumed that all $\overline{\text{TS}}$ or $\overline{\text{AS}}$ signals in a system are tied together and each bus device's $\overline{\text{BD}}$ and $\overline{\text{BR}}$ signals are connected individually to the external arbiter. The external arbiter must ensure that any external masters will have released the bus after the next rising edge of CLKIN before asserting $\overline{\text{BG}}$ to the MCF5407. The MCF5407 does not monitor external bus master operation regarding bus arbitration.

**NOTE:**

The MCF5407 can start a transfer on the rising edge of CLKIN the cycle after $\overline{\text{BG}}$ is asserted. The external arbiter should not assert $\overline{\text{BG}}$ to the MCF5407 until the previous external master stops driving the bus or the part may be damaged.

## 18.10 Reset Operation

The MCF5407 supports two types of reset. Asserting $\overline{\text{RSTI}}$ resets the entire MCF5407. A software watchdog reset resets everything but the internal PLL module.

## 18.10.1 Master Reset

To perform a master reset, an external device asserts $\overline{\text{RSTI}}$. When power is applied to the system, external circuitry should assert $\overline{\text{RSTI}}$ for a minimum of 16 CLKIN cycles after EVcc and IVcc are within tolerance. Figure 18-33 is a functional timing diagram of the master reset operation, showing relationships among E/IVcc, $\overline{\text{RSTI}}$, mode selects, and bus signals. CLKIN must be stable by the time E/IVcc reach the minimum operating specification. See Section 20.1.1, “Supply Voltage Sequencing and Separation Cautions.” CLKIN should start oscillating as E/IVcc are ramped up to clear out contention internal to the MCF5407 caused by the random states of internal flip-flops on power up. $\overline{\text{RSTI}}$ is internally synchronized for two CLKIN cycles before being used and must meet the specified setup and hold times in relationship to CLKIN to be recognized.

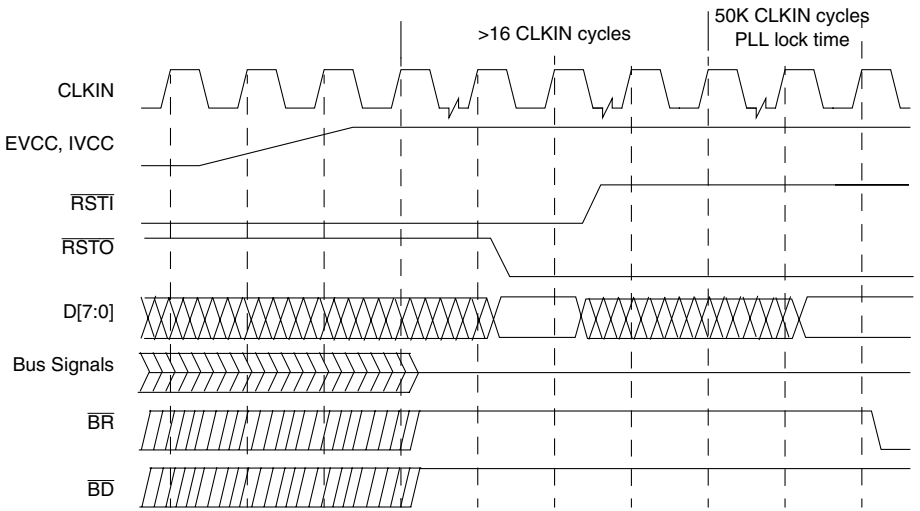



**Figure 18-33. Master Reset Timing**

During the master reset period, all signals capable of being three-stated are driven to a high-impedance; all others are negated. When $\overline{\text{RSTI}}$ negates, all bus signals remain in a high-impedance state until the MCF5407 is granted the bus and the core begins the first bus cycle for reset exception processing. A master reset causes any bus cycle (including DRAM refresh cycle) to terminate and initializes registers appropriately for a reset exception.

Note that during reset D[7:0] are sampled on the negating edge of $\overline{\text{RSTI}}$ for initial MCF5407 configurations listed in Table 18-12.

**Table 18-12. Data Pin Configuration**

| Pin | Function |
|---|---|
| D7 | Auto-Acknowledge Configuration (AA_CONFIG) |
| D[6:5] | Port Size Configuration (PS_CONFIG[1:0]) |
| D4 | Address Configuration (ADDR_CONFIG/D4) |
| D3 | Byte Enable Configuration (BE_CONFIG) |
| D[2:0] | Divide Control (DIVIDE[2:0]) |

See Section 17.5.5, “Data/Configuration Pins (D[7:0]).” Motorola recommends that the data pins be driven rather than using a weak pull-up or pull-down resistor. Table 17-1 lists the encoding of these pins sampled at reset.

After $\overline{\text{RSTI}}$ is negated, 32 bits of CPU configuration information are loaded into data register D0 and 32 bits of internal memory information are loaded in D1. Because D0 and D1 are uninitialized on previous ColdFire devices, this feature allows users to identify the MCF5407 through software. Values D1 = 0x0630_0530 and D0 = 0xCF4*x*_C012 identify the MCF5407, where *x* identifies the core revision number (0x1 for the initial device).

## 18.10.2 Software Watchdog Reset

A software watchdog reset is performed if the executing software does not provide the correct write data sequence with the enable-control bit set. This reset helps prevent runaway software or unterminated bus cycles. Figure 18-34 is a functional timing diagram of the software watchdog reset operation, showing $\overline{\text{RSTO}}$ and bus signal relationships.

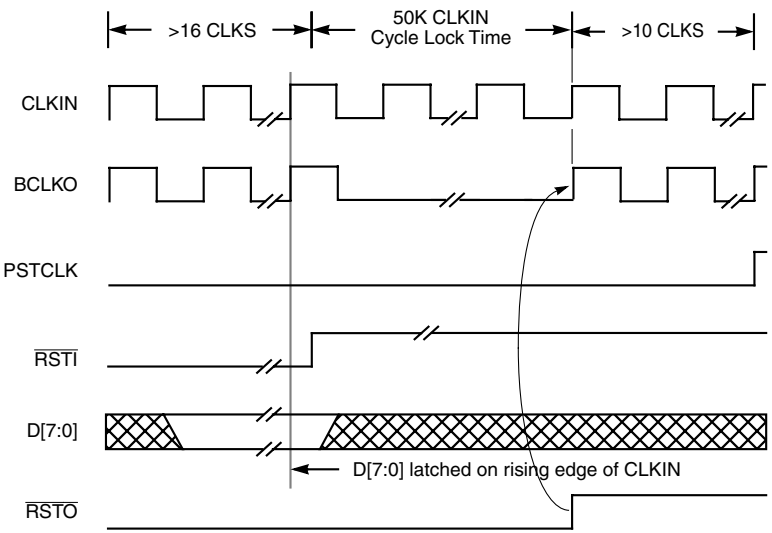



**Figure 18-34. Software Watchdog Reset Timing**

During the software watchdog reset period, all signals that can be are driven to a high-impedance state; all those that cannot be are negated. When $\overline{\text{RSTO}}$ negates, bus signals remain in a high-impedance state until the MCF5407 is granted the bus and the ColdFire core begins the first bus cycle for reset exception processing.

# Chapter 19
# IEEE 1149.1 Test Access Port (JTAG)

This chapter describes configuration and operation of the MCF5407 JTAG test implementation. It describes the use of JTAG instructions and provides information on how to disable JTAG functionality.

## 19.1 Overview

The MCF5407 dedicated user-accessible test logic is fully compliant with the publication *Standard Test Access Port and Boundary-Scan Architecture,* IEEE Standard 1149.1. Use the following description in conjunction with the supporting IEEE document listed above. This section includes the description of those chip-specific items that the IEEE standard requires as well as those items specific to the MCF5407 implementation.

The MCF5407 JTAG test architecture supports circuit board test strategies based on the IEEE standard. 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, $\overline{\text{TRST}}$. Test logic design is static and is independent of the system logic except where the JTAG is subordinate to other complimentary test modes, as described in Chapter 5, “Debug Support.” When in subordinate mode, JTAG test logic is placed in reset and the TAP pins can be used for other purposes, as described in Table 19-1.

The MCF5407 JTAG implementation can do the following:

- Perform boundary-scan operations to test circuit board electrical continuity
- Bypass the MCF5407 by reducing the shift register path to a single cell
- Set MCF5407 output drive pins to fixed logic values while reducing the shift register path to a single cell
- Sample MCF5407 system pins during operation and transparently shift out the result
- Protect MCF5407 system output and input pins from backdriving and random toggling (such as during in-circuit testing) by placing all system pins in high-impedance state

**NOTE:**

IEEE Standard 1149.1 may interfere with system designs that do not incorporate JTAG capability. Section 19.6, “Disabling IEEE Standard 1149.1 Operation,” describes precautions for ensuring

that this logic does not affect system or debug operation.

Figure 19-1 is a block diagram of the MCF5407 implementation of the 1149.1 IEEE standard. The test logic includes several test data registers, an instruction register, instruction register control decode, and a 16-state dedicated TAP controller.

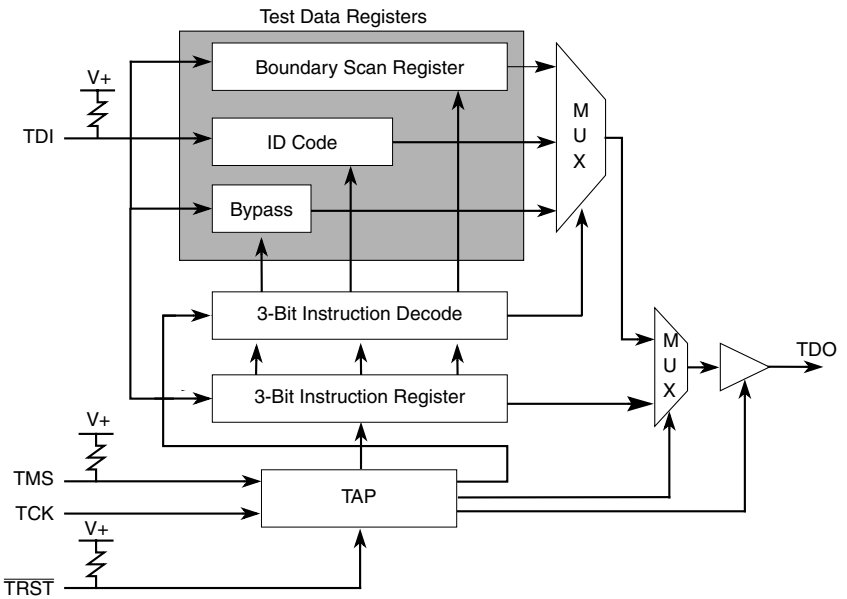



**Figure 19-1. JTAG Test Logic Block Diagram**

## 19.2 JTAG Signal Descriptions

JTAG operation on the MCF5407 is enabled when MTMOD0 is high (logic 1), as described in Table 19-1. Otherwise, JTAG TAP signals, TCK, TMS, TDI, TDO, and $\overline{\text{TRST}}$, are interpreted as the debug port pins. MTMOD0 should not be changed while $\overline{\text{RSTI}}$ is asserted.

**Table 19-1. JTAG Pin Descriptions**

| Pin | Description |
|---|---|
| TCK | Test clock. The dedicated JTAG test logic clock is independent of the MCF5407 processor clock. Various JTAG operations occur on the rising or falling edge of TCK. Internal JTAG controller logic is designed such that holding TCK high or low indefinitely does cause the JTAG test logic to lose state information. If TCK is not used, it should be tied to ground. |
| TMS/ $\overline{\text{BKPT}}$ | Test mode select (MTMOD0 high)/breakpoint (MTMOD0 low). TMS provides the JTAG controller with information to determine the test operation mode. The states of TMS and of the internal 16-state JTAG controller state machine at the rising edge of TCK determine whether the JTAG controller holds its current state or advances to the next state. This directly controls whether JTAG data or instruction operations occur. TMS has an internal pull-up, so if it is not driven low, its value defaults to a logic level of 1. If TMS is not used, it should be tied to VDD. $\overline{\text{BKPT}}$ signals a hardware breakpoint to the processor in debug mode. See Chapter 5, “Debug Support.” |
| TDI/DSI | Test data input (MTMOD0 high)/development serial input (MTMOD0 low). TDI provides the serial data port for loading the JTAG boundary-scan, bypass, and instruction registers. Shifting in of data depends on the state of the JTAG controller state machine and the instruction in the instruction register. This shift occurs on the rising edge of TCK. TDI has an internal pull-up so if it is not driven low its value defaults to a logical 1. If TDI is not used, it should be tied to VDD.<br>DSI provides single-bit communication for debug module commands. See Chapter 5, “Debug Support.” |
| TDO/ DSO | Test data output (MTMOD0 high)/development serial output (MTMOD0 low). TDO is the serial data port for outputting data from JTAG logic. Shifting data out depends on the state of the JTAG controller state machine and the instruction currently in the instruction register. This shift occurs on the falling edge of TCK. When not outputting test data, TDO is three-stated. It can also be placed in three-state mode to allow bussed or parallel connections to other devices having JTAG. DSO provides single-bit communication for debug module commands. See Chapter 5, “Debug Support.” |
| $\overline{\text{TRST}}$/ DSCLK | Test reset (MTMOD0 high)/development serial clock (MTMOD0 low). As $\overline{\text{TRST}}$, this pin asynchronously resets the internal JTAG controller to the test logic reset state, causing the JTAG instruction register to choose the IDCODE instruction. When this occurs, all JTAG logic is benign and does not interfere with normal MCF5407 functionality. Although this signal is asynchronous, Motorola recommends that $\overline{\text{TRST}}$ make only an asserted-to-negated transition while TMS is held at a logic 1 value. $\overline{\text{TRST}}$ has an internal pull-up; if it is not driven low its value defaults to a logic level of 1. However, if $\overline{\text{TRST}}$ is not used, it can either be tied to ground or, if TCK is clocked, to VDD. The former connection places the JTAG controller in the test logic reset state immediately; the latter connection eventually puts the JTAG controller (if TMS is a logic 1) into the test logic reset state after 5 TCK cycles.<br>DSCLK is the development serial clock for the serial interface to the debug module.The maximum DSCLK frequency is 1/2 the CLKIN frequency. See Chapter 5, “Debug Support.” |

## 19.3 TAP Controller

The state of TMS at the rising edge of TCK determines the current state of the TAP controller. The TAP controller can follow two basic two paths, one for executing JTAG instructions and the other for manipulating JTAG data based on JTAG instructions. The various states of the TAP controller are shown in Figure 19-2. For more detail on each state, see the IEEE Standard 1149.1 JTAG document. Note that regardless of the TAP controller state, test-logic-reset can be entered if TMS is held high for at least five rising edges of TCK. Figure 19-2 shows the JTAG TAP controller state machine.

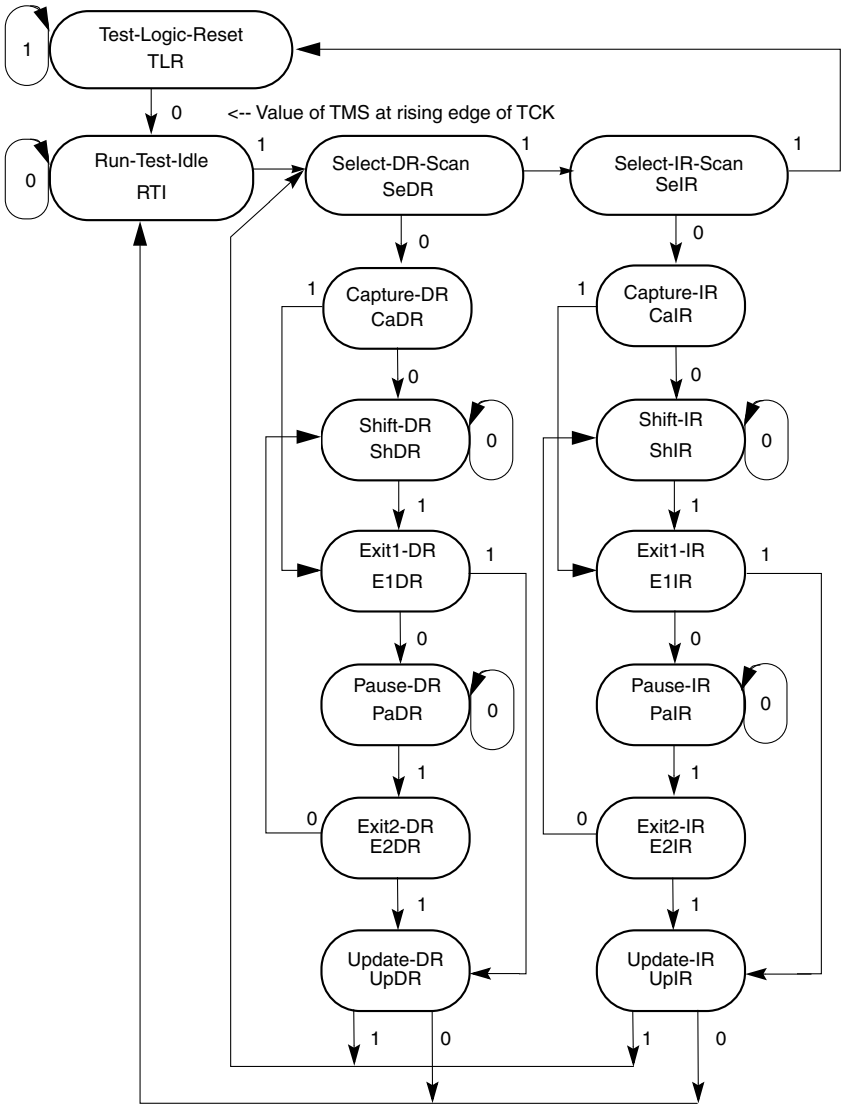



**Figure 19-2. JTAG TAP Controller State Machine**

## 19.4 JTAG Register Descriptions

The following sections describe the JTAG registers implemented on the MCF5407.

### 19.4.1 JTAG Instruction Shift Register

The MCF5407 IEEE Standard 1149.1 implementation uses a 3-bit instruction-shift register (IR) without parity. This register transfers its value to a parallel hold register and applies one of six instructions on the falling edge of TCK when the TAP state machine is in Update-IR state. To load instructions into the shift portion of the register, place the serial data on TDI before each rising edge of TCK. The msb of the instruction shift register is the bit closest to the TDI pin, and the lsb is the bit closest to TDO.

Table 19-2 describes customer-usable instructions.

**Table 19-2. JTAG Instructions**

| Instruction | Class | IR | Description |
|---|---|---|---|
| EXTEST (EXT) | Required | 000 | Selects the boundary-scan register. Forces all output pins and bidirectional pins configured as outputs to the preloaded fixed values (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 becomes active on the falling edge of TCK in the Update-IR state when the data held in the instruction-shift register is equivalent to octal 0. |
| IDCODE (IDC) | Optional | 001 | Selects the IDCODE register for connection as a shift path between TDI and TDO. Interrogates the MCF5407 for version number and other part identification. The IDCODE register is implemented in accordance with IEEE Standard 1149.1 so the lsb of the shift register stage is set to logic 1 on the rising edge of TCK following entry into the capture-DR state. Therefore, the first bit shifted out after selecting the IDCODE register is always a logic 1. The remaining 31-bits are also set to fixed values. See Section 19.4.2, “IDCODE Register.”<br>IDCODE is the default value in the IR when a JTAG reset occurs by either asserting $\overline{\text{TRST}}$ or holding TMS high while clocking TCK through at least five rising edges and the falling edge after the fifth rising edge. A JTAG reset causes the TAP state machine to enter test-logic-reset state (normal operation of the TAP state machine into the test-logic-reset state also places the default value of octal 1 into the instruction register). The shift register portion of the instruction register is loaded with the default value of octal 1 in Capture-IR state and a TCK rising edge occurs. |
| SAMPLE/ PRELOAD (SMP) | Required | 100 | Provides two separate functions. It obtains a sample of the system data and control signals at the MCF5407 input pins and before the boundary-scan cell at the output pins. This sampling occurs on the rising edge of TCK in the capture-DR state when an instruction encoding of octal 4 is in the instruction register. Sampled data is observed by shifting it through the boundary-scan register to TDO by using shift-DR state. The data capture and shift are transparent to system operation. The users must provide external synchronization to achieve meaningful results because there is no internal synchronization between TCK and CLK.<br>SAMPLE/PRELOAD also initializes the boundary-scan register update cells before selecting EXTEST or CLAMP. This is done by ignoring data shifted out of TDO while shifting in initialization data. The Update-DR state in conjunction with the falling edge of TCK can then transfer this data to the update cells. This data is applied to external outputs when an instruction listed above is applied. |
| HIGHZ (HIZ) | Optional | 101 | Anticipates the need to backdrive outputs and protects inputs from random toggling during board testing. Selects the bypass register, forcing all output and bidirectional pins into high-impedance.<br>HIGHZ goes active on the falling edge of TCK in the Update-IR state when instruction shift register data held is equivalent to octal 5. |

**Table 19-2. JTAG Instructions (Continued)**

| Instruction | Class | IR | Description |
|---|---|---|---|
| CLAMP (CMP) | Optional | 110 | Selects the bypass register and asserts functional reset while forcing all output and bidirectional pins configured as outputs to fixed, preloaded values in the boundary-scan update registers. Enhances test efficiency by reducing the overall shift path to one bit (the bypass register) while conducting an EXTEST type of instruction through the boundary-scan register. CLAMP becomes active on the falling edge of TCK in the Update-IR state when instruction-shift register data is equivalent to octal 6. |
| BYPASS (BYP) | Required | 111 | Selects the single-bit bypass register, creating a single-bit shift register path from TDI to the bypass register to TDO. Enhances test efficiency by reducing the overall shift path when a device other than the MCF5407 is under test on a board design with multiple chips on the overall 1149.1 defined boundary-scan chain. The bypass register is implemented in accordance with 1149.1 so the shift register stage is set to logic 0 on the rising edge of TCK following entry into the capture-DR state. Therefore, the first bit shifted out after selecting the bypass register is always a logic 0 (to differentiate a part that supports an IDCODE register from a part that supports only the bypass register).<br>BYPASS goes active on the falling edge of TCK in the Update-IR state when instruction shift register data is equivalent to octal 7. |

The IEEE Standard 1149.1 requires the EXTEST, SAMPLE/PRELOAD, and BYPASS instructions. IDCODE, CLAMP, and HIGHZ are optional standard instructions that the MCF5407 implementation supports and are described in the IEEE Standard 1149.1.

## 19.4.2 IDCODE Register

The MCF5407 includes an IEEE Standard 1149.1-compliant JTAG identification register, IDCODE, which is read by the MCF5407 JTAG instruction encoded as octal 1.

| 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 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Version Number (0000 for initial MCF5407 device) | | | | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |

**Figure 19-3. IDCODE Register**

Table 19-3 describes IDCODE bit assignments.

**Table 19-3. IDCODE Bit Assignments**

| Bits | Description |
|---|---|
| 31–28 | Version number. Indicates the revision number of the MCF5407 |
| 27–22 | Design center. Indicates the ColdFire design center |
| 21–12 | Device number. Indicates an MCF5407 |
| 11–1 | Indicates the reduced JEDEC ID for Motorola. Joint Electron Device Engineering Council (JEDEC) Publication 106-A and Chapter 11 of the IEEE Standard 1149.1 give more information on this field. |
| 0 | Identifies this as the JTAG IDCODE register (and not the bypass register) according to the IEEE Standard 1149.1 |

## 19.4.3 JTAG Boundary-Scan Register

The MCF5407 model includes an IEEE Standard 1149.1-compliant boundary-scan register connected between TDI and TDO when the EXTEST or SAMPLE/PRELOAD instructions are selected. This register captures signal data on the input pins, forces fixed values on the output pins, and selects the direction and drive characteristics (a logic value or high impedance) of the bidirectional and three-state pins. Table 19-4 shows MCF5407 boundary-scan register bits.

**Table 19-4. Boundary-Scan Bit Definitions**

| Bit | Cell Type | Pin Cell | Pin Type | Bit | Cell Type | Pin Cell | Pin Type |
|---|---|---|---|---|---|---|---|
| 0 | O.Ctl | PP0 enable | — | 120 | O.Pin | $\overline{BE0}$ | O |
| 1 | O.Pin | PP0 | I/O | 121 | O.Pin | SCKE | O |
| 2 | I.Pin | PP0 | I/O | 122 | O.Pin | $\overline{SCAS}$ | O |
| 3 | IO.Ctl | PP1 enable | — | 123 | O.Pin | $\overline{SRAS}$ | O |
| 4 | O.Pin | PP1 | I/O | 124 | O.Pin | $\overline{DRAMW}$ | O |
| 5 | I.Pin | PP1 | I/O | 125 | O.Pin | $\overline{CAS3}$ | O |
| 6 | IO.Ctl | PP2 enable | — | 126 | O.Pin | $\overline{CAS2}$ | O |
| 7 | O.Pin | PP2 | I/O | 127 | O.Pin | $\overline{CAS1}$ | O |
| 8 | I.Pin | PP2 | I/O | 128 | O.Pin | $\overline{CAS0}$ | O |
| 9 | IO.Ctl | PP3 enable | — | 129 | O.Pin | $\overline{RAS1}$ | O |
| 10 | O.Pin | PP3 | I/O | 130 | O.Pin | $\overline{RAS0}$ | O |
| 11 | I.Pin | PP3 | I/O | 131 | I.Pin | TIN1 | I |
| 12 | IO.Ctl | PP4 enable | — | 132 | I.Pin | TIN0 | I |
| 13 | O.Pin | PP4 | I/O | 133 | O.Pin | TOUT0 | O |
| 14 | I.Pin | PP4 | I/O | 134 | O.Pin | TOUT1 | O |
| 15 | IO.Ctl | PP5 enable | — | 135 | I.Pin | $\overline{BG}$ | I |
| 16 | O.Pin | PP5 | I/O | 136 | O.Pin | $\overline{BD}$ | O |
| 17 | I.Pin | PP5 | I/O | 137 | O.Pin | $\overline{BR}$ | O |
| 18 | IO.Ctl | PP6 enable | — | 138 | I.Pin | $\overline{IRQ1}$ | I |
| 19 | O.Pin | PP6 | I/O | 139 | I.Pin | $\overline{IRQ3}$ | I |
| 20 | I.Pin | PP6 | I/O | 140 | I.Pin | $\overline{IRQ5}$ | I |
| 21 | IO.Ctl | PP7 enable | — | 141 | I.Pin | $\overline{IRQ7}$ | I |
| 22 | O.Pin | PP7 | I/O | 142 | I.Pin | $\overline{RSTI}$ | I |
| 23 | I.Pin | PP7 | I/O | 143 | O.Pin | $\overline{TS}$ | I/O |
| 24 | O.Pin | PSTDDATA7 | O | 144 | I.Pin | $\overline{TS}$ | I/O |
| 25 | O.Pin | PSTDDATA6 | O | 145 | IO.Ctl | $\overline{TA}$ enable | — |
| 26 | O.Pin | PSTDDATA5 | O | 146 | O.Pin | $\overline{TA}$ | I/O |
| 27 | O.Pin | PSTDDATA4 | O | 147 | I.Pin | $\overline{TA}$ | I/O |
| 28 | O.Pin | PSTDDATA3 | O | 148 | O.Pin | R/$\overline{W}$ | I/O |

## Table 19-4. Boundary-Scan Bit Definitions

| Bit | Cell Type | Pin Cell | Pin Type |
|---|---|---|---|
| 29 | O.Pin | PSTDDATA2 | O |
| 30 | O.Pin | PSTDDATA1 | O |
| 31 | O.Pin | PSTDDATA0 | O |
| 32 | O.Pin | PSTCLK | O |
| 33 | I.Pin | CLKIN | I |
| 34 | IO.Ctl | $\overline{\text{RSTO}}$ enable | — |
| 35 | O.Pin | $\overline{\text{RSTO}}$ | I/O |
| 36 | I.Pin | $\overline{\text{RSTO}}$ | I/O |
| 37 | O.Pin | BCLKO | O |
| 38 | I.Pin | EDGESEL | I |
| 39 | O.Pin | TXD0 | O |
| 40 | I.Pin | RXD0 | I |
| 41 | O.Pin | RTS0 | O |
| 42 | I.Pin | CTS0 | I |
| 43 | O.Pin | TXD1 | O |
| 44 | I.Pin | RXD1 | I |
| 45 | O.Pin | RTS1 | O |
| 46 | I.Pin | CTS1 | I |
| 47 | I.Pin | $\overline{\text{HIZ}}$ | I |
| 48 | IO.Ctl | Data enable | — |
| 49 | O.Pin | D0 | I/O |
| 50 | I.Pin | D0 | I/O |
| 51 | O.Pin | D1 | I/O |
| 52 | I.Pin | D1 | I/O |
| 53 | O.Pin | D2 | I/O |
| 54 | I.Pin | D2 | I/O |
| 55 | O.Pin | D3 | I/O |
| 56 | I.Pin | D3 | I/O |
| 57 | O.Pin | D4 | I/O |
| 58 | I.Pin | D4 | I/O |
| 59 | O.Pin | D5 | I/O |
| 60 | I.Pin | D5 | I/O |
| 61 | O.Pin | D6 | I/O |
| 62 | I.Pin | D6 | I/O |
| 63 | O.Pin | D7 | I/O |
| 64 | I.Pin | D7 | I/O |

| Bit | Cell Type | Pin Cell | Pin Type |
|---|---|---|---|
| 149 | I.Pin | R/$\overline{\text{W}}$ | I/O |
| 150 | O.Pin | $\overline{\text{AS}}$ | I/O |
| 151 | I.Pin | $\overline{\text{AS}}$ | I/O |
| 152 | O.Pin | $\overline{\text{CS7}}$ | O |
| 153 | O.Pin | $\overline{\text{CS6}}$ | O |
| 154 | O.Pin | $\overline{\text{CS5}}$ | O |
| 155 | O.Pin | $\overline{\text{CS4}}$ | O |
| 156 | O.Pin | $\overline{\text{CS3}}$ | O |
| 157 | O.Pin | $\overline{\text{CS2}}$ | O |
| 158 | O.Pin | $\overline{\text{CS1}}$ | O |
| 159 | O.Pin | $\overline{\text{CS0}}$ | O |
| 160 | O.Pin | $\overline{\text{OE}}$ | O |
| 161 | O.Pin | SIZ1 | I/O |
| 162 | I.Pin | SIZ1 | I/O |
| 163 | O.Pin | SIZ0 | I/O |
| 164 | I.Pin | SIZ0 | I/O |
| 165 | IO.Ctl | PP15 enable | — |
| 166 | I.Pin | PP15 | I/O |
| 167 | O.Pin | PP15 | I/O |
| 168 | IO.Ctl | PP14 enable | — |
| 169 | I.Pin | PP14 | I/O |
| 170 | O.Pin | PP14 | I/O |
| 171 | IO.Ctl | PP13 enable | — |
| 172 | I.Pin | PP13 | I/O |
| 173 | O.Pin | PP13 | I/O |
| 174 | IO.Ctl | PP12 enable | — |
| 175 | I.Pin | PP12 | I/O |
| 176 | O.Pin | PP12 | I/O |
| 177 | IO.Ctl | PP11 enable | — |
| 178 | I.Pin | PP11 | I/O |
| 179 | O.Pin | PP11 | I/O |
| 180 | IO.Ctl | PP10 enable | — |
| 181 | I.Pin | PP10 | I/O |
| 182 | O.Pin | PP10 | I/O |
| 183 | IO.Ctl | PP9 enable | — |
| 184 | I.Pin | PP9 | I/O |

## Table 19-4. Boundary-Scan Bit Definitions

| Bit | Cell Type | Pin Cell | Pin Type | Bit | Cell Type | Pin Cell | Pin Type |
|---|---|---|---|---|---|---|---|
| 65 | O.Pin | D8 | I/O | 185 | O.Pin | PP9 | I/O |
| 66 | I.Pin | D8 | I/O | 186 | IO.Ctl | PP8 enable | — |
| 67 | O.Pin | D9 | I/O | 187 | I.Pin | PP8 | I/O |
| 68 | I.Pin | D9 | I/O | 188 | O.Pin | PP8 | I/O |
| 69 | O.Pin | D10 | I/O | 189 | IO.Ctl | $\overline{TS}$/R/$\overline{W}$/SIZ enable | — |
| 70 | I.Pin | D10 | I/O | 190 | IO.Ctl | Address enable | — |
| 71 | O.Pin | D11 | I/O | 191 | O.Pin | A23 | I/O |
| 72 | I.Pin | D11 | I/O | 192 | I.Pin | A23 | I/O |
| 73 | O.Pin | D12 | I/O | 193 | O.Pin | A22 | I/O |
| 74 | I.Pin | D12 | I/O | 194 | I.Pin | A22 | I/O |
| 75 | O.Pin | D13 | I/O | 195 | O.Pin | A21 | I/O |
| 76 | I.Pin | D13 | I/O | 196 | I.Pin | A21 | I/O |
| 77 | O.Pin | D14 | I/O | 197 | O.Pin | A20 | I/O |
| 78 | I.Pin | D14 | I/O | 198 | I.Pin | A20 | I/O |
| 79 | O.Pin | D15 | I/O | 199 | O.Pin | A19 | I/O |
| 80 | I.Pin | D15 | I/O | 200 | I.Pin | A19 | I/O |
| 81 | O.Pin | D16 | I/O | 201 | O.Pin | A18 | I/O |
| 82 | I.Pin | D16 | I/O | 202 | I.Pin | A18 | I/O |
| 83 | O.Pin | D17 | I/O | 203 | O.Pin | A17 | I/O |
| 84 | I.Pin | D17 | I/O | 204 | I.Pin | A17 | I/O |
| 85 | O.Pin | D18 | I/O | 205 | O.Pin | A16 | I/O |
| 86 | I.Pin | D18 | I/O | 206 | I.Pin | A16 | I/O |
| 87 | O.Pin | D19 | I/O | 207 | O.Pin | A15 | I/O |
| 88 | I.Pin | D19 | I/O | 208 | I.Pin | A15 | I/O |
| 89 | O.Pin | D20 | I/O | 209 | O.Pin | A14 | I/O |
| 90 | I.Pin | D20 | I/O | 210 | I.Pin | A14 | I/O |
| 91 | O.Pin | D21 | I/O | 211 | O.Pin | A13 | I/O |
| 92 | I.Pin | D21 | I/O | 212 | I.Pin | A13 | I/O |
| 93 | O.Pin | D22 | I/O | 213 | O.Pin | A12 | I/O |
| 94 | I.Pin | D22 | I/O | 214 | I.Pin | A12 | I/O |
| 95 | O.Pin | D23 | I/O | 215 | O.Pin | A11 | I/O |
| 96 | I.Pin | D23 | I/O | 216 | I.Pin | A11 | I/O |
| 97 | O.Pin | D24 | I/O | 217 | O.Pin | A10 | I/O |
| 98 | I.Pin | D24 | I/O | 218 | I.Pin | A10 | I/O |
| 99 | O.Pin | D25 | I/O | 219 | O.Pin | A9 | I/O |
| 100 | I.Pin | D25 | I/O | 220 | I.Pin | A9 | I/O |

**Table 19-4. Boundary-Scan Bit Definitions**

| Bit | Cell Type | Pin Cell | Pin Type |
|---|---|---|---|
| 101 | O.Pin | D26 | I/O |
| 102 | I.Pin | D26 | I/O |
| 103 | O.Pin | D27 | I/O |
| 104 | I.Pin | D27 | I/O |
| 105 | O.Pin | D28 | I/O |
| 106 | I.Pin | D28 | I/O |
| 107 | O.Pin | D29 | I/O |
| 108 | I.Pin | D29 | I/O |
| 109 | O.Pin | D30 | I/O |
| 110 | I.Pin | D30 | I/O |
| 111 | O.Pin | D31 | I/O |
| 112 | I.Pin | D31 | I/O |
| 113 | O.Pin | SDA | OD |
| 114 | I.Pin | SDA | I |
| 115 | O.Pin | SCL | OD |
| 116 | I.Pin | SCL | I |
| 117 | O.Pin | $\overline{BE3}$ | O |
| 118 | O.Pin | $\overline{BE2}$ | O |
| 119 | O.Pin | $\overline{BE1}$ | O |
| 221 | O.Pin | A8 | I/O |
| 222 | I.Pin | A8 | I/O |
| 223 | O.Pin | A7 | I/O |
| 224 | I.Pin | A7 | I/O |
| 225 | O.Pin | A6 | I/O |
| 226 | I.Pin | A6 | I/O |
| 227 | O.Pin | A5 | I/O |
| 228 | I.Pin | A5 | I/O |
| 229 | O.Pin | A4 | I/O |
| 230 | I.Pin | A4 | I/O |
| 231 | O.Pin | A3 | I/O |
| 232 | I.Pin | A3 | I/O |
| 233 | O.Pin | A2 | I/O |
| 234 | I.Pin | A2 | I/O |
| 235 | O.Pin | A1 | I/O |
| 236 | I.Pin | A1 | I/O |
| 237 | O.Pin | A0 | I/O |
| 238 | I.Pin | A0 | I/O |

### 19.4.4 JTAG Bypass Register

The IEEE Standard 1149.1-compliant bypass register creates a single-bit shift register path from TDI to the bypass register to TDO when the BYPASS instruction is selected.

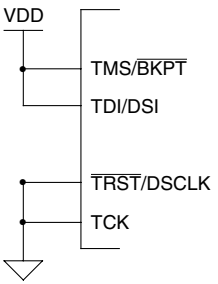
## 19.5 Restrictions

Test logic design is static, so TCK can be stopped in high or low state with no data loss. However, system logic uses a different system clock not internally synchronized to TCK. Operation mixing 1149.1 test logic with system functional logic that uses both clocks must coordinate and synchronize these clocks externally to the MCF5407.

## 19.6 Disabling IEEE Standard 1149.1 Operation

There are two ways to use the MCF5407 without IEEE Standard 1149.1 test logic being active:

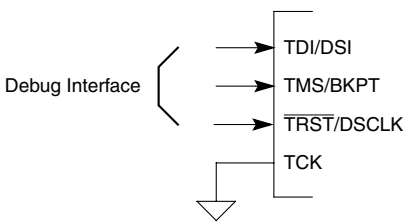
- Nonuse of JTAG test logic by either nontermination (disconnection) or intentionally fixing TAP logic values. The following issues must be addressed if IEEE Standard 1149.1 logic is not to be used when the MCF5407 is assembled onto a board.
  - IEEE Standard 1149.1 test logic must remain transparent and benign to the system logic during functional operation. To ensure that the part enters the test-logic-reset state requires either connecting $\overline{\text{TRST}}$ to logic 0 or connecting TCK to a source that supplies five rising edges and a falling edge after the fifth rising edge. The recommended solution is to connect $\overline{\text{TRST}}$ to logic 0.
  - TCK has no internal pull-up as is required on TMS, TDI, and $\overline{\text{TRST}}$; therefore, it must be terminated to preclude mid-level input values. Figure 19-4 shows pin values recommended for disabling JTAG with the MCF5407 in JTAG mode.




Note: MTMOD0 high allows JTAG mode.

**Figure 19-4. Disabling JTAG in JTAG Mode**

- Disabling JTAG test logic by holding MTMOD0 low during reset (debug mode). This allows the IEEE Standard 1149.1 test controller to enter test-logic-reset state when $\overline{\text{TRST}}$ is internally asserted to the controller. TAP pins function as debug mode pins. In JTAG mode, inputs TDI/DSI, TMS/$\overline{\text{BKPT}}$, and $\overline{\text{TRST}}$/DSCLK have internal pull-ups enabled. Figure 19-5 shows pin values recommended for disabling JTAG in debug mode.




Note: MTMOD0 low prohibits JTAG.

**Figure 19-5. Disabling JTAG in Debug Mode**

## 19.7 Obtaining the IEEE Standard 1149.1

The IEEE Standard 1149.1 JTAG specification is a copyrighted document and must be

obtained directly from the IEEE:

IEEE Standards Department
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA

http://stdsbbs.ieee.org/

FAX: 908-981-9667
Information: 908-981-0060 or 1-800-678-4333

# Chapter 20
# Electrical Specifications

This chapter describes the AC and DC electrical specifications and thermal characteristics for the MCF5407. Note that this information was correct at the time this book was published. As process technologies improve, there is a likelihood that this information may change. To confirm that this is the latest information, see Motorola's ColdFire webpage, http://www.motorola.com/coldfire.

## 20.1 General Parameters

Table 20-1 lists maximum and minimum ratings for supply and operating voltages and storage temperature. Operating outside of these ranges may cause erratic behavior or damage to the processor.

**Table 20-1. Absolute Maximum Ratings**

| Rating | Symbol | Value | Units |
|---|---|---|---|
| External (I/O pads) supply voltage (3.3-V power pins) | $EV_{cc}$ | -0.3 to +4.0 | V |
| Internal logic supply voltage | $IV_{cc}$ | -0.5 to +2.0 [1, 2] | V |
| PLL supply voltage | $PV_{cc}$ | -0.5 to +2.0 [2, 3] | V |
| Internal logic supply voltage, input voltage level | $V_{in}$ | -0.5 to +3.6 [4] | V |
| Storage temperature range | $T_{stg}$ | -55 to +150 | °C |

1 $IV_{cc}$ must not exceed $EV_{cc}$
2 $IV_{cc}$ and $PV_{cc}$ must not differ by more than 0.5 V
3 $PV_{cc}$ must not exceed $EV_{cc}$
4 $V_{in}$ must not exceed $EV_{cc}$

Table 20-2 lists junction and ambient operating temperatures.

**Table 20-2. Operating Temperatures**

| Characteristic | Symbol | Value | Units |
|---|---|---|---|
| Maximum operating junction temperature | $T_j$ | TBD | °C |
| Maximum operating ambient temperature | $T_{Amax}$ | 70 [1] | °C |
| Minimum operating ambient temperature | $T_{Amin}$ | 0 | °C |

1 This published maximum operating ambient temperature should be used only as a system design guideline. All device operating parameters are guaranteed only when the junction temperature lies within the specified range.

Table 20-3 lists DC electrical operating temperatures. This table is based on an operating voltage of EVcc = 3.3 Vdc ± 0.3 Vdc and IVcc of 1.8 ± 0.15 Vdc.

**Table 20-3. DC Electrical Specifications**

| Characteristic | Symbol | Min | Max | Units |
|---|---|---|---|---|
| External (I/O pads) operation voltage range | $EV_{cc}$ | 3.0 | 3.6 | V |
| Internal logic operation voltage range [1] | $IV_{cc}$ | 1.65 | 1.95 | V |
| PLL operation voltage range [1] | $PV_{cc}$ | 1.65 | 1.95 | V |
| Input high voltage | $V_{IH}$ | 2.0 | 3.6 | V |
| Input low voltage | $V_{IL}$ | -0.5 | 0.8 | V |
| Input signal undershoot | — | — | 0.8 | V |
| Input signal overshoot | — | — | 0.8 | V |
| Input leakage current @ 0.5/2.4 V during normal operation | $I_{in}$ | — | 20 | µA |
| High impedance (three-state) leakage current @ 0.5/2.4 V during normal operation | $I_{TSI}$ | — | 20 | µA |
| Signal low input current, $V_{IL}$ = 0.8 V [2] | $I_{IL}$ | 0 | 1 | mA |
| Signal high input current, $V_{IH}$ = 2.0 V [2] | $I_{IH}$ | 0 | 1 | mA |
| Output high voltage $I_{OH}$ = 8 mA [3], 16 mA [4] | $V_{OH}$ | 2.4 | — | V |
| Output low voltage $I_{OL}$ = 8 mA [3], 16 mA [4] | $V_{OL}$ | — | 0.5 | V |
| Load capacitance (all outputs) | $C_L$ | — | 50 | pF |
| Capacitance [5], $V_{in}$ = 0 V, f = 1 MHz | $C_{IN}$ | — | TBD | pF |

1 $IV_{cc}$ and $PV_{cc}$ should be at the same voltage.

2 $\overline{BKPT}$/TMS, DSI/TDI, DSCLK/$\overline{TRST}$

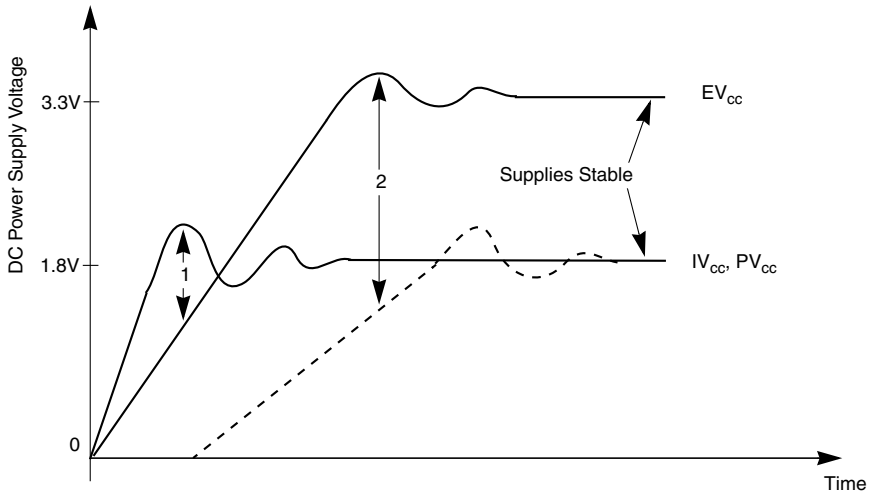
3 D[31:0], A[23:0], PP[15:0],$\overline{TS}$, $\overline{TA}$, SIZ[1:0], R/$\overline{W}$, $\overline{BR}$, $\overline{BD}$, $\overline{RSTO}$, $\overline{AS}$, $\overline{CS}$[7:0], $\overline{BE}$[3:0], $\overline{OE}$, PSTCLK, PSTDDATA[7:0], DSO, TOUT[1:0], SCL, SDA, $\overline{RTS}$[1:0], TXD[1:0]

4 BCLKO, $\overline{RAS}$[1:0], $\overline{CAS}$[3:0], $\overline{DRAMW}$, SCKE, $\overline{SRAS}$, $\overline{SCAS}$

5 Capacitance $C_{IN}$ is periodically sampled rather than 100% tested.

## 20.1.1 Supply Voltage Sequencing and Separation Cautions

Figure 20-1 shows two situations to avoid in sequencing the $IV_{cc}$ and $EV_{cc}$ supplies.




**Notes:**

1 $IV_{cc}$, $PV_{cc}$ rising before $EV_{cc}$

2 $EV_{cc}$ rising much faster than $IV_{cc}$, $PV_{cc}$

**Figure 20-1. Supply Voltage Sequencing and Separation Cautions**

$IV_{cc}$ should not be allowed to rise early (1). This is usually avoided by running the regulator for the $IV_{cc}$ supply (1.8 V) from the voltage generated by the 3.3-V $EV_{cc}$ supply (Figure 20-2). This keeps $IV_{cc}$ from rising faster than $EV_{cc}$.

$IV_{cc}$ should not rise so late that a large voltage difference is allowed between the two supplies (2). Typically this situation is avoided by using external discrete diodes in series between supplies as shown in Figure 20-2. The series diodes forward bias when the difference between $EV_{cc}$ and $IV_{cc}$ reaches approximately 2.1V, causing $IV_{cc}$ to rise as $EV_{cc}$ ramps up. When the $IV_{cc}$ regulator begins proper operation, the difference between supplies should not exceed 1.5 V and conduction through the diode chain reduces to essentially leakage current. During supply sequencing, the following general relationship should be adhered to: $EV_{cc} \geq IV_{cc} \geq (EV_{cc} - 2.1\ V)$. The PLL Vdd ($PV_{cc}$) supply should comply with these constraints just as $IV_{cc}$ does. In practice, $PV_{cc}$ is typically connected directly to $IV_{cc}$ with some filtering.

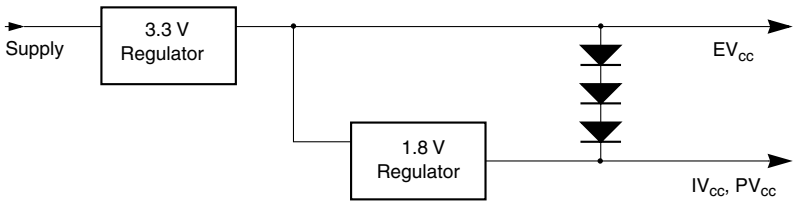



**Figure 20-2. Example Circuit to Control Supply Sequencing**

## 20.2 Clock Timing Specifications

Table 20-4 shows the MCF5407 PLL encodings. Note that they differ from the MCF5307 DIVIDE[1:0] encodings.

**Table 20-4. Divide Ratio Encodings**

| D[2:0]/DIVIDE[2:0] | Input Clock (MHz) | Multiplier | Core Clock (MHz) | PSTCLK (MHz) |
|---|---|---|---|---|
| 00x–010 | Reserved | | | |
| 011 | 40.0–54.0 | 3 | 120.0–162 | 60.0–81.0 |
| 100 | 25.0–40.5 | 4 | 100.0–162 | 50.0–81.0 |
| 101 | 25.0–32.4 | 5 | 125.0–162 | 67.5–81.0 |
| 110 | 25.0–27.0 | 6 | 150.0–162 | 75.0–81.0 |
| 111 | Reserved | | | |

Figure 20-3 correlates CLKIN and core clock frequencies for the 3x–6x multipliers.

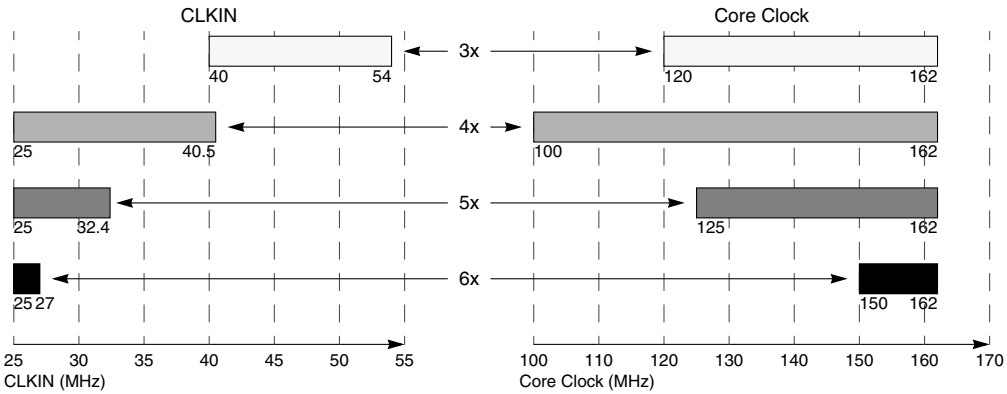



**Figure 20-3. CLKIN-to-Core Clock Frequency Ranges**

Table 20-5 lists specifications for the clock timing parameters shown in Figure 20-4 and Figure 20-5. Motorola recommends that CLKIN be used for the system clock. BCLKO is provided only for compatibility with slower MCF5307 designs. Regardless of the CLKIN frequency driven at power-up, CLKIN (and BCLKO) have the same ratio value to the

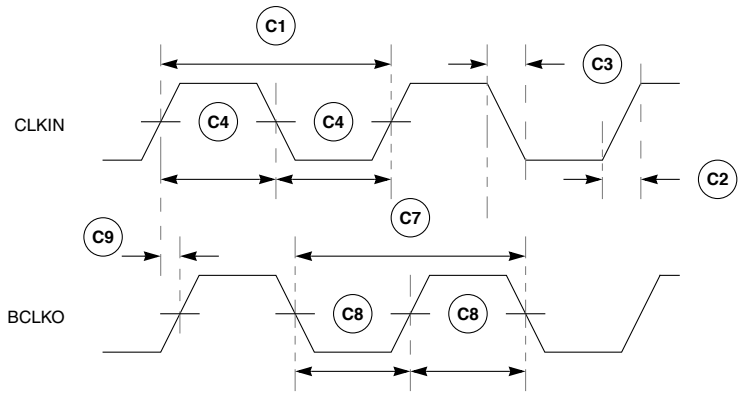
PCLK. Although either signal can be used as a clock reference, CLKIN leaves more room to meet the bus specifications than BCLKO, which is generated as a phase-aligned signal to CLKIN.

**Table 20-5. Clock Timing Specification**

| Num | Characteristic | 54 MHz CLKIN | | Units |
|---|---|---|---|---|
| | | Min | Max | |
| C1 | CLKIN cycle time | 18.5 | Note [1] | nS |
| C2 | CLKIN rise time (0.5V to 2.4 V) | — | 2 | nS |
| C3 | CLKIN fall time (2.4V to 0.5 V) | — | 2 | nS |
| C4 | CLKIN duty cycle (at 1.5 V) | 40 | 60 | % |
| C5 | PSTCLK cycle time | 12.3 | Note [1] | nS |
| C6 | PSTCLK duty cycle (at 1.5 V) | 40 | 60 | % |
| C7 | BCLKO cycle time | 18.5 | Note [1] | nS |
| C8 | BCLKO duty cycle (at 1.5 V) | 45 | 55 | % |
| C9 | CLKIN to BCLKO | -1.5 | 1.5 | nS |

[1] The PLL low-frequency limit depends on the clock divide ratio chosen. See Table 20-4.

Figure 20-4 shows timings for the parameters listed in Table 20-5.




Note: Input and output AC timing specifications are measured to CLKIN with a 50-pF load capacitance (not including pin capacitance).

**Figure 20-4. Clock Timing**

Figure 20-5 shows PSTCLK timings for parameters listed in Table 20-5.

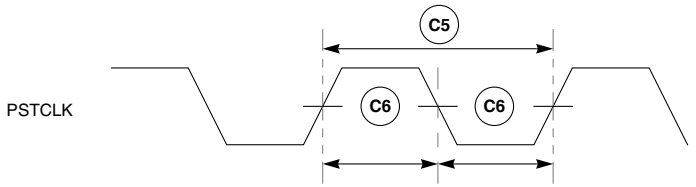



**Figure 20-5. PSTCLK Timing**

## 20.3 Input/Output AC Timing Specifications

Table 20-6 lists specifications for parameters shown in Figure 20-6 and Figure 20-7. Note that inputs $\overline{\text{IRQ}}$[7,5,3,1], $\overline{\text{BKPT}}$, and $\overline{\text{AS}}$ are synchronized internally; that is, the logic level is validated if the value does not change for two consecutive rising CLKIN edges. Setup and hold times must be met only if recognition on a particular clock edge is required.

**Table 20-6. Input AC Timing Specification**

| Num | Characteristic | 54 MHz CLKIN | | Units |
|---|---|---|---|---|
| | | Min | Max | |
| B1 [1] | Valid to CLKIN rising (setup) | 7.5 | — | nS |
| B2 [1] | CLKIN rising to invalid (hold) | 1.0 | — | nS |
| B3 [2] | Valid to CLKIN rising (setup) | 0 | — | nS |
| B4 [2] | CLKIN rising to invalid (hold) | 0.5(C1) + 1.3 | — | nS |
| B5 [3] | CLKIN to input high impedance | — | 2 | Bus clock |
| B6 | CLKIN to EDGESEL delay | 0 | 5.0 | nS |

1 Inputs: $\overline{\text{BG}}$, $\overline{\text{TA}}$, A[23:0], PP[15:0], SIZ[1:0], R/$\overline{\text{W}}$, $\overline{\text{TS}}$, EDGESEL, D[31:0], $\overline{\text{IRQ}}$[7,5,3,1], and $\overline{\text{BKPT}}$

2 Inputs: $\overline{\text{AS}}$

3 Inputs: D[31:0]

Table 20-7 lists specifications for timings in Figure 20-6, Figure 20-7, and Figure 20-13. Although output signals that share a specification number have approximately the same timing, due to loading differences, they do not necessarily change at the same time. However, they have similar timings; that is, minimum and maximum times are not mixed.

**Table 20-7. Output AC Timing Specification**

| Num | Characteristic | 54 MHz CLKIN | | Units |
|---|---|---|---|---|
| | | Min | Max | |
| B10 [1,2,3] | CLKIN rising to valid | — | 8 [4] | nS |
| | | — | 10 [5] | nS |
| B11 [3,4,5] | CLKIN rising to invalid (hold) | 1.0 [5] | — | nS |
| B12 [6,7] | CLKIN to high impedance (three-state) | — | 10 | nS |

**Table 20-7. Output AC Timing Specification (Continued)**

| Num | Characteristic | 54 MHz CLKIN | | Units |
|---|---|---|---|---|
| | | Min | Max | |
| B13 [8,2,3] | CLKIN rising to valid | — | 0.5(C1) +8.0 [9] | nS |
| | | — | 0.5(C1) +10.0 [10] | nS |
| B14 [8,2,3] | CLKIN rising to invalid (hold) | 0.5(C1) + 1.0 | — | nS |
| B15 [2,3] | EDGESEL to valid | — | 12 | nS |
| B16 [2,3] | EDGESEL to invalid (hold) | 2 | — | nS |
| H1 | $\overline{HIZ}$ to high impedance | — | 60 | nS |
| H2 | $\overline{HIZ}$ to low Impedance | — | 60 | nS |

1 Outputs that change only on rising edge of CLKIN: $\overline{RSTO}$, $\overline{TS}$, $\overline{BR}$, $\overline{BD}$, $\overline{TA}$, R/$\overline{W}$, SIZ[1:0], PP[7:0] (and PP[15:8] when configured as parallel port outputs).

2 Outputs that can change on either CLKIN edge depending only on EDGESEL: D[31:0], A[23:0], SCKE, $\overline{SRAS}$, $\overline{SCAS}$, and $\overline{DRAMW}$ and on PP[15:8] when individually configured as A[31:24] outputs.

3 Outputs that can change on either CLKIN edge depending upon EDGESEL and the interface operating mode (DRAM/SDRAM): $\overline{RAS}$[1:0], $\overline{CAS}$[3:0]

4 $\overline{SRAS}$, $\overline{SCAS}$, $\overline{DRAMW}$, $\overline{RAS}$[1:0], $\overline{CAS}$[3:0]

5 D[31:0], A[23:0], TM[2:0], TT[1:0], SIZ[1:0], R/$\overline{W}$, $\overline{TIP}$, $\overline{TS}$, $\overline{BR}$, $\overline{BD}$, and $\overline{TA}$ and PP[15:8] when individually configured as A[31:24] outputs.

6 High impedance (three-state): D[31:0]

7 Outputs that transition to high impedance due to bus arbitration: A[23:0], R/$\overline{W}$, SIZ[1:0], $\overline{TS}$, $\overline{AS}$, and $\overline{TA}$, and PP[15:8] when individually configured as A[31:24] outputs.

8 Outputs that change only on falling edge of CLKIN: $\overline{AS}$, $\overline{CS}$[7:0], $\overline{BE}$[3:0], $\overline{OE}$

9 $\overline{SRAS}$, $\overline{SCAS}$, $\overline{DRAMW}$, $\overline{RAS}$[1:0], $\overline{CAS}$[3:0], $\overline{AS}$, $\overline{CS}$[7:0], $\overline{BE}$[3:0], $\overline{OE}$

10 D[31:0], A[23:0], TM[2:0], TT[1:0], SIZ[1:0], R/$\overline{W}$, $\overline{TIP}$, and $\overline{TS}$ and on PP[15:8] when individually configured as A[31:24] outputs.

Note that these figures show two representative bus operations and do not attempt to show all cases. For explanations of the states, S0–S5, see Section 18.4, “Data Transfer Operation.” Note that Figure 20-7 does not show all signals that apply to each timing specification. See the previous tables for a complete listing.

Figure 20-6 shows AC timings for normal read and write bus cycles.

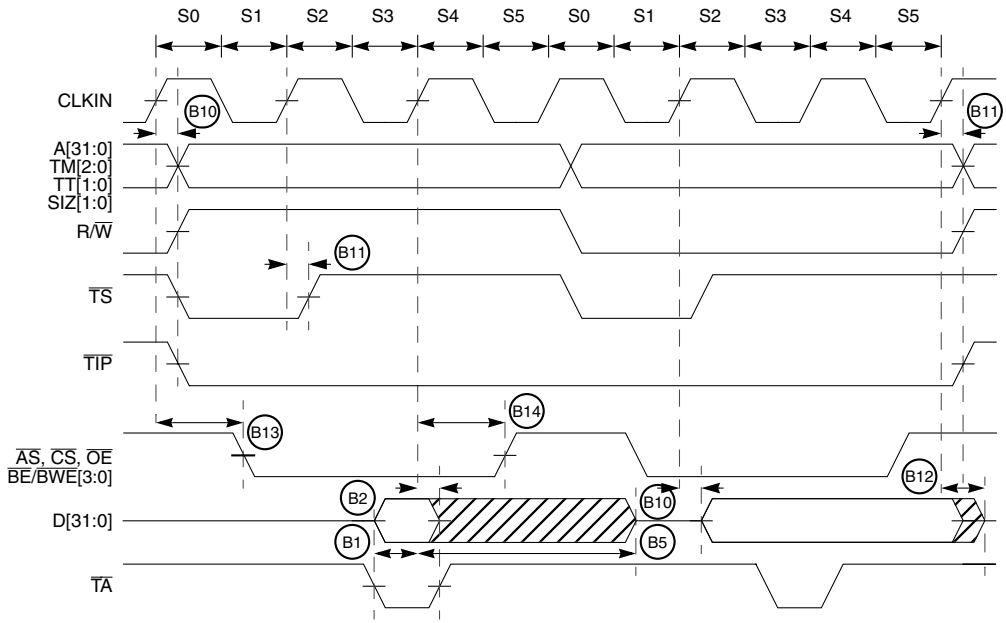



**Figure 20-6. AC Timings—Normal Read and Write Bus Cycles**

Figure 20-7 shows timings for a read cycle with EDGESEL tied to buffered CLKIN.

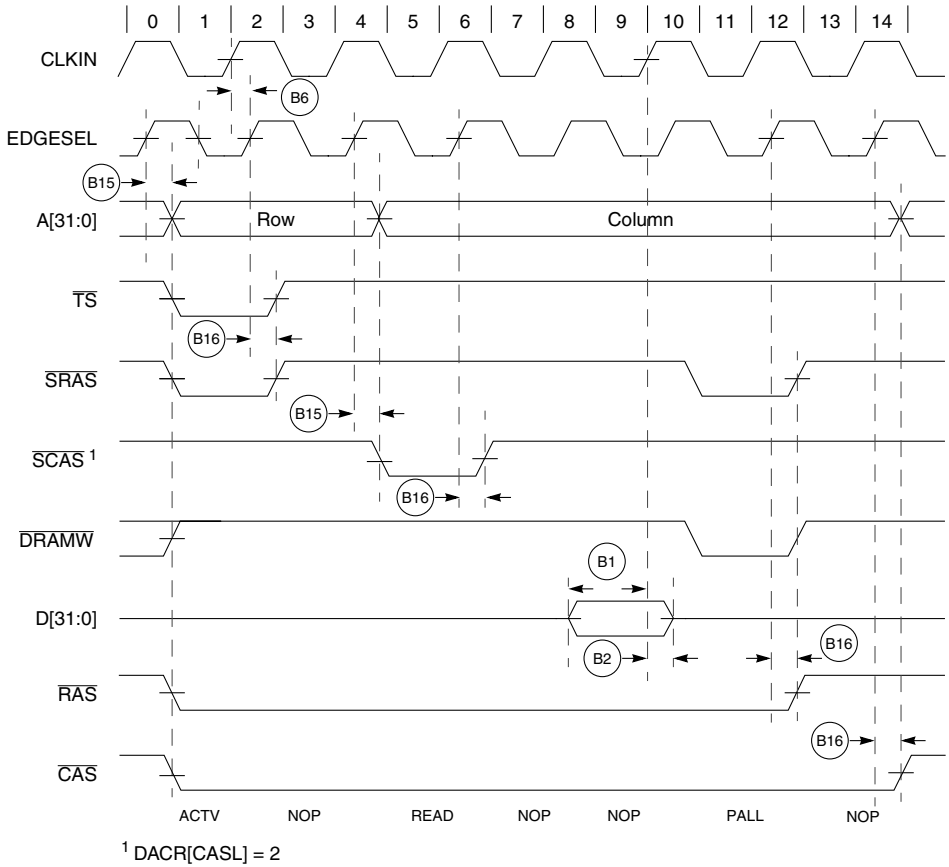



**Figure 20-7. SDRAM Read Cycle with EDGESEL Tied to Buffered CLKIN**

Figure 20-8 shows an SDRAM write cycle with EDGESEL tied to buffered CLKIN.

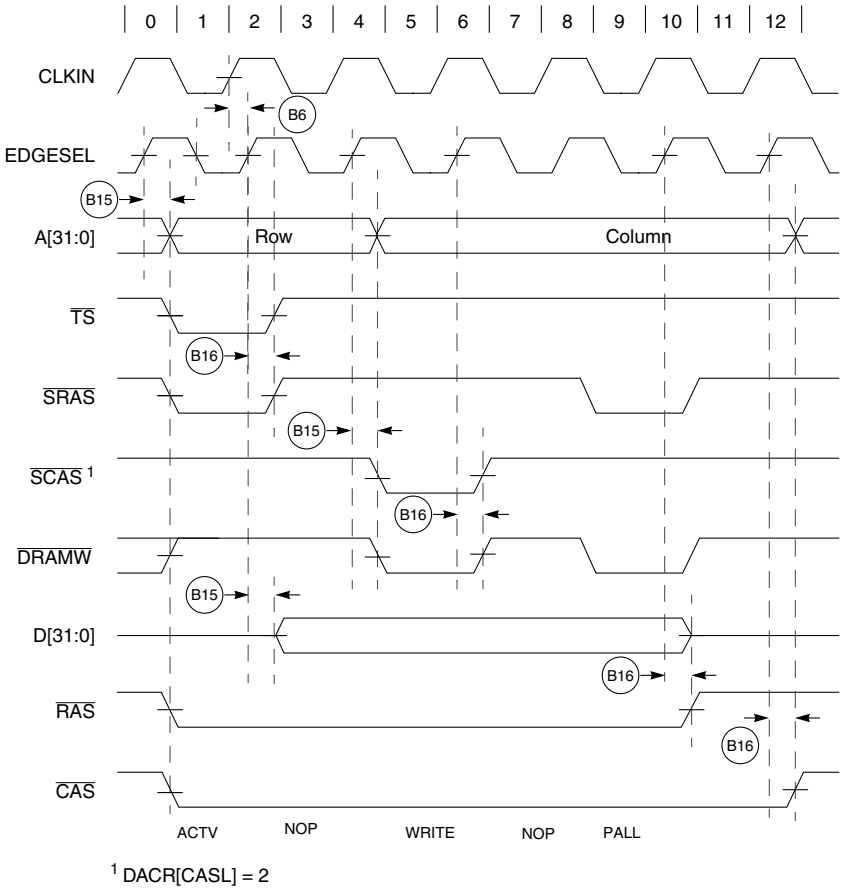



**Figure 20-8. SDRAM Write Cycle with EDGESEL Tied to Buffered CLKIN**

Figure 20-9 shows an SDRAM read cycle with EDGESEL tied high.

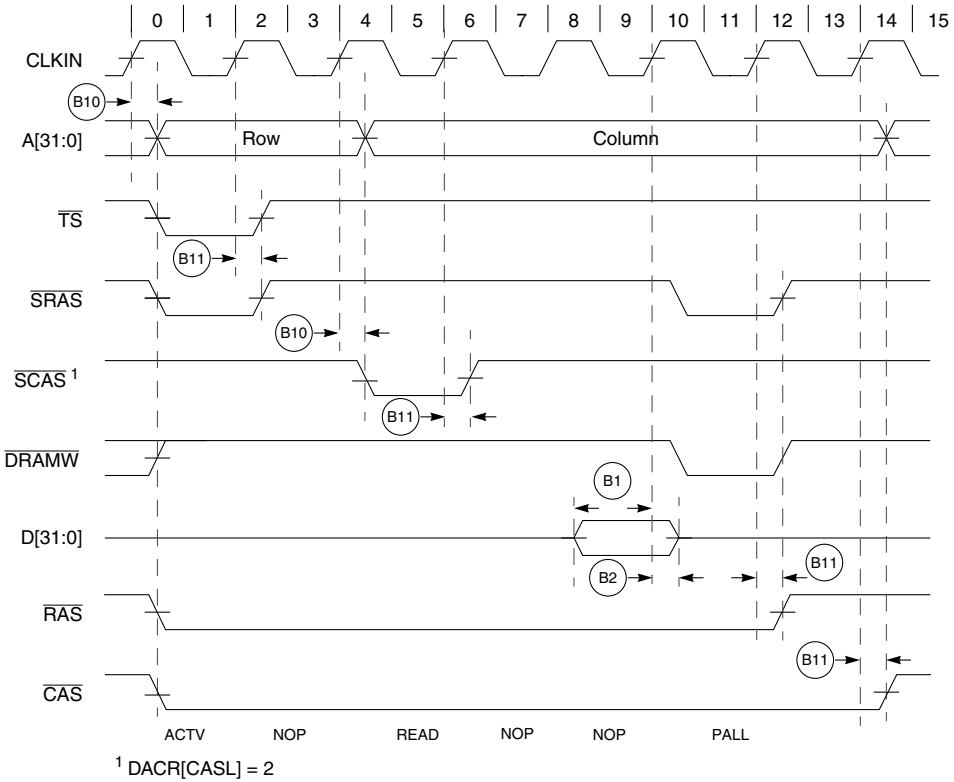



**Figure 20-9. SDRAM Read Cycle with EDGESEL Tied High**

Figure 20-10 shows an SDRAM write cycle with EDGESEL tied high.

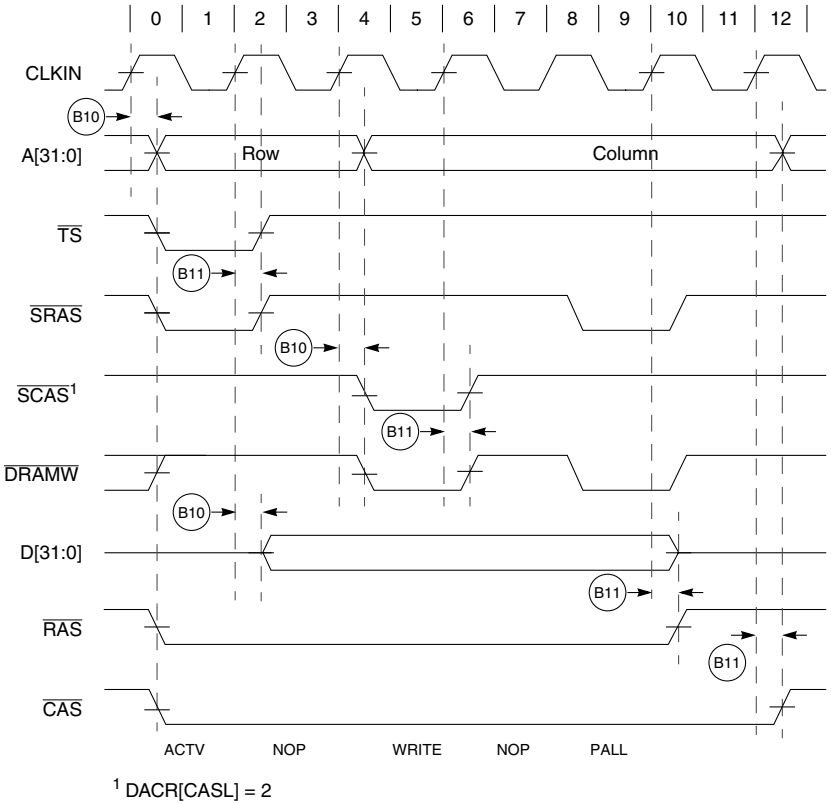



**Figure 20-10. SDRAM Write Cycle with EDGESEL Tied High**

Figure 20-11 shows an SDRAM read cycle with EDGESEL tied low.

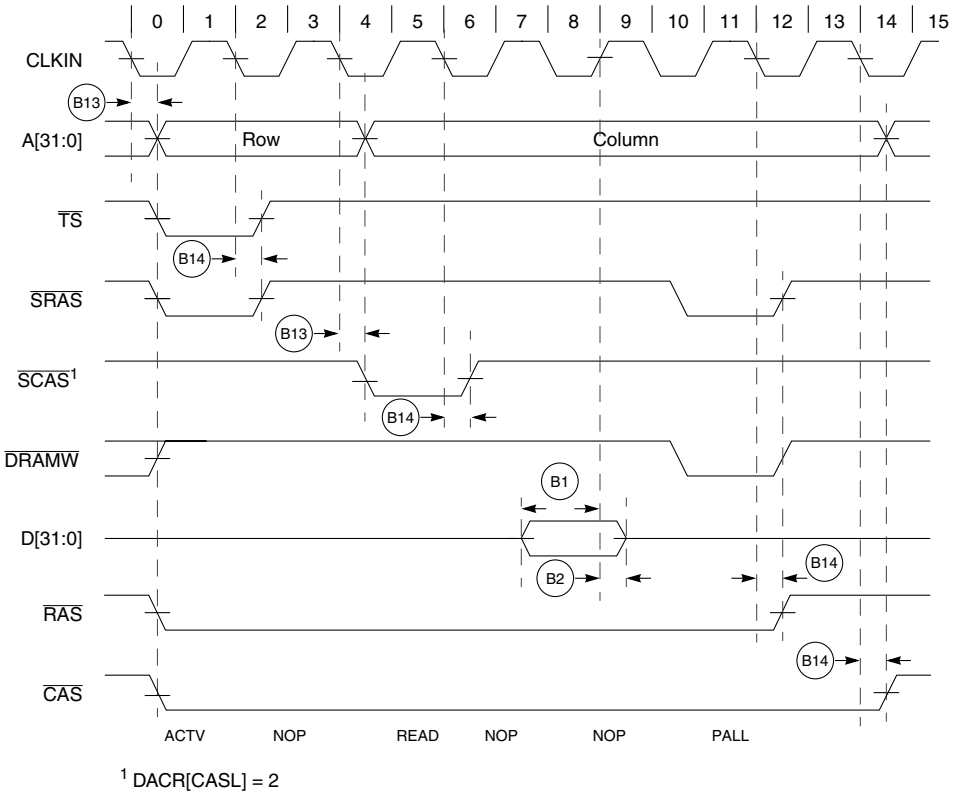



**Figure 20-11. SDRAM Read Cycle with EDGESEL Tied Low**

Figure 20-12 shows an SDRAM write cycle with EDGESEL tied low.

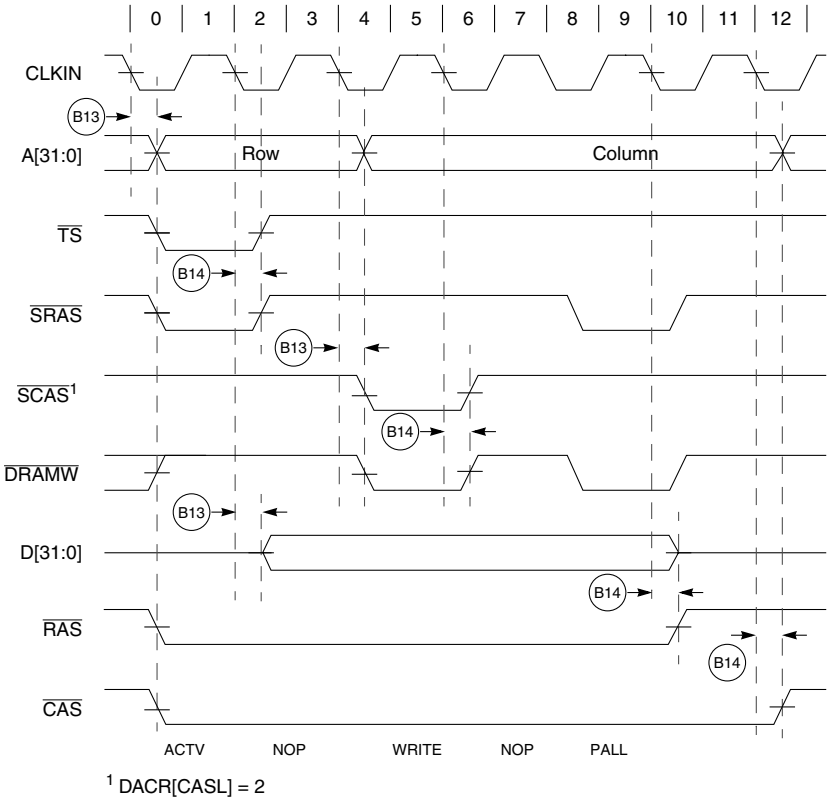



**Figure 20-12. SDRAM Write Cycle with EDGESEL Tied Low**

Figure 20-13 shows AC timing showing high impedance.

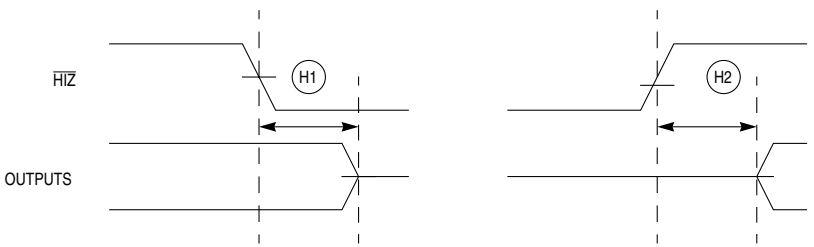



**Figure 20-13. AC Output Timing—High Impedance**

## 20.4 Reset Timing Specifications

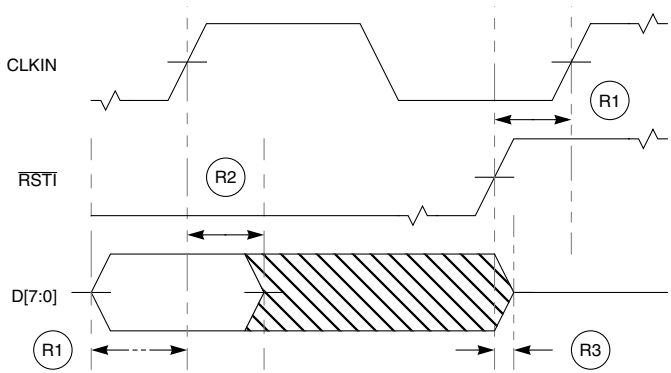
Table 20-8 lists specifications for the reset timing parameters shown in Figure 20-14.

**Table 20-8. Reset Timing Specification**

| Num | Characteristic | 54 MHz CLKIN | | Units |
|---|---|---|---|---|
| | | Min | Max | |
| R1 [1] | Valid to CLKIN (setup) | 7.5 | — | nS |
| R2 | CLKIN to invalid (hold) | 1.0 | — | nS |
| R3 | $\overline{\text{RSTI}}$ to invalid (hold) | 1.0 | — | nS |

[1] $\overline{\text{RSTI}}$ and D[7:0] are synchronized internally. Setup and hold times must be met only if recognition on a particular clock is required.

Figure 20-14 shows reset timing for the values in Table 20-8.




Note: Mode selects are registered on the rising CLKIN edge before the cycle in which $\overline{\text{RSTI}}$ is recognized as being negated.

**Figure 20-14. Reset Timing**

# 20.5 Debug AC Timing Specifications

Table 20-9 lists specifications for the debug AC timing parameters shown in Figure 20-16.

**Table 20-9. Debug AC Timing Specification**

| Num | Characteristic | 54 MHz CLKIN | | Units |
|---|---|---|---|---|
| | | Min | Max | |
| D1 | PSTDDATA to PSTCLK setup | 4.5 | — | nS |
| D2 | PSTCLK to PSTDDATA hold | 4.5 | — | nS |
| D3 | DSI-to-DSCLK setup | 1 | — | PSTCLKs |
| D4 [1] | DSCLK-to-DSO hold | 4 | — | PSTCLKs |
| D5 | DSCLK cycle time | 5 | — | PSTCLKs |

[1] DSCLK and DSI are synchronized internally. D4 is measured from the synchronized DSCLK input relative to the rising edge of PSTCLK.

Figure 20-15 shows real-time trace timing for the values in Table 20-9.

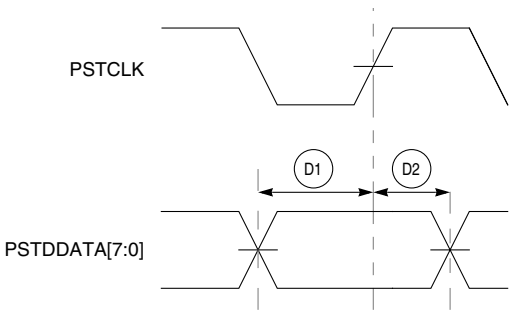



**Figure 20-15. Real-Time Trace AC Timing**

Figure 20-16 shows BDM serial port AC timing for the values in Table 20-9.

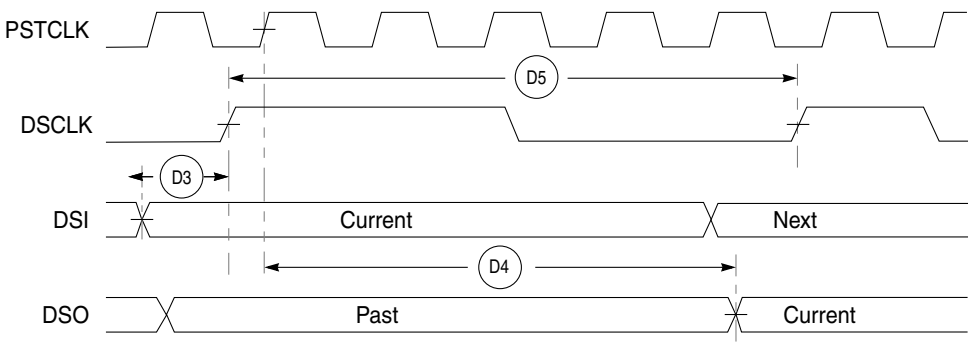



**Figure 20-16. BDM Serial Port AC Timing**

## 20.6 Timer Module AC Timing Specifications

Table 20-10 lists specifications for timer module AC timing parameters shown in Figure 20-17.

**Table 20-10. Timer Module AC Timing Specification**

| Num | Characteristic | 54 MHz CLKIN | | Units |
|---|---|---|---|---|
| | | Min | Max | |
| T1 | TIN cycle time | 3 | — | Bus clocks |
| T2 | TIN valid to CLKIN (input setup) | 7.5 | — | nS |
| T3 | CLKIN to TIN invalid (input hold) | 1.0 | — | nS |
| T4 | CLKIN to TOUT valid (output valid) | — | 10 | nS |
| T5 | CLKIN to TOUT invalid (output hold) | 1.0 | — | nS |
| T6 | TIN pulse width | 1 | — | Bus clocks |
| T7 | TOUT pulse width | 1 | — | Bus clocks |

Figure 20-17 shows timings for Table 20-10.

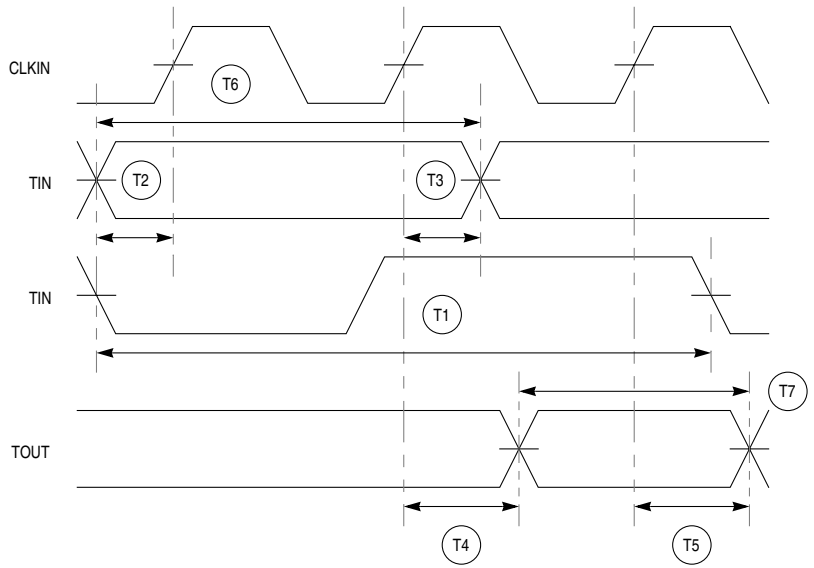



**Figure 20-17. Timer Module AC Timing**

# 20.7 I²C Input/Output Timing Specifications

Table 20-11 lists specifications for the I²C input timing parameters shown in Figure 20-18.

**Table 20-11. I²C Input Timing Specifications between SCL and SDA**

| Num | Characteristic | 54 MHz CLKIN | | Units |
|---|---|---|---|---|
| | | Min | Max | |
| I1 | Start condition hold time | 2 | — | Bus clocks |
| I2 | Clock low period | 8 | — | Bus clocks |
| I3 | SCL/SDA rise time ($V_{IL}$ = 0.5 V to $V_{IH}$ = 2.4 V) | — | 1 | mS |
| I4 | Data hold time | 0 | — | nS |
| I5 | SCL/SDA fall time ($V_{IH}$ = 2.4 V to $V_{IL}$ = 0.5 V) | — | 1 | mS |
| I6 | Clock high time | 4 | — | Bus clocks |
| I7 | Data setup time | 0 | — | nS |
| I8 | Start condition setup time (for repeated start condition only) | 2 | — | Bus clocks |
| I9 | Stop condition setup time | 2 | — | Bus clocks |

Table 20-12 lists specifications for the I²C output timing parameters shown in Figure 20-18.

**Table 20-12. I²C Output Timing Specifications between SCL and SDA**

| Num | Characteristic | 54 MHz CLKIN | | Units |
|---|---|---|---|---|
| | | Min | Max | |
| I1 [1] | Start condition hold time | 6 | — | Bus clocks |
| I2 [1] | Clock low period | 10 | — | Bus clocks |
| I3 [2] | SCL/SDA rise time ($V_{IL}$ = 0.5 V to $V_{IH}$ = 2.4 V) | Note 2 | Note 2 | |
| I4 [1] | Data hold time | 7 | — | Bus clocks |
| I5 [3] | SCL/SDA fall time ($V_{IH}$ = 2.4 V to $V_{IL}$ = 0.5 V) | — | 3 | nS |
| I6 [1] | Clock high time | 10 | — | Bus clocks |
| I7 [1] | Data setup time | 2 | — | Bus clocks |
| I8 [1] | Start condition setup time (for repeated start condition only) | 20 | — | Bus clocks |
| I9 [1] | Stop condition setup time | 10 | — | Bus clocks |

1 Programming IFDR with the maximum frequency (IFDR = 0x20) results in the minimum output timings listed here. The I²C interface is designed to scale the data transition time, moving it to the middle of the SCL low period. The actual position is affected by the prescale and division values programmed in IFDR.

2 Because SCL and SDA are open-collector-type outputs, which the processor can only actively drive low, the time SCL or SDA takes to reach a high level depends on external signal capacitance and pull-up resistor values.

3 Specified at a nominal 50-pF load.

Figure 20-18 shows timing for the values in Table 20-11 and Table 20-12.

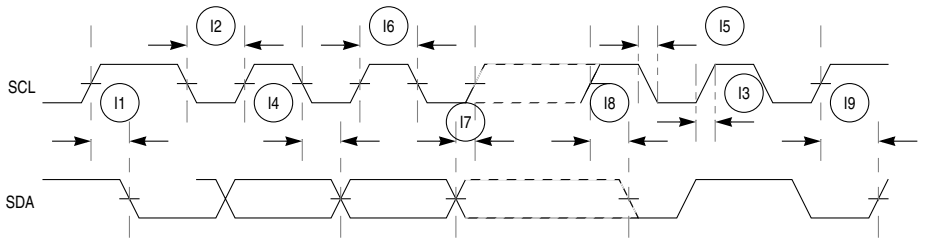



**Figure 20-18. I²C Input/Output Timings**

## 20.8 UART Module AC Timing Specifications

Table 20-13 lists specifications for UART module AC timing parameters in Figure 20-19.

**Table 20-13. UART Module AC Timing Specifications**

| Num | Characteristic | 54 MHz CLKIN | | Units |
|---|---|---|---|---|
| | | Min | Max | |
| U1 | RXD valid to CLKIN (input setup) | 7.5 | — | nS |
| U2 | CLKIN to RXD invalid (input hold) | 1.0 | — | nS |
| U3 | $\overline{CTS}$ valid to CLKIN (input setup) | 7.5 | — | nS |
| U4 | CLKIN to $\overline{CTS}$ invalid (input hold) | 1.0 | — | nS |
| U5 | CLKIN to TXD valid (output valid) | — | 10 | nS |
| U6 | CLKIN to TXD invalid (output hold) | 1.0 | — | nS |
| U7 | CLKIN to $\overline{RTS}$ valid (output valid) | — | 10 | nS |
| U8 | CLKIN to $\overline{RTS}$ invalid (output hold) | 1.0 | — | nS |
| U9 | $\overline{CTS}$ high time | 38 | — | nS |
| U10 | $\overline{CTS}$ low time | 38 | — | nS |
| U11 | $\overline{CTS}$ rising to TxD valid | — | 20 | nS |
| U12 | RxD setup to $\overline{CTS}$ falling | 10 | — | nS |
| U13 | RxD hold from $\overline{CTS}$ falling | — | 5 | nS |
| U14 | TxD to RxD (remote loop back) | — | 15 | nS |
| U15 | TIN1 setup to $\overline{CTS}$ falling | 10 | — | nS |
| U16 | TIN1 hold from $\overline{CTS}$ falling | — | 5 | nS |
| U17 | $\overline{CTS}$ rising to $\overline{RTS}$ asserted | — | 20 | nS |

Figure 20-19 shows UART0 and UART1 timing for the values in Table 20-13.

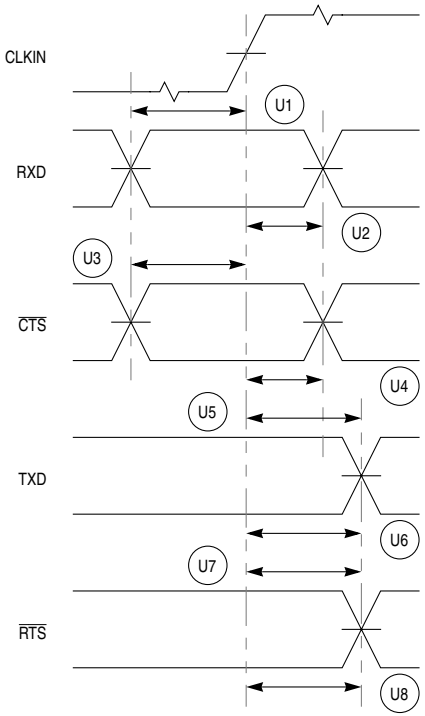



**Figure 20-19. UART0 and UART1 Module AC Timing—UART Mode**

Figure 20-19 shows timing for UART1 in 8- and 16-bit CODEC mode.

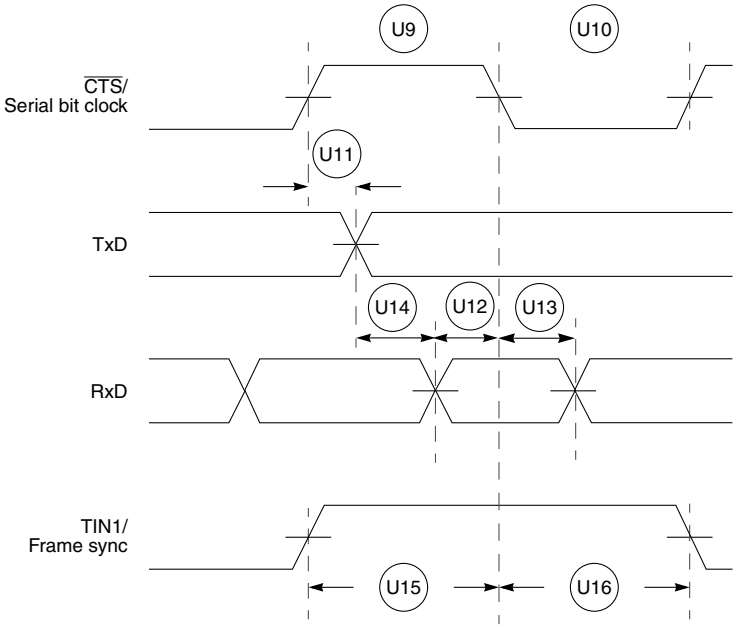



**Figure 20-20. UART1 in 8- and 16-bit CODEC Mode**

Figure 20-21 shows timing for UART1 in AC ‘97 mode.

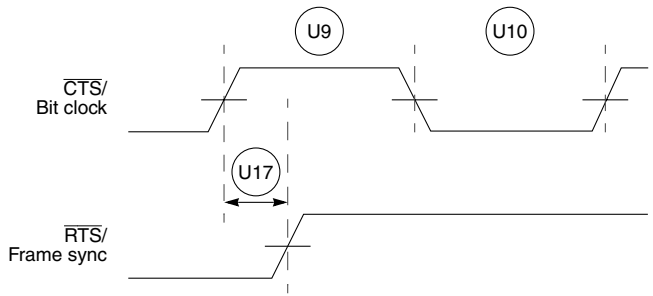



**Figure 20-21. UART1 in AC ‘97 Mode**

# 20.9 Parallel Port (General-Purpose I/O) Timing Specifications

Table 20-14 lists specifications for general-purpose I/O timing parameters in Figure 20-22.

**Table 20-14. General-Purpose I/O Port AC Timing Specifications**

| Num | Characteristic | 54 MHz CLKIN | | Units |
|---|---|---|---|---|
| | | Min | Max | |
| P1 | PP valid to CLKIN (input setup) | 7.5 | — | nS |
| P2 | CLKIN to PP invalid (input hold) | 1.0 | — | nS |
| P3 | CLKIN to PP valid (output valid) | — | 10 | nS |
| P4 | CLKIN to PP invalid (output hold) | 1.0 | — | nS |

Figure 20-22 shows general-purpose I/O timing.

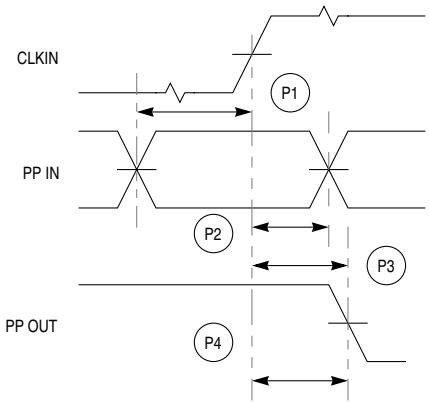



**Figure 20-22. General-Purpose I/O Timing**

## 20.10 DMA Timing Specifications

Table 20-14 lists specifications for DMA timing parameters shown in Figure 20-22.

**Table 20-15. DMA AC Timing Specifications**

| Num | Characteristic | 54 MHz CLKIN | | Units |
|---|---|---|---|---|
| | | Min | Max | |
| M1 | DREQ valid to CLKIN (input setup) | 7.5 | — | nS |
| M2 | CLKIN to DREQ invalid (input hold) | 1.0 | — | nS |
| M3 | CLKIN to DACK valid (output valid) | — | 10 | nS |
| M4 | CLKIN to DACK invalid (output hold) | 1.0 | — | nS |

Figure 20-23 shows DMA AC timing.

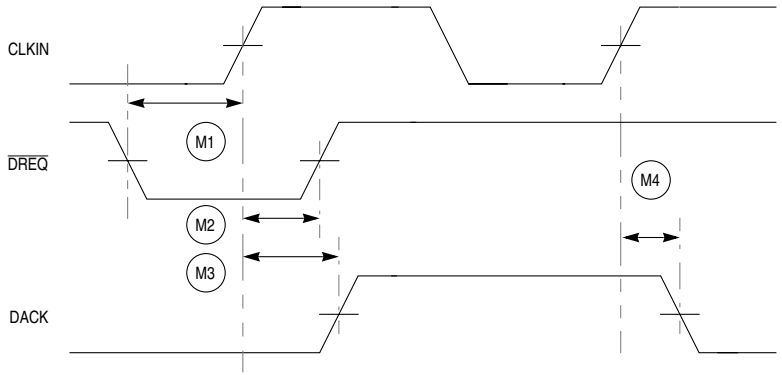



**Figure 20-23. DMA Timing**

# 20.11 IEEE 1149.1 (JTAG) AC Timing Specifications

Table 20-16 lists specifications for JTAG AC timing parameters shown in Figure 20-24.

**Table 20-16. IEEE 1149.1 (JTAG) AC Timing Specifications**

| Num | Characteristic | All Frequencies | | Units |
|---|---|---|---|---|
| | | Min | Max | |
| — | TCK frequency of operation | 0 | 10 | MHz |
| J1 | TCK cycle time | 100 | — | nS |
| J2a | TCK clock pulse high width (measured at 1.5 V) | 40 | — | nS |
| J2b | TCK clock pulse low width (measured at 1.5 V) | 40 | — | nS |
| J3a | TCK fall time ($V_{IH}$ = 2.4 V to $V_{IL}$ = 0.5V) | — | 5 | nS |
| J3b | TCK rise time ($V_{IL}$ = 0.5v to $V_{IH}$ = 2.4V) | — | 5 | nS |
| J4 | TDI, TMS to TCK rising (input setup) | 10 | — | nS |
| J5 | TCK rising to TDI, TMS invalid (hold) | 15 | — | nS |
| J6 | Boundary scan data valid to TCK (setup) | 10 | — | nS |
| J7 | TCK to boundary-scan data invalid (hold) | 15 | — | nS |
| J8 | $\overline{\text{TRST}}$ pulse width (asynchronous to clock edges) | 15 | — | — |
| J9 | TCK falling to TDO valid (signal from driven or three-state) | — | 30 | nS |
| J10 | TCK falling to TDO high impedance | — | 30 | nS |
| J11 | TCK falling to boundary scan data valid (signal from driven or three-state) | — | 30 | nS |
| J12 | TCK falling to boundary scan data high impedance | — | 30 | nS |

Figure 20-24 shows JTAG timing.

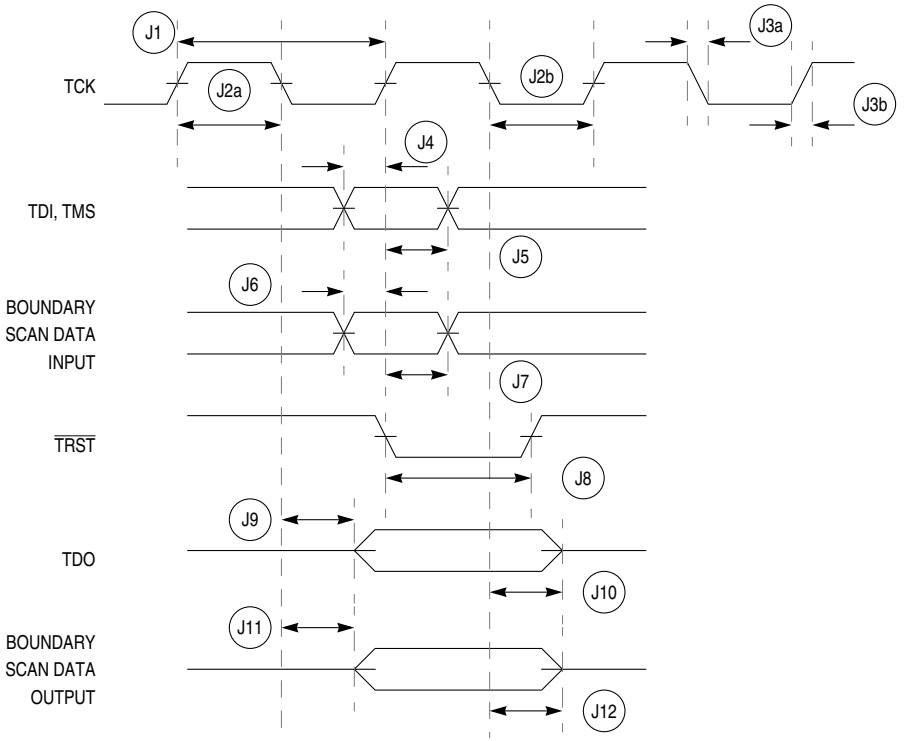



**Figure 20-24. IEEE 1149.1 (JTAG) AC Timing**

# Appendix A
# Migrating from the ColdFire MCF5307 to the MCF5407

This appendix highlights the differences between the MCF5307B and MCF5407. Users of the MCF5307 and MCF5307A should use this document in conjunction with the *MCF5307 User's Manual Mask Set Addendum*. For additional information, see the *MCF5407 Integrated ColdFire Microprocessor Product Brief*.

## A.1 Overview

Customers using the integrated peripherals of the MCF5307 can access the same features on the MCF5407 with the added advantage of increased cache and RAM memories as well as an enhanced instruction set architecture (ISA), DMA, synchronous UART, and debug functionality. Decreased voltage requirements allow designers to take advantage of other low-voltage components on the board for an integrated, low-power system.

To migrate designs from the MCF5307 to the MCF5407, note the minor differences between these code-compatible processors in the initialization code, power supplies, and clock inputs. This document describes the differences between the processors and outlines the steps to upgrade the design. Table A-1 is a quick reference chart of these differences.

**Table A-1. Differences between MCF5307 and MCF5407**

| Feature | MCF5307 | MCF5407 | Reference |
|---|---|---|---|
| Version core | ColdFire Version 3 (V3) | ColdFire Version 4 | — |
| MIPS | 70 MIPS at 90-MHz core clock | 233 MIPS at 162-MHz core clock | — |
| Instruction set | Baseline ColdFire ISA Rev A is used in Version 2 and Version 3 core. | ColdFire ISA Rev B which includes certain instruction enhancements and some instruction additions; V2/V3 ISA Rev A is upward-compatible with ISA Rev B. | Section A.2, “Instruction Set Additions” |

**Table A-1. Differences between MCF5307 and MCF5407**

| Feature | MCF5307 | MCF5407 | Reference |
|---|---|---|---|
| Caches | 8-Kbyte unified cache | 16-Kbyte instruction cache<br>8-Kbyte data cache | Section A.3, "Enhanced Memories" |
| | Two cache access control registers (ACR0/ACR1) | ACR0/ACR1 configure data space;<br>ACR2/ACR3 configure instruction space | |
| | 4-Kbyte SRAM | Two independently configurable 2-Kbyte SRAMs | |
| | No cache locking | Ability to lock all or half of the caches to prevent instructions or data from being cast out. This is useful for deterministic code. | |
| DMA modifications | DMA acknowledge assertion is encoded on TM[2:0]. | DACK[1:0] multiplexed on TM[1:0] can be programmed as separate DMA acknowledge signals.<br>DMA TM[2:0] encodings are different from MCF5307 DMA TM[2:0] encodings. | Section A.4, "On-Chip DMA Modifications" |
| | DMA byte count register (BCR) can be programmed to be 16 or 24 bits. | BCR is 24 bits only. | |
| UART | Both UARTs have identical functionality. No support for synchronous mode. | UART0 is identical to the MCF5307 UARTs; UART1 has been enhanced to provide synchronous operation and a CODEC interface for soft modem support. | Section A.5, "UART Enhancements" |
| Timing relationships | All signal timing with respect to BCLKO; CLKIN rise time = 5 nS. | All signal timings with respect to CLKIN (BCLKO support provides compatibility with MCF5307 designs.)<br>Tighter negative edge bus specifications due to duty cycle; CLKIN rise time = 2 nS. | Section A.6.1, "Phase-Locked Loop (PLL)," and Section A.6, "Timing Differences" |
| Reset initialization | Need to drive D[7:0]/ AA, PS[1:0], ADDR_CONFIG, FREQ[1:0], DIVIDE[1:0] | Need to drive D[7:0]/AA, PS[1:0], ADDR_CONFIG, BE_CONFIG, DIVIDE[2:0] | Section A.7, "Reset Initialization Modifications" |
| Debug module | Debug Revision B. Separate PST[3:0] and DDATA[3:0] | Debug Revision C—Adds breakpoint registers, normal interrupt request service during debug, and combines debug signals into PSTDDATA[7:0] | Section A.8, "Revision C Debug" |
| Voltage input changes | Drives minimum 2.4 V; accepts 5-V input | Drives minimum 2.4 V; accepts 3.3-V input | Section A.9, "Voltage Input Changes" |
| | Requires 3.3-V operating voltage | Requires 1.8-V and 3.3-V operating voltages | |
| Pin assignment | Standard MCF5307 pinout | Compatible with MCF5307 pinout except for power-pad input assignment | Section A.11, "Pin-Assignment Compatibility" |

## A.2 Instruction Set Additions

The MCF5407 implements Revision B (Rev B) of the ColdFire instruction set, which adds instructions and enhances existing ISA Revision A (Rev A) opcodes to support byte- and

word-sized operands and position-independent code. Existing MCF5307 code is completely upward compatible with the MCF5407. However, designers may incorporate the instruction set additions and enhancements, especially when upgrading 68K code that references 8- and 16-bit short operands.

The following list summarizes new and enhanced instructions of Rev B ISA:

- New instructions:
  - INTOUCH loads instructions one cache block at a time for use with cache locking.
  - MOV3Q.L moves 3-bit immediate data to the destination location.
  - MVS.{B,W} moves the sign-extended source operand to the destination register.
  - MVZ.{B,W} zero-fills the source operand and moves it to the destination register.
  - SATS.L updates bit 31 of the destination register depending on the CCR overflow bit.
  - TAS.B tests and sets byte operand being addressed.
- Enhancements to existing Revision A instructions:
  - Longword support for branch instructions (Bcc, BRA, BSR)
  - Byte and word support for compare instructions (CMP, CMPI)
  - Byte and longword support for MOVE where the source is of type #<data> and the destination is of type d16(Ax); that is, move.b #<data>, d16(Ax)

Refer to Section 2.9, “ColdFire Instruction Set Architecture Enhancements” for details of these additions and enhancements.

## A.3 Enhanced Memories

With the introduction of a Harvard memory architecture in the Version 4 core design, the MCF5407 has separate instruction and data caches. The 16-Kbyte instruction cache and 8-Kbyte data cache greatly improve performance on existing systems. On-chip RAMs are also provided to work with the caches. For more details see, Chapter 4, “Local Memory”.

The MCF5307 configuration contains an 8-Kbyte unified cache with a 4-Kbyte SRAM. Configuration registers for these memories include one cache control register (CACR), two access control registers (ACR0 and ACR1), and one RAM base address register (RAMBAR). With the enhanced memory sizes of the MCF5407, more configuration registers have been provided. The new MOVEC register map for the MCF5407 memory configuration registers is given in Table A-2.

**Table A-2. MOVEC CPU Space Register Map**

| Rc[1:0] | Register Definition |
|---|---|
| 0x002 | Cache control register (CACR) |
| 0x004 | Cache access control register 0 (ACR0; data cache) |
| 0x005 | Cache access control register 1 (ACR1; data cache) |
| 0x006 | Cache access control register 2 (ACR2; instruction cache) |
| 0x007 | Cache access control register 3 (ACR3; instruction cache) |
| 0xC04 | RAM base address register 0 (RAMBAR0) [1] |
| 0xC05 | RAM base address register 1 (RAMBAR1) [1] |

[1] Either or both of the RAMBAR registers can be configured for instructions or data through an additional bit, RAMBARn[D/I].

Note that the existing functionality has not changed; new registers and new bits in existing registers have been added to support the enhanced memories and control for the new branch cache. One of the two 2-Kbyte SRAMs can be dedicated to support the instruction cache, and the other can support the data cache. Many designs use one SRAM block as a system stack and the other to hold important interrupt service routines.

The SRAM can also function as a ROM by programming it as a data block while loading configuration information to it and then reprogramming it as a read-only instruction block. The two MCF5407 SRAM blocks can be programmed to provide a contiguous 4-Kbyte memory map similar to the MCF5307's single contiguous 4-Kbyte SRAM.

## A.4 On-Chip DMA Modifications

The MCF5407 integrates the four-channel DMA used in the MCF5307 with changes to pin multiplexing, DMA byte transfer count, and the encoding of transfer acknowledgement. The MCF5307 provides DMA acknowledgement encodings for channels 0 and 1 through the transfer modifier pins, TM[2:1], which are multiplexed with PP[4:3]. For clarification on MCF5307 signal multiplexing, see the pinout tables in the mechanical specifications chapter of the *MCF5307 User's Manual*. The MCF5307 also indicates a DMA single address access through transfer modifier pin TM0, multiplexed with PP2. For more details see, Chapter 12, "DMA Controller Module".

When the pin assignment register (PAR) is programmed to enable the TM signals, the encodings listed in Table A-3 and Table A-4 are driven during transfers by the internal DMA channels of the MCF5307. The condition TT[1:0] = 01 indicates an access by either an internal DMA or an external device.

**Table A-3. TM[2:1] Encoding for MCF5307 Internal DMA as Master (TT = 01)**

| TM[2:1] | Transfer Modifier Encoding |
|---|---|
| 00 | DMA acknowledges negated |
| 01 | DMA acknowledge, channel 0 |

**Table A-3. TM[2:1] Encoding for MCF5307 Internal DMA as Master (TT = 01) (Con-**

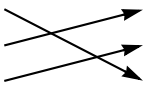
| TM[2:1] | Transfer Modifier Encoding |
|---|---|
| 10 | DMA acknowledge, channel 1 |
| 11 | Reserved |

**Table A-4. TM0 Encoding for MCF5307 Internal DMA as Master (TT = 01)**

| TM0 | Transfer Modifier Encoding |
|---|---|
| 0 | Dual address access |
| 1 | Single address access |

Although the MCF5407 provides similar encodings on TM[2:0], dedicated DMA acknowledgement pins (DACK[1:0]) have been added. Thus, DACK[1:0] are now combined with PP[3:2]/TM[1:0], resulting in a three-to-one multiplexed signal, PP[3:2]/TM[1:0]/DACK[1:0]. TM2 is still multiplexed only with PP4. For further clarification on the multiplexing, see the pinout tables in Section A.11, “Pin-Assignment Compatibility.” When properly connected, TM[2:0] can be used in MCF5407 designs as on MCF5307 designs or DACK[1:0] can be used for DMA transfers, as shown in Figure A-1.

| MCF5307 Function | Pin |
|---|---|
| Single/dual cycle access | TM0 |
| DMA 0 acknowledge configuration | TM1 |
| DMA 1 acknowledge configuration | TM2 |



| Pin | MCF5407 Function |
|---|---|
| TM0 | DMA 0 acknowledge |
| TM1 | DMA 1 acknowledge |
| TM2 | Single/dual cycle access |

**Figure A-1. MCF5307 to MCF5407 TM[2:0] Pin Remapping**

For further details see Section 12.2, “DMA Signal Description”.

Although TM[2:0] can still drive DMA access encoding, the bit positions of these encodings are different from the MCF5307. The MCF5407 encodes single-address accesses on TM2 when the PAR is set to enable the transfer modifier signal and an external master or DMA transfer is occurring. This encoding is driven by TM0 on the MCF5307. Again, more details can be found in Section 12.2, “DMA Signal Description”.

Designers who use MCF5307 DMA channels should also note that the MCF5407 DMA byte count registers (BCRs) for channels 0–3 exclusively support a 24-bit byte count. A 16-bit byte count register is no longer supported; therefore, MPARK[BCR24BIT] has been removed.

## A.5 UART Enhancements

The MCF5407 contains two UARTs that act independently. One of the UARTs on the MCF5407 has been enhanced to provide synchronous operation and a CODEC interface for soft modem support. Each UART can be clocked by the system bus clock, eliminating the need for an external crystal. For more details see, Chapter 14, “UART Modules”.

The UART module interfaces directly to the CPU as shown in Figure A-2. The UART module consists of the following major functional areas:

- Serial communication channel
- 16-bit timer for baud-rate generation
- Internal channel control logic
- Interrupt control logic

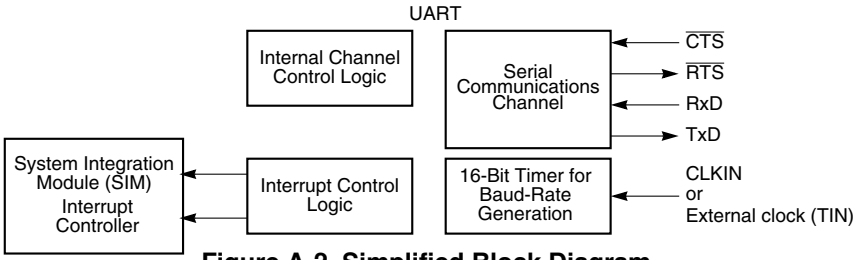



**Figure A-2. Simplified Block Diagram**

In addition, UART1 has been enhanced to provide a CODEC interface for soft modem support. UART1 can be programmed to provide any one of the following functions:

- The original UART (identical to UART0)
- Three modem modes, (see Section 14.5.2.2, “Transmitter in Modem Mode (UART1) for more details)”:
  - An 8-bit CODEC interface
  - A 16-bit CODEC interface
  - An audio CODEC 97 (AC97) digital interface controller

# A.6 Timing Differences

This section explains timing relationships within phase-locked loop registers.

## A.6.1 Phase-Locked Loop (PLL)

The PLL for the MCF5407 is enhanced to support faster processor clock (PCLK) frequencies. The MCF5307 supports PCLK frequencies of 66.7 and 90 MHz with a clock input (CLKIN) of 1/2 PCLK. The MCF5407 offers a larger range of clock input ratios and a higher performance processor clock. For more details see Section 7.1.1, “PLL:PCLK Ratios” and Chapter 20, “Electrical Specifications”.

The MCF5407 PLL module is shown in Figure A-3.

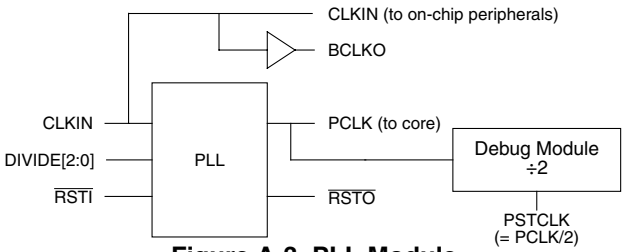



**Figure A-3. PLL Module**

Similar to the MCF5307 functionality, the MCF5407 samples clock ratio encodings on the lower data bits of the bus at reset to determine the CLKIN-to-PCLK ratio at which the device runs. These bits are DIVIDE[1:0] on the MCF5307 and are multiplexed with data bits D[1:0]. Because the MCF5407 offers more divide ratio combinations than the MCF5307, three input bits, D[2:0]/DIVIDE[2:0], have been provided to offer more programming options at reset. Also, note that only specific CLKIN ranges are allowed for each divide ratio on the MCF5407.

Table A-5 shows the new encodings. Note that they differ from the MCF5307 DIVIDE[1:0] encodings.

**Table A-5. Divide Ratio Encodings**

| D[2:0]/DIVIDE[2:0] | Multiplier |
|---|---|
| 00x–010 | Reserved |
| 011 | 3 |
| 100 | 4 |
| 101 | 5 |
| 110 | 6 |
| 111 | Reserved |

## A.6.2 Timing Relationships

For both the MCF5307 and MCF5407, the user provides the clock input signal (CLKIN), which is also used for on-chip peripherals, as shown in Figure A-3. This signal is also the reference from which other clock frequencies are derived, including the bus clock output signal (BCLKO), which on the MCF5407 is provided for compatibility with MCF5307 designs. BCLKO is generated by the PLL and MCF5307 designs should use BCLKO as the bus timing reference for external devices; MCF5407 designs should use CLKIN. On the MCF5407, the CLKIN frequency can be 1/3, 1/4, 1/5, or 1/6 of the PCLK. Furthermore, depending on the MCF5307 configuration, the BCLKO-to-PCLK ratio may not be the same as the CLKIN-to-PCLK ratio. For more details see Section 20.2, “Clock Timing Specifications”.

On the MCF5407, the user-provided CLKIN should be used as the bus clock for the system.

BCLKO runs at the same frequency as CLKIN and is offered as an optional timing reference for backwards compatibility for lower-speed MCF5307 designs.

Regardless of the CLKIN frequency driven at power-up, CLKIN and BCLKO have the same ratio value to PCLK. Although designers can use either BCLKO or CLKIN as a clock reference, Motorola recommends using CLKIN because it leaves more room to meet bus specifications than BCLKO, which is generated as a phase-aligned signal to CLKIN. An MCF5307 user should consider switching to a CLKIN reference clock when upgrading to the MCF5407 if board frequencies exceed 50 MHz.

Although the CLKIN duty cycle remains the same for the MCF5307 and MCF5407, use caution when interfacing signals on the falling edge of CLKIN with only a 4-nS window at high frequencies. Also, note that the MCF5407 input rise time is reduced to 2 nS (5 nS in the MCF5307). For designers who choose to reference signals from CLKIN only, BCLKO can be disabled to save power. For details see Section 7.2.3, “Reduced-Power Mode”.

## A.7 Reset Initialization Modifications

Like the MCF5307, the MCF5407 samples a group of eight input signals, D[7:0], on the rising edge of CLKIN before the rising edge of $\overline{\text{RSTI}}$ to determine the reset configuration of the global chip select, the address bus, and PLL. However, unlike the MCF5307, the frequency range encodings are not sampled on D[3:2], which are replaced by two other reset configuration inputs. First, the CLKIN-to-PCLK ratio allows more combinations. This extra bit is now sampled on D2 so that the clock ratio programming bits encompass D[2:0]/DIVIDE[2:0].

Second, a new reset configuration bit, BE_CONFIG, is now multiplexed with D3 in the MCF5407. This bit enables the four byte enables for the global chip select, CS0, for reads and writes or writes only, depending on the bit value sampled at reset, as shown in Table A-10.

Table A-6 shows the multiplexing of D[7:0] for the MCF5307 and the MCF5407.

**Table A-6. D[7:0] Multiplexing**

| Data Pins | MCF5307 | MCF5407 |
|---|---|---|
| D7 | AA | |
| D[6:5] | PS[1:0] | |
| D4 | ADDR_CONFIG | |
| D3 | FREQ1 | BE_CONFIG, BE[3:0] |
| D2 | FREQ0 | DIVIDE2 |
| D1 | DIVIDE1 | |
| D0 | DIVIDE0 | |

Table A-7 through Table A-10 list the various reset encodings for the configuration signals

multiplexed with D[7:3]. See for D[2:0]/DIVIDE[2:0] encodings sampled at reset. Note that Table A-7 and Table A-8 configure the global, or boot, $\overline{CS0}$ that is used to access boot ROM out of reset. $\overline{CS0}$ is the only chip select active out of reset until other chip selects become valid. Both the wait states and port size of boot memory accessed by boot $\overline{CS0}$ are programmed through these bits.

**Table A-7. D7/AA, Automatic Acknowledge of Boot $\overline{CS0}$**

| D7/AA | Boot $\overline{CS0}$ AA Configuration at Reset |
|---|---|
| 0 | Disabled |
| 1 | Enabled with 15 wait states |

Table A-8 shows configurations for D[6:5]/PS[1:0].

**Table A-8. D[6:5]/PS[1:0], Port Size of Boot $\overline{CS0}$**

| D[6:5]/PS[1:0] | Boot $\overline{CS0}$ Port Size at Reset |
|---|---|
| 00 | 32-bit port |
| 01 | 8-bit port |
| 1x | 16-bit port |

Table A-9 initializes the pin assignment register of the parallel I/O port to be either parallel I/O or to be the upper address bus bits along with various attribute and control signals at reset to give the user the option to access a broader addressing range of memory, if desired.

**Table A-9. D4/ADDR_CONFIG, Address Pin Assignment**

| D4/ADDR_CONFIG | Configuration Pin Assignment Register at Reset |
|---|---|
| 0 | PP[15:0], defaulted to inputs upon reset |
| 1 | ADDR[31:24]/$\overline{TIP}$/$\overline{DREQ}$[1:0]/TM[2:1] |

Table A-10 shows configurations for D3/BE_CONFIG. Because some boot memories require byte enables to be active only during writes, the functionality of byte enables, BE[3:0], can be programmed at reset.

**Table A-10. D3/BE_CONFIG, BE[3:0] Boot Configuration**

| D3/BE_CONFIG | Configuration of Byte Enables for Boot CS0 |
|---|---|
| 0 | BE[3:0] are enabled as byte write enables only |
| 1 | BE[3:0] are enabled as byte enables for reads and writes |

D[2:0]/DIVIDE[2:0] configurations are shown in Table A-5.

After $\overline{RSTI}$ is negated, 32 bits of CPU configuration information are loaded into data register D0 and 32 bits of internal memory information are loaded in D1. Because these registers are completely uninitialized on previous ColdFire devices, this feature allows users to identify the MCF5407 through software. Values D1 = 0x0630_0530 and D0 =

0xCF4*x*_C012 identify the MCF5407, where *x* identifies the core revision number (0x1 for the initial device).

## A.8 Revision C Debug

A number of enhancements to the original ColdFire debug functions were requested by customers and third-party tool developers. As a result, an expanded set of debug functions was implemented in the Version 4 ColdFire and named Revision C, or simply Debug C. Most of the enhancements are included in the MCF5407 debug module and are primarily related to improvements in the real-time debug capabilities.

### A.8.1 Debug Interrupts and Interrupt Requests in Emulator Mode

In the Debug B ColdFire implementation of the MCF5307, the response to a user-defined breakpoint trigger can be configured as one of three possibilities:

- The breakpoint trigger can be displayed on the PSTDDATA bus with no internal reaction to the trigger. The trigger state information is displayed on PSTDDATA in all situations.
- The breakpoint trigger can force the processor to halt and allow BDM activities.
- The breakpoint trigger can generate a special debug interrupt to allow real-time systems to quickly process the interrupt and return to normal system executing as rapidly as possible.

The occurrence of a debug interrupt exception is treated as a special type of interrupt. It is considered to be higher in priority than all normal interrupt requests and has special processor status values to indicate externally that this interrupt occurred.

Additionally, the execution of the debug interrupt service routine is forced to be interrupt-inhibited by the processor hardware. Optionally, it is capable of mapping all instruction and data references while in this service routine into a separate address space, so that an emulator can define the routine dynamically.

Current processor implementations include a state bit, invisible to software, that defines this emulator mode of operation. Note that the interrupt mask level is not modified during the processing of a debug interrupt.

In response to customers with real-time embedded systems asking for the ability to service normal interrupt requests while processing the debug interrupt service routine, this feature has been incorporated in the Revision C debug. To provide this function and service any number of normal interrupt requests, including the possibility of nested interrupts, the processor state signaling emulator mode is now included as part of the exception stack

frame, shown in Figure A-4.

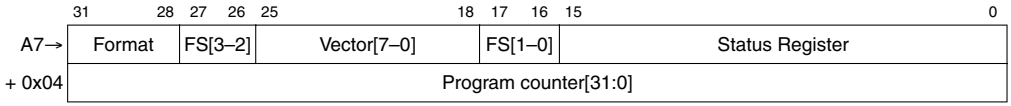



**Figure A-4. Exception Stack Frame Form**

As part of the Debug C enhancement, the operation of the debug interrupt is modified as follows:

- The occurrence of the breakpoint trigger, configured to generate a debug interrupt, is treated exactly as before. The debug interrupt is treated as a higher priority exception relative to the normal interrupt requests encoded on the interrupt priority input signals.
- At the appropriate sample point, the processor initiates debug interrupt exception processing. This event is signaled externally by the generation of a unique PST value (PST = 0xD) asserted for multiple cycles. The processor sets the emulator mode state bit as part of this processing.
- All normal interrupt requests are evaluated and sampled once per instruction during the debug interrupt service routine. If an exception is detected, the processor takes the following steps:

1. In response to the new exception, the processor saves a copy of the current value of the emulator mode state bit and then exits emulator mode by clearing the actual state.
2. The new exception stack frame sets bit 1 of the fault status field, using the saved emulator mode bit, indicating that execution while the processor is in emulator mode was interrupted. This corresponds to bit [17] of the longword at the top of the system stack.
3. Control is passed to the appropriate exception handler.
4. When the exception handler is complete, a Return From Exception (RTE) instruction is executed. During the processing of the RTE, the FS1 bit is reloaded from the system stack. If FS1 = 1, the processor sets the emulator mode state and resumes execution of the original debug interrupt service routine. This is signaled externally by the generation of the PST value that originally identified the occurrence of a debug interrupt exception, that is, PST = 0xD.

Implementation of this revised debug interrupt handling fully supports the servicing of any number of normal interrupt requests while in a debug interrupt service routine. The emulator mode state bit is essentially changed to a program-visible value, stored into memory when the exception stack frame is created, and loaded from memory by the RTE instruction.

## A.8.2 On-Chip Breakpoint Registers

The Debug B core debug module included three basic types of on-chip breakpoint registers:

- A 32-bit PC breakpoint register and a 32-bit PC breakpoint mask
- Two 32-bit address registers, which can be used to specify a single address or a range of addresses
- A 32-bit data breakpoint register and a 32-bit data breakpoint mask

The mask registers can be used to “don’t care” the equivalent bits in the breakpoint registers.

Additions to the breakpoint implementation are as follows:

- Three more 32-bit PC breakpoint registers
- Two more 32-bit address registers (ABLR1, ABHR1) plus an attribute register (AATR1) and mask register, which can be used to specify a single address or a range of addresses
- One more 32-bit data breakpoint register and a 32-bit data breakpoint mask

The addition of these new breakpoint registers also requires the appropriate control and configuration functions be added to the debug programming model. The affected BDM command and new register formats are described below. The revised BDM command is write debug module register (WDMREG)**.**

### A.8.2.1 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 operation. The debug module’s programming model can be accessed either from the serial BDM communication channel or from the processor’s execution of the supervisor-mode WDEBUG instruction. DSCLK must be inactive while WDEBUG executes.

Figure A-5 defines the operand data format.

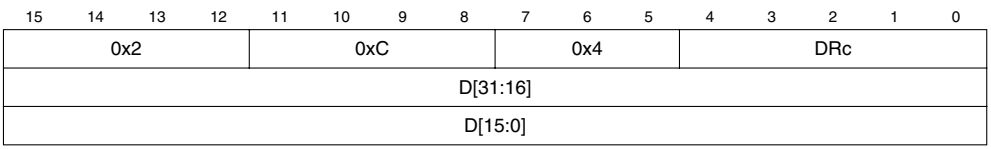



**Figure A-5. Write Debug Module Register Command (WDMREG)**

Table A-11 describes the DRc encoding for the debug registers.

**Table A-11. Definition of DRc Encoding—Write**

| DRc (hex) | Debug Register Definition | Abbreviation | Initial State (hex) |
|---|---|---|---|
| 0x00 | Configuration/Status | CSR | 0x0000 |
| 0x01–0x04 | Reserved | — | — |
| 0x05 | BDM address attributes | BAAR | 0x0005 |
| 0x06 | Bus attributes and mask | AATR | 0x0005 |
| 0x07 | Trigger definition | TDR | 0x0000 |
| 0x08 | PC breakpoint | PBR | — |
| 0x09 | PC breakpoint mask | PBMR | — |
| 0x0A–0x0B | Reserved | — | — |
| 0x0C | Operand address high breakpoint | ABHR | — |
| 0x0D | Operand address low breakpoint | ABLR | — |
| 0x0E | Data breakpoint | DBR | — |
| 0x0F | Data breakpoint mask | DBMR | — |
| 0x10–0x15 | Reserved | — | — |
| 0x16 | Bus attributes and mask 1 | AATR1 | 0x0005 |
| 0x17 | Extended trigger definition | XTDR | 0x0000 |
| 0x18 | PC breakpoint 1 | PBR1 | 0x0000 |
| 0x19 | Reserved | — | — |
| 0x1A | PC breakpoint 2 | PBR2 | 0x0000 |
| 0x1B | PC breakpoint 3 | PBR3 | 0x0000 |
| 0x1C | Operand address high breakpoint 1 | ABHR1 | — |
| 0x1D | Operand address low breakpoint 1 | ABLR1 | — |
| 0x1E | Data breakpoint 1 | DBR1 | — |
| 0x1F | Data breakpoint mask 1 | DBMR1 | — |

Command Sequence:

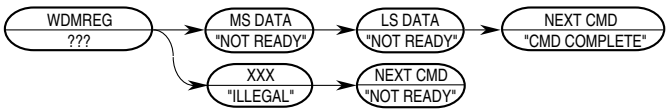



**Figure A-6. WDMREG Command Sequence**

Operand Data:

Longword data is written into the specified debug register. Data is supplied most significant word first.

Result Data:

Command complete status (0x0FFFF) is returned when register write is complete.

## A.8.3 Debug Programming Model

In addition to existing BDM commands that provide access to the processor's registers and the memory subsystem, the debug module contains a number of registers to support the required functionality. These registers are treated as 32-bit quantities, regardless of the number of bits in the implementation. The debug control registers (DRc) are addressed using a 5-bit value as part of two new BDM commands (WDREG and RDREG). These values are shown in Table A-11.

These registers are also accessible from the processor's supervisor programming model through the execution of the WDEBUG instruction. Thus, the breakpoint hardware within the debug module can be accessed by the external development system using the serial interface or by the operating system running on the processor core. It is the software's responsibility to guarantee that all accesses to these resources are serialized and are logically consistent. The 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 IPW).

The following sections describe the newly added breakpoint registers in Debug C.

### A.8.3.1 Address Breakpoint 1 Registers (ABLR1, ABHR1)

The 32-bit address breakpoint 1 registers define an upper (ABHR1) and a lower (ABLR1) boundary for a region in the operand logical address space of the processor that can be used as part of the trigger. The ABLR1 and ABHR1 values are compared with the ColdFire CPU core address signals, as defined by the setting of the trigger definition register (TDR) and the extended trigger definition register (XTDR).

### A.8.3.2 Address Attribute Breakpoint Register 1 (AATR1)

The address attribute breakpoint register 1 (AATR1) defines the address attributes and a mask associated with ABLR1 and ABHR1 to be matched in the trigger. The AATR1 value is compared with the ColdFire CPU core address attribute signals, as defined by the setting of the TDR and XTDR. The format of the AATR1 is the same as the AATR register. For more details about these registers see Section 5.4.1, "Address Attribute Trigger Registers (AATR, AATR1)".

### A.8.3.3 Program Counter Breakpoint Registers 1–3 (PBR1–PBR3)

Each of the program counter (PC) breakpoint registers (PBR, PBR1–PBR3) defines an instruction address that can be used as part of the trigger. PBR*n* registers are compared with the processor's program counter register when the appropriate valid bit is asserted and TDR is configured appropriately. For more details about these registers see Section 5.4.6, "Program Counter Breakpoint/Mask Registers (PBR, PBR1, PBR2, PBR3, PBMR)".

The results of all PC breakpoint registers, PBR/PBMR, PBR1, PBR2, and PBR3, are logically summed to form a single PC breakpoint trigger signal.

- PBR*n*[31:1] = program counter breakpoint address
- PBR*n*[0] = valid bit

### A.8.3.4 Data Breakpoint Register 1 (DBR1, DBMR1)

The data breakpoint register 1 (DBR1) defines a specific data pattern that can be used as part of a trigger. The DBR1 value is masked by DBMR1, allowing only those bits in DBR1 that have a corresponding zero in DBMR1 to be compared with the ColdFire CPU core data signals, as defined in the TDR and the XTDR.

The data breakpoint registers support both aligned and misaligned operand references. The relationship between the processor core address, the access size, and the corresponding location within the 32-bit core data bus is defined in the DBR and DBMR description.

### A.8.3.5 Extended Trigger Definition Register (XTDR)

The XTDR enables the operation as defined by the new breakpoint registers, ABHR1, ABLR1, AATR1, DBR1, and DBMR1, within the debug module and operates in conjunction with the trigger definition register (TDR). The added breakpoint logic can be included as a one- or two-level trigger; XTDR[29–18] define second-level triggers and XTDR[13–2] define first-level triggers. The definition of the XTDR register is exactly the same as the TDR for the control of the ABHR1, ABLR1, DBR1 and DBMR1 breakpoint registers. The XTDR is cleared on reset. For more details about this register see Section 5.4.8, “Extended Trigger Definition Register (XTDR)”.

## A.8.4 Debug Interrupt Exception Vectors

In the Debug B revision, if the occurrence of a hardware breakpoint is configured to generate a debug interrupt, this exception is mapped to vector number 12 (0x030). The actual debug interrupts can be broadly classified into two groups—PC breakpoints and all other types. A PC breakpoint is treated in a precise manner—exception recognition and processing are initiated before the instruction at the given address is executed. Conversely, all other breakpoint events are recognized on the given internal bus transaction, but are made pending to the processor and sampled like other interrupt conditions. As a result, these types of interrupts are imprecise by nature.

In response to a customer request that PC breakpoints be distinguishable from other type of trigger events, the debug interrupt exception vector is expanded in Debug C of the MCF5407 to two unique entries, shown in Table A-12, where the occurrence of a PC breakpoint generates the 0x034 vector. In the case of a two-level trigger, the last breakpoint event determines the exception vector.

**Table A-12. Debug C Exception Vector Assignments**

| Vector | Vector Offset | Stacked Program Counter | Assignment |
|---|---|---|---|
| 12 | 0x030 | Next | Non-PC-breakpoint debug interrupt |
| 13 | 0x034 | Next | PC-breakpoint debug Interrupt |

## A.8.5 Processor Status and Debug Data Output Signals

The Debug B architecture defines processor status, PST[3:0] and debug data DDATA[3:0] signals, which provide information to support real-time trace. In the Debug B design, these signals are output at the processor frequency.

For the Debug C definition, however, the PST and DDATA are combined and redefined to operate at half the processor's operating frequency (provided by PSTCLK). Therefore, PSTDDATA[7:0] are used to output both processor status and captured debug data values.

For more details, including single-cycle instruction timing examples, see Section 5.2.1, "Processor Status/Debug Data (PSTDDATA[7:0])."

A PST marker and its data display are transmitted contiguously. Except for this transmission, the IDLE status (0x0) may appear any time. Again, given the real-time trace information appears as a sequence of 4-bit values, there are no alignment restrictions. That is, PST values and operands may appear on either nibble of PSTDDATA.

In Debug B, the DDATA outputs display the status of the internal breakpoint registers when they are not displaying captured data values. For the Debug C design, any change to this breakpoint state is identified by a PST marker and then the new state value. Specifically, the marker for this breakpoint state change is a single assertion of the value 0xD. Usually, the 0xD status is asserted for multiple cycles, indicating entry into emulator mode in response to a debug interrupt exception. For Debug C, the posting of the 0xD status can signal multiple events, based on the next value.

```
if the PSTDDATA stream includes {0xD, 0x2}
        then Breakpoint state changed to Waiting for Level 1 Trigger

if the PSTDDATA stream includes {0xD, 0x4}
        then Breakpoint state changed to Level 1 Breakpoint Triggered

if the PSTDDATA stream includes {0xD, 0xA}
        then Breakpoint state changed to Waiting for Level 2 Trigger

if the PSTDDATA stream includes {0xD, 0xC}
        then Breakpoint state changed to Level 2 Breakpoint Triggered

if the PSTDDATA stream includes {0xD, 0xD}
        then Entry into Emulator Mode
```

Table A-13 shows the revised definition of the processor status encodings, where the values of {0xC–0xF} are usually asserted for multiple cycles. The behavior of the 0xD value was described previously. The PSTDDATA values of 0x2 and 0x6 are formerly reserved values now needed to support the Version 4 operand execution pipeline.

**Table A-13. Version 4 Debug C Processor Status Encodings**

| PSTDDATA Value | Definition |
|---|---|
| 0x0 | Continue execution |
| 0x1 | Begin execution of one instruction |
| 0x2 | Begin execution of two instructions |
| 0x3 | Entry into user-mode |
| 0x4 | Begin execution of PULSE or WDDATA instruction |
| 0x5 | Begin execution of taken branch |
| 0x6 | Begin execution of an instruction plus a taken branch |
| 0x7 | Begin execution of RTE instruction |
| 0x8 | Begin 1-byte data transfer on PSTDDATA |
| 0x9 | Begin 2-byte data transfer on PSTDDATA |
| 0xA | Begin 3-byte data transfer on PSTDDATA |
| 0xB | Begin 4-byte data transfer on PSTDDATA |
| 0xC | Exception processing |
| 0xD | Breakpoint state change, or entry into emulator mode |
| 0xE | Processor is stopped, waiting for interrupt |
| 0xF | Processor is halted |

### A.8.6 Debug C Summary

The preceding section describes additional functionality requested by ColdFire customers and third-party developers. The Debug C enhancements are designed to retain backward compatibility with the previous definition.

## A.9 Voltage Input Changes

Although the MCF5407 logic operates at 1.8 V, the device pads are standard TTL-compatible and therefore can drive a 2.4-V minimum output and accepts a 3.3-V input. Thus, the MCF5407 requires both 1.8- and 3.3-V power supplies. This specification differs from the MCF5307, which operates at 3.3 V with 5-V-tolerant I/O pads. Although the power and ground pin assignment are the same for both the MCF5307 and MCF5407, the power pin allocation of the MCF5407 is divided between 1.8- and 3.3-V supply levels. Thus, two power rails are necessary to supply power to the MCF5407.

Note that the MCF5407 meets the EIA/JEDEC standard for 1.8-V power supply voltage and interface requirements. See the JEDEC standard (EIA/JESD8-7, February 1997).

## A.10 PLL Power Supply Filter Circuit

To ensure PLL stability, the power supply to the PLL power pin should be filtered using a

circuit similar to the one shown in Figure A-7. The circuit should be as close as possible to the PLL power pin to ensure maximum noise filtering. This filter design can be used for both the MCF5307 and MCF5407.

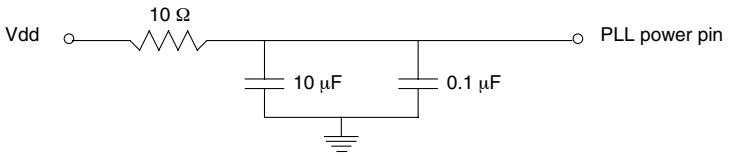



**Figure A-7. PLL Power Supply Filter Circuit**

## A.11 Pin-Assignment Compatibility

The MCF5407 pinout is identical to the MCF5307 except for the power pin allocation and PSTDDATA[7:0], which make available the contents of processor status, PST[3:0], and debug data, DDATA[3:0]. Therefore, when designing-in the MCF5407, note which power pins require 1.8 V and which require 3.3 V. The MCF5407 footprint is the same as the MCF5307, which is a 208-pin plastic quad flat pack (QFP).

# Appendix B
# List of Memory Maps

### Table B-1. SIM Registers

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| 0x000 | Reset status register (RSR) [p. 6-5] | System protection control register (SYPCR) [p. 6-8] | Software watchdog interrupt vector register (SWIVR) [p. 6-9] | Software watchdog service register (SWSR) [p. 6-9] |
| 0x004 | Pin assignment register (PAR) [p. 6-10] | | Interrupt port assignment register (IRQPAR) [p. 9-7] | Reserved |
| 0x008 | PLL control (PLLCR) [p. 7-3] | Reserved | | |
| 0x00C | Default bus master park register (MPARK) [p. 6-11] | Reserved | | |
| 0x010–0x03C | Reserved | | | |

### Table B-2. Interrupt Controller Registers

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| **Interrupt Registers [p. 9-3]** | | | | |
| 0x040 | Interrupt pending register (IPR) [p. 9-6] | | | |
| 0x044 | Interrupt mask register (IMR) [p. 9-6] | | | |
| 0x048 | Reserved | | | Autovector register (AVR) [p. 9-5] |
| **Interrupt Control Registers (ICRs) [p. 9-3]** | | | | |
| 0x04C | Software watchdog timer (ICR0) [p. 6-6] | Timer0 (ICR1) [p. 9-2] | Timer1 (ICR2) [p. 9-3] | $I^2C$ (ICR3) [p. 9-3] |
| 0x050 | UART0 (ICR4) [p. 9-3] | UART1 (ICR5) [p. 9-3] | DMA0 (ICR6) [p. 9-3] | DMA1 (ICR7) [p. 9-3] |
| 0x054 | DMA2 (ICR8) [p. 9-3] | DMA3 (ICR9) [p. 9-3] | Reserved | |

## Table B-3. Chip-Select Registers

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| 0x080 | Chip-select address register—bank 0 (CSAR0) [p. 10-6] | | Reserved[1] | |
| 0x084 | Chip-select mask register—bank 0 (CSMR0) [p. 10-7] | | | |
| 0x088 | Reserved[1] | | Chip-select control register—bank 0 (CSCR0) [p. 10-8] | |
| 0x08C | Chip-select address register—bank 1 (CSAR1) [p. 10-6] | | Reserved[1] | |
| 0x090 | Chip-select mask register—bank 1 (CSMR1) [p. 10-7] | | | |
| 0x094 | Reserved[1] | | Chip-select control register—bank 1 (CSCR1) [p. 10-8] | |
| 0x098 | Chip-select address register—bank 2 (CSAR2) [p. 10-6] | | Reserved[1] | |
| 0x09C | Chip-select mask register—bank 2 (CSMR2) [p. 10-7] | | | |
| 0x0A0 | Reserved[1] | | Chip-select control register—bank 2 (CSCR2) [p. 10-8 | |
| 0x0A4 | Chip-select address register—bank 3 (CSAR3) [p. 10-6] | | Reserved[1] | |
| 0x0A8 | Chip-select mask register—bank 3 (CSMR3) [p. 10-7] | | | |
| 0x0AC | Reserved[1] | | Chip-select control register—bank 3 (CSCR3) [p. 10-8] | |
| 0x0B0 | Chip-select address register—bank 4 (CSAR4) [p. 10-6] | | Reserved[1] | |
| 0x0B4 | Chip-select mask register—bank 4 (CSMR4) [p. 10-7] | | | |
| 0x0B8 | Reserved[1] | | Chip-select control register—bank 4 (CSCR4) [p. 10-8] | |
| 0x0BC | Chip-select address register—bank 5 (CSAR5) [p. 10-6] | | Reserved[1] | |
| 0x0C0 | Chip-select mask register—bank 5 (CSMR5) [p. 10-7] | | | |
| 0x0C4 | Reserved | | Chip-select control register—bank 5 (CSCR5) [p. 10-8] | |
| 0x0C8 | Chip-select address register—bank 6 (CSAR6) [p. 10-6] | | Reserved[1] | |
| 0x0CC | Chip-select mask register—bank 6 (CSMR6) [p. 10-7] | | | |
| 0x0D0 | Reserved[1] | | Chip-select control register—bank 6 (CSCR6) [p. 10-8] | |
| 0x0D4 | Chip-select address register—bank 7 (CSAR7) [p. 10-6] | | Reserved[1] | |
| 0x0D8 | Chip-select mask register—bank 7 (CSMR7) [p. 10-7] | | | |
| 0x0DC | Reserved[1] | | Chip-select control register—bank 7 (CSCR7) [p. 10-8 | |

## Table B-3. Chip-Select Registers (Continued)

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| **MBAR Offset** | **[31:24]** | **[23:16]** | **[15:8]** | **[7:0]** |
| 0x080 | Chip-select address register—bank 0 (CSAR0) [p. 10-6] | | Reserved[1] | |
| 0x084 | Chip-select mask register—bank 0 (CSMR0) [p. 10-7] | | | |
| 0x088 | Reserved[1] | | Chip-select control register—bank 0 (CSCR0) [p. 10-8] | |
| 0x08C | Chip-select address register—bank 1 (CSAR1) [p. 10-6] | | Reserved[1] | |
| 0x090 | Chip-select mask register—bank 1 (CSMR1) [p. 10-7] | | | |
| 0x094 | Reserved[1] | | Chip-select control register—bank 1 (CSCR1) [p. 10-8] | |
| 0x098 | Chip-select address register—bank 2 (CSAR2) [p. 10-6] | | Reserved[1] | |
| 0x09C | Chip-select mask register—bank 2 (CSMR2) [p. 10-7] | | | |
| 0x0A0 | Reserved[1] | | Chip-select control register—bank 2 (CSCR2) [p. 10-8] | |
| 0x0A4 | Chip-select address register—bank 3 (CSAR3) [p. 10-6] | | Reserved[1] | |
| 0x0A8 | Chip-select mask register—bank 3 (CSMR3) [p. 10-7] | | | |
| 0x0AC | Reserved[1] | | Chip-select control register—bank 3 (CSCR3) [p. 10-8] | |
| 0x0B0 | Chip-select address register—bank 4 (CSAR4) [p. 10-6] | | Reserved[1] | |
| 0x0B4 | Chip-select mask register—bank 4 (CSMR4) [p. 10-7] | | | |
| 0x0B8 | Reserved[1] | | Chip-select control register—bank 4 (CSCR4) [p. 10-8] | |

[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.

## Table B-4. DRAM Controller Registers

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| 0x100 | DRAM control register (DCR) [p. 11-3] | | Reserved | |
| 0x104 | Reserved | | | |
| 0x108 | DRAM address and control register 0 (DACR0) [p. 11-3] | | | |
| 0x10C | DRAM mask register block 0 (DMR0) [p. 11-3] | | | |
| 0x110 | DRAM address and control register 1 (DACR1) [p. 11-3] | | | |
| 0x114 | DRAM mask register block 1 (DMR1) [p. 11-3] | | | |

## Table B-5. General-Purpose Timer Registers

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| 0x140 | Timer 0 mode register (TMR0) [p. 13-3] | | Reserved | |
| 0x144 | Timer 0 reference register (TRR0) [p. 13-4] | | Reserved | |
| 0x148 | Timer 0 capture register (TCR0) [p. 13-4] | | Reserved | |
| 0x14C | Timer 0 counter (TCN0) [p. 13-5] | | Reserved | |
| 0x150 | Reserved | Timer 0 event register (TER0) [p. 13-5] | Reserved | |
| 0x180 | Timer 1 mode register (TMR1) [p. 13-3] | | Reserved | |
| 0x184 | Timer 1 reference register (TRR1) [p. 13-4] | | Reserved | |
| 0x188 | Timer 1 capture register (TCR1) [p. 13-4] | | Reserved | |
| 0x18C | Timer 1 counter (TCN1) [p. 13-5] | | Reserved | |
| 0x190 | Reserved | Timer 1 event register (TER1) [p. 13-5] | Reserved | |

## Table B-6. UART0 Control Registers

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| UART0 Control Registers | | | | |
| 0x1C0 | UART mode registers[1]—(UMR1n) [p. 14-5], (UMR2n) [p. 14-7] | — | | |
| 0x1C4 | (Read) UART status registers—(USRn) [p. 14-10] | — | | |
| | (Write) UART clock-select register[1]—(UCSRn) [p. 14-12] | — | | |
| 0x1C8 | (Read) Do not access[2] | — | | |
| | (Write) UART command registers—(UCRn) [p. 14-13] | — | | |
| 0x1CC | (Read) UART receiver buffers—(URBn) [p. 14-15] | — | | |
| | (Write) UART transmitter buffers—(UTBn) [p. 14-16] | — | | |

## Table B-6. UART0 Control Registers (Continued)

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| 0x1D0 | (Read) UART input port change registers—(UIPCRn) [p. 14-17] | — | | |
| | (Write) UART auxiliary control registers[1]—(UACRn) [p. 14-17] | — | | |
| 0x1D4 | (Read) UART interrupt status registers—(UISRn) [p. 14-18] | — | | |
| | (Write) UART interrupt mask registers—(UIMRn) [p. 14-18] | — | | |
| 0x1D8 | UART divider upper registers—(UDUn) [p. 14-19] | — | | |
| 0x1DC | UART divider lower registers—(UDLn) [p. 14-19] | — | | |
| 0x1E0–0x1EC | Do not access[2] | — | | |
| 0x1F0 | UART interrupt vector register—(UIVRn) [p. 14-20] | — | | |
| 0x1F4 | (Read) UART input port registers—(UIPn) [p. 14-20] | — | | |
| | (Write) Do not access[2] | — | | |
| 0x1F8 | (Read) Do not access[2] | — | | |
| | (Write) UART output port bit set command registers—(UOP1n[3]) [p. 14-21] | — | | |
| 0x1FC | (Read) Do not access[2] | — | | |
| | (Write) UART output port bit reset command registers—(UOP0n[3]) [p. 14-21] | — | | |

1 UMR1n, UMR2n, UCSRn, and UACRn[BRG] should be changed only after the receiver/transmitter is issued a software reset command. That is, if channel operation is not disabled, undesirable results may occur.

2 This address is for factory testing. Reading this location results in undesired effects and possible incorrect transmission or reception of characters. Register contents may also be changed.

3 Address-triggered commands

## Table B-7. UART1 Control Registers

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| UART1 Control Registers | | | | |
| 0x200 | UART mode registers[1]—(UMR1n)[p. 14-5], (UMR2n) [p. 14-7 | Rx FIFO threshold register—(RXLVL) [p. 14-8] | Modem control register—(MODCTL) [p. 14-9] | Tx FIFO threshold register—(TXLVL) [p. 14-10] |
| 0x204 | (Read) UART status registers—(USRn) [p. 14-10] | — | (Read) Rx samples available register—(RSMP) [p. 14-12] | (Read) Tx space available register—(TSPC) [p. 14-13] |
| | (Write) UART clock-select register[1]—(UCSRn) [p. 14-12] | | | |
| 0x208 | (Read) Do not access[2] | — | | |
| | (Write) UART command registers—(UCRn) [p. 14-13] | — | | |
| 0x20C | (Read) UART receiver buffers—(URBn) [p. 14-15] | | | |
| | (Write) UART transmitter buffers—(UTBn) [p. 14-16] | | | |
| 0x210 | (Read) UART input port change registers—(UIPCRn) [p. 14-17] | — | | |
| | (Write) UART auxiliary control registers[1]—(UACRn) [p. 14-17] | — | | |
| 0x214 | (Read) UART interrupt status registers—(UISRn) [p. 14-18] | — | | |
| | (Write) UART interrupt mask registers—(UIMRn) [p. 14-18] | — | | |
| 0x218 | UART divider upper registers—(UDUn) [p. 14-19] | — | | |
| 0x21C | UART divider lower registers—(UDLn) [p. 14-19] | — | | |
| 0x220–0x22C | Do not access[2] | — | | |

## Table B-7. UART1 Control Registers (Continued)

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| 0x230 | UART interrupt vector register—(UIVRn) [p. 14-20] | — | | |
| 0x234 | (Read) UART input port registers—(UIPn) [p. 14-20] | — | | |
| | (Write) Do not access[2] | — | | |
| 0x238 | (Read) Do not access[2] | — | | |
| | (Write) UART output port bit set command registers—(UOP1n[3]) [p. 14-21] | — | | |
| 0x23C | (Read) Do not access[2] | — | | |
| | (Write) UART output port bit reset command registers—(UOP0n[3]) [p. 14-21] | — | | |
| 0x200 | UART mode registers[4]—(UMR1n) [p. 14-5], (UMR2n) [p. 14-7] | Rx FIFO threshold register—(RXLVL) [p. 14-10] (UART1 only) | Modem control register—(MODCTL) [p. 14-9] (UART1 only) | Tx FIFO threshold register—(TXLVL) [p. 14-10] (UART1 only) |
| 0x204 | (Read) UART status registers—(USRn) [p. 14-10] | — | (Read) Rx samples available register—(RSMP) [p. 14-12] (UART1 only) | (Read) Tx space available register—(TSPC) [p. 14-13] (UART1 only) |
| | (Write) UART clock-select register[1]—(UCSRn) [p. 14-12] | | | |

1 UMR1n, UMR2n, UCSRn, and UACRn[BRG] should be changed only after the receiver/transmitter is issued a software reset command. That is, if channel operation is not disabled, undesirable results may occur.

2 This address is for factory testing. Reading this location results in undesired effects and possible incorrect transmission or reception of characters. Register contents may also be changed.

3 Address-triggered commands

4 UMR1n, UMR2n, UCSRn, and UACRn[BRG] should be changed only after the receiver/transmitter is issued a software reset command. That is, if channel operation is not disabled, undesirable results may occur.

## Table B-8. Parallel Port Memory Map

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| 0x244 | Parallel port data direction register (PADDR) [p. 15-2] | | Reserved | |
| 0x248 | Parallel port data register (PADAT) [p. 15-2] | | Reserved | |

## Table B-9. I²C Interface Memory Map

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| 0x280 | I²C address register (IADR) [p. 8-6] | Reserved | | |
| 0x284 | I²C frequency divider register (IFDR) [p. 8-6] | Reserved | | |
| 0x288 | I²C control register (I2CR) [p. 8-7] | Reserved | | |
| 0x28C | I²C status register (I2SR) [p. 8-8] | Reserved | | |
| 0x290 | I²C data I/O register (I2DR) [p. 8-9] | Reserved | | |

## Table B-10. DMA Controller Registers

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| 0x300 | Source address register 0 (SAR0) [p. 12-7] | | | |
| 0x304 | Destination address register 0 (DAR0) [p. 12-7] | | | |
| 0x308 | DMA control register 0 (DCR0) [p. 12-8] | | | |
| 0x30C | Reserved | Byte count register 0 (BCR0) [p. 12-7] | | |
| 0x310 | DMA status register 0 (DSR0) [p. 12-10] | Reserved | | |
| 0x314 | DMA interrupt vector register 0 (DIVR0) [p. 12-11] | Reserved | | |
| 0x340 | Source address register 1 (SAR1) [p. 12-7] | | | |
| 0x344 | Destination address register 1 (DAR1) [p. 12-7] | | | |
| 0x348 | DMA control register 1 (DCR1) [p. 12-8] | | | |
| 0x34C | Reserved | Byte count register 1 (BCR1) [p. 12-7] | | |
| 0x350 | DMA status register 1 (DSR1) [p. 12-10] | Reserved | | |
| 0x354 | DMA interrupt vector register 1 (DIVR1) [p. 12-11] | Reserved | | |
| 0x380 | Source address register 2 (SAR2) [p. 12-7] | | | |
| 0x384 | Destination address register 2 (DAR2) [p. 12-7] | | | |
| 0x388 | DMA control register 2 (DCR2) [p. 12-8] | | | |
| 0x38C | Reserved | Byte count register 2 (BCR2) [p. 12-7] | | |
| 0x390 | DMA status register 2 (DSR2) [p. 12-10] | Reserved | | |

## Table B-10. DMA Controller Registers (Continued)

| MBAR Offset | [31:24] | [23:16] | [15:8] | [7:0] |
|---|---|---|---|---|
| 0x394 | DMA interrupt vector register 2 (DIVR2) [p. 12-11] | Reserved | | |
| 0x3C0 | Source address register 3 (SAR3) [p. 12-7] | | | |
| 0x3C4 | Destination address register 3 (DAR3) [p. 12-7] | | | |
| 0x3C8 | DMA control register 3 (DCR3) [p. 12-8] | | | |
| 0x3CC | Reserved | Byte count register 3 (BCR3) [p. 12-7] | | |
| 0x3D0 | DMA status register 3 (DSR3) [p. 12-10] | Reserved | | |
| 0x3D4 | DMA interrupt vector register 3 (DIVR3) [p. 12-11] | Reserved | | |

# Glossary of Terms and Abbreviations

The glossary contains an alphabetical list of terms, phrases, and abbreviations used in this book.

## A

**Architecture.** A detailed specification of requirements for a processor or computer system. It does not specify details of how the processor or computer system must be implemented; instead it provides a template for a family of compatible *implementations*.

**Autovector.** A method of determining the starting address of the service routine by fetching the value from a lookup table internal to the processor instead of requesting the value from the system.

## B

**Branch folding**. The replacement with target instructions of a branch instruction and any instructions along the not-taken path when a branch is either taken or predicted as taken.

**Branch prediction**.The process of guessing whether a branch will be taken. Such predictions can be correct or incorrect; the term 'predicted' as it is used here does not imply that the prediction is correct (successful). **Branch resolution**.The determination of whether a branch is taken or not taken. A branch is said to be resolved when the processor can determine which instruction path to take. If the branch is resolved as predicted, the instructions following the predicted branch that may have been speculatively executed can complete (see completion). If the branch is not resolved as predicted, instructions on the mispredicted path, and any results of speculative execution, are purged from the pipeline and fetching continues from the nonpredicted path.

**Burst**. A multiple-beat data transfer.

## C

**Cache**. High-speed memory containing recently accessed data and/or instructions (subset of main memory).

**Cache coherency**. An attribute wherein an accurate and common view of memory is provided to all devices that share the same memory

system. Caches are coherent if a processor performing a read from its cache is supplied with data corresponding to the most recent value written to memory or to another processor's cache.

**Cache flush**. An operation that removes from a cache any data from a specified address range. This operation ensures that any modified data within the specified address range is written back to main memory.

**Cache line**. The smallest unit of consecutive data or instructions that is stored in a cache. For ColdFire processors a line consists of 16 bytes.

**Caching-inhibited**. A memory update policy in which the *cache* is bypassed and the load or store is performed to or from main memory.

**Cast outs**. *Cache lines* that must be written to memory when a cache miss causes a *cache line* to be replaced.

**Clear**. To cause a bit or bit field to register a value of zero. See also Set.

**Copyback**. A cache memory update policy in which processor write cycles are directly written only to the cache. External memory is updated only indirectly, for example, when a modified cache line is *cast out* to make room for newer data.

## E

**Effective address (EA)**. The 32-bit address specified for an instruction.

**Exception**. A condition encountered by the processor that requires special, supervisor-level processing.

**Exception handler**. A software routine that executes when an exception is taken. Normally, the exception handler corrects the condition that caused the exception, or performs some other meaningful task (that may include aborting the program that caused the exception). The address for each exception handler is identified by an exception vector defined by the ColdFire architecture.

## F

**Fetch**. The act of retrieving instructions from either the cache or main memory and making them available to the instruction unit.

**Flush**. An operation that causes a modified cache line to be invalidated and the data to be written to memory.

## H

**Harvard architecture**. An architectural model featuring separate caches for instruction and data.

## I

**Illegal instructions**. A class of instructions that are not implemented for a particular processor. These include instructions not defined by the ColdFire architecture.

**Implementation**. A particular processor that conforms to the ColdFire architecture, but may differ from other architecture-compliant implementations for example in design, feature set, and implementation of *optional* features. The ColdFire architecture has many different implementations.

**Imprecise mode.** A memory access mode that allows write accesses to a specified memory region to occur out of order.

**Instruction queue**. A holding place for instructions fetched from the current instruction stream.

**Instruction latency**. The total number of clock cycles necessary to execute an instruction and make the results of that instruction available.

**Interrupt**. An *asynchronous exception*. On ColdFire processors, interrupts are a special case of exceptions. See also asynchronous exception.

**Invalid state**. State of a cache entry that does not currently contain a valid copy of a cache line from memory.

## L

**Least-significant bit (lsb)**. The bit of least value in an address, register, data element, or instruction encoding.

**Least-significant byte (LSB)**. The byte of least value in an address, register, data element, or instruction encoding.

**Longword**. A 32-bit data element

## M

**Master.** A device able to initiate data transfers on a bus. Bus mastering refers to a feature supported by some bus architectures that allow a controller connected to the bus to communicate directly with other devices on the bus without going through the CPU.

**Memory coherency**. An aspect of caching in which it is ensured that an accurate view of memory is provided to all devices that share system memory.

**Modified state**. Cache state in which only one caching device has the valid data for that address.

**Most-significant bit (msb)**. The highest-order bit in an address, registers, data element, or instruction encoding.

**Most-significant byte (MSB)**. The highest-order byte in an address, registers, data element, or instruction encoding.

## N

**Nop**. No-operation. A single-cycle operation that does not affect registers or generate bus activity.

## O

**Overflow**. An condition that occurs during arithmetic operations when the result cannot be stored accurately in the destination register(s). For example, if two 16-bit numbers are multiplied, the result may not be representable in 16 bits.

## P

**Pipelining**. A technique that breaks operations, such as instruction processing or bus transactions, into smaller distinct stages or tenures (respectively) so that a subsequent operation can begin before the previous one completes.

**Precise mode.** A memory access mode that ensures that all write accesses to a specified memory region occur in order.

## S

**Set** (v) To write a nonzero value to a bit or bit field; the opposite of *clear*. The term ‘set’ may also be used to generally describe the updating of a bit or bit field.

**Set** (*n*). A subdivision of a *cache*. Cacheable data can be stored in a given location in any one of the sets, typically corresponding to its lower-order address bits. Because several memory locations can map to the same location, cached data is typically placed in the set whose *cache line* corresponding to that address was used least recently. *See* Set-associativity.

**Set-associativity**. Aspect of cache organization in which the cache space is divided into sections, called *sets*. The cache controller associates a particular main memory address with the contents of a particular set, or region, within the cache.

**Slave**. The device addressed by a master device. The slave is identified in the address tenure and is responsible for supplying or latching the requested data for the master during the data tenure.

**Static branch prediction**. Mechanism by which software (for example, compilers) can hint to the machine hardware about the direction a branch is likely to take.

**Superscalar machine**. A machine that can issue multiple instructions concurrently from a conventional linear instruction stream.

**Supervisor mode**. The privileged operation state of a processor. In supervisor mode, software, typically the operating system, can access all control registers and can access the supervisor memory space, among other privileged operations.

**System memory**. The physical memory available to a processor.

## T

**Tenure**. A tenure consists of three phases: arbitration, transfer, termination. There can be separate address bus tenures and data bus tenures.

**Throughput**. The measure of the number of instructions that are processed per clock cycle.

**Transfer termination**. The successful or unsuccessful conclusion of a data transfer.

## U

**Underflow.** A condition that occurs during arithmetic operations when the result cannot be represented accurately in the destination register.

**User mode**. The operating state of a processor used typically by application software. In user mode, software can access only certain control registers and can access only user memory space. No privileged operations can be performed.

## W

**Word**. A 16-bit data element.

**Write-through**. A cache memory update policy in which all processor write cycles are written to both the cache and memory.

# INDEX


## A

Addressing mode summary, 2-15
Arbitration
  between masters, 6-14
  bus control, 6-11
  for internal transfers, 6-12
Architecture
  Harvard memory, 2-6
  instruction set
    additions, 2-18
    enhancements, 2-36

## B

Branch acceleration, 2-4
Branch instruction execution
  timing, 2-30
Bus arbitration control, 6-11
Bus master park register, 6-11
Bus operation
  bus errors, 18-17
  characteristics, 18-2
  control signals, 18-1
  data transfer
    back-to-back cycles, 18-10
    burst cycles
      line read bus, 18-12
      line transfers, 18-12
      line write bus, 18-14
      mixed port sizes, 18-15
      overview, 18-11
    cycle execution, 18-4
    cycle states, 18-5
    fast-termination cycles, 18-9
    operation, 18-2
    read cycle, 18-7
    write cycle, 18-8
  external master transfers
    general, 18-21
    two-device arbitration protocol, 18-25
    two-wire mode, 18-25
  features, 18-1
  interrupt exceptions, 18-17
  misaligned operands, 18-16
  reset operation
    master, 18-34
    overview, 18-33
    software watchdog, 18-35

## C

Cache
  configuration register, 2-12
  registers, access control, 2-12
Chip-select module
  8-, 16-, and 32-bit port sizing, 10-4
  enable signals, 17-15
  operation, 10-2
    general, 10-3
    global, 10-4
  overview, 10-1
  registers, 10-5, 10-6, B-2
    code example, 10-9
    control, 10-8
    mask, 10-7
  signals, 10-1
Clock
  PLL control, 6-10
ColdFire core
  exception stack frame definition, A-11
  features and enhancements, 2-1
Condition code register, 2-9
CPU STOP instruction, 6-10

## D

Data registers, A-13
Debug
  module enhancements, 2-6
  system interface, 1-12
DMA controller module
  byte count registers, 12-7
  programming model, 12-5
  signal description, 12-2
  source address registers, 12-7
  transfer overview, 12-4
DRAM controller
  asynchronous mode signals, 11-4
  asynchronous operation
    burst page mode, 11-12
    continuous page mode, 11-13
    extended data out, 11-15
    general, 11-4
    register set, 11-4
  general guidelines, 11-8
  non-page mode, 11-11
  refresh operation, 11-16
  registers, 11-3

address and control, 11-5
mask, 11-7
signals, 17-16
synchronous operation, 11-16
address and control registers, 11-20
address multiplexing, 11-23
auto-refresh, 11-31
burst page mode, 11-27
continuous page mode, 11-29
controller signals, synchronous mode, 11-17
edge select, 11-18
general guidelines, 11-23
initialization, 11-32
interfacing, 11-27
mask registers, 11-22
mode register settings, 11-33
register set, 11-19
self-refresh, 11-32

## E

Electrical specifications
cautions, 20-3
clock timing, 20-4
debug AC timing, 20-16
DMA timing, 20-23
general parameters, 20-1
I2C input/output timing, 20-18
input/output AC timing specifications, 20-6
JTAG AC timing, 20-24
parallel port timing, 20-22
reset timing, 20-15
timer module AC timing, 20-17
UART module AC timing, 20-19
Exception processing
overview, 2-31
processor exceptions, 2-34
stack frame definition, 2-32
Execution timings
miscellaneous, 2-29
one operand, 2-26
two operands, 2-27

## F

Features
process, 1-7
summary, 1-4

## H

Harvard memory architecture, 2-6

## I

I2C
address register, 8-6
arbitration procedure, 8-4
clock stretching, 8-5
clock synchronization, 8-5
control register, 8-7
data I/O register, 8-9
features, 8-1
frequency divider register, 8-6
handshaking, 8-5
interface memory map, B-8
lost arbitration, 8-13
overview, 8-1
programming examples, 8-10
programming model, 8-6
protocol, 8-3
repeated START generation, 8-12
slave mode, 8-13
software response, 8-11
START generation, 8-10
status register, 8-8
STOP generation, 8-12
system configuration, 8-3
timing specifications, 20-18
IEEE Standard 1149.1 Test Access Port, *see* JTAG
Instruction execution times, 2-29
Instruction set
architecture additions, 2-18
architecture enhancements, 2-36
branch acceleration, 2-4
fetch pipeline, 2-4
MAC summary, 3-4
MAC unit execution times, 3-5
summary, 2-15, 2-19
Integer data formats, 2-13
Integer data formats in memory, 2-14
Integer data formats in registers, 2-13
Interrupt controller
autovector register, 9-5
overview, 9-1
pending and mask registers, 9-6
port assignment register, 9-7

## J

JTAG
AC timing, 20-24
obtaining IEEE Standard 1149.1, 19-11
overview, 19-1
registers
boundary scan, 19-7
bypass, 19-10
descriptions, 19-4
IDCODE, 19-6
instruction shift, 19-5
restrictions, 19-10

# INDEX


signal descriptions, 19-2
TAP controller, 19-3
test logic disabling, 19-10

## M

MAC
data representation, 3-4
hardware support, 2-5
instruction execution timings, 3-5
instruction set summary, 3-4
operation, 3-3
programming model, 2-10, 3-2
Mask registers
DRAM, 11-7, 11-22
MBAR, 6-4
Mechanical data, 16-1
case drawing, 16-9
diagram, 16-8
pinout, 16-1
Memory
integer data formats, 2-14
SIM register, 6-3
Modules, 1-7
base address register, 2-12
core description, 1-7
debug, 2-6
DMA controller, 1-9
DRAM controller, 1-9
$I^2C$, 1-11
PLL, 1-13
system interface, 1-11
16-bit parallel port, 1-12
chip selects, 1-11
debug, 1-12
external bus, 1-11
interrupt controller, 1-12
JTAG, 1-12
Timer, 1-11
UARTs, 1-10
MOVE instructions timing, 2-25

## O

Opcodes
illegal handling, 2-5

## P

Parallel port
code example, 15-4
data direction register, 15-2
data register, 15-2
operation, 15-1
Pin assignment register, 6-10, 15-1
Pipelines, 2-2
instruction fetch, 2-4
operand execution, 2-4
PLL, 7-2
clock control for STOP, 6-10
clock frequency relationships, 7-4
clock-multiplied, 2-2
control register, 7-3
modes
normal, 7-2
reduced power, 7-3
operation, 7-2
overview, 7-1
port list, 7-4
power supply filter circuit, 7-6
reset/initialization, 7-2
timing relationships, 7-4
Power supply
filter circuit, 7-6
Program counter, 2-9
Programming models
MAC, 2-10
overview, 2-7
registers, 1-15
SIM, 6-3
summary, B-1
supervisor, 2-10
user, 2-8

## R

RAM base address registers, 2-12
Registers
A0–A6, 2-9
A7, 2-9
AATR, 5-10
access control, 2-12
address, 2-9
AVR, 9-5
BAAR, 5-12
bus master park, 6-11
cache configuration, 2-12
CACR, 2-12
CCR, 2-9
chip-select
control, 10-8
mask, 10-7
module, 10-5
condition code, 2-9
CSR, 5-13
D0–D7, 2-8
data, 2-8, A-13
DMA byte count, 12-7
DMA source address, 12-7
DRAM

# INDEX


asynchronous
address and control, 11-5
DACR, 11-5
DCR, 11-4
DMR, 11-7
mode signals, 11-4
general operation, 11-3
synchronous
DACR, 11-20
DCR, 11-19
DMR, 11-22
mode settings, 11-33
I$^2$C
address, 8-6
control, 8-7
data I/O, 8-9
frequency divider, 8-6
status, 8-8
I2CR, 8-7
I2DR, 8-9
I2SR, 8-8
IADR, 8-6
IFDR, 8-6
integer data formats in, 2-13
interrupt controller
autovector, 9-5
pending and mask, 9-6
port assignment, 9-7
IPR and IMR, 9-6
IRQPAR, 9-7
JTAG
boundary scan, 19-7
bypass, 19-10
descriptions, 19-4
IDCODE, 19-6
instruction shift, 19-5
MBAR, 2-12, 6-4
MODCTL, 14-9
module base address, 2-12
MPARK, 6-11
PADAT, 15-2
PADDR, 15-2
PAR, 6-10, 15-1
parallel port
data, 15-2
pin assignment, 15-1
PBR, 5-16
pin assignment, 6-10
PLL control, 7-3
PLLCR, 7-3
programming model, 1-15
RAM base address, 2-12
RAMBAR, 2-12
RAREG/RDREG, 5-30
RCREG, 5-42
RDMREG, 5-44
reset status, 6-5
RSMP, 14-12
RSR, 6-5
RXLVL, 14-8
SDRAM mode initialization, 11-38
SIM
base address, 6-4
memory map, 6-3
software watchdog interrupt, 6-9
SR, 2-11
status, 2-11
supervisor, 1-16, 1-16
SWIVR, 6-9
SWSR, 6-9
SYPCR, 6-8
system protection control, 6-8
TCR, 13-4
TDR, 5-18
TER, 13-5
timer module
capture, 13-4
event, 13-5
mode, 13-3
reference, 13-4
TMR, 13-3
TSPC, 14-13
TXLVL, 14-10
UACR, 14-17
UART modules, 14-3–14-21
UCR, 14-13
UCSR, 14-12
UDU/UDL, 14-19
UIP, 14-20
UIPCR, 14-17
UISR, 14-18
UIVR, 14-20
user, 1-15
VB, 2-12
vector base, 2-12
WAREG/WDREG, 5-31
WCREG, 5-43
WDMREG, 5-45
XTDR, 5-19
$\overline{\text{RSTI}}$ timing, 7-5


## S


SDRAM
block diagram and major components, 11-2
controller registers, B-3
DACR initialization, 11-35
DCR initialization, 11-35
definitions, 11-2

# INDEX


DMR initialization, 11-37
example, 11-34
initialization code, 11-39
interface configuration, 11-34
mode register initialization, 11-38
overview, 11-1
Signal descriptions, 17-1
address bus, 17-7
address configuration, 17-15
address strobe, 17-9
bus
arbitration, 17-12
clock output, 17-13
data, 17-8
driven, 17-13
grant, 17-12
request, 17-12
chip-select module, 17-15
clock and reset, 17-13
clock input, 17-13
data bus, 17-8
data/configuration pins, 17-14
debug
high impedance, 17-20
JTAG, 17-21
processor clock output, 17-20
processor status debug data, 17-21
test
clock, 17-22
mode, 17-20
overview, 17-20
debug module/JTAG
test data input/development serial input, 17-22
test data output/development serial output, 17-22
test mode select/breakpoint, 17-21
test reset/development serial clock, 17-21
divide control, 17-15
DMA controller module
general, 17-17
transfer modifier/acknowledge, 17-18
DRAM controller
address strobes, 17-16
overview, 17-16
synchronous
clock enable, 17-17
column address strobe, 17-17
edge select, 17-17
row address strobe, 17-17
synchronous edge select, 17-17
write, 17-16
$I^2C$ module
general, 17-20
serial clock, 17-20
serial data, 17-20
interrupt control signals, 17-12
interrupt request, 17-12
JTAG, 19-2
parallel I/O port, 17-19
read/write, 17-8
reset in, out, 17-13
serial module
clear to Send, 17-19
general, 17-18
receiver serial data input, 17-19
request to send, 17-19
transmitter serial data output, 17-18
size, 17-8
timer module, 17-19
transfer
acknowledge, 17-9
in progress, 17-10
modifier, 17-10
start, 17-9
Signals
overview, 17-1
SIM
features, 6-1
programming model, 6-3
register memory map, 6-3
Software watchdog
interrupt vector register, 6-9
service register, 6-9
timer, 6-6
Stack pointer, 2-9, 2-9
Status register, 2-11
Supervisor
programming model, 2-10
registers, 1-16
System protection control register, 6-8


## T


Timer module
calculating time-out values, 13-7
capture registers, 13-4
code example, 13-6
counters, 13-5
event registers, 13-5
general-purpose programming model, 13-2
general-purpose units, 13-2
mode registers, 13-3
reference registers, 13-4
Timing
branch instruction execution, 2-30
MAC unit instructions, 3-5
MOVE instructions, 2-25
one operand, 2-26
PLL, 7-4
$\overline{\text{RSTI}}$, 7-5
two operands, 2-27

Transfers
internally generated, 6-12

## U

UART Modules
register description and programming
register description
UART module programming model (table 14-1), B-4
UART modules
bus operation, 14-37
interrupt acknowledge cycles, 14-37
read cycles, 14-37
write cycles, 14-37
clock source
baud rates, 14-24
controm registers, B-6
external clock, 14-25
FIFO stack in UART0, 14-32
initialization sequence, 14-38
looping modes, 14-34
automatic echo, 14-34
local loop-back, 14-34
remote loop-back mode, 14-35
mode registers, 14-5
multidrop mode, 14-35
programming, 14-37
receiver, 14-29
receiver in modem mode, 14-31
register descriptions, 14-3
serial overview, 14-2
signal definitions, 14-21
simplified block diagram, A-6
transmitter and receiver modes, 14-25
transmitter in modem mode, 14-27
transmitter/receiver clock source, 14-23
transmitting in UART mode, 14-26
UART1 in UART mode, 14-31
User registers, 1-15

## V

Vector base register, 2-12

Overview 1
Part I: MCF5407 Processor Core Part I
ColdFire Core 2
Hardware Multiply/Accumulate (MAC) Unit 3
Local Memory 4
Debug Support 5
Part II: System Integration Module (SIM) Part II
SIM Overview 6
Phase-Locked Loop (PLL) 7
$I^2C$ Module 8
Interrupt Controller 9
Chip-Select Module 10
Synchronous/Asynchronous DRAM Controller Module 11
Part III: Peripheral Module Part III
DMA Controller Module 12
Timer Module 13
UART Modules 14
Parallel Port (General-Purpose I/O) 15
Part IV: Hardware Interface Part IV
Mechanical Data 16
Signal Descriptions 17
Bus Operation 18
IEEE 1149.1 Test Access Port (JTAG) 19
Electrical Specifications 20
Appendix A: Migration A
Appendix B: Memory Map B
Glossary of Terms and Abbreviations GLO
Index IND

| | |
|---|---|
| 1 | Overview |
| Part I | Part I: MCF5407 Processor Core |
| 2 | ColdFire Core |
| 3 | Hardware Multiply/Accumulate (MAC) Unit |
| 4 | Local Memory |
| 5 | Debug Support |
| Part II | Part II: System Integration Module (SIM) |
| 6 | SIM Overview |
| 7 | Phase-Locked Loop (PLL) |
| 8 | $I^2C$ Module |
| 9 | Interrupt Controller |
| 10 | Chip-Select Module |
| 11 | Synchronous/Asynchronous DRAM Controller Module |
| Part III | Part III: Peripheral Module |
| 12 | DMA Controller Module |
| 13 | Timer Module |
| 14 | UART Modules |
| 15 | Parallel Port (General-Purpose I/O) |
| Part IV | Part IV: Hardware Interface |
| 16 | Mechanical Data |
| 17 | Signal Descriptions |
| 18 | Bus Operation |
| 19 | IEEE 1149.1 Test Access Port (JTAG) |
| 20 | Electrical Specifications |
| A | Appendix A: Migration |
| B | Appendix B: Memory Map |
| GLO | Glossary of Terms and Abbreviations |
| IND | Index |

# Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

NXP:

MCF5407AI162 MCF5407CAI162 MCF5407AI220 MCF5407CAI220