MOTOROLA

ColdFire® 2/2M

Integrated Microprocessor

User's Manual

DOCUMENTATION FEEDBACK

FAX 512-895-8593—Documentation Comments Only (no technical questions please)
http: / / www.mot.com/hpesd/docs_survey.html—Documentation Feedback Only

The Technical Communications Department welcomes your suggestions for improving our documentation and encourages you to complete the documentation feedback form at the World Wide Web address listed above. Your help helps us measure how well we are serving your information requirements.

The Technical Communications Department also provides a fax number for you to submit any questions or comments about this document or how to order other documents. Please provide the part number and revision number (located in upper right-hand corner of the cover) and the title of the document. When referring to items in the manual, please reference by the page number, paragraph number, figure number, table number, and line number if needed. **Please do not fax technical questions to this number.**

When sending a fax, please provide your name, company, fax number, and phone number including area code.

For Internet Access:

Web Only: http: // www.mot.com/aesop

For Hotline Questions:

FAX (US or Canada): 1-800-248-8567

Applications and Technical Information

For questions or comments pertaining to technical information, questions, and applications, please contact one of the following sales offices nearest you.

— Sales Offices —

Field Applications Engineering Available Through All Sales Offices

UNITED STATES

ALABAMA, Huntsville	(205) 464-6800
ARIZONA, Tempe	(602) 897-5056
CALIFORNIA, Agoura Hills	(818) 706-1929
CALIFORNIA, Los Angeles	(310) 417-8848
CALIFORNIA, Irvine	(714) 753-7360
CALIFORNIA, Roseville	(916) 922-7152
CALIFORNIA, San Diego	(619) 541-2163
CALIFORNIA, Sunnyvale	(408) 749-0510
COLORADO, Colorado Springs	(719) 599-7497
COLORADO, Denver	(303) 337-3434
CONNECTICUT, Wallingford	(203) 949-4100
FLORIDA, Maitland	(407) 628-2636
FLORIDA, Pompano Beach/ Fort Lauderdale	(305) 486-9776
FLORIDA, Clearwater	(813) 538-7750
GEORGIA, Atlanta	(404) 729-7100
IDAHO, Boise	(208) 323-9413
ILLINOIS, Chicago/Hoffman Estates	(708) 490-9500
INDIANA, Fort Wayne	(219) 436-5818
INDIANA, Indianapolis	(317) 571-0400
INDIANA, Kokomo	(317) 457-6634
IOWA, Cedar Rapids	(319) 373-1328
KANSAS, Kansas City/Mission	(913) 451-8555
MARYLAND, Columbia	(410) 381-1570
MASSACHUSETTS, Marborough	(508) 481-8100
MASSACHUSETTS, Woburn	(617) 932-9700
MICHIGAN, Detroit	(313) 347-6800
MINNESOTA, Minnetonka	(612) 932-1500
MISSOURI, St. Louis	(314) 275-7380
NEW JERSEY, Fairfield	(201) 808-2400
NEW YORK, Fairport	(716) 425-4000
NEW YORK, Hauppauge	(516) 361-7000
NEW YORK, Poughkeepsie/Fishkill	(914) 473-8102
NORTH CAROLINA, Raleigh	(919) 870-4355
OHIO, Cleveland	(216) 349-3100
OHIO, Columbus/Worthington	(614) 431-8492
OHIO, Dayton	(513) 495-6800
OKLAHOMA, Tulsa	(800) 544-9496
OREGON, Portland	(503) 641-3681
PENNSYLVANIA, Colmar	(215) 997-1020
Philadelphia/Horsham	(215) 957-4100
TENNESSEE, Knoxville	(615) 584-4841
TEXAS, Austin	(512) 873-2000
TEXAS, Houston	(800) 343-2692
TEXAS, Plano	(214) 516-5100
VIRGINIA, Richmond	(804) 285-2100
WASHINGTON, Bellevue	(206) 454-4160
Seattle Access	(206) 622-9960
WISCONSIN, Milwaukee/Brookfield	(414) 792-0122

CANADA

BRITISH COLUMBIA, Vancouver	(604) 293-7605
ONTARIO, Toronto	(416) 497-8181
ONTARIO, Ottawa	(613) 226-3491
QUEBEC, Montreal	(514) 731-6881

INTERNATIONAL

AUSTRALIA, Melbourne	(61-3)887-0711
AUSTRALIA, Sydney	(61(2)906-3855
BRAZIL, Sao Paulo	55(11)815-4200
CHINA, Beijing	86 505-2180
FINLAND, Helsinki	358-0-35161191
Car Phone	358(49)211501
FRANCE, Paris/Vanves	33(1)40 955 900

GERMANY, Langenhagen/ Hanover	49(511)789911
GERMANY, Munich	49 89 92103-0
GERMANY, Nuremberg	49 911 64-3044
GERMANY, Sindelfingen	49 7031 69 910
GERMANY, Wiesbaden	49 611 761921
HONG KONG, Kwai Fong	852-4808333
Tai Po	852-6668333
INDIA, Bangalore	(91-812)627094
ISRAEL, Tel Aviv	972(3)753-8222
ITALY, Milan	39(2)82201
JAPAN, Aizu	81(241)272231
JAPAN, Atsugi	81(0462)23-0761
JAPAN, Kumagaya	81(0485)26-2600
JAPAN, Kyushu	81(092)771-4212
JAPAN, Mito	81(0292)26-2340
JAPAN, Nagoya	81(052)232-1621
JAPAN, Osaka	81(06)305-1801
JAPAN, Sendai	81(22)268-4333
JAPAN, Tachikawa	81(0425)23-6700
JAPAN, Tokyo	81(03)3440-3311
JAPAN, Yokohama	81(045)472-2751
KOREA, Pusan	82(51)4635-035
KOREA, Seoul	82(2)554-5188
MALAYSIA, Penang	60(4)374514
MEXICO, Mexico City	52(5)282-2864
MEXICO, Guadalajara	52(36)21-8977
Marketing	52(36)21-9023
Customer Service	52(36)669-9160
NETHERLANDS, Best	(31)49988 612 11
PUERTO RICO, San Juan	(809)793-2170
SINGAPORE	(65)2945438
SPAIN, Madrid	34(1)457-8204
or	34(1)457-8254
SWEDEN, Solna	46(8)734-8800
SWITZERLAND, Geneva	41(22)7991111
SWITZERLAND, Zurich	41(1)730 4074
TAIWAN, Taipei	886(2)717-7089
THAILAND, Bangkok	(66-2)254-4910
UNITED KINGDOM, Aylesbury	44(296)395-252

FULL LINE REPRESENTATIVES

COLORADO, Grand Junction	
Cheryl Lee Whitely	(303) 243-9658
KANSAS, Wichita	
Melinda Shores/Kelly Greiving	(316) 838 0190
NEVADA, Reno	
Galena Technology Group	(702) 746 0642
NEW MEXICO, Albuquerque	
S&S Technologies, Inc.	(505) 298-7177
UTAH, Salt Lake City	
Utah Component Sales, Inc.	(801) 561-5099
WASHINGTON, Spokane	
Doug Kenley	(509) 924-2322
ARGENTINA, Buenos Aires	
Argonics, S.A.	(541) 343-1787

HYBRID COMPONENTS RESELLERS

Elmo Semiconductor	(818) 768-7400
Minco Technology Labs Inc.	(512) 834-2022
Semi Dice Inc.	(310) 594-4631

**For More Information On This Product,
Go to: www.freescale.com**

Freescale Semiconductor, Inc.

PREFACE

The *ColdFire2/2M Integrated Microprocessor User's Manual* describes the programming, capabilities, and operation of the ColdFire2/2M device. Refer to the *MCF5200 ColdFire Family Programmer's Reference Manual Rev. 1.0* for information on the ColdFire Family of microprocessors.

Throughout this document, the ColdFire2/2M integrated microprocessor is referred to as "the ColdFire2/2M."

CONTENTS

This user manual is organized as follows:

Section 1: Overview
Section 2: Signal Summary
Section 3: Master Bus Operations
Section 4: Exception Processing
Section 5: Integrated Memories
Section 6: Multiply-Accumulate Unit
Section 7: Debug Support
Section 8: Test Operation
Section 9: Instruction Execution Timing
Section 10: Electrical Characteristics
Appendix A: Register Summary
Appendix B: New MAC Instructions
Index

TABLE OF CONTENTS

| **Paragraph
Number** | **Title** | **Page
Number** |

**Section 1
Overview**

Section 2
Signal Summary

Section 3
Master Bus Operation

Freescale Semiconductor, Inc.

TABLE OF CONTENTS (Continued)

**Section 4
Exception Processing**

Section 6
Multiply-Accumulate Unit

Section 7
Debug Support

Section 8
Test Operation

MOTOROLA   ColdFire2/2M User's Manual   xv
For More Information On This Product,
Go to: www.freescale.com

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

Figure Number	**Title**	**Page Number**

Freescale Semiconductor, Inc.

LIST OF ILLUSTRATIONS (Continued)

LIST OF TABLES

LIST OF ACRONYMS

Acronym	Definition
BCD	binary coded decimal
BDM	background debug mode
CAD	computer-aided design
CPU	central processing unit
DSP	digital signal processing
ET	execution time
IACK	interrupt acknowledge
IFP	instruction fetch pipeline
LRU	least recently used
LSB	least significant **bit**
LSW	least significant word
MAC	multiply-accumulate
MARB	master bus arbiter
MSB	most significant bit
MSW	most significant word
OEP	operand execution pipeline
RISC	reduced instruction set computer
ROM	read-only memory
SBC	system bus controller
SRAM	static random access memory
TA	transfer acknowledge
TS	transfer start

SECTION 1
OVERVIEW

This is a summary of the use and operation of the FlexCore ColdFire® microprocessor core (referred to as the ColdFire2) and FlexCore ColdFire microprocessor core with the Multiply-Accumulate unit (MAC), referred to as the ColdFire2M. It also contains a detailed set of timing and electrical specifications. All references to ColdFire2/2M will apply to both the ColdFire2 and the ColdFire2M devices. Refer to the *ColdFire Programmer's Reference Manual Rev 1.0* (MCF5200PRM/AD) for detailed information on the operation of the instruction set and addressing modes.

The ColdFire2/2M is part of the FlexCore Program, a semicustom, standard-cell based design program. Based on the concept of variable-length Reduced Instruction Set Computer (RISC) technology, ColdFire combines the architectural simplicity of conventional 32-bit RISC with a memory-saving, variable-length instruction set. In the FlexCore program, high-volume manufacturers can create their own integrated microprocessor containing a core processor (such as the ColdFire2/2M) and their own proprietary technology. A FlexCore integrated processor allows significant reductions in component count, power consumption, board space, and cost—resulting in higher system reliability and performance.

The main features of the ColdFire2/2M processor include:

- 32-bit address bus which can directly address up to 4 Gbytes
- 32-bit data bus
- Variable-length RISC
- Optimized instruction set for high-level language constructs
- Sixteen general-purpose 32-bit data- and address- registers
- Multiply Accumulate (MAC) unit for DSP applications (ColdFire2M only)
- Supervisor/user modes for system protection
- Vector-base register to relocate exception-vector table
- Special core interfacing signals for integrated memories
- Full debug support

The ColdFire2/2M has 32-bit address and data busses. The 32-bit address bus allows direct addressing of up to 4 Gbytes. A misalignment unit provides support for misaligned data accesses, and an optional bus arbitration unit provides support for additional bus masters. The ColdFire2/2M also supports an integrated instruction cache, SRAM, and ROM (maximum of 32 Kbyte each.)

Freescale Semiconductor, Inc.

The ColdFire2/2M supports a subset of the 68000 instruction set, and the ColdFire2M has the added functionality of a Multiply Accumulate (MAC) unit for DSP applications. The removed instructions include binary coded decimal, bit field, logical rotate, decrement and branch, integer division, and integer multiply with a 64-bit result. In addition, only the twelve addressing modes supported by the 68000 are supported. User-mode code developed for the ColdFire2 processor will run unchanged on 68020, 68030, 68040, and 68060 processors. The new MAC instructions on the ColdFire2M processor will not run on other processors.

A complete debug module is integrated into the ColdFire2/2M. This unit provides real-time trace, background debug mode, and real-time debug support. This includes a parallel processor status port, a subset of the background debug mode (BDM) functionality found on Motorola's 683xx family of parts, and real-time breakpoint capability. This built-in debug support results in a standard debug interface to established tools for all ColdFire-based processors.

1.1 FLEXCORE INTEGRATED PROCESSORS

The FlexCore design methodology allows designers of high-volume digital systems and third-party technology providers to place their proprietary circuitry on-chip with a Motorola microprocessor core. By using FlexCore, a designer can reduce the total system cost, component count, and power consumption while providing higher performance and greater reliability. Custom logic, memory, and peripheral modules can be added to a core processor to produce the most cost-effective solution for a designer's system. Core processors provide special power-management features such as 5 V, 3.3 V, and static operation. The 68000 family of processors have a proven architecture with a broad base of application and system software support, including real-time kernels, operating systems, and compilers, in addition to a wide range of tools to support software development. In the future, additional processing architectures will be included in the FlexCore program, including PowerPC™ and Digital Signal Processing (DSP). Figure 1-1 shows a typical die layout of a FlexCore integrated processor.
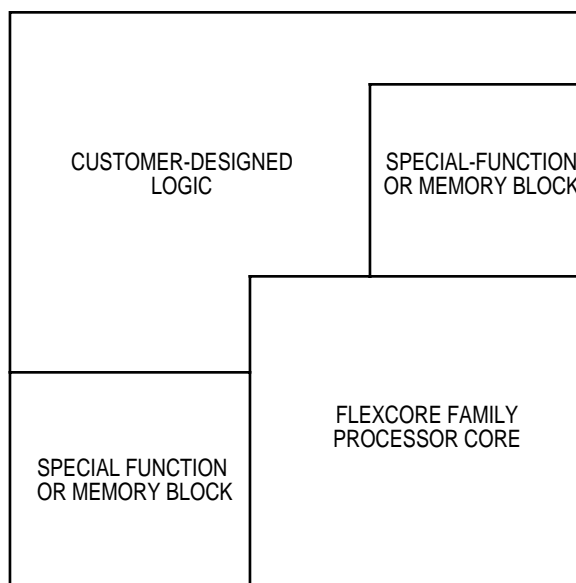
Figure 1-1. FlexCore Integrated Processor Typical Die Layout

Complete product lines can be created using FlexCore by implementing one base design using a variety of core processors. Designers already familiar with 680x0 Family design can easily migrate to FlexCore processors as the core processors use similar interfaces. Additionally, many peripheral modules and memory elements are available for integration. Motorola has developed a complete design system for the customer that includes both a broad cell-based library and effective Computer-Aided Design (CAD) tools. By building on Motorola's proven 680x0 microprocessor architecture and superior manufacturing capabilities, FlexCore offers designers the best path to higher system integration.

FlexCore custom processors are ideal for:

- High-volume users of 8-, 16-, and 32-bit integrated solutions requiring higher system performance whose needs are not met by standard 68300 Family devices.
- Designers of high-volume applications who need to reduce cost, space, and/or power consumption.
- Third-party technology providers who want to deliver their proprietary application-specific technology to a worldwide marketplace.

To develop a solution that best suits system requirements in the shortest time frame, integrated processor design is performed by the designer using a methodology created, tested, and documented by Motorola. The resulting netlist is then laid out by Motorola, verified, and fabricated in silicon. This enables FlexCore integrated processors to be produced quickly and cost-effectively, with the resulting device integrating many of the discrete functions needed in the final system.

To implement the application-specific logic, the designer uses Motorola's standard-cell library. This library offers an extensive range of design elements, memory configurations, and an expanding array of peripheral modules. Each cell in the library has been designed

for optimum size, power, and performance. The added flexibility of high-speed, high-density cells allows the designer to achieve the most cost-effective solution while satisfying critical timing requirements. The standard cell library has been thoroughly characterized and maintained to ensure a smooth transition from a simulated design to working silicon. If both Motorola and the customer have the desire, a custom part may also become a standard product. Standard products are sold on the open market, allowing costs to be spread over additional units, resulting in lower component prices for high-volume users.

Third-party technology providers can use the same methodology to combine their application-specific systems expertise with a core processor. The resulting device is manufactured by Motorola and can be delivered to the marketplace through either the technologist's or Motorola's marketing and sales channels.

Refer to the *Design System User's Guide for Semicustom & FlexCore* for more information on the FlexCore design methodology.

1.1.1 FlexCore Advantages

Developers face challenges in reducing product cost. By incorporating user-designed logic and Motorola-supplied functions into a single FlexCore processor, a system designer can realize significant savings in cost, power consumption, board space, and pin count. The equivalent functionality can easily require 20 separate components. Each component might have 16–64 pins, totaling over 350 connections. Each connection is a candidate for a bad solder joint or misrouted trace. Each component is another part to qualify, purchase, inventory, and maintain. Each component requires a share of the printed circuit board. Each component draws power—often to drive large buffers and circuit board traces to get signals to another chip. Each component must be individually placed and attached to a printed circuit board. The signals between the core processor unit and a peripheral might not be compatible nor run from the same clock, requiring time delays or other special design considerations.

In a FlexCore integrated processor, the major functions and glue logic are all properly connected internally, timed with the same clock, and fully tested. Only essential signals are brought out to pins. The processor is then assembled and tested in a surface-mount package for the smallest possible footprint.

1.1.2 FlexCore Module Types

The three types of FlexCore modules are:

- Hard Module
    - — Not alterable
    - — Laid out
    - — Has a technology file containing timing data
    - — Has a defined test scheme
- Soft Module
    - — Netlist
    - — Not alterable other than by clock tree insertion

— Not laid out
— Has a defined test scheme
— Simulation test fixture is provided

• Parameterizable Module

— Alterable via insertion of predefined parameters
— Behavioral model
— Definition of parameters defines test scheme
— Customer selects parameter values and Motorola synthesizes the design

The ColdFire2/2M is available as a hard module only.

1.2 DEVELOPMENT CYCLE

There are several steps that must be followed in order to create a FlexCore integrated microprocessor with a ColdFire2/2M. Figure 1-2 illustrates the standard cell design flow. These steps include:

• Schematic capture on workstation—Use the engineering workstation to implement the required system functions with a ColdFire2/2M, memory blocks, peripherals, and cells from the Motorola standard cell library.

• Verilog modules—Optionally use Verilog to implement complex user modules or system interconnect of standard cells.

• Generate test patterns—Generate the stimulus and test patterns for the design to be used during functional simulation.

• Module compilation—Use Motorola software to generate compiled modules (SRAM, ROM, etc.)

• Functional simulation—Ensure that the logic of the system is functionally sound by simulating the design. (No timing information is yet associated with the simulations, and all propagation delays are preset to a unit delay.)

• Logic synthesis—The behavioral and structural level description of the design is mapped to a more efficient structural description using Synopsys. This final description is a netlist of standard cell components.

• Electrical rule check—Motorola software verifies the electrical integrity of the design. This includes checking connectivity, fan-out, edge rates, and other electrical rules.

• Calculate node delays— Motorola software calculates the estimated propagation delays of each node in the circuit. The design software estimates delays based on the fanout, input rise/fall times, drive characteristics, and estimated interconnect capacitances of the netlist.

• Path delay analysis—With path delay information, the delays between the clocked elements of the circuit can be determined, and the critical paths that limit the clock rate can be identified. Checking for setup, hold, and pulse-width violations can also be accomplished.

• Perform real-time simulation—The real-time simulation is run to verify full functionality using the estimated propagation delays calculated by the design tools.

• Package and pinout definition—Develop the package and pin definition file.

- Extract test vectors—The simulator records the input/output patterns generated during the real-time simulation. The test vectors that Motorola will use to test the prototypes are derived from these patterns.

- Post testkit verification—Simulate the design using the extracted test vectors to ensure proper operation.

- Perform fault grading—Determine the fault coverage of the extracted test vectors.

- Timing constraints—Chip-level timing constraints are created to be used during floorplanning, clock tree synthesis, placement, and routing.

- Floor planning/Clock tree synthesis—Floor planning and clock tree synthesis are performed by Motorola using the design timing constraints provided by the designer.

- Automatic place & route—The circuit's physical layout is created by Motorola from the netlist using automatic place and route software.

- Post-layout delay calculation—After the cells are placed and routed, the interconnect resistances and capacitances are extracted by Motorola. Extracted elements replace those estimated earlier during the pre-layout calculation of the node delays.

- Re-simulate—The circuit is re-simulated with Verilog to ensure no problems have arisen due to a change in load conditions.

- In-place optimization—If the new delay information causes the simulation to fail, Synopsys is used to further optimize the layout. Placement and routing is then repeated. This cycle continues until the final layout, with post-layout delay, satisfies the design goals.

- Post testkit simulation—Simulate the design using the extracted test vectors to ensure proper operation.

- Re-extract test vectors—Extract the test vectors again in order to account for timing changes due to more accurate delay analysis after layout and routing.

- Power simulation—A power simulation is performed to determine if the design meets the necessary design goals. Power simulation should also be run early in the design cycle to ensure design goals are met.

- Netlist comparison—The netlist after place and route is compared by Motorola to the original netlist to check for connectivity errors.

- Pattern, mask, and wafer generation—Motorola generates the patterns, masks, and wafers.

- Assembly/test—Parts are assembled by Motorola and tested using the test vectors.

- Ship tested prototypes—Tested prototypes are shipped from Motorola to the customer.

- Final test program—The final test is performed by Motorola.

Freescale Semiconductor, Inc.

```
┌──────────────────┐  ┌──────────────────┐  ┌──────────────────┐
│ VERILOG MODULES  │  │ GENERATE TEST    │  │ SCHEMATIC CAPTURE│
│                  │  │ PATTERNS         │  │ ON WORKSTATION   │
└──────────────────┘  └──────────────────┘  └──────────────────┘
                ┌─────────────────────────┐     ┌──────────────────┐
                │ FUNCTIONAL SIMULATION    │◄────│ MODULE COMPILATION│
                └─────────────────────────┘     └──────────────────┘
                ┌─────────────────────────┐
                │ LOGIC SYNTHESIS          │
                └─────────────────────────┘
                ┌─────────────────────────┐
                │ ELECTRICAL RULE CHECK    │
                └─────────────────────────┘
                ┌─────────────────────────┐          ┌──────────────────┐
                │ CALCULATE NODE DELAYS    │ - - - -  │ PATH DELAY ANALYSIS│
                └─────────────────────────┘          └──────────────────┘
                ┌─────────────────────────┐
                │ PERFORM REAL-TIME        │
  ┌──────────────┐  │ SIMULATION          │
  │ TIMING       │  └─────────────────────┘
  │ CONSTRAINTS  │  ┌─────────────────────────┐
  └──────────────┘  │ PACKAGE AND PINOUT      │
  ┌──────────────┐  │ DEFINITION              │
  │ FLOOR        │  └─────────────────────────┘
  │ PLANNING /   │  ┌─────────────────────────┐
  │ CLOCK TREE   │  │ EXTRACT TEST VECTORS    │
  │ SYNTHESIS    │  └─────────────────────────┘
  └──────────────┘  ┌─────────────────────────┐   ┌──────────────────┐
                    │ POST TESTKIT VERIFICATION│═══│ PERFORM FAULT     │
                    └─────────────────────────┘   │ GRADING          │
                                                  └──────────────────┘
                    ┌─────────────────────────┐
                    │ AUTOMATIC PLACE & ROUTE │
                    └─────────────────────────┘
                    ┌─────────────────────────┐   ┌──────────────────┐
                    │ POST-LAYOUT DELAY        │   │ IN-PLACE         │
                    │ CALCULATION              │   │ OPTIMIZATION     │
                    └─────────────────────────┘   └──────────────────┘
                    ┌─────────────────────────┐
                    │ RE-SIMULATE             │
                    └─────────────────────────┘
                    ┌─────────────────────────┐
                    │ POST TESTKIT SIMULATION │
                    └─────────────────────────┘
  ┌──────────────┐  ┌─────────────────────────┐   ┌──────────────────┐
  │ POWER        │  │ EXTRACT TEST VECTORS    │   │ NETLIST          │
  │ SIMULATION   │  └─────────────────────────┘   │ COMPARISON       │
  └──────────────┘                                └──────────────────┘
                    ┌─────────────────────────┐
                    │ PATTERN, MASK AND        │
                    │ WAFER GENERATION         │
                    └─────────────────────────┘
                    ┌─────────────────────────┐
                    │ ASSEMBLY / TEST         │
                    └─────────────────────────┘
                    ┌─────────────────────────┐   ┌──────────────────┐
                    │ SHIP TESTED PROTOTYPES  │◄──│ FINAL TEST PROGRAM│
                    └─────────────────────────┘   └──────────────────┘

- - - ►  OPTIONAL STEPS TO BE DONE IN PARALLEL
─────►  STEPS TO BE DONE IN PARALLEL

▨ MOTOROLA   ☐ CUSTOMER
```

Figure 1-2. Design System Overview

1.3 SYSTEM ARCHITECTURE

Most FlexCore custom processors will be designed according to a standard system architecture. A block diagram of this architecture is shown in Figure 1-3. This architecture contains the standardized busses and modules discussed below.
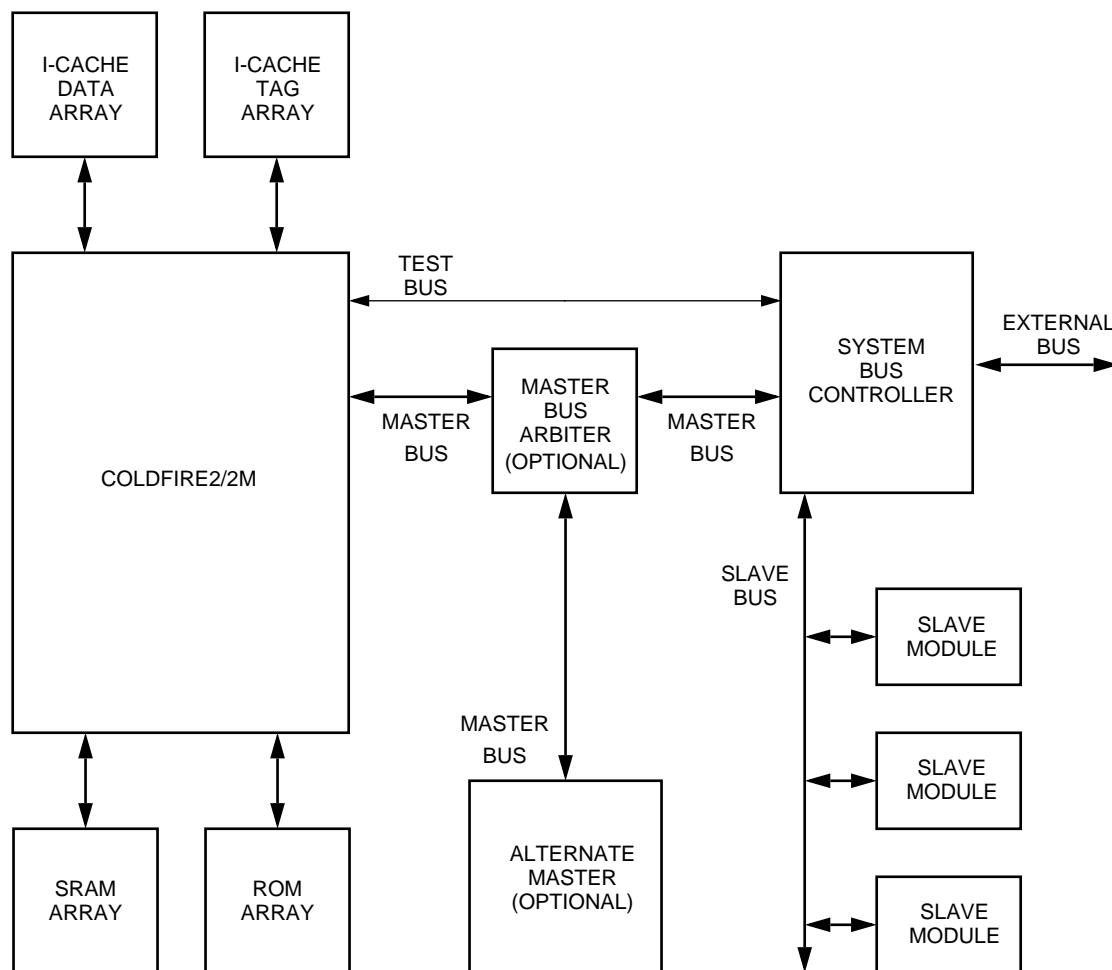
Figure 1-3. ColdFire2/2M System Diagram

For more information on ColdFire system configurations refer to the *MCF5204 User's Manual* (MCF5204UM/AD) and the *MCF5206 User's Manual* (MCF5206UM/AD.)

1.3.1 Internal Bus Structure

1.3.1.1 MASTER BUS. The master bus is the primary data interface for the ColdFire2/2M. It is a basic two-cycle unidirectional bus. Devices on the master bus are capable of initiating bus transactions. This bus includes a 32-bit address bus, 32-bit read data bus, 32-bit write data bus, and various control signals. None of the signals can be tri-stated. As a result, an optional master bus arbiter is required if multiple masters are present in the system. Refer to **Section 3 Master Bus Operation** for more information.

1.3.1.2 SLAVE BUS. The slave bus is a simplified bus that provides the interface between the internal master bus and the on-chip peripheral modules. The system bus controller (SBC) transfers information between the master bus and the slave bus and is the slave bus master. Only the SBC can initiate slave bus transactions. The slave bus includes device select lines, interrupt lines, output enables, write enables, interrupt vector enables, and other control signals to interface to slave modules. All signals are unidirectional and can not be tri-stated.

1.3.1.3 EXTERNAL BUS. The external bus provides the interface between the internal master bus and external resources. This bus has no predefined requirements. It can be asynchronous or synchronous. The address and data bus widths may be different than the internal master bus. The SBC provides the necessary translation between the master and external busses.

1.3.1.4 TEST BUS. The test bus provides an interface for performing extensive tests of the integrated memories. Signals are provided to control reading and writing of the integrated SRAMs, SROMs, instruction cache tag RAM, and the instruction cache data RAM.

1.3.2 System Functional Blocks

1.3.2.1 ALTERNATE MASTER. Any device that is required to initiate bus transactions is required to be on the master bus. All master bus devices except the ColdFire2/2M that initiate bus transactions are considered to be alternate masters. Use of alternate masters requires a master bus arbiter module. Examples of alternate masters include DMA controllers and coprocessors.

1.3.2.2 COLDFIRE2/2M. The ColdFire2/2M is the primary CPU for any ColdFire custom processor. It has dedicated busses for instruction cache and integrated memories. The master bus is the data interface bus used for all data movement to and from the CPU and is usually connected to the system bus controller. Clock and debug signals are connected directly to I/O pins. See **Section 2 Signal Summary** for more information on all of the ColdFire2/2M interface signals. A block diagram of the CPU is shown in Figure 1-4.

Freescale Semiconductor, Inc.
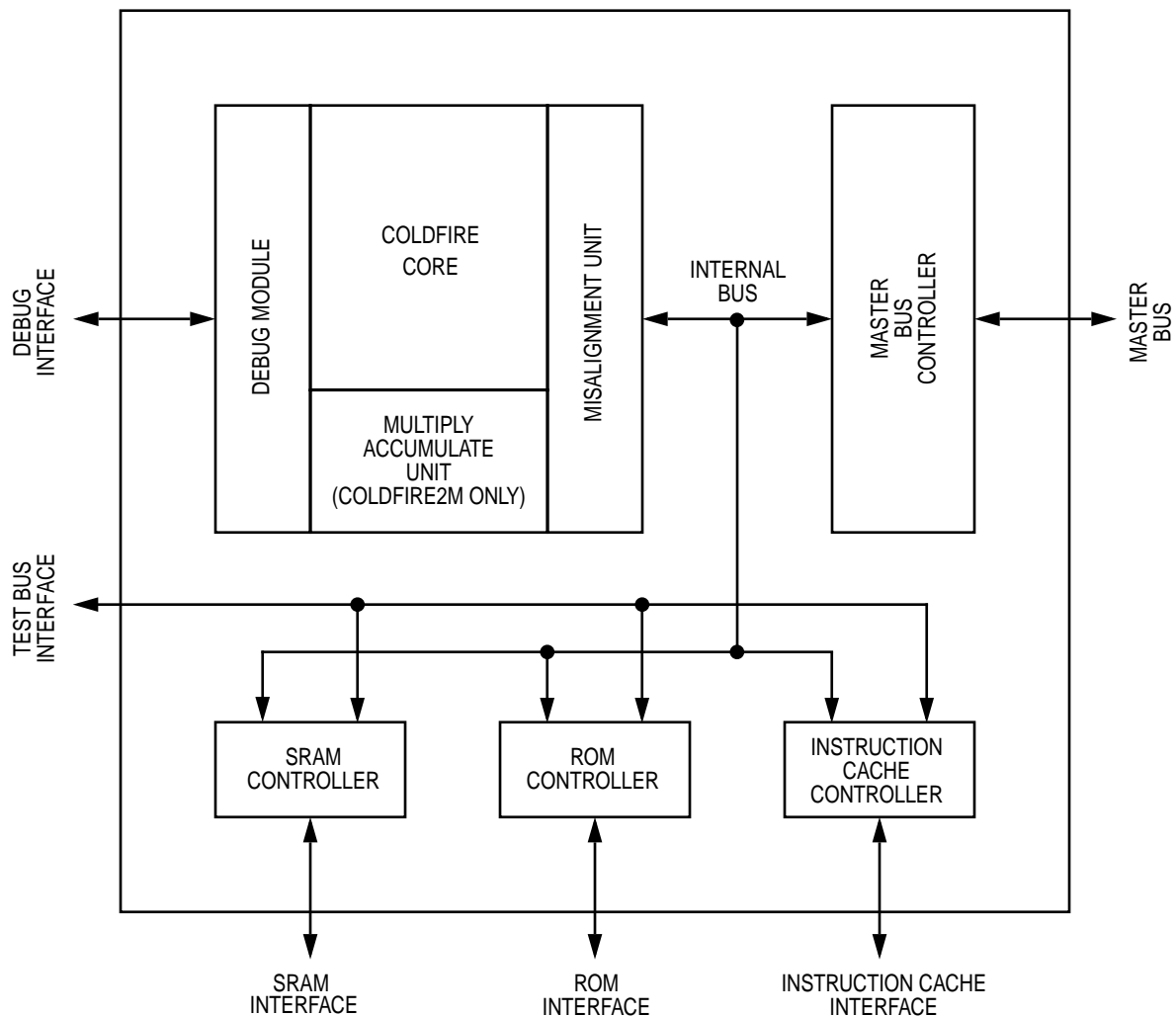
Figure 1-4. ColdFire2/2M Block Diagram

1.3.2.3 I-CACHE DATA ARRAY. The optional instruction cache data array is a compiled RAM used to hold cache data. It is connected directly to the ColdFire2/2M via a dedicated bus. Refer to **Section 5.1 Instruction Cache** for more information on the cache configuration and interface signals.

1.3.2.4 I-CACHE TAG ARRAY. The optional instruction cache tag array is a compiled RAM used to hold tag information for the cache. It is connected directly to the ColdFire2/2M via a dedicated bus. Refer to **Section 5.1 Instruction Cache** for more information on the cache configuration and the interface signals.

1.3.2.5 MASTER BUS ARBITER (MARB). The optional master bus arbiter is required to support multiple masters on the master bus. It performs bus control and multiplexing for the unidirectional signals **on** the master bus. Refer to **Section 3.10 Master Bus Arbitration** for more information.

For More Information On This Product,
Go to: www.freescale.com

Freescale Semiconductor, Inc.

1.3.2.6 ROM ARRAY. The optional ROM array is a compiled ROM used to hold boot code, critical code, and monitor code. It is connected directly to the ColdFire2/2M via a dedicated bus. Refer to **Section 5.3 ROM Module** for more information on the ROM configuration and the interface signals.

1.3.2.7 SLAVE MODULES. The slave modules are on-chip peripherals. They communicate with the ColdFire2/2M via the slave bus and SBC. Slave modules are always bus slaves and cannot initiate bus transactions except via interrupts. Examples of slave modules include serial ports, parallel ports, and timers.

1.3.2.8 SRAM ARRAY. The optional SRAM array is a compiled RAM used to hold critical variables and the stack. It is connected directly to the ColdFire2/2M processor via a dedicated bus. Refer to **Section 5.4 SRAM Module** for more information on the SRAM configuration and the interface signals.

1.3.2.9 SYSTEM BUS CONTROLLER (SBC). The system bus controller is responsible for providing overall control of the slave and external busses. The system bus controller is the single slave-bus master and interrupt controller, a possible external bus master, bus arbiter and interrupt controller, and a master-bus slave and interrupt controller. The SBC provides programmable registers to configure the memory map and interrupt control. The SBC provides master bus cycle termination for accesses to slave modules on the slave bus. It also generates interrupts on the master bus when requested by slaves on the slave bus, and it responds to interrupt acknowledge cycles on the master bus.

1.4 PROGRAMMING MODEL

The ColdFire2/2M programming model consists of three register groups: integer unit user, MAC unit user, and supervisor. Programs executing in the user mode use only the registers in the integer and MAC groups. System software executing in the supervisor mode can access all registers and use the control registers in the supervisor group to perform supervisor functions. The following paragraphs provide a brief description of the registers in the user and supervisor models. Refer to **Appendix A Register Summary**.

1.4.1 Integer Unit User Programming Model

Figure 1-5 illustrates the integer portion of the user programming model. It consists of the following registers:

- 16 general-purpose 32-bit registers (D0 – D7, A0 – A7)
- 32-bit Program Counter (PC)
- 8-bit Condition Code Register (CCR)

Freescale Semiconductor, Inc.

31	16	15	8	7	0	

Data Register 0 (D0)
Data Register 1 (D1)
Data Register 2 (D2)
Data Register 3 (D3)
Data Register 4 (D4)
Data Register 5 (D5)
Data Register 6 (D6)
Data Register 7 (D7)
Address Register 0 (A0)
Address Register 1 (A1)
Address Register 2 (A2)
Address Register 3 (A3)
Address Register 4 (A4)
Address Register 5 (A5)
Address Register 6 (A6)
Stack Pointer (SP,A7)
Program Counter (PC)
Condition Code Register (CCR)

Figure 1-5. Integer Unit User Programming Model

1.4.1.1 DATA REGISTERS (D0 – D7) . Registers D0–D7 are used as data registers for bit (1 bit), byte (8 bits), word (16 bits), and long word (32 bits) operations and may also be used as index registers.

1.4.1.2 ADDRESS REGISTERS (A0 – A6) . These registers can be used as software stack pointers, index registers, or base address registers and may be used for word and long word operations.

1.4.1.3 STACK POINTER (A7,SP) . ColdFire2/2M supports a single hardware stack pointer (A7) used during stacking for subroutine calls, returns, and exception handling. The initial value of A7 is loaded from the reset exception vector, address $0. The same register is used for user mode and supervisor mode, and may be used for word and long word operations.

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

1.4.1.4 PROGRAM COUNTER (PC). The PC contains the address of the currently executing instruction. During instruction execution and exception processing, the processor automatically increments the contents of the PC or places a new value in the PC, as appropriate. For some addressing modes, the PC can be used as a pointer for PC-relative operand addressing.

1.4.1.5 CONDITION CODE REGISTER (CCR). The CCR is the least significant byte of the processor Status Register (SR), as shown in Figure 1-6. Bits 4–0 represent indicator flags

based on results generated by processor operations. Bit 4, the extend bit (X bit), is also used as an input operand during multiprecision arithmetic computations.

BITS	7	6	5	4	3	2	1	0
FIELD	-	-	-	X	N	Z	V	C
RESET	-	-	-	-	-	-	-	-
R/W	R	R	R	R/W	R/W	R/W	R/W	R/W

Figure 1-6. Condition Code Register (CCR)

Field Definitions:

X[4]—Extend Condition Code

Assigned the value of the carry bit for arithmetic operations; otherwise not affected or set to a specified result.

N[3]—Negative Condition Code

Set if the most significant bit of the result is set; otherwise cleared.

Z[2]—Zero Condition Code

Set if the result equals zero; otherwise cleared.

V[1]—Overflow Condition Code

Set if an arithmetic overflow occurs implying that the result cannot be represented in the operand size; otherwise cleared.

C[0]—Carry Condition Code

Set if a carryout of the most significant bit of the operand occurs for an addition, or if a borrow occurs in a subtraction; otherwise cleared.

1.4.2 MAC Unit User Programming Model

Figure 1-7 illustrates the MAC portion of the user programming model available on the ColdFire2M only. It consists of the following registers:

- 32-bit accumulator (ACC)
- 16-bit mask register (MASK)
- 8-bit MAC status register (MACSR)
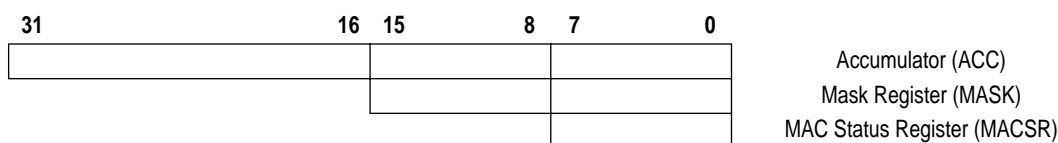
Figure 1-7. MAC Unit User Programming Model

Freescale Semiconductor, Inc.

See **Section 6.2 MAC Programming Model** for more details.

1.4.2.1 ACCUMULATOR (ACC). This is a 32-bit special purpose register used to accumulate the results of MAC operations.

1.4.2.2 MASK REGISTER (MASK). This is a 16-bit special purpose register used to hold the address mask for MAC load operations.

1.4.2.3 MAC STATUS REGISTER (MACSR). This is an 8-bit special purpose register used to hold the status of MAC operations.

1.4.3 Supervisor Programming Model

System programers use the supervisor programming model to implement sensitive operating system functions. The following paragraphs briefly describe the registers in the supervisor programming model. All accesses that affect the control features of the ColdFire2/2M are in the supervisor programming model, which consist of the registers available to users as well as the registers listed in Figure 1-8.
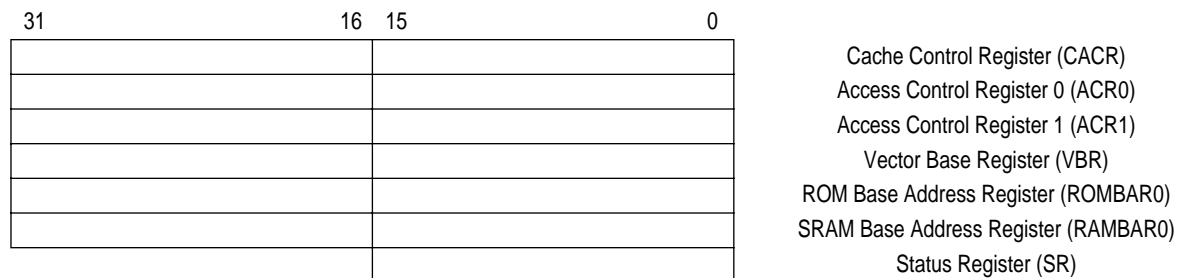
Figure 1-8. Supervisor Programming Model

Most of the control registers are accessed via the MOVEC instruction using the control register definitions shown in Table 1-1. These are a subset of the registers defined in the *ColdFire Programmer's Reference Manual Rev. 1.0* (MCF5200PRM/AD).

Table 1-1. MOVEC Register Map

RC[11:0]	REGISTER DEFINITION
$002	Cache Control Register (CACR)
$004	Access Control Register 0 (ACR0)
$005	Access Control Register 1 (ACR1)
$801	Vector Base Register (VBR)
$C00	ROM Base Address Register (ROMBAR0)
$C04	RAM Base Address Register 0 (RAMBAR0)

Refer to **Appendix A Register Summary** for a description of the access methods for all of the register in the ColdFire2/2M.

1.4.3.1 STATUS REGISTER (SR). Figure 1-9 illustrates the SR, which stores the processor status and contains the condition codes that reflect the results of a previous operation. The low-order byte of the SR is the condition code register (CCR.)

BITS	15	14	13	12	11	10	8	7	5	4	3	2	1	0
FIELD	T	-	S	M	-	I		-		X	N	Z	V	C
RESET	0	0	1	1	0	7		0		0	0	0	0	0
R/W	R/W	R	R/W	R/W	R	R/W		R		R/W	R/W	R/W	R/W	R/W

Figure 1-9. Status Register (SR)

Field Definitions:

T[15]—Trace Enable

When set, the processor will perform a trace exception after every instruction; otherwise no trace exception is performed.

S[13]—Supervisor / User State

Denotes the processor privilege mode: supervisor mode (S set) or user mode (S cleared).

M[12]—Master / Interrupt State

This bit is cleared by an interrupt exception, and can be set by software during execution of the RTE or move to SR instructions.

I[10:8]—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 level seven request, which cannot be masked.

1.4.3.2 CACHE CONTROL REGISTER (CACR). The CACR controls the cache operation. This includes cache enable, cache freeze, cache invalidate, cache mode, and default write protect. See **Section 5.1.11 Cache Control Register (CACR)** for more information.

1.4.3.3 ACCESS CONTROL REGISTERS (ACR0, ACR1). The ACRs allow definition of access attributes for two definable memory regions. These attributes include burst control, instruction caching, and write protection. These attributes override the defaults in the CACR. See **Section 5.2.1 ACR Programming Model** for more information.

1.4.3.4 VECTOR BASE REGISTER (VBR). The VBR contains 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. Only the upper 12 bits of the VBR are used and the lower 20 bits are filled with zeros. This forces the exception vector table to be aligned on a 1 MByte boundary. This register is reset to zero.

1.4.3.5 ROM BASE ADDRESS REGISTER (ROMBAR0). The ROMBAR0 register configures the internal ROM module. This includes the base address, code space masks, and enable. See **Section 5.3.2 ROM Programming Model** for more information.

1.4.3.6 SRAM BASE ADDRESS REGISTER (RAMBAR0). The RAMBAR0 register configures the internal SRAM module. This includes the base address, write protect, code

space masks, and enable. See **Section 5.4.3 RAM Programming Model** for more information.

1.5 INTEGER DATA FORMATS

Table 1-2 lists the operand data formats that are supported by the integer unit. Integer unit 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 1-2. Integer Data Formats

OPERAND DATA FORMAT	SIZE
Bit	1 Bit
Byte Integer	8 Bits
Word Integer	16 Bits
Long-Word Integer	32 Bits

1.6 ORGANIZATION OF DATA IN REGISTERS

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

1.6.1 Organization of Integer Data Formats in Registers

Figure 1-10 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. Long-word 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.
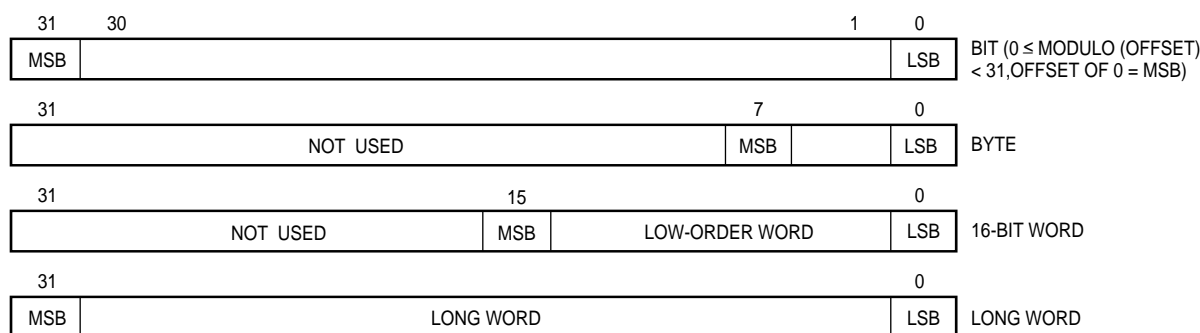
Figure 1-10. Organization of Integer Data Formats in Data Registers

Because address registers and stack pointers are 32-bits wide, address registers cannot be used for byte-size operands. When an address register is a source operand, either the low-

For More Information On This Product,
Go to: www.freescale.com

order word or the entire longword operand is used, depending on the operation size. When an address register is the destination operand, the entire register becomes affected, despite the operation size. If the source operand is a word size, it is sign-extended to 32 bits and then used in the operation to an address register destination. Address registers are primarily for addresses and address computation support. The instruction set (See **Section 1.8 Instruction Set Summary**) explains how to add, compare, and move the contents of address registers. Figure 1-11 illustrates the integer format for address registers.

31	16	15	0
SIGN-EXTENDED		16-BIT ADDRESS OPERAND	

31	0
FULL 32-BIT ADDRESS OPERAND	
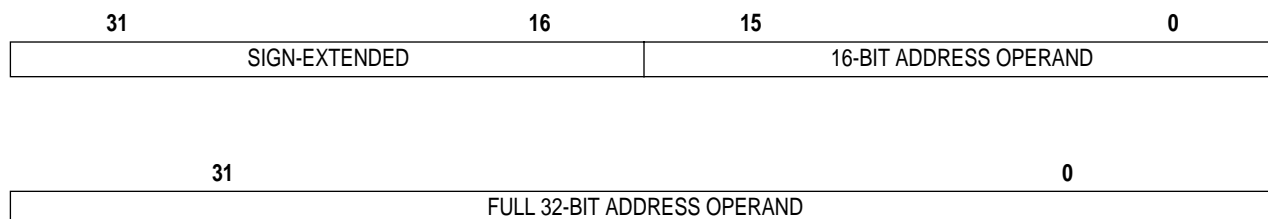
Figure 1-11. Organization of Integer Data Formats in Address Registers

Control registers vary in size according to function. Some control registers 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, despite privilege mode.

1.6.2 Organization of Integer Data Formats in Memory

The ColdFire2/2M uses 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 long-word 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 1-12.

| 31 | | 23 | | 15 | | 7 | | 0 |
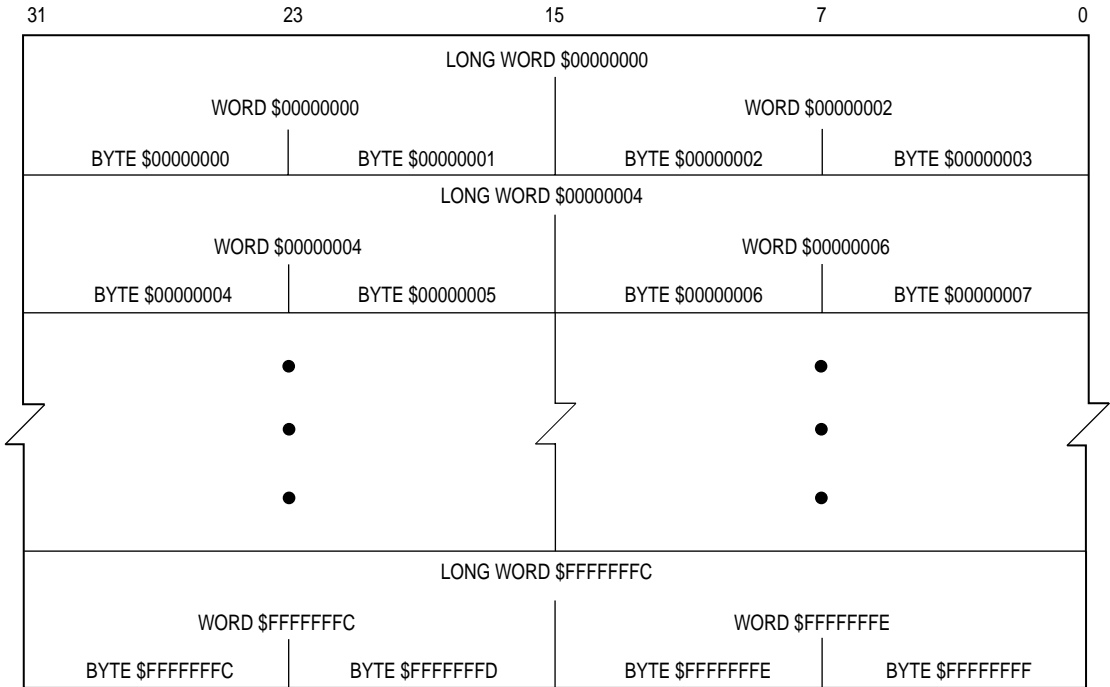
Figure 1-12. Memory Operand Addressing

1.7 ADDRESSING MODE SUMMARY

The addressing modes are grouped into categories according to the mode of use. Data addressing modes refer to data operands. Memory addressing modes refer to memory operands. Alterable addressing modes refer to alterable (writable) operands. Control addressing modes refer to memory operands without an associated size.

These categories sometimes combine to form new categories that are more restrictive. Two combined classifications are alterable memory (both alterable and memory) and data alterable (both alterable and data). Table 1-3 lists a summary of effective addressing modes and their categories. Only the twelve addressing modes supported by the 68000 are available on the ColdFire2/2M.

Table 1-3. Effective Addressing Modes and Categories

ADDRESSING MODES	SYNTAX	MODE FIELD	REG. FIELD	CATEGORY			
				DATA	MEMORY	CONTROL	ALTERABLE
Register Direct Data Address	Dn An	000 001	reg. no. reg. no.	X —	— —	— —	X X
Register Indirect Address Address with Postincrement Address with Predecrement Address with Displacement	(An) (An)+ –(An) (d_{16}, An)	010 011 100 101	reg. no. reg. no. reg. no. reg. no.	X X X X	X X X X	X — — X	X X X X
Address Register Indirect with Index 8-Bit Displacement	(d_8, An, Xn)	110	reg. no.	X	X	X	X
Program Counter Indirect with Displacement	(d_{16}, PC)	111	010	X	X	X	—
Program Counter Indirect with Index 8-Bit Displacement	(d_8, PC, Xn)	111	011	X	X	X	—
Absolute Data Addressing Short Long	(xxx).W (xxx).L	111 111	000 001	X X	X X	X X	— —
Immediate	#<xxx>	111	100	X	X	—	—

1.8 INSTRUCTION SET SUMMARY

Table 1-4 lists the notational conventions used throughout this manual unless otherwise specified. Table 1-5 lists the ColdFire2/2M instruction set by opcode. This instruction set is a reduced version of the 68000 instruction set. The removed instructions include BCD, bit field, logical rotate, decrement and branch, integer division, and integer multiply with a 64-bit result. Nine new MAC instructions have been added.

Table 1-4. Notational Conventions

OPCODE WILDCARDS	
cc	Logical Condition (example: NE for not equal)
REGISTER OPERANDS	
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
Rn	Any Address or Data Register
Ry,Rx	Any source and destination registers, respectively
Rw	Any second source register
Rc	Any Control Register (example VBR is the vector base register)
REGISTER/PORT NAMES	
ACC	MAC Accumulator
DDATA	Debug Data Port
CCR	Condition Code Register (lower byte of status register)
MACSR	MAC Status Register
MASK	Mask Register
PC	Program Counter
PST	Processor Status Port
SR	Status Register

Table 1-4. Notational Conventions (Continued)

MISCELLANEOUS OPERANDS	
#<data>	Immediate data following the instruction word(s).
<ea>	Effective Address
<label>	Assemble Program Label
<list>	List of registers (example: D3–D0)
<shift>	Shift operation: Shift left (<<), Shift Right (>>)
<size>	Operand data size: Byte (B), Word (W), Longword (L)
OPERATIONS	
+	Arithmetic addition or postincrement indicator
−	Arithmetic subtraction or predecrement indicator
¥	Arithmetic multiplication
~	Invert; operand is logically complemented
L	Logical AND
V	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.
&	and
|	or
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: MSB of D0)
LSW	Least Significant Word
MSB	Most Significant Bit
MSW	Most Significant Word
CONDITION CODE REGISTER BIT NAMES	
C	Carry Bit in CCR
N	Negative Bit in CCR
V	Overflow Bit in CCR
X	Extend Bit in CCR
Z	Zero Bit in CCR

Table 1-5. Instruction Set Summary

INSTRUCTION	OPERAND SYNTAX	OPERAND SIZE	OPERATION
ADD	Dy,<ea>,x <ea>y,Dx	32 32	Source + Destination → Destination
ADDA	<ea>y,Ax	32	Source + Destination → Destination
ADDI	#<data>,Dx	32	Immediate Data + Destination → Destination
ADDQ	#<data>,<ea>x	32	Immediate Data + Destination → Destination
ADDX	Dy,Dx	32	Source + Destination + X → Destination
AND	Dy,<ea>x <ea>y,Dx	32 32	Source L Destination → Destination
ANDI	#<data>,Dx	32	Immediate Data L Destination → Destination
ASL	Dx,Dy #<data>,Dn	32 32	X/C ← (Dy << Dx) ← 0 X/C ← (Dy << #<data>) ← 0
ASR	Dx,Dy <data>,Dx	32 32	MSB → (Dy >> Dx) → X/C MSB → (Dy >> #<data>) → X/C
Bcc	<label>	8,16	If Condition True, Then PC + d_n → PC
BCHG	Dy,<ea>x #<data>,<ea>x	8,32 8,32	~(<Bit Number> of Destination) → Z, Bit of Destination
BCLR	Dy,<ea>x #<data>,<ea>x	8,32 8,32	~(<Bit Number> of Destination) → Z, 0 → Bit of Destination
BRA	<label>	8,16	PC + d_n → PC
BSET	Dy,<ea>x #<data>,<ea>x	8,32 8,32	~(<Bit Number> of Destination) → Z, 1→ Bit of Destination
BSR	<label>	8,16	SP – 4 → SP; PC → (SP); PC + d_n → PC
BTST	Dy,<ea>x #<data>,<ea>x	8,32 8,32	~(<Bit Number> of Destination) → Z
CLR	<ea>x	8,16,32	0 → Destination
CMPI	#<data>,Dx	8,16,32	Destination – Immediate Data
CMP	<ea>y,Dx	32	Destination - Source
CMPA	<ea>y,Ax	32	Destination - Source
CPUSH	(An)	32	Push and Invalidate Cache Line
EOR	Dy,<ea>x	32	Source ⊕ Destination → Destination
EORI	#<data>,Dx	32	Immediate Data ⊕ Destination → Destination
EXT	Dx Dx	8 → 16 16 → 32	Sign-Extended Destination → Destination
EXTB	Dx	8 → 32	Sign-Extended Destination → Destination
HALT	none	none	Enter Halted State
JMP	<ea>y	none	<ea>y → PC
JSR	<ea>y	none	SP – 4 → SP; Next PC → (SP); <ea>y → PC
LEA	<ea>y,Ax	32	Address of <ea> → An
LINK	Ax,#<data>	32	SP – 4 → SP; Ax → (SP); SP → Ax; SP + d16 → SP
LSL	Dx,Dy #<data>,Dx	32 32	X/C ← (Dy << Dx) ← 0 X/C ← (Dx << #<data>) ← 0
LSR	Dx,Dy #<data>,Dx	32 32	0 → (Dy >> Dx) → X/C 0 → (Dx >> #<data>) → X/C
MAC†	Rw,Rx ,<shift>	16 × 16 + 32 → 32 32 × 32 + 32 → 32	ACC + (Rw × Rx){ << 1 | >> 1} → ACC
MACL†	Rw,Rx,<shift>,<ea>,Ry	16 × 16 + 32 → 32 32 × 32 + 32 → 32	ACC + (Rw × Rx){ << 1 | >> 1} → ACC; (<ea>{& MASK}) → Ry
MOVE	<ea>y,<ea>x	8,16,32	Source → Destination
MOVE from ACC†	ACC,Rx	32	ACC → Rn
MOVE from CCR	Dx	16	CCR → Destination

NOTE: †Available on the ColdFire2M only.

Freescale Semiconductor, Inc.

Table 1-5. Instruction Set Summary (Continued)

INSTRUCTION	OPERAND SYNTAX	OPERAND SIZE	OPERATION
MOVE from MACSR†	MACSR,Rx MACSR,CCR	32 8	MACSR → Rx MACSR → CCR
MOVE from MASK†	MASK,Rx	32	MASK → Rx
MOVE from SR	Dx	16	SR → Destination
MOVE to ACC†	Rx,ACC #<data>,ACC	32 32	Rx → ACC #<data> → ACC
MOVE to CCR	Dy,CCR, #<data>,CCR	16	Source → CCR #<data> → CCR
MOVE to MACSR†	Rn,MACSR #<data>,MACSR	32	Rn → MACSR #<data> → MACSR
MOVE to MASK†	Rn,MASK #<data>,MASK	32 32	Rn → MASK #<data> → MASK
MOVE to SR	Dy,SR #<data>,SR	16	Source → SR #<data> → SR
MOVEA	<ea>y,Ax	16,32 → 32	Source → Destination
MOVEC	Rn,Rc	32	Rn → Rc
MOVEM	list,<ea>x <ea>y,list	32 32	Listed Registers → Destination Source→ Listed Registers
MOVEQ	#<data>,Dx	8 → 32	Sign-extended Immediate Data → Destination
MSAC†	Rw,Rx,<shift>	32 - 16 × 16 → 32 32 - 32 × 32 → 32	ACC - (Rw × Rx){ << 1 | >> 1} SF → ACC
MSACL†	Rw,Rx,<shift>,<ea>,Ry	32 - 16 × 16 → 32 32 - 32 × 32 → 32	ACC - (Rw × Rx){ << 1 | >> 1} SF → ACC; (<ea>{& MASK}) → Ry
MULS	<ea>y,Dx <ea>y, DI	16 x 16 → 32 32 x 32 → 32	Source × Destination → Destination (Signed or unsigned)
MULU	<ea>y,Dx <ea>y, DI	16 x 16 → 32 32 x 32 → 32	Source x Destination → Destination (Signed or unsigned)
NEG	<ea>x	32	0 – Destination → Destination
NEGX	<ea>x	32	0 – Destination – X → Destination
NOP	none	none	PC + 2 → PC; Pipeline synchronized
NOT	<ea>x	32	~ Destination → Destination
OR	Dy,<ea>x <ea>y,Dx	32	Source V Destination → Destination
ORI	#<data>,Dx	32	Immediate Data V Destination → Destination
PEA	<ea>y	32	SP – 4 → SP; <ea>y → (SP)
PULSE	none	none	Generate unique PST value
RTE	none	none	(SP+2) → SR; (SP+4) → PC; SP+ 8 → PC
RTS	none	none	(SP) → PC; SP + 4 → SP
Scc	Dx	8	If condition true, then 1's → Destination; Else 0's → Destination
STOP	#<data>	16	Immediate data → SR; Enter Stopped State
SUB	Dy,<ea>x <ea>y,Dx	32 32	Destination - Source→ Destination
SUBA	<ea>,Ax	32	Destination - Source→ Destination
SUBI	#<data>,Dx	32	Destination – Immediate data → Destination
SUBQ	#<data>,<ea>x	32	Destination – Immediate data → Destination
SUBX	Dy,Dx	32	Destination – Source – X → Destination
SWAP	Dx	16	MSW of Dx ←→ LSW of Dx
TRAP	none	none	SP – 4 → SP; PC → (SP); SP – 2 → SP; SR → (SP); SP – 2 → SP; Format → (SP); Vector Address → PC

NOTE: †Available on the ColdFire2M only.

ColdFire2/2M User's Manual

MOTOROLA

Table 1-5. Instruction Set Summary (Continued)

INSTRUCTION	OPERAND SYNTAX	OPERAND SIZE	OPERATION
TRAPF	none #<data>	none 16 32	PC + 2→ PC; PC + 4 → PC; PC + 6→ PC
TST	<ea>y	8,16,32	Set Integer Condition Codes
UNLK	Ax	32	Ax → SP; (SP) → Ax; SP + 4 → SP
WDDATA	<ea>y	8,16,32	(<ea>y) → DDATA port
WDEBUG	<ea>y	64	<ea>y → DEBUG; <ea>y + 4 → DEBUG

NOTE: †Available on the ColdFire2M only.

SECTION 2
SIGNAL SUMMARY

2.1 INTRODUCTION

This section describes the ColdFire2/2M input and output signals. Figure 2-1 shows the ColdFire2/2M along with the signal interface. The signals are listed alphabetically in Table 2-1. All ColdFire2/2M signals are unidirectional and synchronous.
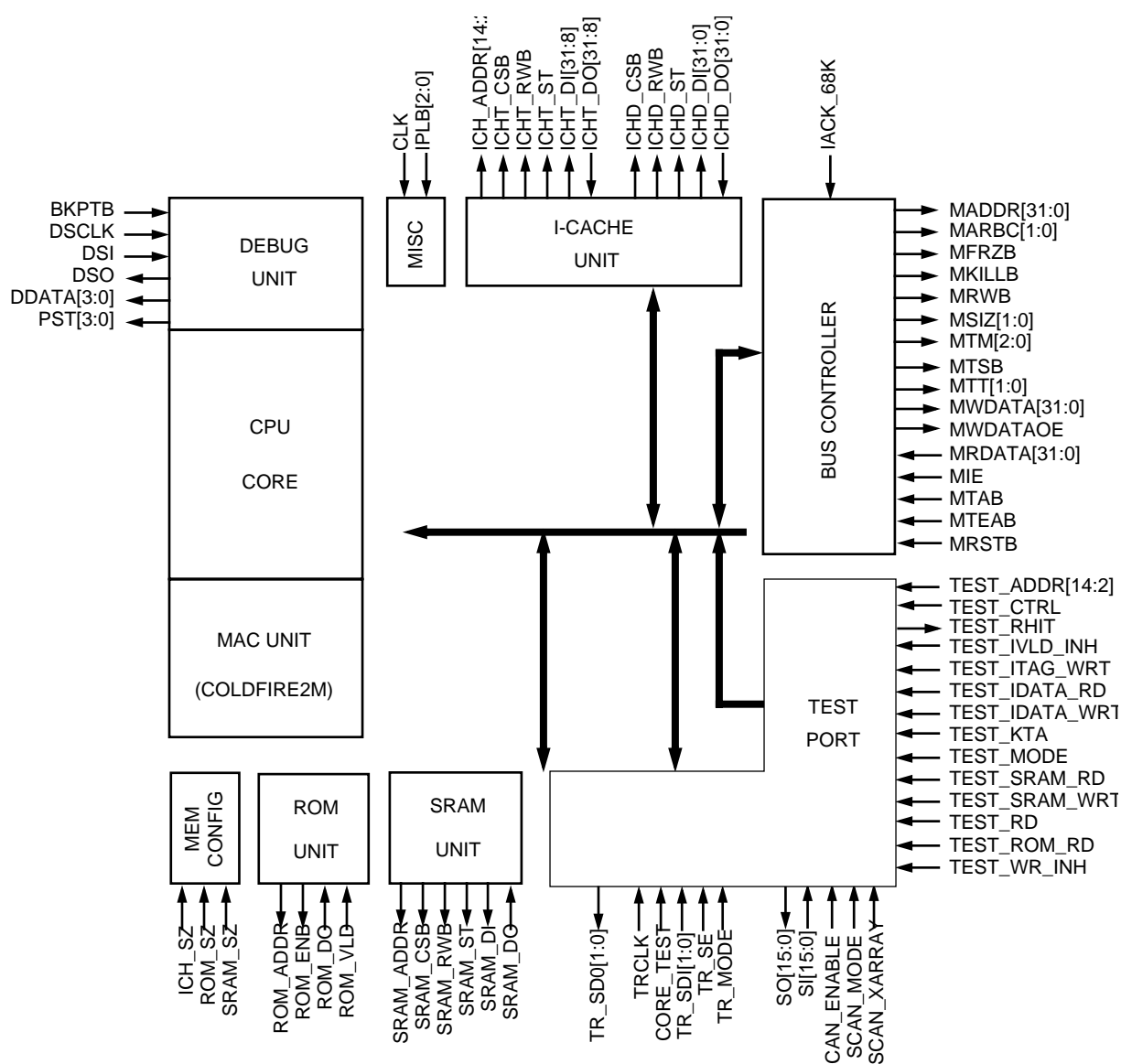
Figure 2-1. ColdFire2/2M Detailed Block Diagram

Freescale Semiconductor, Inc.

Table 2-1. Signal Summary

SIGNAL	MNEMONIC	INPUT/OUTPUT	ACTIVE STATE
68K Interrupt Acknowledge Mode	IACK_68K	Input	High
Break Point	BKPTB	Input	Low
Clock	CLK	Input	High
Debug Data	DDATA[3:0]	Output	High
Development Serial Clock	DSCLK	Input	High
Development Serial Data Input	DSI	Input	High
Development Serial Data Output	DSO	Output	High
Instruction Cache Address Bus	ICH_ADDR[14:2]	Output	High
Instruction Cache Data Chip-Select	ICHD_CSB	Output	Low
Instruction Cache Data Input Bus	ICHD_DI[31:0]	Output	High
Instruction Cache Data Output Bus	ICHD_DO[31:0]	Input	High
Instruction Cache Data Strobe	ICHD_ST	Output	High
Instruction Cache Data Write Enable	ICHD_RWB	Output	Low
Instruction Cache Size	ICH_SZ[2:0]	Input	High
Instruction Cache Tag Chip-Select	ICHT_CSB	Output	Low
Instruction Cache Tag Input Bus	ICHT_DI[31:8]	Output	High
Instruction Cache Tag Output Bus	ICHT_DO[31:8]	Input	High
Instruction Cache Tag Strobe	ICHT_ST	Output	High
Instruction Cache Tag Write Enable	ICHT_RWB	Output	Low
Interrupt Priority Level	IPLB[2:0]	Input	Low
Master Address Bus	MADDR[31:0]	Output	High
Master Arbiter Control	MARBC[1:0]	Output	High
Master Freeze	MFRZB	Output	Low
Master Kill	MKILLB	Output	Low
Master Read Data Bus	MRDATA[31:0]	Input	High
Master Read Data Input Enable	MIE	Input	High
Master Read/Write	MRWB	Output	Low
Master Reset	MRSTB	Input	Low
Master Size	MSIZ[1:0]	Output	High
Master Transfer Acknowledge	MTAB	Input	Low
Master Transfer Error Acknowledge	MTEAB	Input	Low
Master Transfer Modifier	MTM[2:0]	Output	High
Master Transfer Start	MTSB	Output	Low
Master Transfer Type	MTT[1:0]	Output	High
Master Write Data Bus	MWDATA[31:0]	Output	High
Master Write Data Output Enable	MWDATAOE	Output	High
Processor Status	PST[3:0]	Output	High
ROM Address Bus	ROM_ADDR[14:2]	Output	High
ROM Data Output Bus	ROM_DO[31:0]	Input	High
ROM Enable	ROM_ENB[1:0]	Output	Low
ROM Size	ROM_SZ[2:0]	Input	High
ROM Valid	ROM_VLD	Input	High
Scan Enable	SCAN_ENABLE	Input	High
Scan Exercise Array	SCAN_XARRAY	Input	High
Scan Input	SI[15:0]	Input	High
Scan Mode	SCAN_MODE	Input	High
Scan Output	SO[15:0]	Input	High
Scan Test Ring Clock	TR_CLK	Input	High

ColdFire2/2M User's Manual MOTOROLA

Table 2-1. Signal Summary (Continued)

SIGNAL	MNEMONIC	INPUT/OUTPUT	ACTIVE STATE
Scan Test Ring Core Mode Enable	CORE_TEST	Input	High
Scan Test Ring Data Input	TR_SDI[1:0]	Input	High
Scan Test Ring Data Output	TR_SDO[1:0]	Output	High
Scan Test Ring Enable	TR_SE	Input	High
Scan Test Ring Mode	TR_MODE	Input	High
SRAM Address Bus	SRAM_ADDR[14:2]	Output	High
SRAM Chip-Select	SRAM_CSB	Output	Low
SRAM Data Input Bus	SRAM_DI[31:0]	Output	High
SRAM Data Output Bus	SRAM_DO[31:0]	Input	High
SRAM Size	SRAM_SZ[2:0]	Input	High
SRAM Strobe	SRAM_ST[3:0]	Output	High
SRAM Read/Write	SRAM_RWB[3:0]	Output	Low
Test Address Bus	TEST_ADDR[14:2]	Input	High
Test Control	TEST_CTRL	Input	High
Test Invalidate Inhibit	TEST_IVLD_INH	Input	High
Test ITAG Write	TEST_ITAG_WRT	Input	High
Test Instruction Cache Read Hit	TEST_RHIT	Output	High
Test IDATA Read	TEST_IDATA_RD	Input	High
Test IDATA Write	TEST_IDATA_WRT	Input	High
Test KTA Mode Enable	TEST_KTA	Input	High
Test Mode Enable	TEST_MODE	Input	High
Test SRAM Read	TEST_SRAM_RD	Input	High
Test SRAM Write	TEST_SRAM_WRT	Input	High
Test Read	TEST_RD	Input	High
Test ROM Read	TEST_ROM_RD	Input	High
Test Write Inhibit	TEST_WR_INH	Input	High

2.2 MASTER BUS SIGNALS

2.2.1 68K Interrupt Acknowledge Mode Enable (IACK_68K)

This active-high input signal enables the 68K interrupt acknowledge mode. In this mode, the Master Address Bus, MADDR[31:0], MTT[1:0], and MTM[2:0] signals mimic the 68K address bus and function codes during interrupt acknowledge and CPU space bus cycles. This is a static input. Refer to **Section 3.7.1 Interrupt Acknowledge Bus Cycle** for more information.

2.2.2 Master Address Bus (MADDR[31:0])

During a normal bus cycle, this 32-bit output bus provides the address of the first item of a bus transfer. It is capable of addressing 4 Gbytes of address space.

2.2.3 Master Arbiter Control (MARBC[1:0])

These output signals can be used to specify the mode of operation for an optional arbitration module. They reflect the bit positions [17:16] in the CACR. If an optional arbitration module is not used, these signals can be used as general purpose output signals. The CACR is accessible in supervisor mode as control register $002 using the MOVEC instruction.

2.2.4 Master Freeze (MFRZB)

This active-low output signal indicates that the core has been halted. MFRZB is not part of the M-Bus protocol. It is simply a signal that can be used to alert timers or other peripheral modules that the core has been halted.

2.2.5 Master Kill (MKILLB)

This active-low output signal qualifies MTSB (i.e. it can assert in other cycles but is only significant in a cycle where MSTB is asserted). When MKILLB is asserted simultaneously with MTSB assertion, this indicates a hit in a K-Bus memory and that the external cycle must be inhibited. This means the current master bus transaction is no longer required and should be ignored (MTAB should not be asserted). MKILLB is asserted late in the MTSB cycle. Note that if there is no K-Bus resident memory (ICACHE, SRAM, or ROM), MKILLB never asserts. See Table 2-2 for MTSB/MKILLB interaction in table format.

Table 2-2. M-Bus Protocol with respect to MTSB and MKILLB

MTSB	MKILLB	M-BUS PROTOCOL
0	1	Initiate M-Bus transfer
0	0	Nop
1	X	Nop

2.2.6 Master Read Data Bus (MRDATA[31:0])

These input signals provide the read data path between the system and the ColdFire2/2M. The read data bus is 32-bits wide and can transfer 8, 16 or 32 bits of data per bus transfer. During a line transfer, the data bus is time-multiplexed across multiple clock cycles to transfer 128 bits.

2.2.7 Master Read Data Input Enable (MIE)

This active-high input signal enables the capturing of MRDATA[31:0]. Power consumption can be reduced by minimizing signal switching in the ColdFire2/2M by negating MIE when MRDATA[31:0] is invalid. MIE must be asserted during all functional operation with one master and during K-Bus memory testing.

2.2.8 Master Read/Write (MRWB)

This output signal indicates the direction of the data transfer for the current bus cycle. A high level indicates a read cycle and a low level indicates a write cycle.

2.2.9 Master Reset (MRSTB)

This active-low input signal instructs all master bus modules, including the ColdFire2/2M, to enter reset mode. The ColdFire2/2M will then initiate a reset exception.

2.2.10 Master Size (MSIZ[1:0])

These output signals indicate the data size for the bus transfer. Refer to Table 2-3 for the bus size encoding.

Table 2-3. Master Bus Transfer Size Encoding

MSIZ[1:0]	TRANSFER SIZE
00	Longword (4 bytes)
01	Byte (1 byte)
10	Word (2 bytes)
11	Line (16 bytes)

2.2.11 Master Transfer Acknowledge (MTAB)

This active-low input signal is asserted by master bus slaves to indicate the successful completion of a requested bus transfer.

2.2.12 Master Transfer Error Acknowledge (MTEAB)

This active-low input signal is asserted by master bus slaves to indicate an error condition for the current bus transfer. If MTAB and MTEAB are both asserted, the cycle is terminated with an error because MTEAB always has precedence over MTAB.

2.2.13 Master Transfer Modifier (MTM[2:0])

These output signals provide supplemental information for each transfer type. The exact meaning depends on the MTT[1:0], and IACK_68K signals as shown in Table 2-4.

Table 2-4. Master Bus Transfer Modifier Encoding

MTM[2:0]	MTT[1:0]	COLDFIRE IACK MODE[1]	68K IACK MODE[2]
000	0x	Reserved	Reserved
001	0x	User Data Access	User Data Access
010	0x	User Code Access	User Code Access
011 - 100	0x	Reserved	Reserved
101	0x	Supervisor Data Access	Supervisor Data Access
110	0x	Supervisor Code Access	Supervisor Code Access
111	0x	Reserved	Interrupt Acknowledge/CPU Space Access
000 - 100	10	Reserved	Reserved
101	10	Emulator Mode Data Access	Emulator Mode Data Access
110	10	Emulator Mode Code Access	Emulator Mode Code Access
111	10	Reserved	Reserved
000	11	CPU Space Access	Reserved
001	11	Interrupt Acknowledge Level 1	Reserved
010	11	Interrupt Acknowledge Level 2	Reserved
011	11	Interrupt Acknowledge Level 3	Reserved
100	11	Interrupt Acknowledge Level 4	Reserved
101	11	Interrupt Acknowledge Level 5	Reserved
110	11	Interrupt Acknowledge Level 6	Reserved
111	11	Interrupt Acknowledge Level 7	Reserved

NOTES: 1. 68K interrupt acknowledge mode signal negated (IACK_68K = 0)

2. 68K interrupt acknowledge mode signal asserted (IACK_68K = 1)

2.2.14 Master Transfer Start (MTSB)

This active-low output signal indicates the start of each bus transfer.

2.2.15 Master Transfer Type (MTT[1:0])

These output signals indicate the type of access of the current bus cycle. The exact meaning depends on the IACK_68K signal as shown in Table 2-5.

Table 2-5. Master Bus Transfer Type Encoding

MTT[1:0]	COLDFIRE IACK MODE[1]	68K IACK MODE[2]
00	ColdFire2/2M Access	Acknowledge/CPU Space/ColdFire2/2M Access
01	Alternate Master Access	Alternate Master Access
10	Emulator Mode Access	Emulator Mode Access
11	Acknowledge/CPU Space Access	Reserved

NOTES: 1. 68K interrupt acknowledge mode signal negated (IACK_68K = 0)

2. 68K interrupt acknowledge mode signal asserted (IACK_68K = 1)

2.2.16 Master Write Data Bus (MWDATA[31:0])

These output signals provide the write data path between the ColdFire2/2M and the system. The write data bus is 32-bits wide and can transfer 8, 16 or 32 bits of data per bus transfer. During a line transfer, the data bus is time-multiplexed across multiple clock cycles to carry 128 bits.

2.2.17 Master Write Data Output Enable (MWDATAOE)

This active-high output signal indicates that the ColdFire2/2M is driving the master write data bus. This is used to control optional bidirectional data bus three-state drivers.

2.3 GENERAL CONTROL SIGNALS

2.3.1 Clock (CLK)

This input signal is the synchronous clock for the ColdFire2/2M. CLK is used to clock or sequence the ColdFire2/2M internal logic.

2.3.2 Interrupt Priority Level (IPLB[2:0])

These active-low input signals indicate the encoded priority level of the requested interrupt. Level seven, which cannot be masked, has the highest priority. Level zero indicates no interrupt has been requested. These signals must maintain the interrupt request until the ColdFire2/2M acknowledges the interrupt to guarantee that the interrupt is recognized. Table 2-6 lists the interrupt levels, the IPLB[2:0] states, and the mask value that allows an interrupt at each level.

Table 2-6. Interrupt Levels and Mask Values

REQUESTED INTERRUPT LEVEL	CONTROL LINE STATUS			INTERRUPT MASK LEVEL REQUIRED FOR RECOGNITION
	IPLB[2]	IPLB[1]	IPLB[0]	
0	High	High	High	No Request
1	High	High	Low	0
2	High	Low	High	0-1
3	High	Low	Low	0-2
4	Low	High	High	0-3
5	Low	High	Low	0-4
6	Low	Low	High	0-5
7	Low	Low	Low	0-7

2.4 INTEGRATED MEMORY SIGNALS

These signals interface the ColdFire2/2M to an integrated instruction cache, ROMs and SRAMs.

2.4.1 Instruction Cache Signals

The signals interface the ColdFire2/2M to an optional integrated instruction cache.

2.4.1.1 INSTRUCTION CACHE ADDRESS BUS (ICH_ADDR[14:2]). These registered output signals provide the address of the current bus cycle (i.e. fetch cycle) to the integrated cache RAMs. ICH_ADDR is only updated on fetch cycles (i.e. ICH_ADDR does not get updated on SRAM or ROM hits). This bus should be connected to the address bus (A) of the two compiled cache RAMs.

2.4.1.2 INSTRUCTION CACHE DATA CHIP-SELECT (ICHD_CSB). This active-low, output signal indicates the cache data RAM is currently selected to perform a data transfer with the ColdFire2/2M. This bus should be connected to the chip-select (CSB) signal of the compiled cache data RAM.

2.4.1.3 INSTRUCTION CACHE DATA INPUT BUS (ICHD_DI[31:0]). These output signals provide the write data path between the ColdFire2/2M and the cache data RAM. The data bus is 32-bits wide and should be connected to the data inputs (DBI) of the compiled cache data RAM.

2.4.1.4 INSTRUCTION CACHE DATA OUTPUT BUS (ICHD_DO[31:0]). These input signals provide the read data path between the cache data RAM and the ColdFire2/2M. The data bus is 32-bits wide and should be connected to the data outputs (DBO) of the compiled cache data RAM.

2.4.1.5 INSTRUCTION CACHE DATA STROBE (ICHD_ST). This output signal initiates a read or write cycle to the cache data RAM on a low-to-high transition. This signal should be connected to the strobe input (ST) signal of the compiled cache data RAM.

2.4.1.6 INSTRUCTION CACHE DATA READ/WRITE (ICHD_RWB). This output signal indicates the direction of the data transfer to the cache data RAM. A high level indicates a

Freescale Semiconductor, Inc.

read cycle and a low level indicates a write cycle. It should be connected to the read/write (RWB) signal of the compiled cache data RAM.

2.4.1.7 INSTRUCTION CACHE SIZE (ICH_SZ[2:0]). These static inputs specify the size of the compiled cache RAMs connected to the ColdFire2/2M processor. Table 2-7 lists the possible cache configurations. ICH_SZ[2:0] does not affect the CACR in any way; thus a MOVEC instruction will write the CACR regardless of the ICH_SZ specification (which is contrary to the SRAM_SZ and ROM_SZ effect during RAMBAR and ROMBAR loading).

Table 2-7. Cache Configuration Encoding

CACHE SIZE (BYTES)	ICH_SZ[2:0]	TAG RAM (BITS)		DATA RAM (BITS)	
		ADDRESS	DATA	ADDRESS	DATA
None	000	-	-	-	-
512	001	5	24	7	32
1 K	010	6	23	8	32
2 K	011	7	22	9	32
4 K	100	8	21	10	32
8 K	101	9	20	11	32
16K†	110	10	19	12	32
32K†	111	11	18	13	32

NOTE: †16K and 32K RAMS may require a reduced operating frequency in HPF65 ColdFire2 Hard Macro.

2.4.1.8 INSTRUCTION CACHE TAG CHIP-SELECT (ICHT_CSB). This active-low output signal indicates the cache tag RAM is currently selected to perform a data transfer with the ColdFire2/2M. This signal should be connected to the chip-select (CSB) signal of the compiled cache tag RAM.

2.4.1.9 INSTRUCTION CACHE TAG INPUT BUS (ICHT_DI[31:8]). These output signals provide the write data path between the ColdFire2/2M and the cache tag RAM. The data bus is 24-bits wide. Bit eight is always the valid bit and is always used as seen in the cache configuration shown in Table 2-8. This bus should be connected to the data inputs (DBI) of the compiled cache tag RAM. Functionally, ICH_ADDR[31:9] is what gets written onto ICHT_DI[31:9] and ICHT_DI[8] is written with the valid state of the entry.

Table 2-8. Valid Tag RAM Data Signals

CACHE SIZE (BYTES)	VALID DATA BITS
512	ICHT_Dx[31:8]
1 K	ICHT_Dx[31:10,8]
2 K	ICHT_Dx[31:11,8]
4 K	ICHT_Dx[31:12,8]
8 K	ICHT_Dx[31:13,8]
16K	ICHT_Dx[31:14,8]
32K	ICHT_Dx[31:15,8]

2.4.1.10 INSTRUCTION CACHE TAG OUTPUT BUS (ICHT_DO[31:8]). These input signals provide the read data path between the cache tag RAM and the ColdFire2/2M. The data bus is 24-bits wide. Bit eight is always the valid bit and is always used regardless of the cache configuration as shown in Table 2-8. This bus should be connected to the data outputs (DBO) of the compiled cache tag RAM. Unused signals must be tied low.

2.4.1.11 INSTRUCTION CACHE TAG STROBE (ICHT_ST). This output signal initiates a read or write cycle to the cache tag RAM on a low-to-high transition. This signal should be connected to the strobe input (ST) signal of the compiled cache tag RAM.

2.4.1.12 INSTRUCTION CACHE TAG READ/WRITE (ICHT_RWB). This output signal indicates the direction of the data transfer to the cache tag RAM. A high level indicates a read cycle and a low level indicates a write cycle. It should be connected to the read/write (RWB) signal of the compiled cache tag RAM.

2.4.2 Integrated ROM Signals

These signals interface the ColdFire2/2M to the optional integrated ROMs.

2.4.2.1 ROM ADDRESS BUS (ROM_ADDR[14:2]). These output signals provide the address of the current bus cycle to the integrated ROMs. This bus should be connected to the address bus (A) of the compiled ROMs. The number of valid address signals depends on the total ROM size as shown in Table 2-9.

Table 2-9. Valid ROM Address Bits

TOTAL ROM SIZE	VALID ROM_ADDR BITS
0	None
512	ROM_ADDR[8:2]
1 K	ROM_ADDR[9:2]
2 K	ROM_ADDR[10:2]
4 K	ROM_ADDR[11:2]
8 K	ROM_ADDR[12:2]
16K	ROM_ADDR[13:2]
32K	ROM_ADDR[14:2]

2.4.2.2 ROM DATA OUTPUT BUS (ROM_DO[31:0]). These input signals provide the read data path from the integrated ROMs to the ColdFire2/2M. The data bus is 32-bits wide and can transfer 8, 16, or 32 bits of data per bus transfer. During a line transfer, the data lines are time-multiplexed across multiple cycles to carry 128 bits. This bus should be connected to the data outputs (DO) of the compiled ROMs.

2.4.2.3 ROM ENABLE (ROM_ENB[1:0]). These active-low, output signals indicate the ROMs are currently selected to drive the ROM_DO[31:0] bus. These signals should be connected individually to the enable signal (ROMENB) signal of the compiled ROMs. Both are asserted for 32-bit accesses. ROM_ENB[0] connects to the MSW while ROM_ENB[1] connects to the LSW.

2.4.2.4 ROM SIZE (ROM_SZ[2:0]). These static inputs specify the size of the compiled ROMs connected to the ColdFire2/2M. These pins need to stay valid during all operation. Table 2-10 lists the possible ROM configurations. If the ROM_SZ pins are zero, the ROM cannot be enabled via a CPU space write to ROMBAR. Therefore if the ROM is enabled while the ROM_SZ pins are at zero, the processor behaves as if no ROM module existed.

Table 2-10. ROM Configuration Encoding

TOTAL ROM SIZE (BYTES)	ROM_SZ[2:0]	ADDRESS (BITS)	DATA[1] (BITS)
None	000	-	-
512	001	7	2 @ 16
1 K	010	8	2 @ 16
2 K	011	9	2 @ 16
4 K	100	10	2 @ 16
8 K	101	11	2 @ 16
16K[2]	110	12	2 @ 16
32K[2]	111	13	2 @ 16

NOTES: 1. 2 ROMs, each 16-bits wide

2. 16K and 32K ROMs may require a reduced operating frequency.

2.4.2.5 ROM VALID (ROM_VLD). This active-high input signal determines if the ROM module should be active immediately after a hard reset. Thus, if asserted, the first fetches ($0, $4) go to ROM instead of external memory. ROM_VLD controls the reset value of the ROM base address register (i.e. the ROM must be based at $0000 if ROM_VLD is asserted).

2.4.3 Integrated SRAM Signals

These signals interface the ColdFire2/2M to the optional integrated SRAMs.

2.4.3.1 SRAM ADDRESS BUS (SRAM_ADDR[14:2]).

These registered output signals provide the address of the current bus cycle to the integrated SRAMs. This bus should be connected to the address bus (A) of the four compiled SRAMs. The number of valid address signals depends on the total SRAM size as shown in Table 2-11.

Table 2-11. Valid SRAM Address Bits

TOTAL SRAM SIZE	VALID SRAM_ADDR BITS
0	None
512	SRAM_ADDR[8:2]
1 K	SRAM_ADDR[9:2]
2 K	SRAM_ADDR[10:2]
4 K	SRAM_ADDR[11:2]
8 K	SRAM_ADDR[12:2]
16K	SRAM_ADDR[13:2]
32K	SRAM_ADDR[14:2]

2.4.3.2 SRAM CHIP-SELECT (SRAM_CSB). This active-low output signal indicates the SRAMs are currently selected to perform a data transfer with the ColdFire2/2M. This signal should be connected to the chip-select (CSB) signal of the four compiled SRAMs.

2.4.3.3 SRAM DATA INPUT BUS (SRAM_DI[31:0]). These output signals provide the write data path between the ColdFire2/2M and the integrated SRAM. The data bus is 32-bits wide and can transfer 8, 16, or 32 bits of data per bus transfer. During a line transfer, the data lines are time-multiplexed across multiple cycles to carry 128 bits. This bus should be connected to the data inputs (DBI) of the four compiled SRAMs. If only one byte is being written, the byte will be replicated on all 4 lines likewise a word will be replicated in both word positions.

2.4.3.4 SRAM DATA OUTPUT BUS (SRAM_DO[31:0]). These input signals provide the read data path between the integrated RAM and the ColdFire2/2M. The data bus is 32-bits wide and can transfer 8, 16 or 32 bits of data per bus transfer. During a line transfer, the data lines are time-multiplexed across multiple cycles to carry 128 bits. This bus should be connected to the data outputs (DBO) of the four compiled SRAMs.

2.4.3.5 SRAM SIZE (SRAM_SZ[2:0]). These static inputs specify the size of the compiled SRAMs connected to the ColdFire2/2M. These pins need to stay valid during all operation. If the SRAM_SZ pins are zero, the SRAM cannot be enabled via a CPU space write to RAMBAR. Therefore if the RAM is enabled while the SRAM_SZ pins are at zero, the processor behaves as if no SRAM module existed.

Table 2-12 lists the possible SRAM configurations.

Table 2-12. SRAM Configuration Encoding

TOTAL SRAM SIZE (BYTES)	RAM_SZ[2:0]	ADDRESS (BITS)	DATA[1] (BITS)
None	000	-	-
512	001	7	4@8
1 K	010	8	4@8
2 K	011	9	4@8
4 K	100	10	4@8
8 K	101	11	4@8
16K[2]	110	12	4@8
32K[2]	111	13	4@8

NOTES: 1.   4 RAMs, each 8-bits wide

2.   16K and 32K RAMs may require a reduced operating frequency.

2.4.3.6 SRAM STROBE (SRAM_ST[3:0]). These output signals initiate a read or write cycle to the integrated SRAMs on a low-to-high transition. These signals should be connected individually to the strobe input (ST) signals of the four compiled SRAMs. The ST[0] signal connects to the high-order byte and ST[3] connects to the low-order byte.

Freescale Semiconductor, Inc.

2.4.3.7 SRAM READ/WRITE (SRAM_RWB[3:0]). These output signals indicate the direction of the data transfer to the to the integrated SRAMs. A high level indicates a read cycle and a low level indicates a write cycle. They should be connected individually to the read/write (RWB) signal of the four compiled SRAMs. Like SRAM_ST[3:0], the SRAM_RWB[3] signal connects to the high-order byte and SRAM_ST[0] connects to the low-order byte.

2.5 DEBUG SIGNALS

2.5.1 Break Point (BKPTB)

This active-low, unidirectional input signal is used to request a manual break point. It will cause the processor to enter a halted state after the completion of the current instruction. This status will be reflected on the processor status (PST) outputs.

2.5.2 Debug Data (DDATA[3:0])

These output signals display the captured processor status and break point status.

2.5.3 Development Serial Clock (DSCLK)

This input signal is used as the development serial clock for the serial interface to the Debug Module.The maximum frequency is 1/2 the clock (CLK) frequency.

2.5.4 Development Serial Input (DSI)

This input signal provides the single-bit communication for the Debug Module commands.

2.5.5 Development Serial Output (DSO)

This output signal provides single-bit communication for the Debug Module responses.

2.5.6 Processor Status (PST[3:0])

These output signals report the processor status. Table 2-13 shows the encoding of these signals. These signals indicate the current status of the processor pipeline and, as a result, are not related to the current bus transfer.

Table 2-13. Processor Status Encoding

PST[3:0]		DEFINITION
(HEX)	(BINARY)	
$0	0000	Continue execution
$1	0001	Begin execution of an instruction
$2	0010	Reserved
$3	0011	Entry into user-mode
$4	0100	Begin execution of **PULSE** and **WDDATA** instructions
$5	0101	Begin execution of taken branch
$6	0110	Reserved
$7	0111	Begin execution of **RTE** instruction
$8	1000	Begin 1-byte transfer on DDATA
$9	1001	Begin 2-byte transfer on DDATA
$A	1010	Begin 3-byte transfer on DDATA
$B	1011	Begin 4-byte transfer on DDATA
$C	1100	Exception processing†
$D	1101	Emulator-mode entry exception processing†
$E	1110	Processor is stopped, waiting for interrupt†
$F	1111	Processor is halted †

NOTE: †These encodings are asserted for multiple cycles.

2.6 TEST SIGNALS

2.6.1 SIGNALS REQUIRED TO PERFORM SCAN TEST

This section describes the ColdFire2/2M signals dedicated to the scan testing of the ColdFire2/2M. All ColdFire2/2M signals are unidirectional and synchronous.

2.6.1.1 SCAN ENABLE (SCAN_ENABLE). This active-high input signal enables scan testing of the ColdFire2/2M. It forces all internal flip-flops to be linked together into sixteen parallel scan chains. This signal must be negated for functional operation.

2.6.1.2 SCAN EXERCISE ARRAY (SCAN_XARRAY). This active-high input signal is used to exercise the integrated memory arrays during scan testing. This signal causes random writes to the internal RAMs by strobing the write strobes while scanning.

2.6.1.3 SCAN INPUT (SI[15:0]). These input signals are connected to the16 internal ColdFire2/2M scan chain inputs.

2.6.1.4 SCAN MODE (SCAN_MODE). This active-high, unidirection input signal gates off all memory array outputs during scan testing.SCAN_MODE should be held asserted for the duration of scan testing.

2.6.1.5 SCAN OUTPUT (SO[15:0]). These output signals are connected to the 16 internal ColdFire2/2M scan chain outputs.

2.6.1.6 IO TEST RING CLOCK (TRCLK). This input signal is the synchronous clock used to transition the test ring during scan testing. TR_CLK is connected to the clock input of all IO test ring registers.

2.6.1.7 IO TEST RING CORE MODE ENABLE (CORE_TEST). This active-high input signal enables the core mode of the test ring during scan testing. The test ring is in scan core mode if CORE_TEST is asserted and in scan ASIC mode if CORE_TEST is negated.

2.6.1.8 IO TEST RING DATA INPUT (TR_SDI[1:0]). These input signals are the serial data inputs for test ring chain one and zero.

2.6.1.9 IO TEST RING DATA OUTPUT (TR_SDO[1:0]). These output signals are the serial output data from test ring chain one and zero.

2.6.1.10 IO TEST RING ENABLE (TR_SE). This active-high input signal enables the test ring. TR_SE is connected to the scan enable input of all IO test ring scannable registers.

2.6.1.11 IO TEST RING MODE (TR_MODE). This active-high input signal enables the scan test mode of the test ring. The test ring is in scan test mode if TR_MODE is asserted and in normal functional mode if negated. TR_MODE should be asserted for the duration of scan testing, and be held negated for the duration of memory testing and during functional operation of the device.

2.6.2 Integrated Memory Test Signals

This section describes the ColdFire2/2M signals dedicated to testing the integrated memories. Other signals are required to be either controlled or brought out (muxed) to package pins as well (See **Section 8 Test Operation**).

2.6.2.1 TEST ADDRESS BUS (TEST_ADDR[14:2]). These input signals specify an address when testing the integrated memories.

2.6.2.2 TEST CONTROL (TEST_CTRL). This active-high input signal indicates the test address bus (TEST_ADDR[14:2]) will be latched on the next positive clock edge.

2.6.2.3 TEST IDATA READ (TEST_IDATA_RD). This active-high input signal tests the instruction cache data memory read operation.

2.6.2.4 TEST IDATA WRITE (TEST_IDATA_WRT). This active-high input signal tests the instruction cache data memory write operation.

2.6.2.5 TEST INSTRUCTION CACHE READ HIT (TEST_RHIT). This active-high output signal indicates a hit has occurred when accessing the instruction cache during memory array testing.

2.6.2.6 TEST INVALIDATE INHIBIT (TEST_IVLD_INH). This active-high input signal inhibits the invalidate operation when testing the instruction cache.

2.6.2.7 TEST ITAG WRITE (TEST_ITAG_WRT). This active-high input signal tests the instruction cache tag memory write operation.

2.6.2.8 TEST KTA MODE ENABLE (TEST_KTA). This active-high input signal allows the instruction cache tag and data arrays to be read in parallel, mimicking the functional operation. This allows testing of the speed path from the tag and data arrays to the core.

2.6.2.9 TEST MODE ENABLE (TEST_MODE). This active-high input signal enables all of the integrated memory test signals. TEST_MODE should be asserted for the duration of memory testing.

2.6.2.10 TEST SRAM READ (TEST_SRAM_RD). This active-high input signal tests the integrated SRAM memory read operation.

2.6.2.11 TEST SRAM WRITE (TEST_SRAM_WRT). This active-high input signal tests the integrated SRAM memory write operation.

2.6.2.12 TEST READ (TEST_RD). This active-high input signal tests read operations on all of the integrated memories.

2.6.2.13 TEST ROM READ (TEST_ROM_RD). This active-high input signal tests the integrated ROM memory read operation.

2.6.2.14 TEST WRITE INHIBIT (TEST_WR_INH). This active-high input signal disables the write strobes to the SRAM and instruction cache compiled RAMS. TEST_WR_INH should be negated for the duration of memory test.

SECTION 3
MASTER BUS OPERATION

The master bus provides a basic two cycle bus protocol, similar to that used by previous generations of M68000 microprocessors. Basic cycles are defined as the transfer start (TS) cycle and the transfer acknowledge (TA) cycle. The address and control information is driven onto the bus during the TS cycle, and the data is valid during the subsequent TA cycle. By delaying the assertion of the transfer acknowledge signal, the bus automatically inserts wait states to easily accommodate any slave response speed. The following sections detail the signal descriptions, data transfer mechanism, and bus transfer protocols.

3.1 SIGNAL DESCRIPTION

This section describes the ColdFire2/2M signals associated with the master bus. All ColdFire2/2M signals are unidirectional and synchronous.

3.1.1 68K Interrupt Acknowledge Mode Enable (IACK_68K)

This active-high input signal enables the 68K interrupt acknowledge mode. In this mode, the MADDR[31:0], MTT[1:0], and MTM[2:0] signals mimic the 68K address bus and function codes during interrupt acknowledge and CPU space bus cycles. This is a static input. Refer to **Section 3.7.1 Interrupt Acknowledge Bus Cycle** for more information.

3.1.2 Master Address Bus (MADDR[31:0])

During a normal bus cycle, this 32-bit output bus provides the address of the first item of a bus transfer. It is capable of addressing four Gbytes of address space.

3.1.3 Master Arbiter Control (MARBC[1:0])

These output signals can be used to specify the mode of operation for an optional arbitration module. They reflect the bit positions [17:16] in the CACR. If an optional arbitration module is not used, these signals can be used as general purpose output signals.

3.1.4 Master Freeze (MFRZB)

This active-low output signal indicates that the core has been halted. MFRZB is not part of the M-Bus protocol. It is simply a signal that can be used to alert timers or other peripheral modules that the core has been halted.

3.1.5 Master Kill (MKILLB)

This active-low output signal qualifies MTSB (i.e. it can assert in other cycles but is only significant in a cycle where MSTB is asserted). When MKILLB is asserted simultaneously with MTSB assertion, this indicates a hit in a K-Bus memory and that the external cycle must

be inhibited. This means the current master bus transaction is no longer required and should be ignored (MTAB should not be asserted). MKILLB is asserted late in the MTSB cycle. Note that if there is no K-Bus resident memory (ICACHE, SRAM, or ROM), MKILLB never asserts. See Table 3-1 for MTSB/MKILLB interaction in table format.

Table 3-1. M-Bus Protocol with respect to MTSB and MKILLB

MTSB	MKILLB	M-BUS PROTOCOL
0	1	Initiate M-Bus transfer
0	0	Nop
1	X	Nop

3.1.6 Master Read Data Bus (MRDATA[31:0])

These input signals provide the read data path between the system and the ColdFire2/2M device. The read data bus is 32-bits wide and can transfer 8, 16 or 32 bits of data per bus transfer. During a line transfer, the data bus is time-multiplexed across multiple clock cycles to transfer 128 bits.

3.1.7 Master Read Data Input Enable (MIE)

This active-high input signal enables the capturing of MRDATA[31:0]. Power consumption can be reduced by minimizing signal switching in the ColdFire2/2M by negating MIE when MRDATA[31:0] is invalid. MIE must be asserted during all one-master functional operation and during K-Bus memory testing.

3.1.8 Master Read/Write (MRWB)

This output signal indicates the direction of the data transfer for the current bus cycle. A high level indicates a read cycle and a low level indicates a write cycle.

3.1.9 Master Reset (MRSTB)

This active-low input signal instructs all master bus modules, including the ColdFire2/2M device, to enter reset mode. The ColdFire2/2M processor will then initiate a reset exception.

3.1.10 Master Size (MSIZ[1:0])

These output signals indicate the data size for the bus transfer. Refer to Table 3-2 for the bus size encoding.

Table 3-2. Master Bus Transfer Size Encoding

MSIZ[1:0]	TRANSFER SIZE
00	Longword (4 bytes)
01	Byte (1 byte)
10	Word (2 bytes)
11	Line (16 bytes)

3.1.11 Master Transfer Acknowledge (MTAB)

This active low input signal is asserted by master bus slaves to indicate the successful completion of a requested bus transfer.

3.1.12 Master Transfer Error Acknowledge (MTEAB)

This active low input signal is asserted by master bus slaves to indicate an error condition for the current bus transfer. If MTAB and MTEAB are both asserted, the cycle is terminated with an error because MTEAB always has precedence over MTAB.

3.1.13 Master Transfer Modifier (MTM[2:0])

These output signals provide supplemental information for each transfer type. The exact meaning depends on the MTT[1:0], and IACK_68K signals as shown in Table 3-3.

Table 3-3. Master Bus Transfer Modifier Encoding

MTM[2:0]	MTT[1:0]	COLDFIRE IACK MODE[1]	68K IACK MODE[2]
000	0x	Reserved	Reserved
001	0x	User Data Access	User Data Access
010	0x	User Code Access	User Code Access
011 - 100	0x	Reserved	Reserved
101	0x	Supervisor Data Access	Supervisor Data Access
110	0x	Supervisor Code Access	Supervisor Code Access
111	0x	Reserved	Interrupt Acknowledge/CPU Space Access
000 - 100	10	Reserved	Reserved
101	10	Emulator Mode Data Access	Emulator Mode Data Access
110	10	Emulator Mode Code Access	Emulator Mode Code Access
111	10	Reserved	Reserved
000	11	CPU Space Access	Reserved
001	11	Interrupt Acknowledge Level 1	Reserved
010	11	Interrupt Acknowledge Level 2	Reserved
011	11	Interrupt Acknowledge Level 3	Reserved
100	11	Interrupt Acknowledge Level 4	Reserved
101	11	Interrupt Acknowledge Level 5	Reserved
110	11	Interrupt Acknowledge Level 6	Reserved
111	11	Interrupt Acknowledge Level 7	Reserved

NOTES: 1. 68K interrupt acknowledge mode signal negated (IACK_68K = 0)

2. 68K interrupt acknowledge mode signal asserted (IACK_68K = 1)

3.1.14 Master Transfer Start (MTSB)

This active low output signal indicates the start of each bus transfer.

3.1.15 Master Transfer Type (MTT[1:0])

These output signals indicate the type of access of the current bus cycle. The exact meaning depends on the IACK_68K signal as shown in Table 3-4.

Table 3-4. Master Bus Transfer Type Encoding

MTT[1:0]	COLDFIRE IACK MODE[1]	68K IACK MODE[2]
00	ColdFire2/2M Access	Acknowledge/CPU Space/ColdFire2/2M Access
01	Alternate Master Access	Alternate Master Access
10	Emulator Mode Access	Emulator Mode Access
11	Acknowledge/CPU Space Access	Reserved

NOTES: 1.   68K interrupt acknowledge mode signal negated (IACK_68K = 0)

2.   68K interrupt acknowledge mode signal asserted (IACK_68K = 1)

3.1.16 Master Write Data Bus (MWDATA[31:0])

These output signals provide the write data path between the ColdFire2/2M and the system. The write data bus is 32-bits wide and can transfer 8, 16 or 32 bits of data per bus transfer. During a line transfer, the data bus is time-multiplexed across multiple clock cycles to carry 128 bits.

3.1.17 Master Write Data Output Enable (MWDATAOE)

This active high output signal indicates that the ColdFire2/2M is driving the master write data bus. This is used to control optional bidirectional data bus three-state drivers.

3.2 DATA TRANSFER MECHANISM

3.2.1 Transfer Type Control Signals

The transfer type control signals indicate the type of bus transaction occurring on the master bus. This includes the master transfer type (MTT[1:0]) and master transfer modifier (MTM[2:0]) signals. The MTT[1:0] signals indicates the type of transfer and the MTM[2:0] signals provide supplemental information. The encodings for MTT[1:0] and MTM[2:0] are shown in Table 3-3 and Table 3-4. The transfer type attributes for accesses made through the debug module are determined by the programming of the debug module (see **Section 7.4.2.2 Address Attribute Register (AATR)**).

3.2.1.1 COLDFIRE2/2M ACCESS. If the ColdFire2/2M requests a master bus transfer, it drives the MTT[1:0] signals with a ColdFire2/2M access encoding. The MTM[2:0] encoding will depend on the privilege mode and address space of the transfer:

Privilege Mode—Supervisor/User Mode Access

When the supervisor (S) bit in the status register (SR) is set, the ColdFire2/2M will drive MTM[2:0] with one of the supervisor mode encodings during a master bus transfer. When the S bit in the SR is clear, the ColdFire2/2M will drive MTM[2:0] with one of the user mode access encodings.

Address Space—Code/Data Access

If the ColdFire2/2M accesses the code space, it will drive MTM[2:0] with one of the code access encodings during a master bus transfer. Code space accesses are opcode fetches or operand fetches in one of the PC relative addressing modes. If the ColdFire2/2M accesses the data space, it will drive MTM[2:0] with one of the data access encodings. Data

space accesses are operand fetches that are not in one of the PC relative addressing modes.

3.2.1.2 ALTERNATE MASTER ACCESS. When an alternate master requests a master bus transfer, the MTT[1:0] signals should be driven with the alternate master access encoding by the alternate master. The MTM[2:0] encoding is the same as that for the ColdFire2/2M access.

3.2.1.3 EMULATOR MODE ACCESS. Accesses made while in emulator mode generate emulator mode accesses on the master bus. This is controlled by the configuration/status register (CSR) in the Debug module. Refer to **Section 7.4.1.1 Emulator Mode**.

Emulator mode accesses result in he MTT[1:0] signals being driven with the emulator mode access encoding. The encoding of the MTM[2:0] signals will be one of the emulator mode encodings depending on the address space as defined for the ColdFire2/2M access.

3.2.1.4 INTERRUPT ACKNOWLEDGE ACCESS. Interrupt acknowledge bus cycles are indicated as an acknowledge/CPU space access on the MTT[1:0] signals (the encoding depends on the interrupt acknowledge mode). If the ColdFire2/2M is in the ColdFire interrupt acknowledge mode, the MTM[2:0] signals are driven with the pending interrupt level number. In the 68K interrupt acknowledge mode, the MTM[2:0] signals will be driven high. Refer to **Section 3.7 Interrupt Acknowledge Bus Cycles**.

3.2.1.5 CPU SPACE ACCESS. CPU space accesses are indicated as a acknowledge/ CPU space access on the MTT[1:0] signals. The MTM[2:0] signals are driven with a CPU space encoding. The specific encoding depends on the interrupt acknowledge mode. Many debug commands and MOVEC instructions result in CPU space accesses.

3.2.2 Data Bus Requirements

The ColdFire2/2M designates all operands for transfers on a byte-boundary system. A line-sized operand (16 bytes) is four longwords. For all data transfers, MADDR[31:2] indicates the longword base address of the first byte of the reference item. MADDR[1:0] indicates the byte offset from this base address. The MSIZ[1:0] signals along with the low-order two address signals are used to determine how the data bus is used. Table 3-5 indicates the MRDATA[31:0] requirements for slave devices when responding to read transfers. A "-" indicates a "don't care", i.e. the value is ignored.

Table 3-5. MRDATA Requirements for Read Transfers

TRANSFER SIZE	MSIZ[1:0]	MADDR[1:0]	MRDATA[31:24]	MRDATA[23:16]	MRDATA[15:8]	MRDATA[7:0]
Byte	01	00	Byte Data	-	-	-
	01	01	-	Byte Data	-	-
	01	10	-	-	Byte Data	-
	01	11	-	-	-	Byte Data
Word	10	00	Word Data		-	
	10	10	-		Word Data	
Long	00	00	Longword Data			
Line	11	00	First Longword Data			

Freescale Semiconductor, Inc.

Some of the system bus controllers (SBC) within the ColdFire architecture support dynamically-sized external data transfers, i.e., the slave indicates the width of the data port at the time of the transfer. To support this bus sizing feature, there are certain data replication functions which must be performed by all master devices, including the ColdFire2/2M, during write cycles. Table 3-6 indicates the MWDATA[31:0] requirements for master devices when initiating write transfers.

Table 3-6. MWDATA Bus Requirements for Write Transfers

TRANSFER SIZE	MSIZ[1:0]	MADDR[1:0]	MWDATA[31:24]	MWDATA[23:16]	MWDATA[15:8]	MWDATA[7:0]
Byte	01	--	Byte Data	Byte Data	Byte Data	Byte Data
Word	10	-0	Word Data		Word Data	
Long	00	00	Longword Data			
Line	11	00	First Longword Data			

Note that the ColdFire2/2M device, as well as all masters, places the byte operand on all byte lanes for a byte write cycle, and the word operand on both word lanes for a word write cycle.

3.3 DATA TRANSFERS

3.3.1 Byte, Word, and Longword Read Transfers

For byte, word, and longword read accesses, the ColdFire2/2M requests data from a slave device. The bus operations are similar for the different sized accesses, with the MSIZ[1:0] signals defining the access size. Based on the transfer size, the data is placed on the appropriate byte lanes of the read data bus (MRDATA[31:0]). This read transfer is defined in Figure 3-1.

MASTER   SLAVE

1) DRIVE ADDRESS ON MADDR[31:0]

2) SET MRWB TO READ (MRWB = 1)

3) SET MSIZ[1:0] TO BYTE, WORD, OR LONG

4) SET MTT[1:0], MTM[2:0] TO SIGNAL THE
   APPROPRIATE ACCESS TYPE

5) ASSERT MTSB FOR ONE CLK CYCLE

1) DECODE THE ADDRESS AND SELECT
   THE APPROPRIATE SLAVE DEVICE.
   INSERT NECESSARY WAIT-STATES.

1) DRIVE THE DATA ON THE APPROPRIATE
   BYTE LANES OF THE MRDATA[31:0] BUS,
   BASED ON MSIZ[1:0], MADDR1:0].

2) ASSERT MIE FOR ONE CLK CYCLE.

2) ASSERT MTAB FOR ONE CLK CYCLE.

1) REGISTER THE MRDATA[31:0] BUS

2) RECOGNIZE THE TRANSFER IS DONE

Figure 3-1. Byte, Word, and Longword Read Transfer Flowchart

See Figure 3-2 for an example of normal read and write master bus transfers without wait states.

Freescale Semiconductor, Inc.

Figure 3-2. Normal Transfer (without Wait States)

Clock 1 (C1)

The read cycle starts in C1. During the first half of C1, the ColdFire2/2M places valid values on the master address bus (MADDR[31:0]) and transfer control signals. The MTT[1:0] and MTM[2:0] signals identify the specific access type. The master read/write (MRWB) signal is driven high for a read cycle, and the master size signals (MSIZ[1:0]) indicate transfer size. The ColdFire2/2M asserts the master transfer start (MTSB) signal during C1 to indicate the beginning of a bus cycle.

Clock 2 (C2)

During the first half of C2, the ColdFire2/2M negates MTSB. The selected device uses the MRWB and MSIZ signals to place the data on the master read data bus (MRDATA[31:0]) and assert the master input enable (MIE) signal. Concurrently, the selected device asserts the master transfer acknowledge (MTAB) signal. At the end of C2, the ColdFire2/2M samples the level of MTAB and latches the current value on MRDATA[31:0]. If MTAB is

asserted, the transfer terminates. If MTAB is not asserted, the ColdFire2/2M processor ignores the data and inserts wait states instead of terminating the transfer. The ColdFire2/2M continues to sample MTAB on successive rising edges of CLK until it is asserted. The selected device negates the MIE and MTAB signals in the first half of the next CLK cycle.

3.3.2 Normal Tranfers with MKILLB

Figure3-3 illustrates normal tranfers (no wait states) with MKILLB asserted and negated:
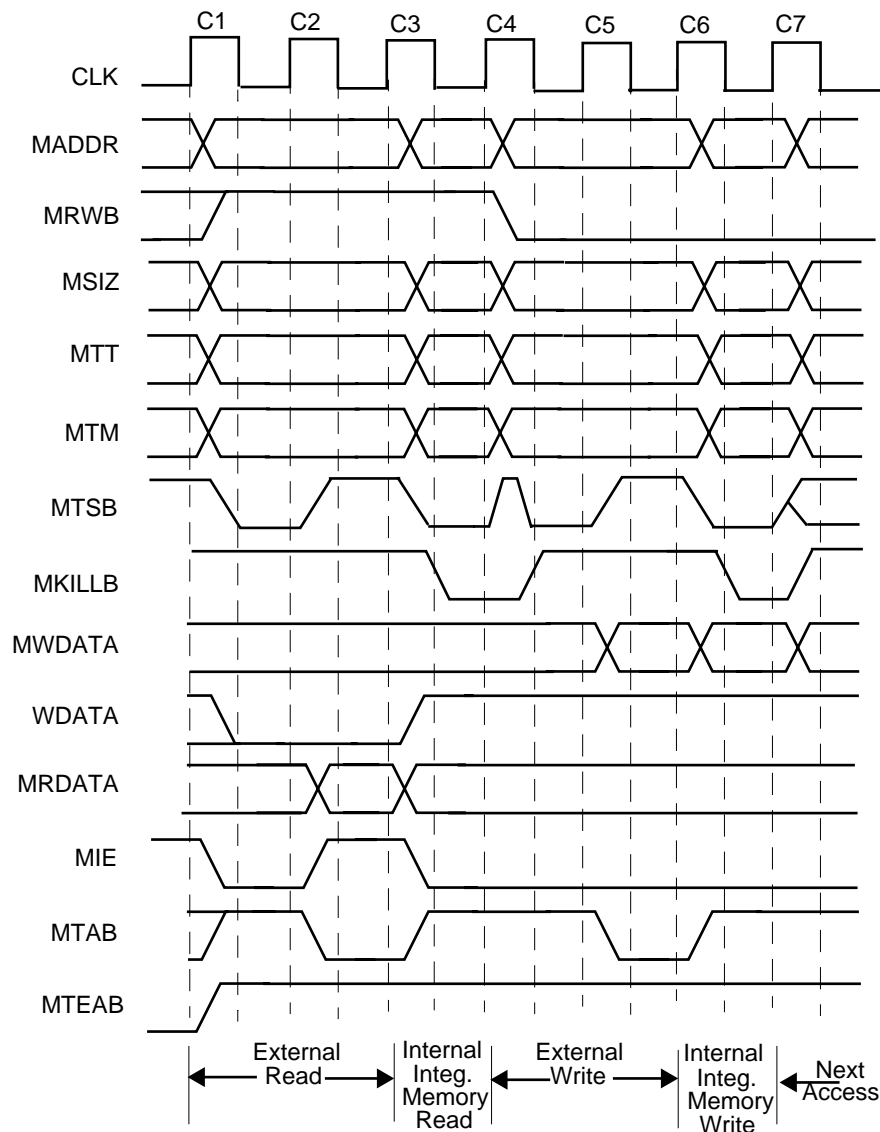
Figure 3-3. Normal Transfer with MKILLB Timing Diagram (without wait states)

Clock 1 (C1)

The read cycle starts in C1. During the first half of C1, the ColdFire2/2M places valid values on the master address bus (MADDR[31:0]) and transfer control signals. The MTT[1:0] and MTM[2:0] signals identify the specific access type. The master read/write (MRWB) signal is

Freescale Semiconductor, Inc.

driven high for a read cycle, and the master size signals (MSIZ[1:0]) indicate transfer size. The ColdFire2/2M asserts the master transfer start (MTSB) signal during C1 to indicate the beginning of a bus cycle. MKILLB does not assert during this cycle so the access will go external via M-Bus.

Clock 2 (C2)

During the first half of C2, the ColdFire2/2M processor negates MTSB. The selected device uses the MRWB and MSIZ signals to place the data on the master read data bus (MRDATA[31:0]) and assert the master input enable (MIE) signal. Concurrently, the selected device asserts the master transfer acknowledge (MTAB) signal. At the end of C2, the ColdFire2/2M processor samples the level of MTAB and latches the current value on MRDATA[31:0]. If MTAB is asserted, the transfer terminates. If MTAB is not asserted, the ColdFire2/2M ignores the data and inserts wait states instead of terminating the transfer. The ColdFire2/2M continues to sample MTAB on successive rising edges of CLK until it is asserted. The selected device negates the MIE and MTAB signals in the first half of the next CLK cycle.

Clock 3 (C3)

Another read cycle begins in C3; however this read cycle will be to internal integrated memory. During the first half of C3, the ColdFire2/2M places valid values on the master address bus (MADDR[31:0]) and transfer control signals. The MTT[1:0] and MTM[2:0] signals identify the specific access type. The master read/write (MRWB) signal is driven high for a read cycle, and the master size signals (MSIZ[1:0]) indicate transfer size. The ColdFire2/2M asserts the master transfer start (MTSB) signal during C3 to indicate the beginning of a bus cycle. MKILLB asserts late in the cycle. The combination of the assertion of both (MTSB and MKILLB signifies that the access will occur internally in integrated memory and the M-Bus transaction is not needed. The internal access will complete in one cycle; therefore another access can begin immediately on the following cycle.

Clock 4 (C4)

A write cycle starts in C4. During the first half of C4, the ColdFire2/2M places valid values on the master address bus (MADDR[31:0]) and transfer control signals. The MTT[1:0] and MTM[2:0] signals identify the specific access type. The master read/write (MRWB) signal is driven low for a write cycle, and the master size signals (MSIZ[1:0]) indicate transfer size. The ColdFire2/2M asserts the master transfer start (MTSB) signal during C4 to indicate the beginning of a bus cycle. MKILLB does not assert during this cycle so the access will go external via M-Bus.

Cycle 5 (C5)

During the first half of C5, the ColdFire2/2M negates MTSB. The selected device uses the MRWB and MSIZ signals to take the data off the master write data bus (MWDATA[31:0]. Concurrently, the selected device asserts the master transfer acknowledge (MTAB signal. At the end of C5, if MTAB is asserted, the transfer terminates. If MTAB is not asserted, the ColdFire2/2M inserts wait states instead of terminating the transfer. The ColdFire2/2M

continues to sample MTAB on successive rising edges of CLK until it is asserted. The selected device negates the (MTAB signal in the first half of the next CLK cycle.

Cycle 6 (C6)

Another write cycle begins in C6; however this write cycle will be to internal integrated memory. During the first half of C6, the ColdFire2/2M places valid values on the master address bus (MADDR[31:0]) and transfer control signals. The MTT[1:0] and MTM[2:0] signals identify the specific access type. The master read/write (MRWB) signal is driven low for a write cycle, and the master size signals (MSIZ[1:0]) indicate transfer size. The ColdFire2/2M asserts the master transfer start (MTSB) signal during C6 to indicate the beginning of a bus cycle. MKILLB asserts late in the cycle. The combination of the assertion of both (MTSB and MKILLB signifies that the access will occur internally in integrated memory and the M-Bus transaction is not needed. The internal access will complete in one cycle; therefore another access can begin immediately on the following cycle.

Cycle 7 (C7)

Another access can begin in this cycle ((MTSB can assert).

3.3.3 Byte, Word, and Longword Write Transfers

For byte, word, and longword write accesses, the ColdFire2/2M transfers data to a slave device. The bus operations are similar for the different sized accesses, with the MSIZ[1:0] signals defining the access size. Based on the transfer size, the data is placed on the appropriate byte lanes of the write data bus (MWDATA[31:0]). This write transfer is defined in Figure 3-4.

Freescale Semiconductor, Inc.

MASTER   SLAVE

1) DRIVE ADDRESS ON MADDR[31:0]

2) SET MRWB TO WRITE (MRWB = 0)

3) SET MSIZ[1:0] TO BYTE, WORD, OR LONG

4) SET MTT[1:0], MTM2:0] TO SIGNAL THE APPROPRIATE ACCESS TYPE

5) ASSERT MTSB FOR ONE CLK CYCLE

1) DECODE THE ADDRESS AND SELECT THE APPROPRIATE SLAVE DEVICE. INSERT NECESSARY WAIT-STATES.

1) DRIVE DATA ONTO MWDATA[31:0]

2) ASSERT MWDATAOE

1) CAPTURE THE DATA FROM THE APPROPRIATE BYTE LANES OF THE MWDATA[31:0] BUS, BASED ON MSIZ[1:0], MADDR[1:0]

2) ASSERT MTAB FOR ONE CLK CYCLE

1) RECOGNIZE THE TRANSFER IS DONE

2) NEGATE MWDATAOE

Figure 3-4. Byte, Word, and Longword Write Transfer Flowchart

See Figure 3-2 for an example of a normal write transfer without wait states. Figure 3-5 shows an example of a normal write master bus transfer with wait states. Note the Cxw nomenclature is used to define a wait state during the bus cycle—e.g., C2w.

ColdFire2/2M User's Manual
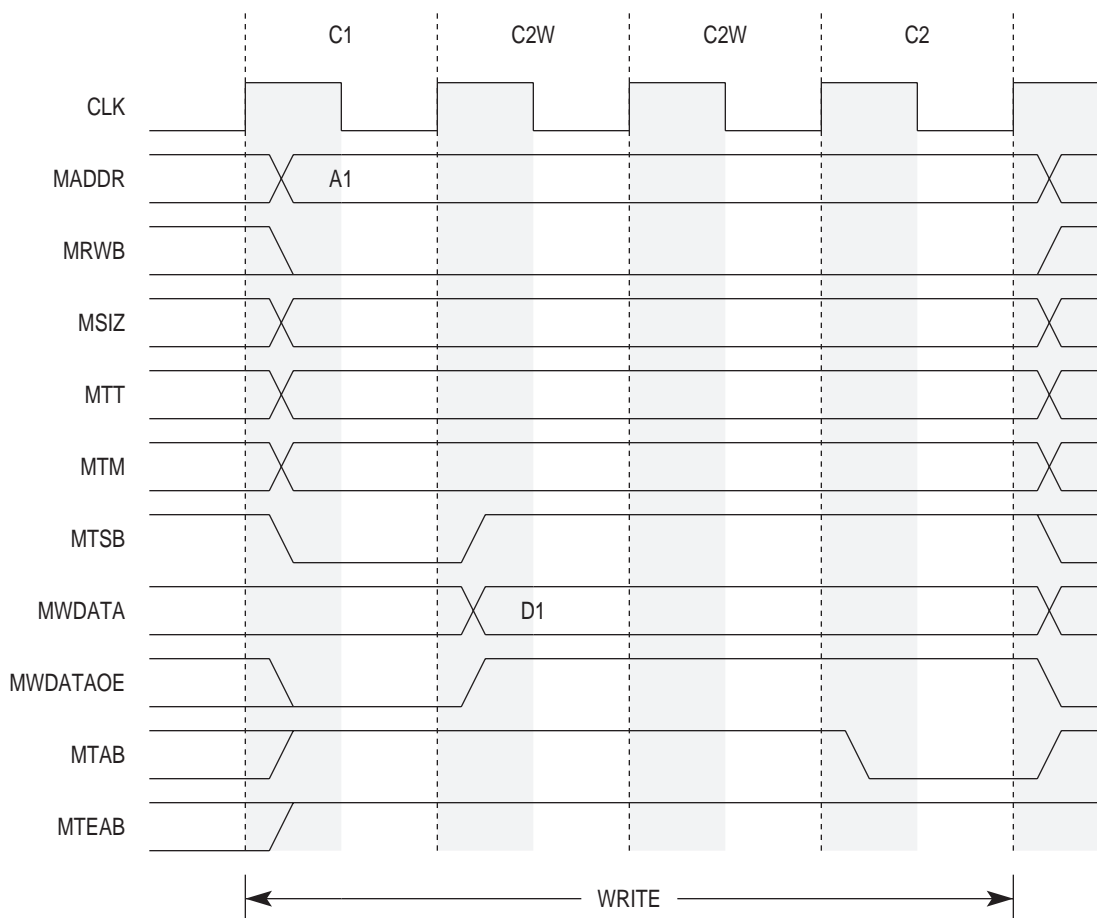
Figure 3-5. Normal Write Transfer (with Wait States)

Clock 1 (C1)

The write cycle starts in C1. During the first half of C1, the ColdFire2/2M places valid values on the master address bus (MADDR[31:0]) and transfer control signals. The master transfer type (MTT[1:0]) and master transfer modifier (MTM[2:0]) signals identify the specific access type. The master read/write (MRWB) signal is driven low for a write cycle, and the master size signals (MSIZ[1:0]) indicate transfer size. The ColdFire2/2M asserts the master transfer start (MTSB) signal during C1 to indicate the beginning of a bus cycle.

Clock 2 (C2)

During the first half of C2, the ColdFire2/2M negates MTSB, places the data on the master write data bus (MWDATA[31:0]), and asserts the master write data output enable (MWDATAOE) signal. Concurrently, the selected device asserts the master transfer acknowledge (MTAB) signal if it is ready to latch the data. At the end of C2, the selected device latches the current value on MWDATA[31:0], and the ColdFire2/2M samples the level of MTAB. If MTAB is asserted, the bus cycle is terminated and the ColdFire2/2M negates MWDATAOE in the first half on the next CLK cycle. If MTAB is not asserted, the ColdFire2/2M inserts wait states instead of terminating the transfer. The ColdFire2/2M continues to sample MTAB on successive rising edges of CLK until it is asserted.

Freescale Semiconductor, Inc.

3.3.4 Line Read Transfer

The ColdFire2/2M uses a line read access to fetch a four-longword operand using a burst transfer. A line read accesses a block of four longwords, aligned to a 16-byte memory boundary, by supplying a starting address that points to the critical longword in the four-longword block. The address and attributes driven by the ColdFire2/2M remain stable throughout the entire transfer. As a result, the slave device must increment MADDR[3:2] internally to sequence for each transfer with the address wrapping around at the end of the block. The allowable longword fetch patterns during a line access are shown in Table 3-7.

Table 3-7. Allowable Line Access Patterns

MADDR[3:2]	LONGWORD ACCESS ORDER
00	0 - 4 - 8 - C
01	4 - 8 - C - 0
10	8 - C - 0 - 4
11	C - 0 - 4 - 8

The responding slave device terminates each longword transfer on the MRDATA[31:0] bus by asserting the transfer acknowledge control signal, MTAB. All devices on the master bus must support burst accesses. The assertion of MTEAB aborts the line read access. See **Section 3.8.1 Access Errors** for more information on MTEAB.

A line read burst is defined in Figure 3-6.

MASTER   SLAVE

1) DRIVE ADDRESS ON MADDR[31:0]

2) SET MRWB TO READ (MRWB = 1)

3) SET MSIZ[1:0] TO LINE

4) SET MTT[1:0] AND MTM[2:0] TO SIGNAL THE
   APPROPRIATE ACCESS TYPE

5) ASSERT MTSB FOR ONE CLK CYCLE

1) DECODE THE ADDRESS AND SELECT
   THE APPROPRIATE SLAVE DEVICE.
   INSERT NECESSARY WAIT-STATES.

1) DRIVE THE DATA ON MRDATA[31:0].

2) ASSERT MIE FOR ONE CLK CYCLE.

3) ASSERT MTAB FOR ONE CLK CYCLE.

1) REGISTER THE MRDATA[31:0] BUS

2) RECOGNIZE THE 1ST TRANSFER IS DONE

1) DRIVE THE DATA ON MRDATA[31:0].

2) ASSERT MIE FOR ONE CLK CYCLE.

3) ASSERT MTAB FOR ONE CLK CYCLE.

1) REGISTER THE MRDATA[31:0] BUS

2) RECOGNIZE THE 2ND TRANSFER IS DONE

1) DRIVE THE DATA ON MRDATA[31:0].

2) ASSERT MIE FOR ONE CLK CYCLE.

3) ASSERT MTAB FOR ONE CLK CYCLE.

1) REGISTER THE MRDATA[31:0] BUS

2) RECOGNIZE THE 3RD TRANSFER IS DONE

1) DRIVE THE DATA ON MRDATA[31:0].

2) ASSERT MIE FOR ONE CLK CYCLE.

3) ASSERT MTAB FOR ONE CLK CYCLE.

1) REGISTER THE MRDATA[31:0] BUS

2) RECOGNIZE THE 4TH TRANSFER IS DONE

Figure 3-6. Line Read Transfer Flowchart

See Figure 3-7 for an example of a line read burst master bus transfer without wait states.
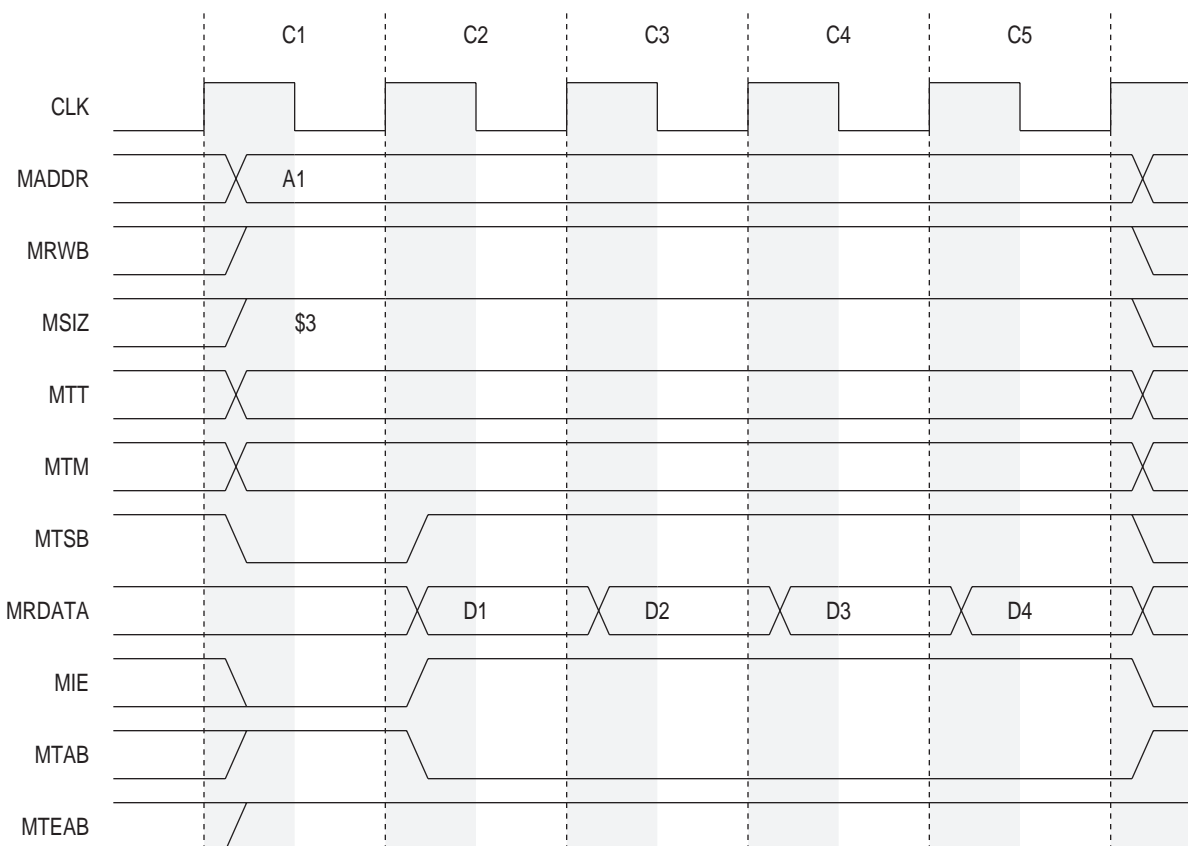
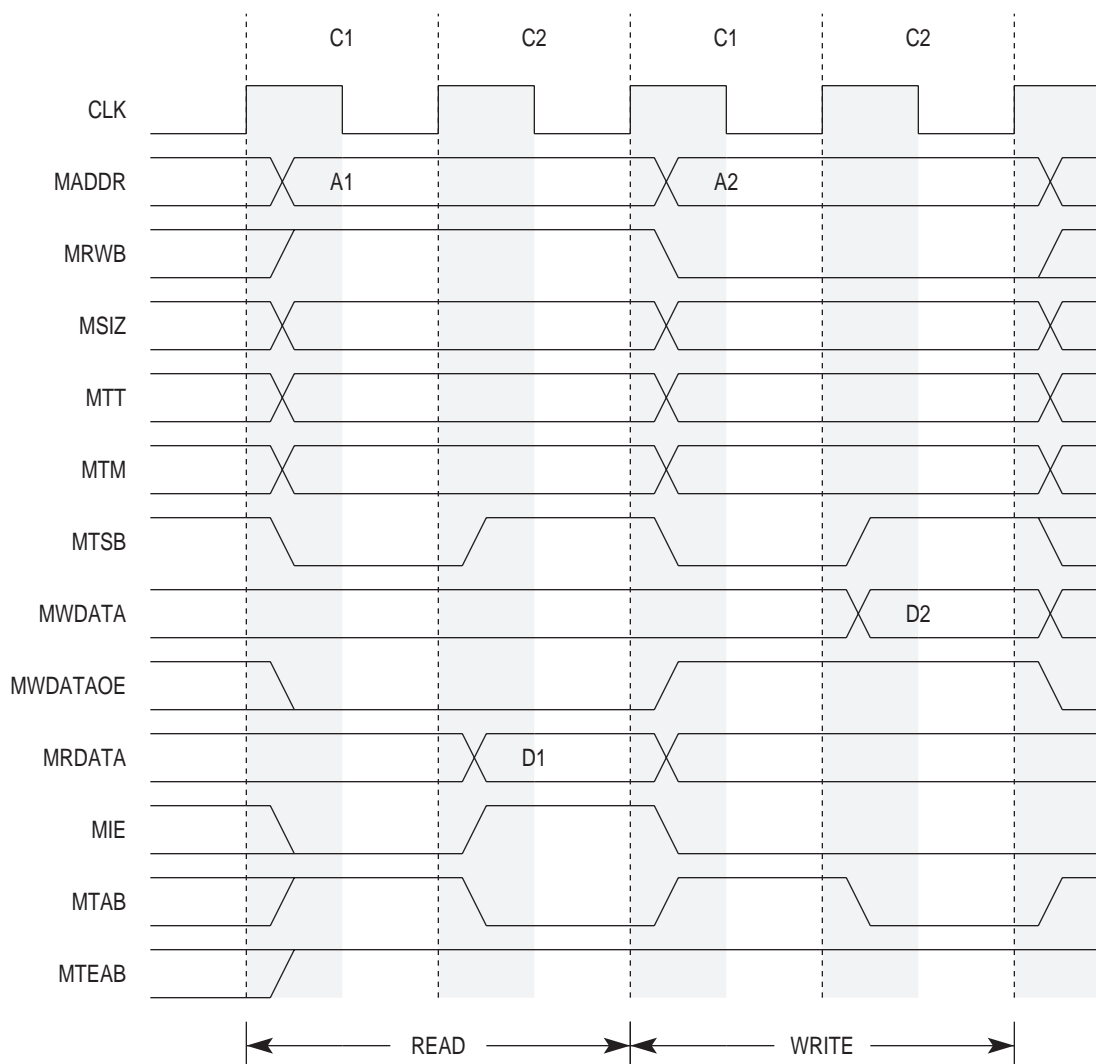**Figure 3-7. Line Read Transfer (without Wait States)**

Clock 1 (C1)

The line read cycle starts in C1. During the first half of C1, the ColdFire2/2M places valid values on the master address bus (MADDR[31:0]) and transfer control signals. The master transfer type (MTT[1:0]) and master transfer modifier (MTM[2:0]) signals identify the specific access type. The master read/write (MRWB) signal is driven high for a read cycle, and the master size signals (MSIZ[1:0]) indicate transfer size (Line = $3). The ColdFire2/2M asserts the master transfer start (MTSB) signal during C1 to indicate the beginning of a bus cycle.

Clock 2 (C2)

During the first half of C2, the ColdFire2/2M negates MTSB. The selected device uses the MRWB and MSIZ signals to place the data on the master read data bus (MRDATA[31:0]) and assert the master input enable (MIE) signal. The first transfer must supply the longword at the corresponding long-word boundary. Concurrently, the selected device asserts master transfer acknowledge (MTAB) signal. At the end of C2, the ColdFire2/2M samples the level of MTAB and latches the current value on MRDATA[31:0]. If MTAB is asserted, the transfer of the first longword terminates. If MTAB is not asserted, the ColdFire2/2M ignores the data and inserts wait states instead of terminating the transfer. The ColdFire2/2M continues to sample MTAB on successive rising edges of CLK until it is asserted.

Clock 3 (C3)

The ColdFire2/2M holds the address and transfer control signals constant during C3. The selected device must increment the MADDR[3:2] signals to reference the next longword to transfer, place the data on MRDATA[31:0], assert MIE, and assert MTAB. At the end of C3, the ColdFire2/2M samples the level of MTAB and latches the current value on the MRDATA[31:0] signals. If MTAB is asserted, the transfer terminates. If MTAB is not asserted at the end of C3, the ColdFire2/2M ignores the latched data and inserts wait states instead of terminating the transfer. The ColdFire2/2M continues to sample MTAB on successive rising edges of CLK until it is asserted.

Clock 4 (C4)

This clock is identical to C3 except that once MTAB is recognized as being asserted, the latched value corresponds to the third longword of data for the burst.

Clock 5 (C5)

This clock is identical to C3 except that once MTAB is recognized, the latched value corresponds to the fourth longword of data for the burst. This is the last clock cycle of the line read transaction. The selected device negates the MIE and MTAB signals in the first half of the next CLK cycle.

3.3.5 Line Write Transfers

The line write transfer is similar to the line read transfer. This 16-byte burst write transfer is defined in Figure 3-8.

MASTER   SLAVE

1) DRIVE ADDRESS ON MADDR[31:0]

2) SET MRWB TO WRITE (MRWB = 0)

3) SET MSIZ[1:0] TO LINE

4) SET MTT[1:0] AND MTM[2:0] TO SIGNAL THE APPROPRIATE ACCESS TYPE

5) ASSERT MTSB FOR ONE CLK CYCLE

1) DECODE THE ADDRESS AND SELECT THE APPROPRIATE SLAVE DEVICE

2) INSERT NECESSARY WAIT-STATES

1) DRIVE DATA ONTO MWDATA[31:0]

2) ASSERT MWDATAOE

1) CAPTURE THE DATA FROM MWDATA[31:0]

2) ASSERT MTAB FOR ONE CLK CYCLE

1) RECOGNIZE THE 1ST TRANSFER IS DONE

1) DRIVE DATA ONTO MWDATA[31:0]

1) CAPTURE THE DATA FROM MWDATA[31:0]

2) ASSERT MTAB FOR ONE CLK CYCLE

1) RECOGNIZE THE 2ND TRANSFER IS DONE

1) DRIVE DATA ONTO MWDATA[31:0]

1) CAPTURE THE DATA FROM MWDATA[31:0]

2) ASSERT MTAB FOR ONE CLK CYCLE

2) RECOGNIZE THE 3RD TRANSFER IS DONE

1) DRIVE DATA ONTO MWDATA[31:0]

1) CAPTURE THE DATA FROM MWDATA[31:0]

2) ASSERT MTAB FOR ONE CLK CYCLE

1) RECOGNIZE THE 4TH TRANSFER IS DONE

2) NEGATE MWDATAOE

Figure 3-8. Line Write Transfer Flowchart

The assertion of MTEAB will abort a line write transfer. See **Section 3.8.1 Access Errors** for more information on MTEAB. It is the slave's responsibility to increment and wrap the MADDR[3:2] signals internally.

See Figure 3-9 for an example of a line write master bus transfer without wait states.
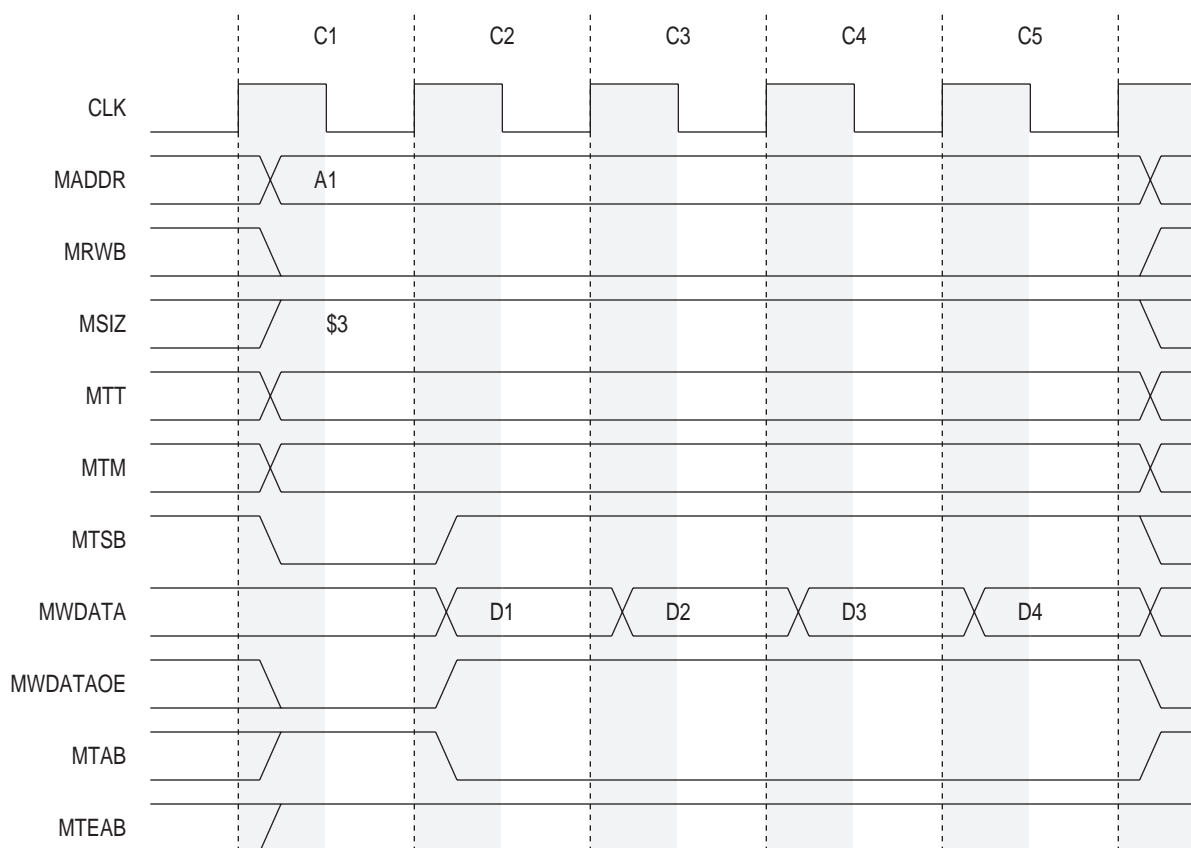
Figure 3-9. Line Write Transfer (without Wait States)

Clock 1 (C1)

The line write cycle starts in C1. During the first half of C1, the ColdFire2/2M places valid values on the master address bus (MADDR[31:0]) and transfer control signals. The master transfer type (MTT[1:0]) and master transfer modifier (MTM[2:0]) signals identify the specific access type. The master read/write (MRWB) signal is driven low for a write cycle, and the master size signals (MSIZ[1:0]) indicate line size ($3). The ColdFire2/2M asserts the master transfer start (MTSB) signal during C1 to indicate the beginning of a bus cycle.

Clock 2 (C2)

During the first half of C2, the ColdFire2/2M negates MTSB, places the first longword of data on the master write data bus (MWDATA[31:0]), and asserts the master write data output enable (MWDATAOE) signal. Concurrently, the selected device asserts the master transfer acknowledge (MTAB) signal if it is ready to latch the data. At the end of C2, the ColdFire2/ 2M samples the level of MTAB, and the selected device latches MWDATA[31:0]. If MTAB is asserted, the transfer of the first longword terminates. If MTAB is not asserted, the ColdFire2/2M inserts wait states instead of terminating the transfer. The ColdFire2/2M continues to sample MTAB on successive rising edges of CLK until it is recognized asserted.

Freescale Semiconductor, Inc.

Clock 3 (C3)

The ColdFire2/2M holds the address and transfer control signals constant during C3, but drives MWDATA[31:0] with the second longword of data. The selected device must increment MADDR[3:2] to reference the second longword address, and assert MTAB. At the end of C3, the ColdFire2/2M samples the level of MTAB, and the selected device latches MWDATA[31:0]. If MTAB is asserted, the transfer terminates. If MTAB is not recognized asserted at the end of C3, the ColdFire2/2M inserts wait states instead of terminating the transfer. The ColdFire2/2M continues to sample MTAB on successive rising edges of CLK until it is recognized.

Clock 4 (C4)

This clock is identical to C3 except that once MTAB is asserted, the value corresponds to the third longword of data for the burst.

Clock 5 (C5)

This clock is identical to C3 except that once MTAB is asserted, the data value corresponds to the fourth longword of data for the burst. This is the last clock cycle of the line write transaction and the ColdFire2/2M negates MWDATAOE in the first half of the next CLK cycle.

See Figure 3-10 for an example of a line write master bus transfer with wait states. Note the Cxw nomenclature that is used to define a wait state during the bus cycle, e.g., C2w.

ColdFire2/2M User's Manual
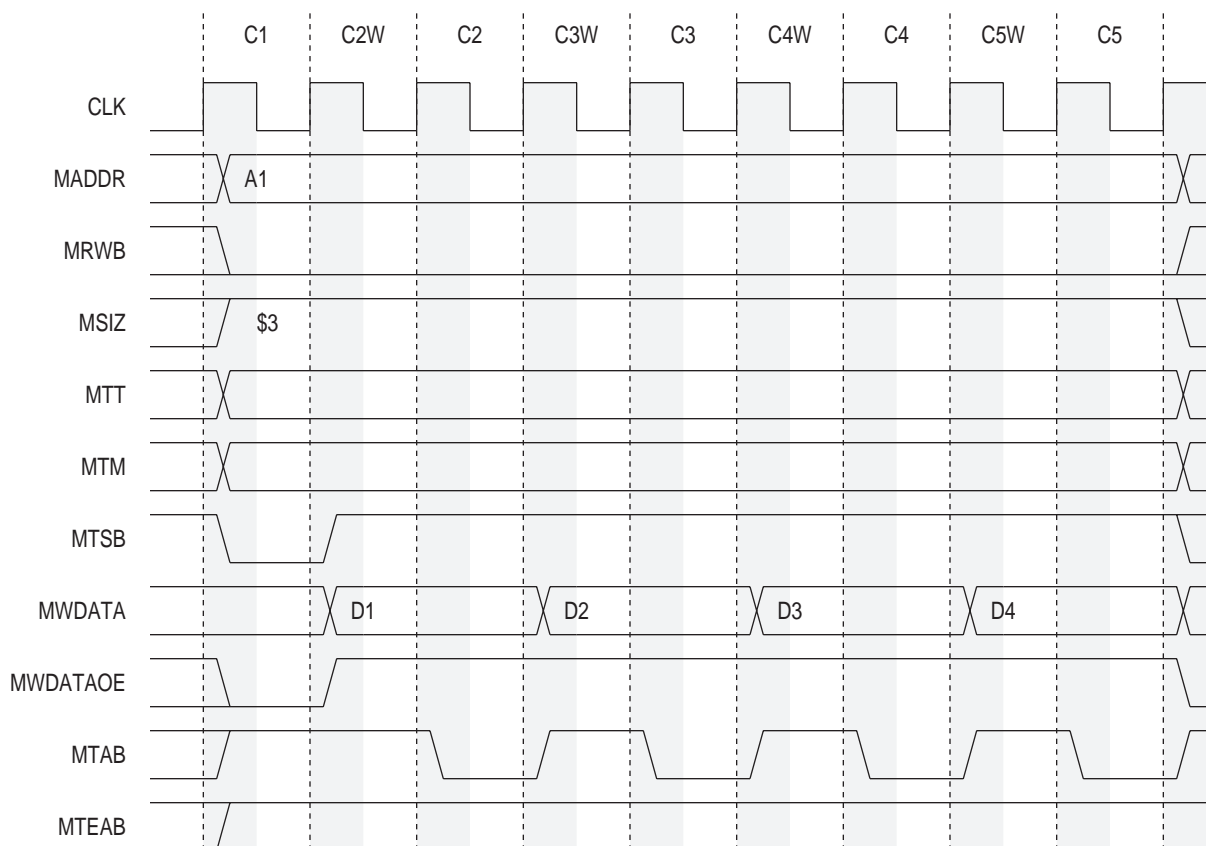
Figure 3-10. Line Write Transfer (with Wait States)

3.4 MISALIGNED OPERANDS

All ColdFire2/2M data formats can be located in memory on any byte boundary. 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 that is not evenly divisible by four. However, since operands can reside at any byte boundary, they can be misaligned. Although the ColdFire2/2M does not enforce any alignment restrictions for data operands, including PC relative data addressing, some performance degradation occurs when additional bus cycles are required for longword or word operands that are misaligned. For maximum performance, data items should be aligned on their natural boundaries. All instruction words and extension words must reside on word boundaries. Attempting to prefetch an instruction word at an odd address causes an address error exception. See **Section 4.2.3 Address Error Exception** for more information.

The ColdFire2/2M misalignment unit converts misaligned operand accesses that are noncachable to a sequence of aligned accesses. Figure 3-11 illustrates the transfer of a longword operand from an odd address requiring more than one bus cycle. In this example, the MSIZ[1:0] signals specify a byte transfer, and the byte offset is $1. The slave device supplies the byte and acknowledges the data transfer. When the ColdFire2/2M starts the second cycle, the MSIZ[1:0] signals specify a word transfer with a byte offset of $2. The next two bytes are transferred during this cycle. The ColdFire2/2M then initiates the third cycle,

Freescale Semiconductor, Inc.

with the MSIZ[1:0] signals indicating a byte transfer. The byte offset is now $0; the port supplies the final byte and the operation is complete. This example is similar to the one illustrated in Figure 3-12 except that the operand is word sized and the transfer requires only two bus cycles. Figure 3-13 illustrates a functional timing diagram for a misaligned word read transfer.

	31 24	23 16	15 8	7 0
Transfer 1	-	Byte 3	-	-
Transfer 2	-	-	Byte 2	Byte 1
Transfer 3	Byte 0	-	-	-

Figure 3-11. Example of a Misaligned Longword Transfer

	31 24	23 16	15 8	7 0
Transfer 1	-	-	-	Byte1
Transfer 2	Byte 0	-	-	-

Figure 3-12. Example of a Misaligned Word Transfer
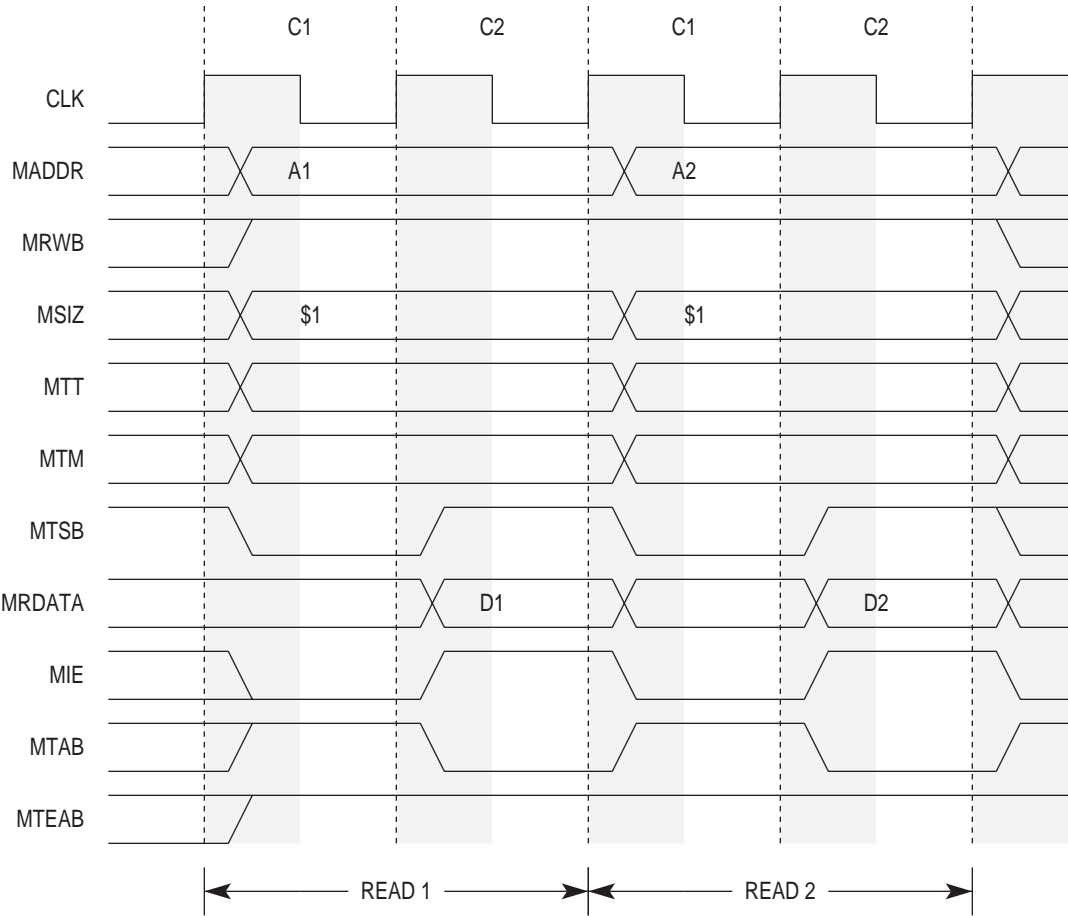
Figure 3-13. Misaligned Word Read Transfer

The combination of operand size and alignment determines the number of bus cycles required to perform a particular memory access. Table 3-8 lists the number of bus cycles required for different operand sizes with all possible alignment conditions for read and write cycles. The table confirms that alignment significantly affects bus cycle throughput for noncachable accesses. For example, in Figure 3-11 the misaligned longword operand took three bus cycles because the first byte offset = $1. If the byte offset = $0, then it would have taken one bus cycle. The ColdFire2/2M system designer and programmer should account for these effects, particularly in time-critical applications.

Table 3-8. Memory Alignment Cycles

TRANSFER SIZE	$0†	$1†	$2†	$3†
Instruction	1	-	-	-
Byte Operand	1	1	1	1
Word Operand	1	2	1	2
Longword Operand	1	3	2	3

NOTE: †Where the byte offset (MADDR[1:0]) equals this encoding.

3.5 INVALID MASTER BUS CYCLES

The ColdFire2/2M starts a master bus transaction before it determines if the address hits in the cache or is mapped to the internal memories. As a result, the ColdFire2/2M will assert the MKILLB signal (late in that MTSB cycle) to indicate the current bus transaction resulted in a hit in the cache or internal memories. When this signal is asserted, master bus slaves must stop driving the master bus and must not return a MTAB or MTEAB. The current master bus cycle must be inhibited.

The general priority scheme is as follows:

        if (SRAM "hits")
                SRAM supplies data to the processor
        else if (ROM "hits")
                ROM supplies data to the processor
        else if (line fill buffer "hits")
                line fill buffer supplies data to the processor
        else if (icache "hits")
                icache supplies data to the processor
        else
                master bus cycle accesses to reference data from non-local memory

Clock 1 (C1)

The read cycle starts in C1. During the first half of C1, the ColdFire2/2M places valid values on the master address bus (MADDR[31:0]) and transfer control signals. The ColdFire2/2M asserts the master transfer start (MTSB) signal during C1 to indicate the beginning of a bus cycle.

Clock 2 (C2)

During the first half of C2, the ColdFire2/2M negates MTSB. Because the MKILLB signal was not asserted during the rising clock edge of C2, the selected device uses the MRWB and MSIZ signals to place the data on the master read data bus (MRDATA[31:0]) and assert the master input enable (MIE) signal. Concurrently, the selected device asserts the master transfer acknowledge (MTAB) signal. At the end of C2, the ColdFire2/2M samples the level of MTAB and captures the current value on MRDATA[31:0]. If MTAB is asserted, the transfer terminates. If MTAB is not asserted, the ColdFire2/2M ignores the data and inserts wait states instead of terminating the transfer. The ColdFire2/2M continues to sample MTAB on successive rising edges of CLK until it is asserted. The selected device negates the MIE and MTAB signals in the first half of the next CLK cycle.

Clock 3 (C3)

The read cycle starts in C3. The ColdFire2/2M asserts the MTSB signal, and then the MKILLB signal. This signifies that the M-Bus cycle must be inhibited; i.e. the access has "hit" in an internal-bus memory.

Clock 4 (C4)

Nop.

Clock 5 (C5)

The read cycle starts in C5. The ColdFire2/2M asserts the MTSB signal, and then the MKILLB signal. This signifies that the M-Bus cycle must be inhibited; i.e. the access has "hit" in an internal K-Bus memory.

Clock 6 (C6)

Another read cycle starts in C6. The C1 description applies here.

Clock 7 (C7)

During the first half of C7, the ColdFire2/2M negates MTSB. Because the MKILLB signal was not asserted during the rising clock edge of C7, the selected device uses the MRWB and MSIZ signals to place the data on the master read data bus (MRDATA[31:0]). The rest of the C2 description applies here.

3.6 PIPELINE STALLS

In an idealized environment for maximum performance, all ColdFire2/2M references are mapped to integrated memory resources and complete in a single clock cycle. Any memory reference that generates a master bus access stalls the processor pipeline since the internal transfer cannot be completed in a single clock cycle. This performance degradation factor can be expressed as:

Pipeline Stall = Master Bus Clock Cycles - 1

where the stall is measured in clock cycles, and **Master Bus Clock Cycles** represents the entire master bus transfer time in clock cycles. In the examples shown in Figure 3-14, the read cycle with one wait state stalls the ColdFire2/2M for two clock cycles, and the zero wait state write transfer produces a stall of one clock cycle.
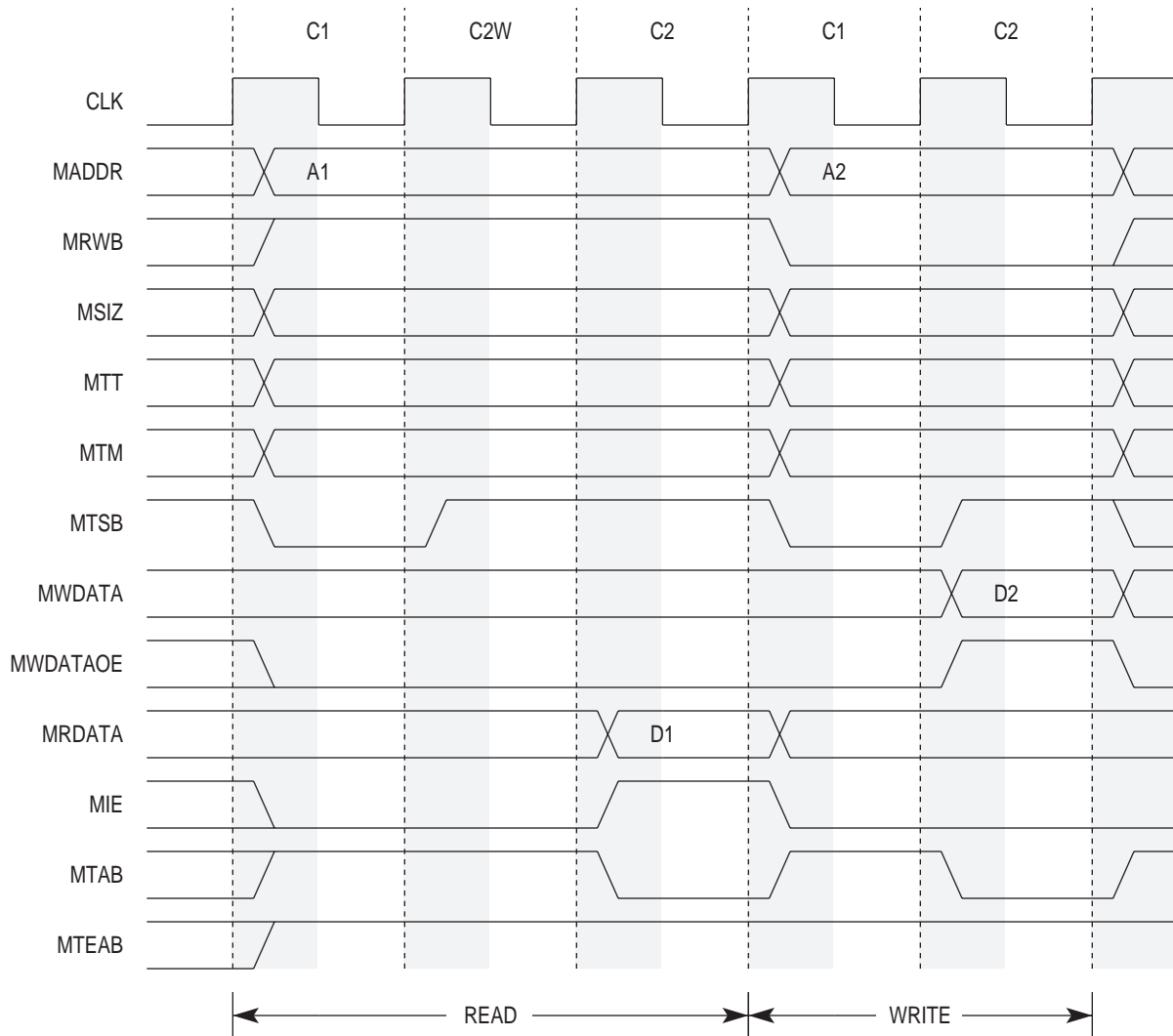
Figure 3-14. Example Master Bus Wait State

3.7 INTERRUPT ACKNOWLEDGE BUS CYCLES

When a peripheral device requires the services of the ColdFire2/2M or is ready to send information that the ColdFire2/2M requires, it can signal the ColdFire2/2M to take an interrupt exception. The interrupt exception transfers control to a routine that responds appropriately. The peripheral device uses the active-low interrupt priority level signals (IPLB[2:0]) to signal an interrupt condition to the ColdFire2/2M and to specify the priority level for the condition. Refer to **Section 4 Exception Processing** for a discussion on the IPLB[2:0] levels.

The status register of the ColdFire2/2M, described in **Section 1.4.3.1 Status Register (SR)**, contains an interrupt priority mask (IPLB[2:o]). The value in the interrupt mask is the highest priority level that the ColdFire2/2M ignores. When an interrupt request has a priority higher than the value in the mask, the ColdFire2/2M makes the request a pending interrupt. IPLB[2:0] must maintain the interrupt request level until the ColdFire2/2M acknowledges the interrupt to guarantee that the interrupt is recognized. The ColdFire2/2M continuously samples IPLB[2:0] on consecutive rising edges of CLK. As a result, the IPLB[2:0] signals are synchronous and must meet setup and hold times to CLK. If the external IPLB[2:0] signals are asynchronous, flip-flops should be used to synchronize them before they drive the IPLB[2:0] signals on the ColdFire2/2M.

The ColdFire2/2M takes an interrupt exception for a pending interrupt within one instruction boundary after processing any other pending exception with a higher priority. Thus, the ColdFire2/2M executes at least one instruction in an interrupt exception handler before recognizing another interrupt request. The following paragraphs describe the two kinds of interrupt acknowledge bus cycles that can be executed as part of interrupt exception processing.

3.7.1 Interrupt Acknowledge Bus Cycle (Terminated Normally)

When the ColdFire2/2M 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. Most interrupting devices have programmable vector registers that contain the interrupt vectors for the exception handlers they use. Others may have fixed vector numbers. The values driven on MADDR[31:0], MTT[1:0], and MTM[2:0] are dependent on the interrupt acknowledge mode. The interrupt acknowledge mode is statically determined by the connection of the 68K interrupt acknowledge mode enable (IACK_68K) signal.

The interrupt acknowledge bus cycle is a read transfer. If the ColdFire2/2M is in the ColdFire interrupt acknowledge mode (IACK_68K negated), it differs from a normal read cycle in the following respects:

1. MTT[1:0]= $3 to indicate a acknowledge/CPU space bus cycle.
2. Address signals MADDR[31:5] are set to all ones ($3FFFFFF). The MADDR[4:2] signals are set to the pending interrupt number, and the MADDR[1:0] signals are driven low.
3. MTM[2:0] are set to the interrupt request level, the inverted values of IPLB[2:0]. This will be nonzero for all interrupt acknowledge cycles.

If the ColdFire2/2M is in the 68K interrupt acknowledge mode (IACK_68K asserted), it differs in the following respects:

1. MTT[1:0] = $0 to indicate a Acknowledge/CPU space bus cycle.
2. Address signals MADDR[31:4] are set to all ones ($7FFFFFF). The MADDR[3:1] signals are set to the pending interrupt number, and the MADDR[0] signal is driven high.

3. MTM[2:0] = $7 to indicate an interrupt acknowledge cycle.

The responding device places the vector number on MRDATA[31:24] during the interrupt acknowledge bus cycle, and the cycle is terminated normally with MTAB. Figure 3-15 illustrates a flowchart diagram for an interrupt acknowledge cycle terminated with MTAB.

MASTER   SLAVE

```
┌─────────────────────────────┐        ┌─────────────────────────────┐
│ 1) RECOGNIZE PENDING        │ ◄───── │ REQUEST INTERRUPT ON        │
│    INTERRUPT                │        │ IPLB[2:0]                   │
│ 2) WAIT FOR INSTRUCTION     │        └─────────────────────────────┘
│    BOUNDARY                 │
└─────────────────────────────┘
              │
              ▼
┌─────────────────────────────┐        ┌─────────────────────────────┐
│ 1) DRIVE APPROPRIATE VALUE  │        │ 1) DECODE THE ADDRESS AND   │
│    ON MADDR[31:0]           │        │    SELECT THE INTERRUPTING  │
│ 2) SET MRWB TO READ ($1)    │        │    SLAVE DEVICE.            │
│ 3) SET MSIZ[1:0] TO BYTE    │ ─────► │ 2) INSERT NECESSARY         │
│    ($1)                     │        │    WAIT-STATES.            │
│ 4) SET MTT[1:0] TO          │        │ 3) REMOVE INTERRUPT ON      │
│    ACKNOWLEDGE              │        │    IPLB[2:0]                │
│ 5) DRIVE VALUE ON MTM[2:0]  │        └─────────────────────────────┘
│ 6) ASSERT MTSB FOR ONE CLK  │                       │
│    CYCLE                    │                       ▼
└─────────────────────────────┘        ┌─────────────────────────────┐
                                        │ 1) DRIVE THE VECTOR NUMBER  │
┌─────────────────────────────┐        │    ON MRDATA[31:24]         │
│ 1) LATCH VECTOR NUMBER      │ ◄───── │ 2) ASSERT MTAB FOR ONE CLK  │
│ 2) RECOGNIZE THE TRANSFER   │        │    CYCLE.                  │
│    IS DONE                  │        │ 3) ASSERT MIE FOR ONE CLK   │
└─────────────────────────────┘        │    CYCLE.                  │
                                        └─────────────────────────────┘
```

Figure 3-15. Interrupt Acknowledge Bus Cycle Flowchart

See Figure 3-16 for an example of a ColdFire mode (IACK_68K negated) interrupt acknowledge cycle. The interrupt shown occurs during a normal master bus read cycle with no lower-order interrupts pending.
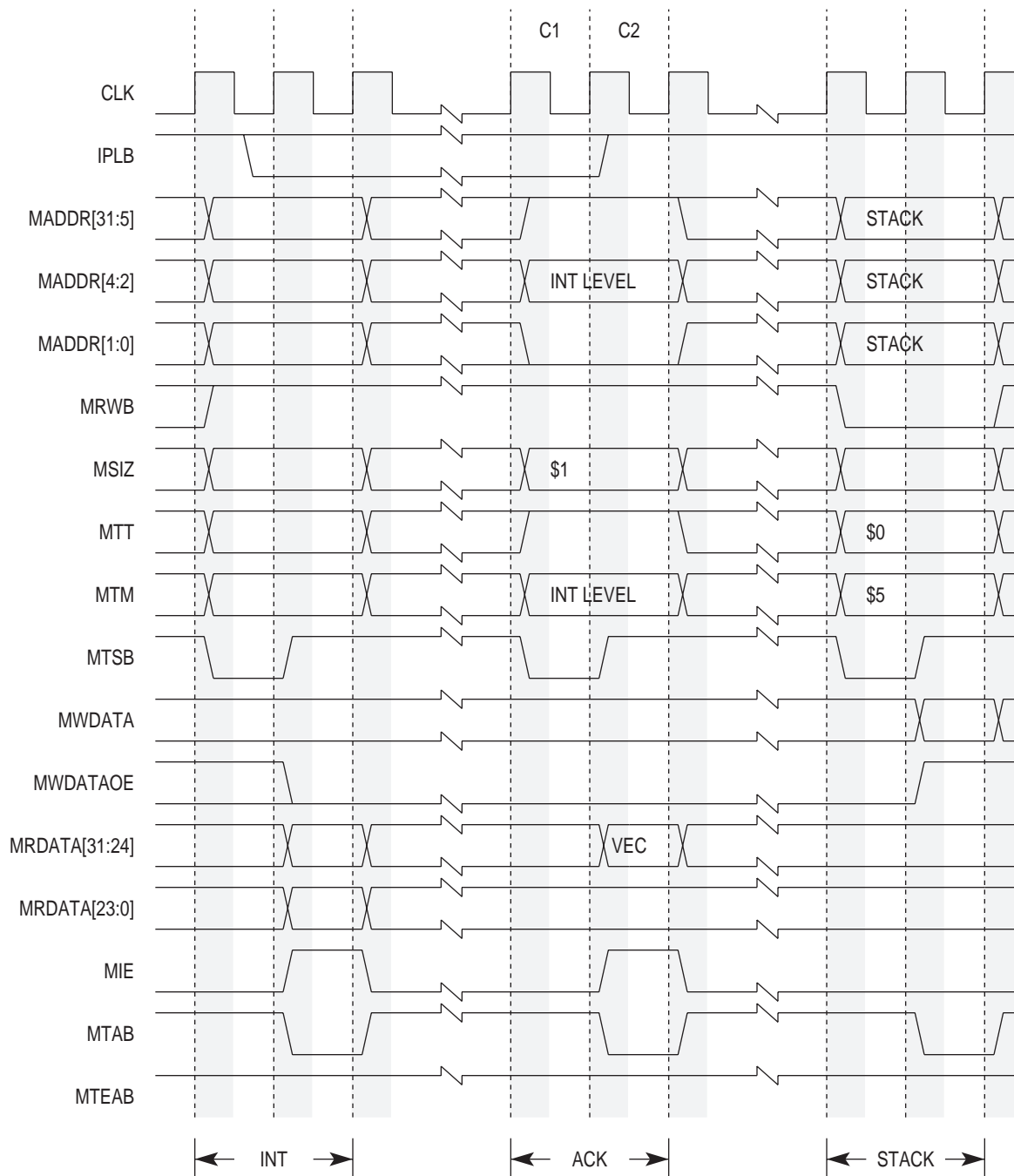
Figure 3-16. ColdFire Mode Interrupt Acknowledge Bus Cycle

Clock 1 (C1)

The interrupt acknowledge cycle starts in C1. During the first half of C1, the ColdFire2/2M drives the master address bus (MADDR[31:5]) high, MADDR[1:0] low, the MTT[1:0] signals to $3, and the MADDR[4:2] and MTM[2:0] signals to the interrupt level. The master read/write (MRWB) signal is driven high for a read cycle, and the master size signals (MSIZ[1:0]) indicate transfer size (Byte = $1). The ColdFire2/2M asserts the master transfer start (MTSB) signal during C1 to indicate the beginning of a bus cycle.

Clock 2 (C2)

During the first half of C2, the ColdFire2/2M negates MTSB. The interrupting device uses the MRWB and MSIZ signals to place the vector number on the high-order byte of the master read data bus (MRDATA[31:24]) and assert the master input enable (MIE) signal. Concurrently, the interrupting device asserts the master transfer acknowledge (MTAB) signal. At the end of C2, the ColdFire2/2M samples the level of MTAB and latches the current value on MRDATA[31:24]. If MTAB is asserted, the transfer terminates. If MTAB is not asserted, the ColdFire2/2M ignores the data and inserts wait states instead of terminating the transfer. The ColdFire2/2M continues to sample MTAB on successive rising edges of CLK until it is asserted. The interrupting device negates the MIE and MTAB signals in the first half of the next CLK cycle.

See Figure 3-17 for an example of a 68K mode (IACK_68K asserted) interrupt acknowledge cycle. The interrupt is shown occurring during a normal master bus read cycle with no lower-order interrupts pending.

Freescale Semiconductor, Inc.
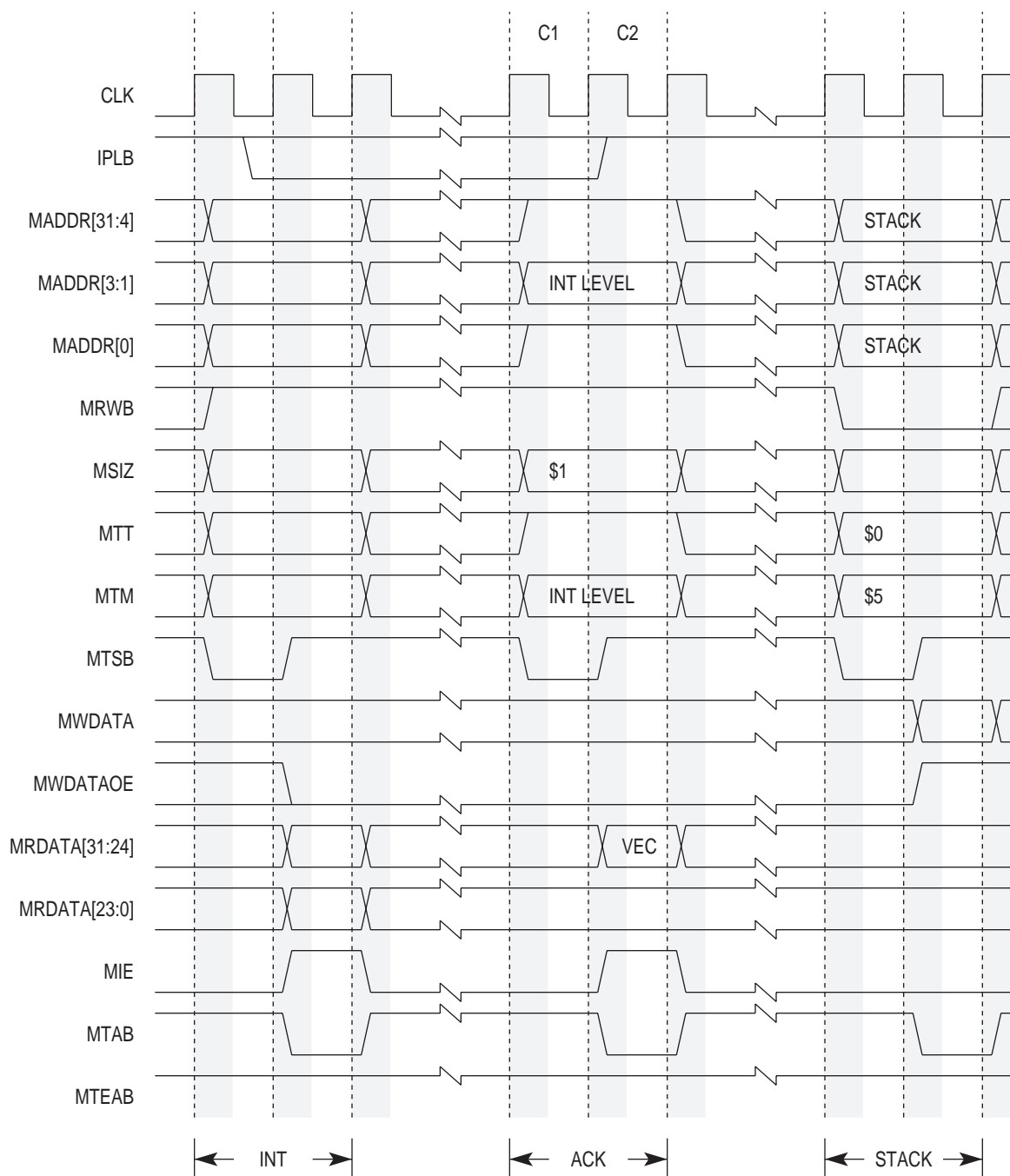
Figure 3-17. 68K Mode Interrupt Acknowledge Bus Cycle

Clock 1 (C1)

The interrupt acknowledge cycle starts in C1. During the first half of C1, the ColdFire2/2M drives the master address bus (MADDR[31:4]), MADDR[0], and the MTM[2:0] signals high, the MTT[1:0] signals low, and MADDR[3:1] to the interrupt level. The master read/write (MRWB) signal is driven high for a read cycle, and the master size signals (MSIZ[1:0]) indicate transfer size (Byte = $1). The ColdFire2/2M asserts the master transfer start (MTSB) signal during C1 to indicate the beginning of a bus cycle.

ColdFire2/2M User's Manual   MOTOROLA

Clock 2 (C2)

During the first half of C2, the ColdFire2/2M negates MTSB. The interrupting device uses the MRWB and MSIZ signals to place the vector number on the high-order byte of the master read data bus (MRDATA[31:24]) and assert the master input enable (MIE) signal. Concurrently, the interrupting device asserts the master transfer acknowledge (MTAB) signal. At the end of C2, the ColdFire2/2M samples the level of MTAB and latches the current value on MRDATA[31:24]. If MTAB is asserted, the transfer terminates. If MTAB is not asserted, the ColdFire2/2M ignores the data and inserts wait states instead of terminating the transfer. The ColdFire2/2M continues to sample MTAB on successive rising edges of CLK until it is asserted. The interrupting device negates the MIE and MTAB signals in the first half of the next CLK cycle.

3.7.2 Spurious Interrupt Acknowledge Bus Cycle

When a device does not respond to an interrupt acknowledge bus cycle with MTAB, the external logic typically returns the transfer error acknowledge signal (MTEAB). In this case, it is the responsibility of the external logic to return the spurious interrupt vector number 24 ($18) on MRDATA[31:24] and assert MIE. The vector number will be latched on the first rising edge of CLK after MTEAB is asserted. Because the spurious interrupt vector number is returned on MRDATA[31:24] in the same manor as a normal interrupt acknowledge, MTAB may be asserted instead of MTEAB.

3.8 MASTER BUS EXCEPTION CONTROL CYCLES

The ColdFire2/2M bus architecture requires assertion of MTAB from an external device to signal that a bus cycle is complete. MTAB is not asserted in the following cases:

- The external device does not respond to a normal bus cycle.
- No interrupt vector is provided during an interrupt acknowledge cycle.
- Various other application-dependent errors occur.

External circuitry should assert MTEAB when no device asserts MTAB within an appropriate period of time after the ColdFire2/2M begins the bus cycle. This terminates the cycle and allows the ColdFire2/2M to enter exception processing for the error condition.

To properly control termination of a bus cycle for a access error, MTAB and/or MTEAB must be asserted and negated for the same rising edge of CLK. Table 3-9 lists the control signal combinations and the resulting bus cycle terminations. Note that the access error Exception taken upon an MTEAB assertion cannot be masked and will occur for both reads and writes that result in MTEAB being asserted. access error terminations during burst cycles operate as described in **Section 3.3.4 Line Read Transfer** and **Section 3.3.5 Line Write Transfers**.

Table 3-9. MTAB and MTEAB Assertion Results

MTAB	MTEAB	RESULT
Don't Care	Asserted	Access Error—Terminate and Take Access Error Exception This exception cannot be masked
Asserted	Negated	Normal Cycle Terminate and Continue
Negated	Negated	Insert Wait States

3.8.1 Access Errors

The system hardware can use the MTEAB signal to abort the current bus cycle when a fault is detected as shown in Figure 3-18. An access error is recognized during a bus cycle in which MTEAB is asserted. When the ColdFire2/2M recognizes an access error condition, the access is terminated immediately. A line access that has MTEAB asserted for one of the four longword transfers aborts without completing the remaining transfers.

When MTEAB is asserted to terminate a bus cycle, the ColdFire2/2M can enter access error exception processing immediately following the bus cycle, or it can defer processing the exception in the following manner. The instruction prefetch mechanism requests instruction words from the instruction memory unit before it is ready to execute them. If an access error occurs on an instruction fetch, the ColdFire2/2M does not take the exception until it attempts to use the instruction. Should an intervening instruction cause a branch or should a task switch occur, the access error exception for the unused access does not occur. Similarly, if an access error is detected on the second, third, or fourth longword transfer for a line read access, an access error exception is taken only if the execution unit is specifically requesting that longword. Otherwise, the line is not placed in the cache, and the ColdFire2/2M repeats the line access when another access references the line. If a misaligned operand spans two longwords in a line, an access error on either the first or second transfer for the line causes exception processing to begin immediately. An access error termination for any write accesses or for read accesses that reference data specifically requested by the execution unit causes the ColdFire2/2M to begin exception processing immediately. Refer to **Section 4Exception Processing** for details of access error exception processing.

When an access error terminates an access, the contents of the corresponding cache can be affected. For a cache line read to replace a valid instruction line, the cache line being filled is invalidated before the bus cycle begins and remains invalid if the replacement line access is terminated with an access error.

Note that if an access is made to a space that is masked, it simply becomes mapped to the next valid space; no access error is generated. See Section 5.5, Interactions Between K-Bus Memories, for further information. When a write error occurs on a buffered write, an access error will be generated but will not be reported on the instruction that generated the write.

Access errors only occur because of the following:
1) MTEAB asserts
2) MMU error; i.e. trying to write in a write protected space

(RAMBAR, ROMBAR, ACR1, ACR0, and CACR have write protect bits)

3) Accessing CPU space from the debug module while the processor in NOT halted or stopped
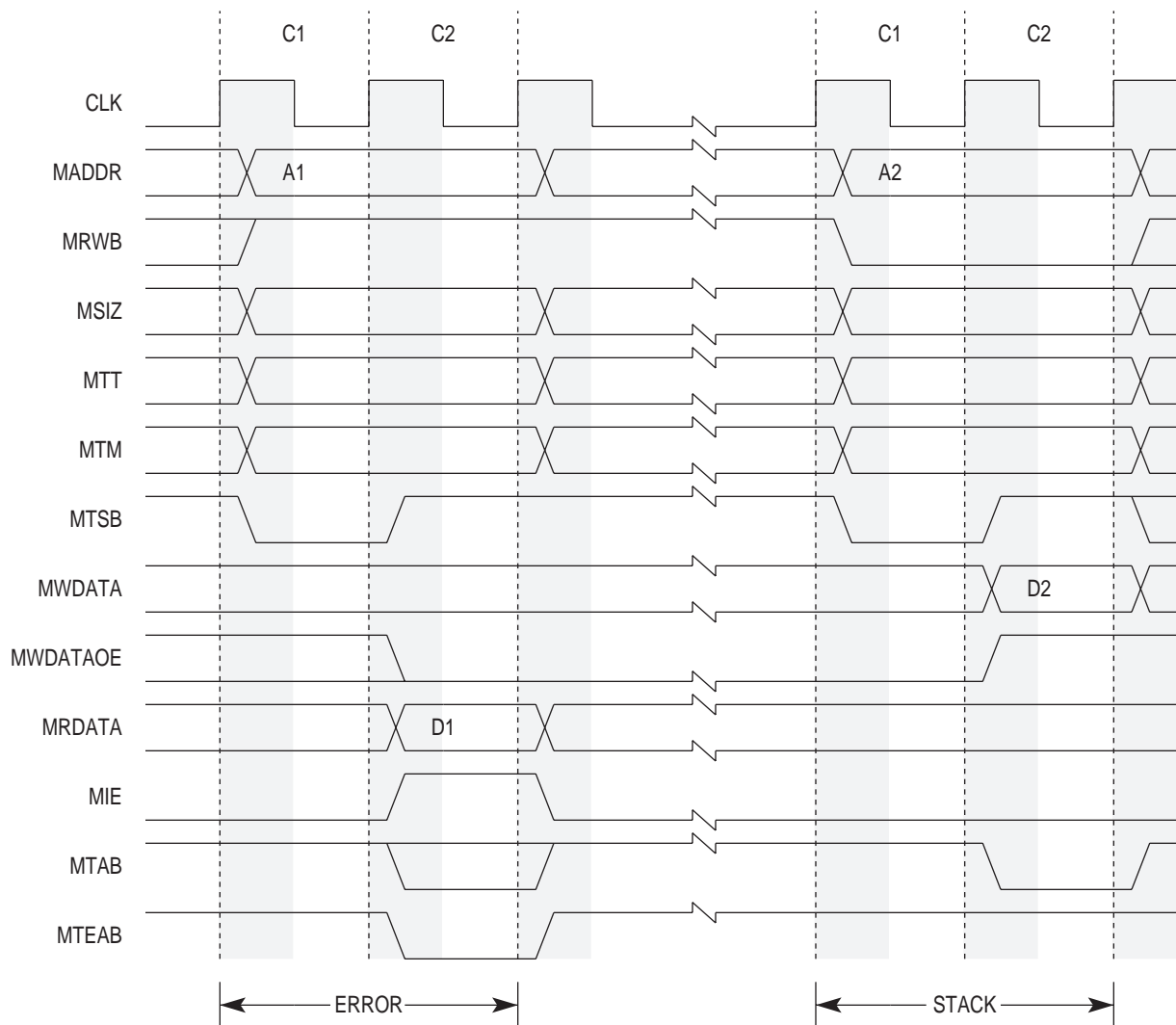
Figure 3-18. Bus Exception Cycle

3.8.2 Fault-on-Fault Halt

A fault-on-fault error occurs when an access or address error occurs during the exception processing sequence - e.g., the ColdFire2/2M attempts to stack several words containing information about the state of the machine while processing an access error exception. If an access error occurs during the stacking operation, the second error is considered a fault-on-fault error. A second access or address error that occurs during execution of an exception handler or later, does not cause a fault-on-fault condition. The ColdFire2/2M indicates a fault-on-fault condition by continuously driving the processor status (PST[3:0]) signals with an encoded value of $F until it is reset. Only an external reset operation can restart a halted ColdFire2/2M. See **Section 7.3.1 CPU Halt** for more information.

3.9 RESET OPERATION

An external device asserts the MRSTB signal to reset the ColdFire2/2M. When power is applied to the system, external circuitry should assert MRSTB for a minimum of six clock cycles after V_{DD} and CLK are within tolerance. The MRSTB signal is not internally synchronized and must meet the specified setup and hold times to CLK. If the external reset is asynchronous, two flip-flops should be used to synchronize the signal before it drives MRSTB.

Figure 3-19 shows the general relationship between V_{DD}, MRSTB, and the bus signals during the power-on reset operation. Resets during normal operation must follow the same requirements as those for power-on reset.
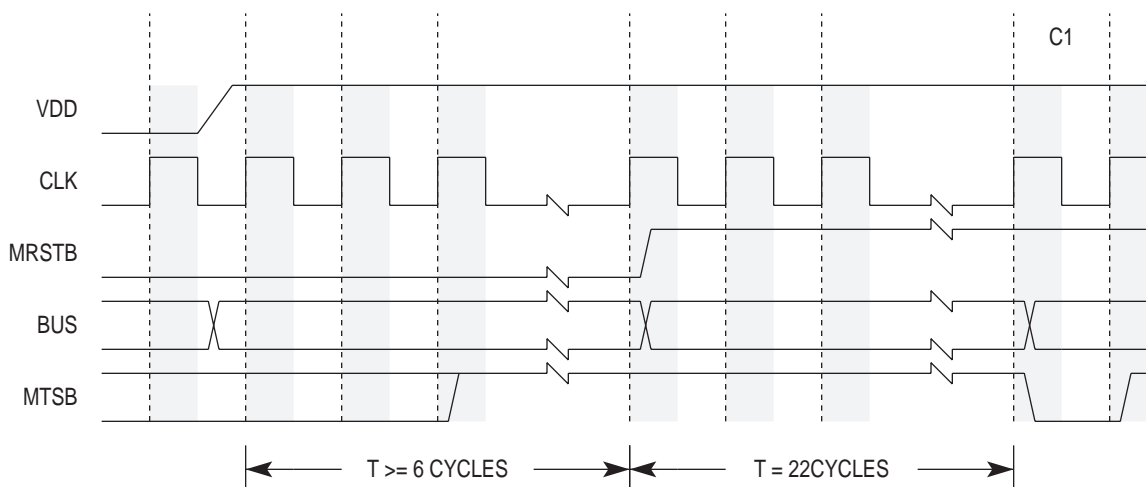
Figure 3-19. Initial Power-On Reset

3.10 MASTER BUS ARBITRATION

The ColdFire2/2M implementation is fully compatible with other, optional alternate Master's. The M-Bus supports arbitration; i.e. where the arbitration is performed in an external module to the ColdFire2/2M. This arbitration module could function as a multiplexer-based master switch. The ColdFire2/2M and the alternate masters generate master bus requests via their individual bus connections, with the arbitration logic selecting between the masters on a transfer-by-transfer basis.

SECTION 4
EXCEPTION PROCESSING

Exception processing is the activity performed by the ColdFire2/2M in preparing to execute a special routine, called an exception handler, for any condition that causes an exception. Exception processing does not include execution of the routine itself.

This section describes the processing for each type of integer unit exception, exception priorities, the return from an exception, and bus fault recovery. Also described are the formats of the exception stack frames.

4.1 EXCEPTION PROCESSING OVERVIEW

Exception processing is the transition from the normal processing of a program to the processing required for any special internal or external condition that preempts normal processing. External conditions that cause exceptions are interrupts from external devices, access errors, resets, and the assertion of the breakpoint signal. Internal conditions that cause exceptions are address errors, instruction traps, illegal instructions, privilege violations, tracing, format errors, and interrupts from the debug module. Exception processing uses an exception vector table and an exception stack frame.

Exception processing for the ColdFire2/2M is streamlined for performance. Differences from prior 68000 family processors are:

- A simplified exception vector table
- Reduced relocation capabilities using the vector base register (VBR)
- A fixed-length exception stack frame format
- Elimination of simultaneous multiple exception support
- Use of a single self-aligning system stack

Exception processing is comprised of four major steps and can be defined as the time from the detection of the fault condition until the fetch of the first instruction of the exception handler has been initiated. Figure 4-1 illustrates a general flowchart for the steps taken by the ColdFire2/2M during exception processing.

```
                        ┌──────────┐
                        │  ENTRY   │
                        └────┬─────┘
                             ▼
                   ┌───────────────────┐
                   │  SAVE INTERNAL    │
                   │  COPY OF SR       │
                   └─────────┬─────────┘
                             ▼
                   ┌───────────────────┐
                   │ ENABLE SUPERVISOR │
                   │ MODE & DISABLE    │
                   │ TRACE             │
                   └─────────┬─────────┘
                             ▼
                   ┌───────────────────┐
                   │   DETERMINE       │
                   │ VECTOR NUMBER     │
                   └─────────┬─────────┘
                             ▼
                   ┌───────────────────┐
                   │ SAVE CONTENTS TO  │
                   │ EXCEPTION FRAME   │
                   │ (SEE NOTE)        │
                   └─────────┬─────────┘
                             ▼
                        ◇ ACCESS ERROR? ◇──Y──► (FAULT-ON-FAULT)
                             │N
                             ▼
                   ┌───────────────────┐
                   │ CALCULATE ADDRESS │
                   │       OF          │
                   │ EXCEPTION HANDLER │
                   └─────────┬─────────┘
                             ▼
                   ┌───────────────────┐
                   │  FETCH FIRST      │
                   │  INSTRUCTION OF   │
                   │ EXCEPTION HANDLER │
                   └─────────┬─────────┘
                             ▼
                    ◇ ACCESS/ADDRESS ◇──Y──► (FAULT-ON-FAULT)
                    ◇    ERROR?      ◇
                             │N
                             ▼
                   ┌───────────────────┐         ┌──────────┐
                   │ BEGIN EXECUTION   │         │ HALTED   │
                   │      OF           │         │ STATE    │
                   │ EXCEPTION HANDLER │         └────┬─────┘
                   └─────────┬─────────┘              ▼
                             ▼                   ┌──────────┐
                        ┌──────────┐             │   EXIT   │
                        │   EXIT   │             └──────────┘
                        └──────────┘
```
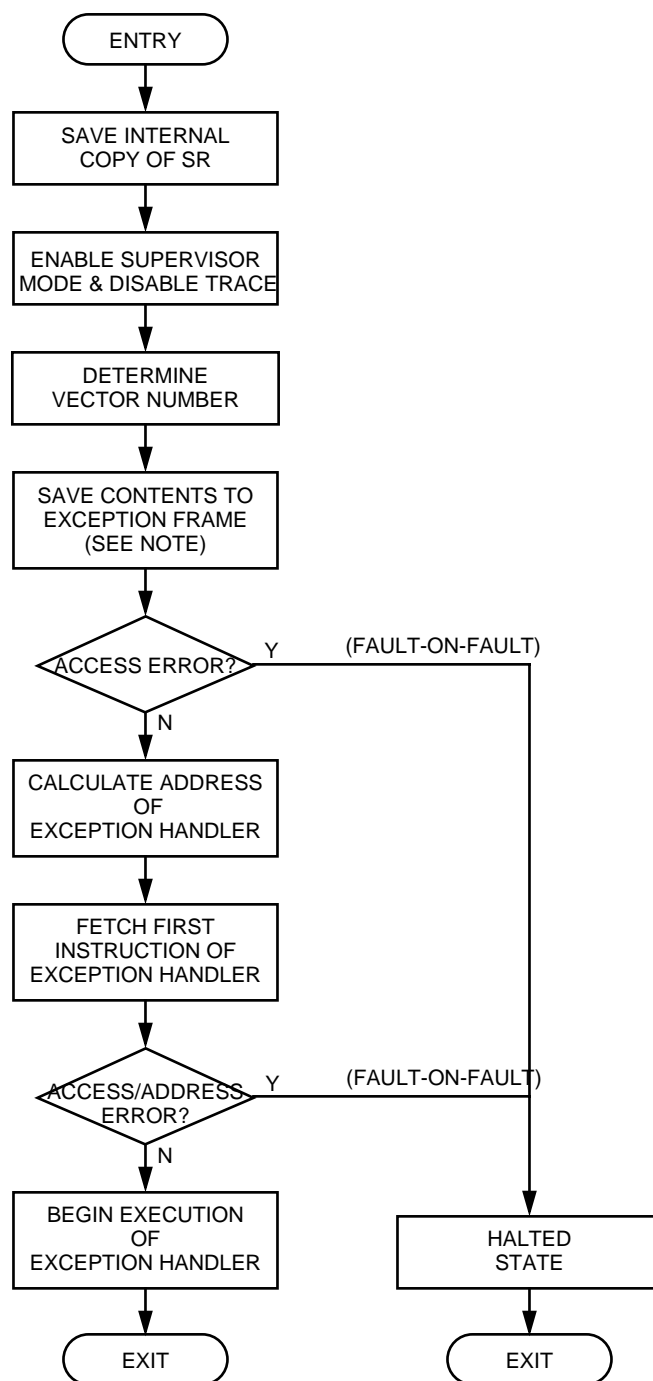
NOTE: THIS BLOCK VARIES FOR RESET AND INTERRUPT EXCEPTI[ON]

Figure 4-1. Exception Processing Flowchart

The processing of an exception on the ColdFire2/2M is composed of the following steps:

1. The ColdFire2/2M makes an internal copy of the status register (SR) and then enters supervisor mode by setting the S-bit and disabling trace mode by clearing the T-bit in the SR. The occurrence of an interrupt exception also forces the master/interrupt bit,

M-bit, in the SR to be cleared and the interrupt priority mask, I[2:0], in the SR to be set to the level of the current interrupt request

2. The ColdFire2/2M determines the exception vector number. For all faults except interrupts, the ColdFire2/2M performs this calculation based on the exception type. For interrupts, the ColdFire2/2M 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. Refer to Section 3.7 Interrupt Acknowledge Bus Cycles.

3. The ColdFire2/2M saves the current context by creating an exception stack frame on the system stack. The ColdFire2/2M supports a single stack pointer (SP) in the A7 address register, i.e., there is no notion of separate supervisor- or 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 ColdFire2/2M uses a simplified fixed-length exception stack frame for all exceptions. The exception type determines whether the program counter placed in the exception stack frame defines the location of the faulting instruction or the address of the next instruction to be executed.

4. The ColdFire2/2M calculates the address of the first instruction of the exception handler. By definition, the exception vector table is aligned on a 1 MByte boundary. This instruction address is generated by fetching an exception vector from the table located at the address defined in the vector base register. The index into the exception table is calculated as (4 x vector_number). Once the exception vector has been fetched, the contents of the vector determine the address of the first instruction of the desired exception handler. After the instruction fetch for the first opcode of the exception handler has been initiated, exception processing terminates and normal instruction processing continues in the exception handler.

4.1.1 Exception Stack Frame Definition

A diagram of the exception stack frame is shown in Figure 4-2. The first long word of the exception stack frame, pointed to by SP, contains the 16-bit format/vector word (F/V) and the 16-bit status register, and the second long word contains the 32-bit program counter address.

31 28	27 26 25	18	17 16	15 0
FORMAT	FS[3:2]	VECTOR[3:2]	FS[1:0]	STATUS REGISTER
PROGRAM COUNTER[31:0] PC				
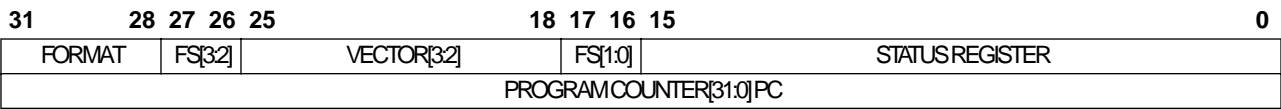
Figure 4-2. Exception Stack Frame Form

Freescale Semiconductor, Inc.

Field Definitions:

FORMAT—Exception Frame Format

This field is always written with a value of $4, $5, $6, or $7 by the ColdFire2/2M indicating a two long-word frame format. The specific value depends on the value of the stack pointer (SP) before the exception occurred.

    0100 = Original SP[1:0] set to 00
    0101 = Original SP[1:0] set to 01
    0110 = Original SP[1:0] set to 10
    0111 = Original SP[1:0] set to 11

FS—Fault Status

This field is defined for access and address errors only and written as zeros for all other types of exceptions.

    0000 = Not address or access error
    0001 = Reserved
    001x = Reserved
    0100 = Error on instruction fetch
    0101 = Reserved
    011x = Reserved
    1000 = Error on operand write
    1001 = Attempted write to write-protected space
    101x = Reserved
    1100 = Error on operand read
    1101 = Reserved
    111x = Reserved

VECTOR—Exception Vector Number

This 8-bit vector number defines the exception vector number used to index into the exception vector table. For internal exceptions, this number is calculated by the ColdFire2/2M, and for external interrupts, this number is read during the IACK cycle as described in Section 3.7 Interrupt Acknowledge Bus Cycles. Refer to Table 4-2 for exception vector assignments.

PC—Program Counter

This field contains the 32-bit program counter at the time of the exception. The instruction at this address is the faulting instruction or the next instruction depending on the type of exception.

4.1.1.1 SELF-ALIGNING STACK. The ColdFire2/2M aligns the stack to a longword boundary before the exception stack frame is placed on the stack. As a result, the difference between the SP before the exception and the SP at the start of the exception handler may be more than two longwords, but the location of the exception stack frame is always at the top of the stack. As seen in Table 4-1, the format field reflects the SP alignment and is used when an RTE instruction executes to correctly restore the SP to its original value.

Table 4-1. Stack Pointer Alignment

ORIGINAL SP [1:0] @ TIME OF EXCEPTION,	FORMAT FIELD	SP @ 1ST INSTRUCTION OF EXCEPTION HANDLER
00	0100	(Original SP) - 8
01	0101	(Original SP) - 9
10	0110	(Original SP) - 10
11	0111	(Original SP) - 11

4.1.2 Exception Vectors

The ColdFire2/2M supports a 1024-byte vector table aligned on any 1 MByte address boundary (see Table 4-2). The table contains 256 exception vectors where the first 64 are defined by Motorola and the remaining 192 are user-defined.The exception table order does not have any priority. It is an area in memory reserved for the storage of the first address used within the different exception handler routines. The reserved areas in the exception table are for future expansion and can be used as long as the application does not care if these areas become valid regions in future ColdFire processors.

Table 4-2. Exception Vector Assignments

VECTOR NUMBER		VECTOR OFFSET	STACKED PROGRAM COUNTER	ASSIGNMENT
HEX	DECIMAL	HEX		
$0	0	$000	-	Reset Initial Stack Pointer
$1	1	$004	-	Reset Initial Program Counter
$2	2	$008	Fault	Access Error
$3	3	$00C	Fault	Address Error
$4	4	$010	Fault	Illegal Instruction
$5-$7	5-7	$014-$01C	-	Reserved
$8	8	$020	Fault	Privilege Violation
$9	9	$024	Next	Trace
$A	10	$028	Fault	Unimplemented Line-A Opcode
$B	11	$02C	Fault	Unimplemented Line-F Opcode
$C	12	$030	Next	Debug Interrupt
$D	13	$034	-	Reserved
$E	14	$038	Fault	Format Error
$F	15	$03C	Next	Optional Uninitialized Interrupt
$10-$17	16-23	$040-$05C	-	Reserved
$18	24	$060	Next	Optional Spurious Interrupt
$19-$1F	25-31	$064-$07C	Next	Optional Level 1-7 Autovectored Interrupts
$20-$2F	32-47	$080-$0BC	Next	TRAP # 0-15 Instructions
$30-$3F	48-63	$0C0-$0FC	-	Reserved
$40-$FF	64-255	$100-$3FC	Next	User-Defined Interrupts

NOTES: 1. "Fault" refers to the PC of the instruction that caused the exception

2. "Next" refers to the PC of the next instruction that follows the instruction that caused the fault.

The ColdFire2/2M inhibits sampling for interrupts during exception processing, including the first instruction of the exception handler. This allows the first instruction of any exception handler to effectively disable interrupts, if desired, by raising the interrupt mask level contained in the status register.

Normally, the end of an exception handler contains an RTE instruction. When the ColdFire2/2M executes the RTE instruction, it examines the exception stack frame on top of the stack to determine if it is a valid frame. If the ColdFire2/2M determines that it is a valid frame, the SR and PC fields are loaded from the exception stack frame and control is passed to the specified instruction address. If the frame is invalid, a format exception is taken.

All exception vectors are located in the supervisor data space. Since the vector base register (VBR) provides the base address of the exception vector table, the exception vector table can be located anywhere in memory; it can even be dynamically relocated for each task that an operating system executes. The VBR is reset to zero. Refer to Section 1.4.3.4 Vector Base Register (VBR).

4.1.3 Multiple Exceptions

Within a ColdFire2/2M system, more than one exception can occur at the same time. When this occurs, only the exception with the highest priority will be processed. Exceptions can be divided into the four basic groups identified in Table 4-3.

Table 4-3. Exception Priority Groups

GROUP PRIORITY	EXCEPTION AND RELATIVE PRIORITY	CHARACTERISTICS
0.0	Reset	The ColdFire2/2M aborts all processing (instruction or exception) and does not save old context.
1.0 1.1 1.2	Address Error Instruction Access Error Data Access Error	The ColdFire2/2M suspends processing and saves the context.
2.0 2.1 2.2 2.3	A-Line Unimplemented F-Line Unimplemented Illegal Instruction Privilege Violation	Exception processing begins before the instruction is executed.
3.0	TRAP Instruction Format Error	Exception processing is part of the instruction execution and begins after instruction execution.
4.0 4.1 4.2	Trace Debug Interrupt Interrupt	Exception processing begins when the current instruction is completed.

NOTES: 1.   0.0 is the highest priority.

2.   4.2 is the lowest priority.

These groups are defined by specific characteristics and the priority with which they are handled. As far as the ColdFire2/2M is concerned, the interrupt exception will never appear to occur at the same time as another exception. Interrupt exceptions will remain pending until the other exception is processed. Refer to **Section 4.2.6 Trace Exception** for an example of simultaneous exceptions.

4.1.4 Fault-on-Fault Halt

If the ColdFire2/2M encounters any type of fault during the exception processing of another fault, it immediately halts execution with the catastrophic fault-on-fault condition. The ColdFire2/2M indicates a fault-on-fault condition by continuously driving the processor status (PST[3:0]) signals with an encoded value of $F until it is reset. Only an external reset operation can restart a halted ColdFire2/2M. See **Section 7.3.1 CPU Halt** for more information.

4.2 EXCEPTIONS

The following paragraphs describe the external interrupt exceptions and the different types of exceptions generated internally by the integer unit. The following exceptions are discussed:

- Reset
- Access Error
- Address Error
- Illegal Instruction
- Privilege Violation
- Trace
- Unimplemented Opcode
- Debug Interrupt
- Format Error
- TRAP Instruction
- External Interrupt

4.2.1 Reset Exception

Asserting the reset input signal to the ColdFire2/2M causes a reset exception, vector number $0. The reset exception has the highest priority of any exception; it provides for system initialization. Reset also aborts any processing in progress when the reset input is recognized, and the aborted processing cannot be recovered.

The reset exception places the ColdFire2/2M in the supervisor mode by setting the S-bit and disables tracing by clearing the T-bit in the SR. This exception also clears the M-bit and sets the ColdFire2/2M's interrupt priority mask in the SR to the highest level (level 7). Next, the VBR, CACR, ACRs, RAMBAR0, and ROMBAR0 are initialized to their reset value. This will disable the cache, SRAM, and optionally the ROM (see **Section 5.3.2 ROM Programming Model**.)

Once the ColdFire2/2M is granted the bus, and it does not detect any other alternate masters taking the bus, the core then performs two longword read bus cycles. Because the VBR is reset to zero, the first longword is always loaded into the stack pointer from address zero, and the second longword is always loaded into the program counter from address four. 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 begins execution, the ColdFire2/2M enters the fault-on-fault halted state. Refer to **Section 3.9 Reset Operation** for more information on initiating a reset exception.

4.2.2 Access Error Exception

An access error exception, vector number $2, occurs when a bus cycle terminates with an error condition such as the assertion of the master transfer error acknowledge (MTEAB) signal. Refer to Section 3.8 Master Bus Exception Control Cycles. The exact response to an access error is dependent on the type of memory reference being performed.

For an instruction fetch, the ColdFire2/2M postpones the reporting of an error until the faulted reference is needed by an instruction to be executed. Thus, faults which occur during instruction prefetches which are then followed by a change of instruction flow will not generate an exception. When the ColdFire2/2M attempts to execute an instruction with a faulted opword and/or extension words, the access error will be signaled and the instruction aborted. For this type of exception, the programming model has not been altered by the instruction generating the access error.

If the access error occurs on an operand read, the ColdFire2/2M immediately aborts the current instruction's execution and initiates exception processing. In this situation, any address register updates due to the auto-addressing modes, {e.g., (An)+,-(An)}, will already have been performed. Thus, the programming model contains the updated An value. In addition, if an access error occurs during the execution of a MOVEM instruction loading registers from memory, any registers already updated before the fault occurs will contain the operands from memory.

The ColdFire2/2M uses an imprecise reporting mechanism for access errors on write operations. Since the actual write cycle may be decoupled from the ColdFire2/2M's execution of the instruction requesting the write, the signaling of an access error appears to be decoupled from the instruction which generated the write. Accordingly, the PC contained in the exception stack frame merely represents the location in the program when the access error was signaled, not when the offending instruction was executed. All programming model updates associated with the write instruction are completed. The NOP instruction can be used for purposes of collecting access errors for writes. This instruction delays its execution until all previous operations, including all pending write operations to internal memory resources, are complete. Noncachable writes to the master bus may be buffered but will be completed before the execution of a NOP instruction. The NOP instruction waits until the pipeline is cleared out, including buffered writes before executing. Generally, buffering can provide higher performance, although issues concerning recovery from physical write-errors may become more difficult to resolve.

4.2.3 Address Error Exception

Any attempted execution transferring control to an odd instruction address (i.e., if bit 0 of the target address is set) results in an address error exception, vector number $3. For conditional branch instructions, the exception is generated regardless of the taken/not-taken resolution of the branch condition.

The ColdFire2/2M has a misalignment unit which will generate a series of aligned bus cycles to access data requested from a misaligned address. As a result, misaligned operand fetches will not cause an address error exception.

Any attempted use of a word-sized index register (Xn.w) or an invalid scale factor on an indexed effective addressing mode generates an address error. The setting of the extension word valid (EV) bit on an indexed addressing mode instruction will also generate an address error. Refer to the *ColdFire Programmer's Reference Manual Rev. 1.0* (MCF5200PRM/AD).

4.2.4 Illegal Instruction Exception

The attempted execution of the $0000 and the $4AFC opwords generates an illegal instruction exception, vector number $4. The ColdFire2/2M does not provide illegal instruction detection on the extension words on any instruction, including MOVEC. If any other non-supported opcode is executed, the resulting operation is undefined and the ColdFire2/2M's behavior is unpredictable.

4.2.5 Privilege Violation Exception

The attempted execution of a supervisor mode instruction while in user mode generates a privilege violation exception, vector number $8. See the *ColdFire Programmer's Reference Manual Rev. 1.0* for lists of supervisor- and user- mode instructions.

4.2.6 Trace Exception

To aid in program development, the ColdFire2/2M provides an instruction-by-instruction tracing capability. While in trace mode, indicated by the setting of the T-bit in the status register (SR[15] = 1), the completion of an instruction execution triggers a trace exception, vector number $9. This functionality allows a debugger to monitor program execution.

The single exception to this definition is the STOP instruction. A STOP instruction that begins execution in trace mode (T-bit in SR set) or enters trace mode because of the STOP instruction execution (bit 15 of the STOP operand is set) forces a trace exception after it loads the SR. Upon return from the trace exception handler, execution continues with the instruction following the STOP instruction, and the ColdFire2/2M never enters the stopped condition. A STOP instruction will enter the stopped state only if the ColdFire2/2M is not in trace mode before the STOP instruction is executed and the STOP instruction does not place the ColdFire2/2M in the trace mode.

Since theColdFire2/2M does 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 the execution of a TRAP instruction while in trace mode. The ColdFire2/2M will initiate the TRAP exception and then pass control to the corresponding exception handler, clearing the trace bit (T-bit) in the SR. If the system requires that a trace exception be processed, it is the responsibility of the TRAP exception handler to check for this condition, SR[15] in the exception stack frame, and pass control to the trace exception handler before returning from the original exception.

A trace exception does not occur immediately after the execution (MOVE to SR, RTE) of the instruction that puts the ColdFire2/2M in trace mode. The first trace exception occurs after the subsequent instruction.

4.2.7 Unimplemented Opcode Exception

The attempted execution of line A opcodes not used for MAC instructions and line F opcodes not used for the debug instructions, including $FFFF, generates their unique exception types, vector numbers $A and $B respectively. Refer to **Appendix B New MAC Instructions** for a list of the MAC opcodes and the *ColdFire Programmer's Reference Manual Rev. 1.0* for a list of the debug instruction opcodes.

4.2.8 Debug Interrupt

The debug module is the only internal interrupt source for the ColdFire2/2M. This type of program interrupt, vector number $C, is discussed in detail in **Section 7.4 Real-Time Debug Support**. This exception is generated in response to a hardware breakpoint register trigger. The ColdFire2/2M does not generate an IACK cycle, but rather calculates the vector number internally.

4.2.9 RTE & Format Error Exceptions

When an RTE instruction is executed, the ColdFire2/2M first examines the 4-bit format field in the exception stack frame on the stack to validate the frame type. Any attempted execution of an RTE when the format is not equal to $4, $5, $6, or $7 generates a format error, vector number $E. The exception stack frame for the format error is created without disturbing the original exception stack frame. The PC field in the new exception stack frame will point to the RTE instruction.

The selection of the format value provides some limited debug support for porting code from 68000 applications. On 680x0 family processors, the status register (SR) was located at the top of the stack. On those processors, bit[30] of the longword addressed by the system stack pointer (SP) is typically zero. Thus, if an RTE is attempted with a 680x0-type exception stack frame, the ColdFire2/2M will generate a format exception.

If the format field defines a valid type, the ColdFire2/2M: (1) reloads the SR operand, (2) fetches the second long word operand (PC), (3) adjusts the stack pointer by adding the format value to the auto-incremented address after the fetch of the first long word, and then (4) transfers control to the instruction address defined by the PC (fetched in step 2.)

4.2.10 TRAP Instruction Exceptions

The TRAP instruction always forces an exception as part of its execution and is useful for implementing system calls. The instruction adds the immediate operand (vector) of the instruction to 32 to obtain the vector number. The range of vector values is 0 - 15, which provides 16 vectors numbered $20 - $2F. Refer to the *ColdFire Programmers Reference Manual Rev. 1.0* (MCF5200PRM/AD) for more information on the TRAP instruction.

4.2.11 Interrupt Exception

When a peripheral device requires the services of the ColdFire2/2M or is ready to send information that the ColdFire2/2M requires, it can signal the ColdFire2/2M to take an interrupt exception. Seven levels of interrupt priorities are provided, numbered 1–7. Devices can be chained externally within interrupt priority levels, allowing an unlimited number of peripheral devices to interrupt the ColdFire2/2M. The status register contains a 3-bit mask indicating the current interrupt priority, and interrupts are inhibited for all priority levels less than or equal to the current priority (see Section 1.4.3.1 Status Register (SR).)

An interrupt request is made to the ColdFire2/2M by encoding the interrupt request levels 1–7 on the three interrupt request level (IPLB[2:0]) signals; all signals high indicate no interrupt request. Table 4-4 shows the relationship between the actual requested interrupt and the state of the IPLB[2:0] signals as well as the interrupt mask levels required for recognition of the requested level.

Table 4-4. Interrupt Levels and Mask Values

REQUESTED INTERRUPT LEVEL	CONTROL LINE STATUS			INTERRUPT MASK LEVEL REQUIRED FOR RECOGNITION
	IPLB[2]	IPLB[1]	IPLB[0]	
0	High	High	High	No Request
1	High	High	Low	0
2	High	Low	High	0-1
3	High	Low	Low	0-2
4	Low	High	High	0-3
5	Low	High	Low	0-4
6	Low	Low	High	0-5
7	Low	Low	Low	0-7

Interrupt requests arriving at the ColdFire2/2M do not force immediate exception processing, but the requests are made pending. Pending interrupts are detected between instruction executions. If the priority of the pending interrupt is lower than or equal to the current ColdFire2/2M priority, execution continues with the next instruction, and the requesting interrupt is postponed until the priority of the pending interrupt becomes greater than the current ColdFire2/2M priority.

If the priority of the pending interrupt is greater than the current ColdFire2/2M priority, the exception processing sequence for the requesting interrupt is started. A copy of the status register is saved internally; the privilege mode is set to supervisor mode; tracing is suppressed; and the ColdFire2/2M priority level is set to the level of the interrupt being acknowledged. The ColdFire2/2M fetches the vector number from the interrupting device by executing an interrupt acknowledge cycle, which displays the level number of the interrupt being acknowledged on the address bus (see Section 3.7 Interrupt Acknowledge Bus Cycles). The ColdFire2/2M then proceeds with the usual exception processing, saving the exception stack frame on the supervisor stack. The saved value of the program counter is the address of the instruction that would have been executed had the interrupt not been taken. The appropriate interrupt vector is fetched and loaded into the program counter, and normal instruction execution commences in the interrupt handling routine.

Interrupt requests should be maintained on the IPLB[2:0] signals until the conclusion of the interrupt acknowledge (IACK) cycle from the processor to guarantee that the interrupt will be recognized. The interrupt request can be maintained without being recognized again, until the interrupt priority level in the SR is lowered below the currently requested level. This is usually the result of the RTE instruction at the end of the interrupt exception handler.

Once the conclusion of an Interrupt Acknowledge (IACK) cycle occurs, another interrupt may be asserted on the IPLB lines.The processor will start fetching the interrupt handler code for the acknowledged interrupt. In the handler, the processor looks at the interrupt mask level and determines if the second interrupt is masked or allowed.If the second interrupt is not masked, the processor will start another IACK cycle for the second interrupt. It will go to the second interrupt handler and sample IPLB inputs, compare mask, and execute this handler if a higher level interrupt is not pending. Once the second interrupt has completed the return from exception (RTE) in the handler, the processor will return to the first interrupt handler, where the IPLB lines are sampled and compared to the interrupt mask. If no interrupts have higher priority, it will execute this handler and return (RTE) to the place in the code where the first interrupt was allowed to be taken.

Thus, interrupts can be nested, and higher interrupts given priority by the interrupt mask register once they are executing code within the handlers.

4.2.11.1 LEVEL 7 INTERRUPTS. Level 7 interrupts are handled differently than interrupt levels one through six. A level 7 interrupt is a nonmaskable interrupt; therefore, a 7 in the interrupt mask does not disable a level 7 interrupt.

Level 7 interrupts are edge-triggered by a transition from a lower priority request to the level 7 request, as opposed to interrupt levels one through six, which are level sensitive. Therefore, if the interrupt priority level (IPLB[2:0]) signals remain at level 7, the ColdFire2/2M will only recognize one level 7 interrupt since only one transition from a lower level request to a level 7 request occurred. For the ColdFire2/2M to recognize a level 7 interrupt followed by another level 7 interrupt, one of the two following sequences must occur:

1. The interrupt request level on the IPLB[2:0] signals changes from a lower request level to level 7 and remains at level 7 until the interrupt acknowledge bus cycle begins. Later, the interrupt request level returns to a lower interrupt request level and then back to level 7, causing a second transition on the IPLB[2:0] signals.

2. The interrupt request level on the IPLB[2:0] signals changes from a lower request level to level 7 and remains at level 7. If the interrupt handling routine for the level 7 interrupt lowers the interrupt mask level, a second level 7 interrupt will be recognized even though no transition has occurred on the interrupt control pins. After the level 7 interrupt handling routine completes, the ColdFire2/2M will compare the interrupt mask level to the interrupt request level on the IPLB[2:0] signals. Since the interrupt mask level will be lower than the requested level, the interrupt mask will be set back to level 7. The level 7 request on the IPLB[2:0] signals must be held until the second interrupt acknowledge bus cycle has begun to ensure that the interrupt is recognized.

4.2.11.2 SPURIOUS, AUTOVECTORED, AND UNINITIALIZED INTERRUPTS. If When the external logic indicates an access error during the interrupt acknowledge cycle, the

Freescale Semiconductor, Inc.

interrupt is considered spurious (refer to Section 3.7.2 Spurious Interrupt Acknowledge Bus Cycle.) It is the responsibility of the external logic to return the spurious interrupt vector number, $18. External hardware may also be designed to generate autovector interrupts, $64 - $7C, for each interrupt level. Many M68000 family peripherals use programmable interrupt vector numbers as part of the interrupt-acknowledge operation for the system. If this vector number is not initialized after reset and the peripheral must acknowledge an interrupt request, the peripheral usually returns the vector number for the uninitialized interrupt vector, $F.

SECTION 5
INTEGRATED MEMORIES

The ColdFire2/2M has dedicated buses to support three integrated memories: instruction cache, RAM, and ROM.

5.1 INSTRUCTION CACHE

The ColdFire2/2M has a dedicated bus to support an integrated instruction cache with the following features:

- 0 - 32 Kbyte direct-mapped instruction cache
- Instruction cache byte size programmed with ICH_SZ[2:0] static signals
- Single-cycle access on cache hits
- Physically located on processor's high-speed local bus
- Non-blocking design to maximize performance
- Configurable cache-miss fetch algorithm

The cache module services instruction-fetch requests from the ColdFire2/2M by either returning matching 32-bit cache entries in a single clock, or by initiating memory requests to service accesses that miss in the cache. The instruction cache size is specified via the ICH_SZ[2:0] pins which are static signals that need to stay valid for all operation. Only the ColdFire2/2M can access the cache. A fetch is defined as a read from user or supervisor code space only.

5.1.1 Instruction Cache Hardware Organization

The instruction cache is an optional direct-mapped single-cycle memory, organized as 32 (512 byte) to 2K (32 Kbyte) lines, each containing 4 longwords or 16 bytes. The data array is organized in longwords, 4 bytes per entry. The tag array is organized in lines, one entry per four longwords or line. The cache size is determined by the encoding of the ICH_SZ[2:0] inputs as shown in Table 5-1. Thus the memory storage consists of a N-entry tag array (where N corresponds to the number of lines in the data array) containing addresses and a valid bit, and the data array containing M bytes of instruction data (where M= 512,1K, 2K, 4K, 8K, 16K, or 32K), organized as M/4 x 32 bits.

The two memory arrays are accessed in parallel: bits [X:4] of the instruction fetch address providing the index into the tag array, and bits [X:2] addressing the data array (where X ranges from 8 to 14 for 512-byte to 8K-byte I-cache size, see Table 5-1).The tag array outputs the address mapped to the given cache location along with the valid bit for the line. This address field is compared to bits [31:Y] of the instruction fetch address (where Y = X +

ColdFire2/2M User's Manual

1 for a given cache size) from the local bus to determine if a cache hit in the memory array has occurred. If the desired address is mapped into the cache memory, the output of the data array is driven onto the processor's local data bus completing the access in a single cycle.

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

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

If the referenced address is not contained in the memory array nor the line fill buffer, the instruction cache initiates the required external fetch operation. In most situations, this is a 16-byte line-sized burst reference. An external bus cycle is always started simultaneously with the fetch cycle to the instruction cache. If a "hit" occurs in the instruction cache, the MKILLB signal is asserted late in the cycle to "kill" the external bus cycle. Thus an asserted MTSB and MKILLB direct the external bus controller to ignore the MTSB. If no "hit" occurred in the instruction cache (or other K-Bus memory), the M-Bus cycle uses the normal number of clock cycles beginning with the MTSB issued for the instruction fetch. See **Section 5.5, Interactions Between K-Bus Memories** for more details.

The hardware implementation is a non-blocking design, meaning the processor's local bus is released after the initial access of a miss. Thus, the cache, RAM, or ROM module can service subsequent requests while the remainder of the line is being fetched and loaded into the fill buffer.

5.1.2 Instruction Cache Operation

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

5.1.3 Instruction Cache Signal Description

The following signals interface the ColdFire2/2M to an integrated instruction cache. The cache is comprised of two compiled RAMs: tag and data. Figure 5-1 illustrates an 8 Kbyte configuration. All ColdFire2/2M signals are unidirectional and synchronous. Instruction cache outputs are registered and the Instruction cache data is latched into the ColdFire2/2M on the falling edge of the clock.
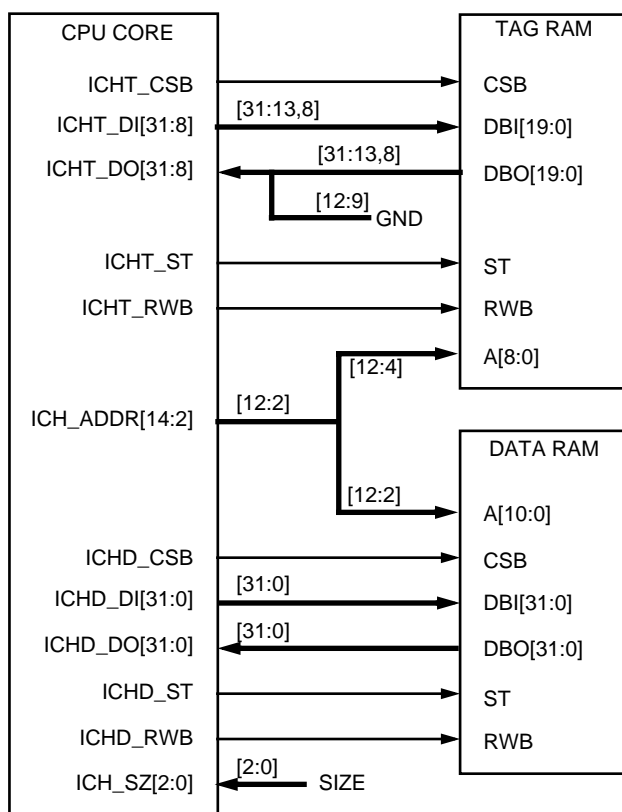
Figure 5-1. Example of 8 Kbyte Instruction Cache Interface Diagram

5.1.3.1 INSTRUCTION CACHE ADDRESS BUS (ICH_ADDR[14:2]). These registered output signals provide the address of the current bus cycle (i.e. fetch cycle) to the integrated cache RAMs. ICH_ADDR is only updated on fetch cycles (i.e. ICH_ADDR does not get updated on RAM or ROM hits). This bus should be connected to the address bus (A) of the two compiled cache RAMs. ICH_ADDR[N:M] is being provided via MADDR[N:M].

5.1.3.2 INSTRUCTION CACHE DATA CHIP-SELECT (ICHD_CSB). This active-low, output signal indicates the cache data RAM is currently selected to perform a data transfer with the ColdFire2/2M. This bus should be connected to the chip-select (CSB) signal of the compiled cache data RAM.

5.1.3.3 INSTRUCTION CACHE DATA INPUT BUS (ICHD_DI[31:0]). These output signals provide the write data path between the ColdFire2/2M and the cache data RAM. The data bus is 32-bits wide and should be connected to the data inputs (DBI) of the compiled cache data RAM.

5.1.3.4 INSTRUCTION CACHE DATA OUTPUT BUS (ICHD_DO[31:0]). These input signals provide the read data path between the cache data RAM and the ColdFire2/2M. The data bus is 32-bits wide and should be connected to the data outputs (DBO) of the compiled cache data RAM.

5.1.3.5 INSTRUCTION CACHE DATA STROBE (ICHD_ST). This output signal initiates a read or write cycle to the cache data RAM on a low-to-high transition. This signal should be connected to the strobe input (ST) signal of the compiled cache data RAM.

5.1.3.6 INSTRUCTION CACHE DATA READ/WRITE (ICHD_RWB). This output signal indicates the direction of the data transfer to the cache data RAM. A high level indicates a read cycle and a low level indicates a write cycle. It should be connected to the read/write (RWB) signal of the compiled cache data RAM.

5.1.3.7 INSTRUCTION CACHE SIZE (ICH_SZ[2:0]). These static inputs specify the size of the compiled cache RAMs connected to the ColdFire2/2M. Table 5-1 lists the possible cache configurations. ICH_SZ[2:0] does not affect the CACR in any way; thus a MOVEC instruction will write the CACR regardless of the ICH_SZ specification (which is contrary to the RAM_SZ and ROM_SZ effect during RAMBAR and ROMBAR loading).

Table 5-1. Cache Configuration Encoding

ICACHE SZ (BYTES)	ICH_SZ[2:0]	TAG ARRAY SIZE	TAG ARRAY ADDRESS	DATA ARRAY SIZE	DATA ARRAY ADDRESS
None	000	-	-	-	-
512	001	32x24	ICH_ADDR[8:4]	128x32	ICH_ADDR[8:2]
1 K	010	64x23	ICH_ADDR[9:4]	256x32	ICH_ADDR[9:2]
2 K	011	128x22	ICH_ADDR[10:4]	512x32	ICH_ADDR[10:2]
4 K	100	256x21	ICH_ADDR[11:4]	1Kx32	ICH_ADDR[11:2]
8 K	101	512x20	ICH_ADDR[12:4]	2Kx32	ICH_ADDR[12:2]
16K†	110	1Kx19	ICH_ADDR[13:4]	4Kx32	ICH_ADDR[13:2]
32K†	111	2Kx18	ICH_ADDR[14:4]	8Kx32	ICH_ADDR[14:2]

NOTE: †HPF65 ColdFire2 Hard Macro may require a reduced operating frequency for 16K and 32K sized ICACHE.

5.1.3.8 INSTRUCTION CACHE TAG CHIP-SELECT (ICHT_CSB). This active-low output signal indicates the cache tag RAM is currently selected to perform a data transfer with the ColdFire2/2M. This signal should be connected to the chip-select (CSB) signal of the compiled cache tag RAM.

5.1.3.9 INSTRUCTION CACHE TAG INPUT BUS (ICHT_DI[31:8]). These output signals provide the write data path between the ColdFire2/2M and the cache tag RAM. The data bus depends upon the size of the CACHE; see Table 5-1. Bit eight is always the valid bit and is always used as seen in the cache configuration shown in Table 5-2. This bus should be connected to the data inputs (DBI) of the compiled cache tag RAM. Functionally, MADDR[31:9] is written onto ICHT_DI[31:9] and ICHT_DI[8] is written with the valid state of the entry.

Table 5-2. Valid Tag RAM Data Signals

CACHE SIZE (BYTES)	VALID DATA BITS	CPU DATA PLACED ON VALID DATA BITS
512	ICHT_Dx[31:8]	{MADDR[31:9], VALID}
1 K	ICHT_Dx[31:10,8]	MADDR[31:10], VALID}
2 K	ICHT_Dx[31:11,8]	MADDR[31:11], VALID}
4 K	ICHT_Dx[31:12,8]	MADDR[31:12], VALID}
8 K	ICHT_Dx[31:13,8]	MADDR[31:13], VALID}
16K	ICHT_Dx[31:14,8]	MADDR[31:14], VALID}
32K	ICHT_Dx[31:15,8]	MADDR[31:15], VALID}

5.1.3.10 INSTRUCTION CACHE TAG OUTPUT BUS (ICHT_DO[31:8]). These input signals provide the read data path between the cache tag RAM and the ColdFire2/2M. The data bus size depends upon the size of the CACHE; see Table 5-1. Bit eight is always the valid bit and is always used regardless of the cache configuration as shown in Table 5-2. This bus should be connected to the data outputs (DBO) of the compiled cache tag RAM. Unused signals must be tied low.

5.1.3.11 INSTRUCTION CACHE TAG STROBE (ICHT_ST). This output signal initiates a read or write cycle to the cache tag RAM on a low-to-high transition. This signal should be connected to the strobe input (ST) signal of the compiled cache tag RAM.

5.1.3.12 INSTRUCTION CACHE TAG READ/WRITE (ICHT_RWB). This output signal indicates the direction of the data transfer to the cache tag RAM. A high level indicates a read cycle and a low level indicates a write cycle. It should be connected to the read/write (RWB) signal of the compiled cache tag RAM.

5.1.4 Interaction With Other Modules

Since the instruction cache, high-speed ROM and RAM modules are connected to the processor's local data bus, certain user-defined configurations may result in simultaneous instruction fetch processing. If the referenced address is mapped into the ROM or RAM module, that module will service the request in a single cycle. In this case, data accessed from the instruction cache is simply discarded, and no external memory references are generated. The RAM module has higher priority over the ROM module. If the address is not mapped into the RAM or ROM space, then the request is handled by the instruction cache in the normal fashion.

5.1.5 Memory Reference Attributes

For every memory reference generated by the processor or the debug module, a set of "effective attributes" is determined based on the address and the Access Control Registers (ACR0, ACR1). This set of attributes includes the cacheable/non-cacheable definition, the precise/imprecise handling of operand writes and the write-protect capability.

In particular, each address is compared to the values programmed in the Access Control Registers. If the address matches one of the ACR values, the access attributes from that

Freescale Semiconductor, Inc.

ACR are applied to the reference. If the address does not match either ACR, then the default value defined in the Cache Control Register (CACR) is used. The specific algorithm is:

> if (address = ACR0_address including mask)
>> Effective Attributes = ACR0 attributes
> else if (address = ACR1_address including mask)
>> Effective Attributes = ACR1 attributes
>> else Effective Attributes = CACR default attributes

5.1.6 Cache Coherency and Invalidation

The instruction cache does not monitor processor data references for accesses to cached instructions. Therefore it is necessary for software to maintain cache coherence by invalidating the appropriate cache entries after modifying code segments.

The cache invalidation can be performed in two ways:

1. The assertion of bit 24 in the Cache Control Register, via a CPU space write, forces the entire instruction cache to be marked as invalid. The invalidation operation requires N (N = # of lines) cycles since the cache sequences through the entire tag array clearing a single location (the valid location) each cycle. Any subsequent instruction fetch accesses are postponed until the invalidation sequence is complete; i.e. the processor can continue running as long as it does not try and fetch from the instruction cache. If the instruction cache is accessed, processing halts and waits for the invalidation to complete. The CACR can be loaded to not turn on the instruction cache to let processing continue unblocked.

2. The privileged CPUSHL instruction can be used to invalidate a single cache line. When this instruction is executed, the cache entry defined by bits [X:4] of the source address register in invalidated, provided bit 28 of the CACR is cleared.

These invalidation operations may be initiated from the processor or the debug module.

5.1.7 Reset

A hardware reset clears the CACR disabling the instruction cache. The contents of the tag array are not affected by the reset. Accordingly, the system start-up code must explicitly perform a cache invalidation by setting CACR[24] before the cache may be enabled.

5.1.8 Line Fill Buffer and Cache Miss Fetch Algorithm

As discussed in Section 5.1.1, the instruction cache hardware includes a 16-byte line fill buffer for providing temporary storage for the last fetched instruction.

With the cache enabled as defined by CACR[31], a cacheable instruction fetch that misses in both the tag memory and the line-fill buffer generates an external fetch. The size of the external fetch is determined by the value contained in the 2-bit CLNF field of the CACR, and the miss address. Table 5-3 shows the relationship between the CLNF bits, the miss address and the size of the external fetch:

Table 5-3. Initial Fetch Size Based on Miss Address & CLNF

CLNF[1:0]	MISS ADDRESS [3:2] (LONGWORD ADDRESS BITS)			
	00	01	10	11
00	Line	Line	Line	Longword
01	Line	Line	Longword	Longword
1X	Line	Line	Line	Line

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

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

> if miss address[3:2] = 00
>> fetch sequence = {$0, $4, $8, $C}
> if miss address[3:2] = 01
>> fetch sequence = {$4, $8, $C, $0}
> if miss address[3:2] = 10
>> fetch sequence = {$8, $C, $0, $4}
> if miss address[3:2] = 11
>> fetch sequence = {$C, $0, $4, $8}

Once an external fetch has been initiated and the data loaded into the line-fill buffer, the instruction cache maintains a special most-recently-used indicator which tracks the contents of the fill buffer versus its corresponding cache location. At the time of the miss, the hardware indicator is set, marking the fill buffer as "most recently used". If a subsequent access to the cache location defined by bits [X:4] of the fill buffer address occurs, the data in the cache memory array is now most-recently-used, so the hardware indicator is cleared. In all cases, the indicator defines whether the contents of the line-fill buffer or the memory data array are most-recently-used. At the time of the next cache miss, the contents of the line-fill buffer are written into the memory array if the entire line is present, and the fill buffer data is still most-recently-used compared to the memory array. Only a complete line can be transferred to the icache array (the array only has one valid bit while the line-fill buffer has a valid bit per longword). This transfer can only occur if the icache is not locked or frozen. This write takes four cycles with ICH_ADDR[3:2] incrementing each cycle. Generally, when a fetch misses the cache, the previously fetched instruction line in the line-fill buffer is written to the cache while the master bus is running a fetch cycle.

The fill buffer can also be utilized as temporary storage for line-sized bursts of non-cacheable references under control of CACR [10]. With this bit set, a non-cacheable instruction fetch is processed as defined by Table 5-3. For this condition, the fill buffer is loaded and subsequent references can "hit" in the buffer, but the data is never loaded into the memory array.

Freescale Semiconductor, Inc.

Line fills begin with the longword containing the requested instruction and wrap around, as needed, to complete the full 16-byte line request. Noncachable accesses can still result in line requests to the master bus because they are buffered in the cache line-fill buffer, but they are not copied into the cache. Requests which result in a longword fetch will not be written to the cache.

The relationship between CACR bits 31 and 10, and the type of instruction fetch is shown below.

Table 5-4. Instruction Cache Operation as Defined by CACR[31, 10]

CACR [31]	CACR [10]	TYPE OF INST FETCH	DESCRIPTION
0	0	-	Instruction Cache is completely disabled; All fetches are word, longword in size.
0	1	-	All fetches are treated as non-cacheable and loaded into the line-fill buffer as defined by Table 5-3.
1	-	Cacheable	Fetch size is defined by Table 5-3, and contents of the line fill buffer can be written into the memory array.
1	0	Non-cacheable	All fetches are longword in size, and not loaded into the line-fill buffer.
1	1	Non-cacheable	Fetch size is defined by Table 5-3, and loaded into the line-fill buffer, but are never written into the memory array.

5.1.9 Instruction Cache Programming Model

The operation of the instruction cache and local bus controller are defined by three supervisor registers: the Cache Control Register (CACR) and two Access Control Registers (ACR0, ACR1). All three registers may be written from the processor using the privileged MOVEC instruction. Additionally, the registers may be accessed from the debug module. From the debug module, the registers may be read or written. In all cases, undefined bits in a register are reserved. These bits should be written as zeroes, and return zeroes when read from the debug module.

5.1.10 Cacheability

Cacheability of instruction accesses is controlled by either the access control registers ACR0 and ACR1, for accesses matching the address ranges defined by these registers, or by the default cache mode bits in the CACR for all other accesses. Only instruction fetches are cached (i.e. code space accesses.)

5.1.11 Cache Control Register (CACR)

The operation of the instruction cache is controlled by the Cache Control Register (CACR). The CACR also provides a set of default memory access attributes used when a reference address does not map into the space defined by the Access Control Registers.

The CACR is accessed as control register $002 using the privileged MOVEC instruction. This instruction provides write-only access to this register from the processor. Additionally,

the CACR register may be accessed from the debug module in a similar manner. The entire register is cleared by a hardware reset.

BITS	31	30 29	28	27	26 25	24	23 18	17 16
FIELD	CENB	-	CPSH	CFRZ	-	CINV	-	PARK
RESET	0	-	0	0	-	0	-	0
R/W	W	-	W	W	-	W	-	W

BITS	15 11	10	9	8	7 6	5	4 2	1 0
FIELD	-	CBEN	DCM	DBWE	-	DWP	-	CLNF
RESET	-	0	0	0	-	0	-	0
R/W	-	W	W	W	-	W	-	W

Cache Control Register (CACR)

Field Definitions:

5.1.11.1 CENB: CACR[31]—CACHE ENABLE.

    0 = Cache disabled
    1 = Cache enabled

The memory array of the instruction cache is enabled only if CENB is asserted. If ICH_SZ[2:0] is set to zero, this bit is set to zero and the ICACHE module is disabled. Any attempt to set the CENB bit is disabled when ICH_SZ[2:0] is set to zero.

5.1.11.2 CPSH: CACR[28] - CPUSH DISABLE INVALIDATE.

    0 = Enable Invalidation
    1 = Disable Invalidation

When the privileged CPUSHL instruction is executed, the cache entry defined by bits [X:4] of the address is invalidated if CPSH = 0. If CPSH = 1, no operation is performed.

5.1.11.3 CFRZ: CACR[27]—CACHE FREEZE.

    0 = Normal operation
    1 = Freeze valid cache lines

Setting this bit effectively freezes the contents of the cache. Line fetches may still be initiated and loaded into the line fill buffer, but a valid cache entry is never overwritten. If a given cache location is invalid, the contents of the line-fill buffer may be written into the memory array while CFRZ is asserted.

5.1.11.4 CINV: CACR[24]—CACHE INVALIDATE.

    0 = No operation
    1 = Invalidate all cache lines

Setting this bit forces the cache to invalidate each tag array entry. The invalidation process requires N (N = #lines) machine cycles, with a single cache entry cleared per machine cycle. The state of this bit is always read as a zero. After a hardware reset, the cache must be invalidated before it is enabled.

5.1.11.5 PARK: CACR[17:16]—OPTIONAL EXTERNAL ARBITER CONTROL.

This field can be used to drive an external master arbiter control module via the MARBC[1:0] signals. Otherwise, these bits can use used as general purpose output bits. Refer to **Section 3.10 Master Bus Arbitration**. If an external master arbiter module is not used, the MARBC[1:0] signals may be used as general purpose control signals.

5.1.11.6 CBEN:CACR[10]—CACHE ENABLE NON-CACHEABLE INSTRUCTION BURSTING.

    0 = Disable burst fetches on non-cacheable accesses
    1 = Enable burst fetches on non-cacheable accesses

Setting this bit allows the line-fill buffer to be loaded with burst transfers under control of CLNF[1:0] for non-cacheable accesses. Non-cacheable accesses are never written into the memory array. CBEN in conjunction with CACR[31], CENB, determines the line buffer and ICACHE array status. See Table 5-5 for guidance in setting CACR[10].

Table 5-5. CACR[31] and CACR[10] CONFIGURATION

CACR[31]	CACR[10]	ICACHE/LINE FILL BUFFER CONFIGURATION
0	0	ICACHE disabled, LINE FILL BUFFER disabled
0	1	ICACHE disabled, LINE FILL BUFFER enabled
1	0	ICACHE enabled, LINE FILL BUFFER enabled on cachable accesses
1	1	ICACHE enabled, LINE FILL BUFFER enabled, but get the line-fill buffer even on non-cacheable accesses

5.1.11.7 DCM: CACR[9]—DEFAULT INSTRUCTION FETCH CACHE MODE.

    0 = Caching enabled
    1 = Caching disabled

This bit defines the default cache mode: 0 is cacheable, 1 is non-cacheable. For more information on the selection of the effective memory attributes, see Section 5.1.5.

5.1.11.8 DBWE:CACR[8]—DEFAULT BUFFERED WRITE ENABLE.

    0 = Disable buffered writes
    1 = Enable buffered writes

ColdFire2/2M User's Manual

This bit defines the default value for enabling buffered writes. If DBWE = 0, the termination of an operand write cycle on the processor's local bus is delayed until the external bus cycle is completed. If DBWE = 1, the write cycle on the local bus in terminated immediately, and the operation buffered in the bus controller. In this mode, operand write cycles are effectively decoupled between the processor's local bus and the external bus.

Generally, the enabling of buffered writes provides higher system performance, but recovery from access errors may be more difficult. For the ColdFire CPU, the reporting of access errors on operand writes is always imprecise, and enabling buffered writes simply decouples the write instruction from the signaling of the fault even more.

5.1.11.9 DWP: CACR[5]—DEFAULT WRITE PROTECTION.

    0 = Read and write access permitted
    1 = Only read access permitted

The DWP bit defines the default write-protection attribute. If the effective memory attributes for a given access select the DWP bit, then any attempted write with this bit set is terminated with an access error.

5.1.11.10 CLNF:CACR[1:0]—CACHE LINE FILL.

This two-bit field determines the size of the instruction fetch memory transfer based on various CACR bits. See Section 5.1.8 for additional information.

The encoding is shown in Table 5-6.

Table 5-6. External Fetch Size Based on Miss Address & CLNF

CLNF[1:0]	MISS ADDRESS[3:2]			
	00	01	10	11
00	Line	Line	Line	Longword
01	Line	Line	Longword	Longword
1X	Line	Line	Line	Line

5.2 ACCESS CONTROL REGISTERS (ACR0, ACR1)

The two access Control Registers (ACR0, ACR1) provide a definition of memory reference attributes for two memory regions (one per ACR). This set of effective attributes is defined for every memory reference using the ACRs or the set of default attributes contained in the CACR. The ACRs are examined for every memory reference that is NOT mapped to the RAM or ROM module.

The ACRs are accessed as control registers $004 and $005 using the privileged MOVEC instruction (ACR0 = $004, ACR1 = $005). This instruction provides write-only access to these registers from the processor. Additionally, the ACRs may be accessed from the debug module in a similar manner. Each ACR is disabled by a hardware reset.

5.2.1 ACR Programming Model

The ACRs are accessible in supervisor mode using the MOVEC instruction to access control register $004 for ACR0 and $005 for ACR1. All undefined bits are reserved and should always be written as zero. A hardware reset clears all bits in the ACRs.

BITS	31	24	23	16
FIELD	AB		AM	
RESET	0		0	
R/W	W		W	

BITS	15	14	13	12	8	7	6	5	4	3	2	1	0
FIELD	EN	SM		-		ENIB	CM	BUFW	-		WP	-	
RESET	0	0		-		0	0	0	-		0	-	
R/W	W	W		-		W	W	W	-		W	-	

Access Control Register (ACR0, ACR1)

Field Definitions:

5.2.1.1 AB: ACR[31:24]—ADDRESS BASE [31:24].

This 8-bit field is compared to address bits [31:24] from the processor's local bus, under control of the ACR address mask. If the address matches, the attributes for the memory reference are sourced from the given ACR.

5.2.1.2 AM: ACR [23:16]—ADDRESS MASK [23:16].

This 8-bit field masks comparison of the access address with the ACR address base bits. Setting an AM bit ignores the comparison of the corresponding address base bits.

5.2.1.3 EN: ACR [15]—ENABLE .

    0 = ACR disabled
    1 = ACR enabled

The EN bit defines the ACR enable. This bit is cleared by hardware reset, disabling the ACR.

5.2.1.4 SM: ACR [14:13]—SUPERVISOR MODE .

    00 = Match if user mode
    01 = Match if supervisor mode
    1x = Match always; ignore user/supervisor mode

This two-bit field allows the given ACR to be applied to references based on operating privilege mode of the ColdFire processor. The field allows use of the ACR for user-references only, supervisor-references only, or all accesses.

5.2.1.5 CM: ACR [6]—CACHE MODE.

> 0 = Caching enabled
> 1 = Caching disabled

This bit defines the cache mode: 0 is cacheable, 1 is non-cacheable.

5.2.1.6 BWE: ACR [5]—BUFFERED WRITES.

> 0 = Disable buffered writes
> 1 = Enable buffered writes

This bit defines the value for enabling buffered writes. If BWE = 0, the termination of an operand write cycle on the processor's local bus is delayed until the external bus cycle is completed. If BWE = 1, the write cycle on the local bus is terminated immediately, and the operation buffered in the bus controller. In this mode, operand write cycles are effectively decoupled between the processor's local bus and the external bus.

Generally, the enabling of buffered writes provides higher system performance, but recovery from access errors may be more difficult. For the ColdFire CPU, the reporting of access errors on operand writes is always imprecise, and enabling buffered writes simply decouples the write instruction from the signaling of the fault even more.

5.2.1.7 WP: ACR [2]—WRITE PROTECT.

> 0 = Read and write accesses permitted
> 1 = Only read accesses permitted

The WP bit defines the write-protection attribute. If the effective memory attributes for a given access select the WP bit, then any attempted write with this bit set is terminated with an access error.

5.3 ROM MODULE

The ColdFire2/2M has a dedicated bus to support an integrated ROM with the following features:

- 0 - 32 Kbytes ROM, organized by ROM size/4 x 32 bits
- ROM byte-size programmed with ROM_SZ[2:0] static signals
- Single-cycle access
- Physically located on processor's high-speed bus
- Byte, word, longword addressable
- Memory-mapping defined by user
- Configurable at reset as boot memory

The ROM module provides a general-purpose memory block that the ColdFire2/2M can access in a single clock, and can store either code or data structures. The ROM can be programmed to function as a boot memory; this configurable option is available at reset time.

The ROM size is specified via the ROM_SZ[2:0] pins which are static signals that need to stay valid for all operation. Only the ColdFire2/2M can access the ROM module.

The ColdFire2/2M issues the instruction fetches in parallel to all K-Bus memories. Fetches from the ROM module are not cached because they require only one clock cycle to complete. Two ROMs are used to lower power consumption by only driving the required portions of the data bus.

5.3.1 ROM Signal Description

These signals interface the ColdFire2/2M to two integrated ROMs. Figure 5-2 illustrates an 8 Kbyte configuration. All ColdFire2/2M signals are unidirectional and synchronous.
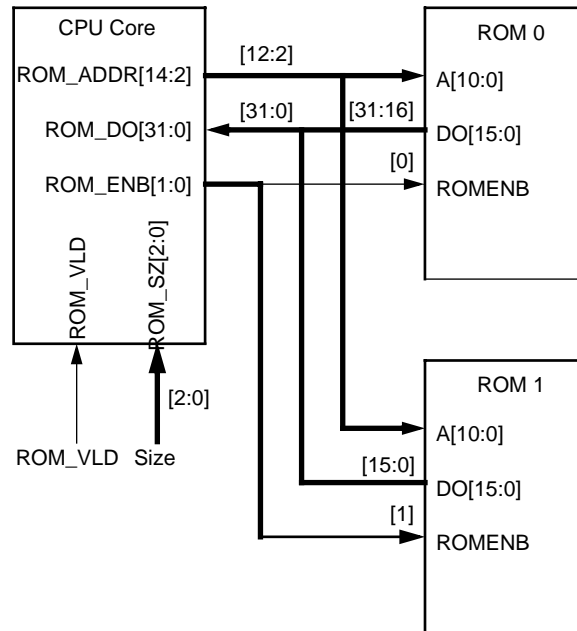
Figure 5-2. Example 8 Kbyte ROM Interface Diagram

5.3.1.1 ROM ADDRESS BUS (ROM_ADDR[14:2]). These output signals provide the address of the current bus cycle to the integrated ROMs. This bus should be connected to the address bus (A) of the compiled ROMs. The number of valid address signals depends on the total ROM size as shown in Table 5-7.

Table 5-7. Valid ROM Address Bits

TOTAL ROM SIZE	VALID ROM_ADDR BITS
0	None
512	ROM_ADDR[8:2]
1 K	ROM_ADDR[9:2]
2 K	ROM_ADDR[10:2]
4 K	ROM_ADDR[11:2]
8 K	ROM_ADDR[12:2]
16K	ROM_ADDR[13:2]
32K	ROM_ADDR[14:2]

5.3.1.2 ROM DATA OUTPUT BUS (ROM_DO[31:0]). These input signals provide the read data path from the integrated ROMs to the ColdFire2/2M. The data bus is 32-bits wide and can transfer 8, 16, or 32 bits of data per bus transfer. During a line transfer, the data lines are time-multiplexed across multiple cycles to carry 128 bits. This bus should be connected to the data outputs (DO) of the compiled ROMs.

5.3.1.3 ROM ENABLE (ROM_ENB[1:0]). These active-low, output signals indicate the ROMs are currently selected to drive the ROM_DO[31:0] bus. These signals should be connected individually to the enable signal (ROMENB) signal of the compiled ROMs. Both are asserted for 32-bit accesses. ROM_ENB[0] connects to the MSW while ROM_ENB[1] connects to the LSW.

5.3.1.4 ROM SIZE (ROM_SZ[2:0]). These static inputs specify the size of the compiled ROMs connected to the ColdFire2/2M. These pins need to stay valid during all operation. Table 5-8 lists the possible ROM configurations. If the ROM_SZ pins are zero, the ROM cannot be enabled via a CPU space write to ROMBAR. Therefore, if the ROM is enabled while the ROM_SZ pins are at zero, the processor behaves as if no ROM module existed.

Table 5-8. ROM Configuration Encoding

TOTAL ROM SIZE (BYTES)	ROM_SZ[2:0]	ADDRESS (BITS)	DATA[1] (BITS)
None	000	-	-
512	001	7	2 @ 16
1 K	010	8	2 @ 16
2 K	011	9	2 @ 16
4 K	100	10	2 @ 16
8 K	101	11	2 @ 16
16K[2]	110	12	2 @ 16
32K[2]	111	13	2 @ 16

NOTES: 1. 2 ROMs, each 16-bits wide

2. 16K and 32K ROMs may require a reduced operating frequency.

5.3.1.5 ROM VALID (ROM_VLD). This active-high input signal determines if the ROM module should be active immediately after a hard reset. Thus, if asserted, the first fetches ($0, $4) go to ROM instead of external memory. ROM_VLD controls the reset value of the

ROM base address register (i.e. the ROM must be based at $0000 if ROM_VLD is asserted).

5.3.2 ROM Programming Model

The configuration information in the ROM base address register (ROMBAR) controls the ROM module operation. The ROMBAR is accessible in supervisor mode as control register $C00 using the MOVEC instruction. All undefined bits are reserved and should always be written as zero. A hardware reset clears only the valid bit if ROM_VLD is negated. If ROM_VLD is asserted, ROMBAR is reset to $0121. This activates the ROM with a starting address of $0000 and all address spaces, except the CPU space, are allowed. This is shown below where the reset values in parenthesis are valid if ROM_VLD is asserted during a reset.

BITS	31							16
FIELD	BA							
RESET	-(0)							
R/W	W							

BITS	15	9	8	7	6	5			1	0
FIELD	BA		WP	-		AS				V
RESET	-(0)		-(1)	-		-				0(1)
R/W	W		W	-		W				W

NOTE: The reset values in parenthesis are valid if the ROM_VLD signal is asserted during reset.

ROM Base Address Register (ROMBAR)

Field Definitions:

5.3.2.1 BA: ROMBAR[31:9]—BASE ADDRESS.

Defines the base address for the ROM module address range. The number of valid base address bits in this field is a function of the ROM size as shown in Table 5-9. The base address is reset to $0 if ROM_VLD is asserted during reset.

Table 5-9. Valid ROM Base Address Bits

ROM SIZE	VALID BA BITS
512	BA[31:9]
1 K	BA[31:10]
2 K	BA[31:11]
4 K	BA[31:12]
8 K	BA[31:13]
16K	BA[31:14]
32K	BA[31:15]

5.3.2.2 WP: ROMBAR[8]—WRITE PROTECT.

This bit is reserved for future use. This bit can be used for debug purposes; i.e. it could set a trip point when a write to ROM occurred.

   0 = No effect
   1 = An attempted write access will generate an access error exception in the processor.

5.3.2.3 AS ROMBAR[5:1]—ADDRESS SPACE MASKS.

This five-bit field, specified by ROMBAR[5:1], allows certain types of accesses to be "masked" or inhibited, from accessing the RAM module. The mask bits are defined as:

   AS5 - Mask CPU Space and Interrupt Acknowledge Accesses
   AS4 - Mask Supervisor Code Accesses
   AS3 - Mask Supervisor Data Accesses
   AS2 - Mask User Code Accesses
   AS1 - Mask User Data Accesses

If a given mask bit is set, then references of that type are NOT allowed to access the ROM module.

If ASn = 0, then accesses of the given type are allowed by the ROM.

If ASn = 1, then accesses of the given type are not allowed by the ROM. If an access is made to a space that is masked, it simply becomes mapped to the next valid space.

5.3.2.4 V: ROMBAR [0]—VALID.

The valid bit is specified by ROMBAR[0]. This bit is cleared by a hardware reset. When set, this bit enables the ROM module, otherwise the module is disabled.

If ROM_SZ is set to zero, this bit is set to zero and the ROM module is disabled. Any attempt to set the valid bit is disabled when ROM_SZ is set to zero.

5.3.3 ROM INITIALIZATION / ROM BOOT

After a hardware reset, the contents of the ROMBAR depend upon the ROM_VLD signal polarity and the ROM_SZ[2:0] configuration. If ROM_SZ[2:0] = 0, the valid bit is held cleared, even if a write is attempted during a load to ROMBAR.

ROM_VLD determines if the ROM module should be active immediately after a hard reset. Thus, if ROM_VLD is asserted, the first fetches ($0, $4) go to ROM instead of external memory. Note, if ROM_VLD is asserted, ROM_SZ cannot be set to zero. ROM_VLD controls the reset value of the ROM base address register (i.e. the ROM must be based at $0000 if ROM_VLD is asserted).

To map the ROMBAR, a load to the ROMBAR mapping the ROM module to the desired location within the address space, must be performed.

5.3.4 Power Management

As noted previously, depending upon the configuration defined by the ROMBAR, instruction fetch accesses may be sent to the ROM module and the I-Cache simultaneously. If the access is mapped to the ROM module, it sources the read data and the I-Cache access is discarded. If the ROM is used only for data operands, power dissipation can be lowered by asserting the ASn bits associated with instruction fetches. Additionally, if the ROM contains only instructions, power dissipation can be reduced by masking operand accesses.

Consider the following examples of typical ROMBAR settings:

Data contained in ROM	ROMBAR[7:0]
Only code	$2B
Only data	$35
Both code and data	$21

5.4 RAM MODULE

The ColdFire2/2M has a dedicated bus to support an integrated RAM with the following features:

- 0 - 32 Kbytes RAM, organized by RAM byte size/4 x 32 bits
- RAM byte-size programmed with RAM_SZ[2:0] static signals
- Single-cycle access
- Physically located on processor's high-speed local bus
- Byte, word, longword address capability
- Memory mapping defined by user

5.4.1 RAM Operation

The RAM module provides a general-purpose memory block that the ColdFire2/2M can access in a single cycle. The RAM size is specified via the RAM_SZ[2:0] pins which are static signals that need to stay valid for all operation. The location of the memory block may be specified to any 0-modulo-[ICACHE byte size] address within the four gigabyte address space. The memory is ideal for storing critical code or data structures, or for use as the system stack. Since the RAM module physically is connected to the processor's high-speed local bus, it can service CPU-initiated accesses, or memory referencing commands from the debug module.

Depending on configuration information, instruction fetches can be sent to both the instruction cache and the RAM block simultaneously. If the instruction fetch address is mapped into the region defined by the RAM, the RAM provides data back to the processor, and the I-cache data is discarded. Accesses from the RAM module are not cached.

Generally, the RAM is loaded by copying a hex image from another memory region into the RAM address space. This copy function can be performed by the processor during

initialization, or can be performed by an emulator during system debug. The emulator approach uses the BDM serial communication channel to download the hex image from a host machine into the RAM directly.

5.4.2 RAM Signal Description

These signals interface the ColdFire2/2M to the four integrated arrays that comprise the RAM. Figure 5-3 illustrates an 8 Kbyte configuration. All ColdFire2/2M signals are unidirectional and synchronous.
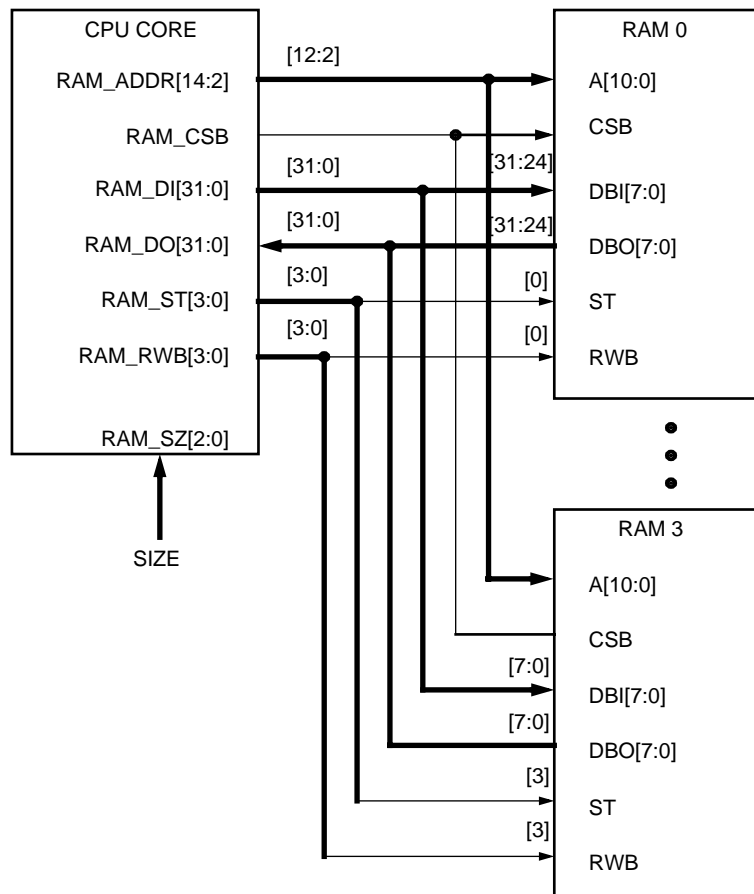
Figure 5-3. Example 8 Kbyte RAM Interface Diagram

5.4.2.1 RAM ADDRESS BUS (RAM_ADDR[14:2]).

These registered output signals provide the address of the current bus cycle to the integrated RAMs. This bus should be connected to the address bus (A) of the four compiled

Freescale Semiconductor, Inc.

RAMs. The number of valid address signals depends on the total RAM size as shown in
Table 5-10.

Table 5-10. Valid RAM Address Bits

TOTAL RAM SIZE	VALID RAM_ADDR BITS
0	None
512	RAM_ADDR[8:2]
1 K	RAM_ADDR[9:2]
2 K	RAM_ADDR[10:2]
4 K	RAM_ADDR[11:2]
8 K	RAM_ADDR[12:2]
16K	RAM_ADDR[13:2]
32K	RAM_ADDR[14:2]

5.4.2.2 RAM CHIP-SELECT (RAM_CSB). This active-low output signal indicates the
RAMs are currently selected to perform a data transfer with the ColdFire2/2M. This signal
should be connected to the chip-select (CSB) signal of the four compiled RAMs.

5.4.2.3 RAM DATA INPUT BUS (RAM_DI[31:0]). These output signals provide the write
data path between the ColdFire2/2M and the integrated RAM. The data bus is 32-bits wide
and can transfer 8, 16, or 32 bits of data per bus transfer. During a line transfer, the data
lines are time-multiplexed across multiple cycles to carry 128 bits. This bus should be
connected to the data inputs (DBI) of the four compiled RAMs. If only one byte is being
written, the byte will be replicated on all 4 lines; likewise a word will be replicated in both
word positions.

5.4.2.4 RAM DATA OUTPUT BUS (RAM_DO[31:0]). These input signals provide the read
data path between the integrated RAM and the ColdFire2/2M. The data bus is 32-bits wide
and can transfer 8, 16 or 32 bits of data per bus transfer. During a line transfer, the data lines
are time-multiplexed across multiple cycles to carry 128 bits. This bus should be connected
to the data outputs (DBO) of the four compiled RAMs.

5.4.2.5 RAM SIZE (RAM_SZ[2:0]). These static inputs specify the size of the compiled
RAMs connected to the ColdFire2/2M. These pins need to stay valid during all operation. If
the RAM_SZ pins are zero, the RAM cannot be enabled via a CPU space write to RAMBAR.
Therefore if the RAM is enabled while the RAM_SZ pins are at zero, the processor behaves
as if no RAM module existed.

Table 5-11 lists the possible RAM configurations.

Table 5-11. RAM Configuration Encoding

TOTAL RAM SIZE (BYTES)	RAM_SZ[2:0]	ADDRESS (BITS)	DATA[1] (BITS)
0	000	-	-
512	001	7	4@8
1 K	010	8	4@8
2 K	011	9	4@8
4 K	100	10	4@8
8 K	101	11	4@8
16K[2]	110	12	4@8
32K[2]	111	13	4@8

NOTES: 1.   4 RAMs, each 8-bits wide

2.   16K and 32K RAMs may require a reduced operating frequency.

5.4.2.6 RAM STROBE (RAM_ST[3:0]). These output signals initiate a read or write cycle to the integrated RAMs on a low-to-high transition. These signals should be connected individually to the strobe input (ST) signals of the four compiled RAMs. The ST[0] signal connects to the high-order byte and ST[3] connects to the low-order byte.

5.4.2.7 RAM READ/WRITE (RAM_RWB[3:0]). These output signals indicate the direction of the data transfer to the integrated RAMs. A high level indicates a read cycle and a low level indicates a write cycle. They should be connected individually to the read/write (RWB) signal of the four compiled RAMs. Like RAM_ST[3:0], the RAM_RWB[3] signal connects to the high-order byte and RAM_ST[0] connects to the low-order byte.

5.4.3 RAM Programming Model

The configuration information in the RAM base address register (RAMBAR) controls the operation of the RAM module. The RAMBAR is accessed as control register $C04 using the privileged MOVEC instruction. The MOVEC instruction provides write-only access to this register. Additionally, the RAMBAR register may be accessed from the debug module in a similar manner. From the debug module, the register may be read or written. All undefined bits are reserved. These bits should be written as zeroes, and return zeroes when read from the debug module. Only the valid bit is cleared by a hardware reset.

5.4.4 RAM Base Address Register

BITS	31							16
FIELD				BA				
RESET				-				
R/W				W				

BITS	15	9	8	7	6	5	1	0
FIELD	BA		WP	-		AS		V
RESET	-		-	-		-		0
R/W	W		W	-		W		W

RAM Base Address Register (RAMBAR)

Field Definitions:

5.4.4.1 BA: RAMBAR [31:9]—BASE ADDRESS.

Defines the base address for the RAM module address range. The number of valid base address bits in this field are a function of the RAM size as shown in . By programming this field, the RAM may be located anywhere within the four gigabyte address space of ColdFire.

Table 5-12. Valid RAM Base Address Bits

RAM BYTE SIZE	VALID BA BITS
512	BA[31:9]
1 K	BA[31:10]
2 K	BA[31:11]
4 K	BA[31:12]
8 K	BA[31:13]
16K	BA[31:14]
32K	BA[31:15]

5.4.4.2 WP: RAMBAR [8]—WRITE PROTECT.

0 = RAM module supports read and write references
1 = RAM module supports only read accesses.

The write protect field is defined by RAMBAR [8]. If set, this bit allows only read accesses to the RAM. Any attempted write access will generate an access error exception in the processor.

If cleared, the RAM supports read and write references.

5.4.4.3 AS: RAMBAR [5:1]—ADDRESS SPACE MASKS.

This five bit field, specified by RAMBAR [5:1], allows certain types of accesses to be "masked", or inhibited, from accessing the RAM module. The mask bits are defined as:

        AS5 - Mask CPU Space and Interrupt Acknowledge Accesses
        AS4 - Mask Supervisor Code Accesses
        AS3 - Mask Supervisor Data Accesses
        AS2 - Mask User Code Accesses
        AS1 - Mask User Data Accesses

If a given mask bit is set, then references of that type are NOT allowed to access the RAM module. These bits are useful for power management as detailed in section 5.4.6.

If ASn = 0, then accesses of the given type are allowed by the RAM.

If ASn = 1, then accesses of the given type are not allowed by the RAM. If an access is made to a space that is masked, it simply becomes mapped to the next valid space.

5.4.4.4 V: RAMBAR [0]—VALID.

This bit indicates when the contents of the RAMBAR are valid. The base address value is not used, and the RAM module is not accessible until the V-bit is set. An external bus cycle is generated if the base address field matches the internal core address, and the V-bit is cleared.

    0 = Contents of RAMBAR are not valid
    1 = Contents of RAMBAR are valid

The valid bit is specified by RAMBAR [0]. This bit is cleared by a hardware reset. When set, this bit enables the RAM module, otherwise the module is disabled.

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

        if (RAMBAR[0] = 1)
                if (requested address [31:9]* = RAMBAR[31:9]*
                        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

* See Table 5-10 for address bit map pertaining to RAM sizes.

If RAM_SZ is set to zero, this bit is set to zero, and the RAM module is disabled. Any attempt to set the valid bit is disabled when RAM_SZ is set to zero.

5.4.5 RAM Initialization

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

1.   Load the RAMBAR mapping the RAM module to the desired location within the address space.

2.   Read the source data and write it to the RAM. There are various instructions to support this function, including memory-to-memory move instructions, or the MOVEM opcode. The MOVEM instruction is optimized to generate line-sized burst fetches on 0-modulo-16 addresses, so this opcode generally provides maximum performance.

3.   After the data has been loaded into the RAM, it may be appropriate to load a revised value into the RAMBAR with a new set of "attributes". These attributes consist of the write-protect and address space mask fields.

These initialization functions can be performed by the ColdFire processor, or from an external emulator using the debug module.

5.4.6 Power Management

As noted previously, depending upon the configuration defined by the RAMBAR, instruction fetch accesses may be sent to the RAM module and the I-Cache simultaneously. If the access is mapped to the RAM module, it sources the read data and the I-Cache access is discarded. If the RAM is used only for data operands, power dissipation can be lowered by asserting the ASn bits associated with instruction fetches. Additionally, if the RAM contains only instructions, power dissipation can be reduced by masking operand accesses.

Consider the following examples of typical RAMBAR settings:

Data contained in RAM	RAMBAR[7:0]
Only code	$2B
Only data	$35
Both code and data	$21

5.5 INTERACTIONS BETWEEN KBUS MEMORIES

Depending on configuration information, instruction fetches and operand accesses may be sent to all of the K-Bus memories (i.e. RAM, ROM, and ICACHE) simultaneously. There needs to be consistency between the ACRs and the default modes defined by CACR (CACR[9], CACR[8], and CACR[5]).

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 I-CACHE data discarded. Accesses from the RAM module are not cached. The ROM behaves similarly. The RAM has priority over the ROM. The full priority scheme is as follows:

if (RAM "hits")

       RAM supplies data to the processor

else if (ROM "hits")

       ROM supplies data to the processor

else if (line-fill buffer "hits")

       line-fill buffer supplies data to the processor

else if (icache "hits")

       icache supplies data to the processor

else

       master bus cycle accesses to reference data from non-K-Bus memory

SECTION 6
MULTIPLY-ACCUMULATE UNIT

This section details the hardware multiply-accumulate (MAC) support functions within the ColdFire2M. The MAC is not present in the ColdFire2 hard macro, but is an optional high-speed execution unit that is available within the ColdFire core environment. If the MAC is present, it executes the integer multiplies at an accelerated speed. The MAC unit is designed to provide a common set of DSP operations and to enhance the current multiply instructions in the ColdFire architecture.

The MAC unit provides functionality in three related areas:

- Faster signed and unsigned integer multiplies
- New multiply-accumulate operations supporting signed and unsigned operands
- New miscellaneous register operations

Each is addressed in detail in the following sections.

6.1 INTRODUCTION

The MAC unit is designed to optimize the current ColdFire multiply instructions, MULS and MULU, and provide additional instructions for algorithms that use multiply-accumulate operations. As shown in Figure 6-1, these new instructions multiply two numbers, followed by the addition/subtraction of this product to/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. All arithmetic operations use register-based input operands, and summed values are stored in the internal accumulator. For word operations, the upper or lower word of each input register can be chosen as a source operand. As a result, two word operations will require only one register load operation. Some instructions also allow a parallel load to be performed in addition to the MAC operation.
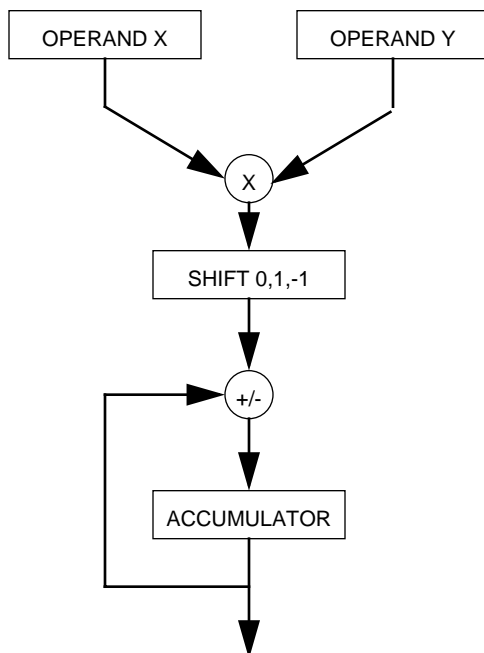
Figure 6-1. MAC Flow Diagram

The MAC module has been designed for 16-bit multiplies to optimize the die size. Word operations require two signed or unsigned 16-bit operands and produce a 32-bit result. By iteratively performing multiple 16-bit operations, the MAC can perform 32-bit operations. Longword operations require two signed or unsigned 32-bit operands and produce a 32-bit result. Because the longword operations are iteratively computed, the MAC instructions have an effective issue rate of one clock for word operations and three clocks for longword operations.

6.2 MAC PROGRAMMING MODEL

The MAC unit includes three new registers: a 32-bit accumulator (ACC), an 8-bit status register (MACSR), and a 16-bit mask register (MASK).

6.2.1 Accumulator (ACC)

This is a 32-bit special purpose register used to accumulate the results of MAC operations. The accumulator is not cleared upon reset.

6.2.2 MAC Status Register (MACSR)

The status register contains the saturation mode control bit, the negative, zero and overflow flags, as well as the signed/unsigned operation control bit as shown in the next register.

BITS	7	6	5	4	3	2	1	0
FIELD	OMC	S/U	-	-	N	Z	V	C
RESET	0	0	0	0	-	-	-	0
R/W	R/W	R/W	R	R	R/W	R/W	R/W	R

MAC Status Register (MACSR)

Field Definitions:

OMC[7]–Overflow Mode Control

   0 = normal mode
   1 = saturation mode

This bit is used to enable or disable saturation mode on overflow. This bit is cleared by system reset. Refer to **Section 6.4 Overflow Mode**.

S/U[6]–Signed/Unsigned Operations

The S/U bit defines the type of multiply operations that are performed.

   0 = signed numbers
   1 = unsigned numbers

This bit is cleared by system reset.

N[3]–Negative

This bit is set if the most significant bit of the result is set, cleared otherwise. This bit is affected only by MAC operations. The MULS and MULU instructions do not change this value.

Z[2]–Zero

This bit is set if the result equals zero, otherwise, it is cleared. This bit is affected only by MAC operations. MULS and MULU instructions do not change this value.

V[1]–Overflow

This bit is set if an arithmetic overflow occurs implying that the result cannot be represented in 32 bits. Once set, this bit remains set until the ACC register is loaded with a new value using the MOVE to ACC instruction, or the MACSR is explicitly loaded using the MOVE to MACSR instruction. The MULS and MULU instructions do not change this value.

C[0]–Carry

This field is always zero and is reserved for future use.

6.2.3 Mask Register (MASK)

This is a 16-bit special purpose register used to mask the lower 16 address bits during MAC load operations. The field definition follows.

BITS	15	0
FIELD	MASK	
RESE T	-	
R/W	R/W	

MAC Mask Register (MASK)

Field Definitions:

MASK[15:0]–Address Mask

This field is logically ANDed with the lower 16 bits of the effective address during MAC instructions with a register load performed using the mask addressing mode modifier.

The actions of the mask addressing mode depend on the effective addressing mode as shown in .

Table 6-1. Mask Addressing Mode

EFFECTIVE ADDRESSING MODE	OUTPUT ADDRESS	NEW AN
(An)	An & {0xFFFF, MASK}	-
(An)+	An	(An +4) & {0xFFFF, MASK}
-(An)	(An - 4) & {0xFFFF, MASK}	(An - 4) & {0xFFFF, MASK}
(d16, An)	(An + d16) & {0xFFFF, MASK}	-

NOTE: {upper, lower} notation indicates upper and lower order words

When loading data into the data registers, the MASK register can be used to implement a very efficient circular queue.

6.3 SHIFTING OPERATIONS

The MAC unit is capable of shifting a product before the result is added to or subtracted from the accumulator (ACC). Since there is the possibility of overflowing a 32-bit product, use the following guidelines when performing MAC instructions:

- For both word and longword unsigned operations, a zero is shifted into the product on right shifts.

- For signed word operations, the sign bit is shifted into the product on right shifts, unless the product is zero.

- For signed, longword operations, the sign bit is shifted into the product unless an overflow occurs or the product is zero, in which case a zero is shifted in.

6.4 OVERFLOW MODE

When dealing with potential overflow conditions, software overhead can be minimized by enabling hardware support for saturation arithmetic. Saturation mode is controlled by the

OMC bit in the MACSR. In saturation mode, if a MAC instruction overflows 32 bits during the multiply portion of an operation, the overflow bit (V) will be set in the MACSR and the accumulator (ACC) will contain the most positive or the most negative value possible. As seen in Table 6-2, the value depends on the signed/unsigned mode (S/U bit in the MACSR), multiply result, and the addition/subtraction operation.

Table 6-2. Accumulator Result in Saturation Mode

SIGNED/UNSIGNED	MULTIPLY/ RESULT	MAC INSTRUCTION	MAC OPERATION	ACCUMULATOR RESULT
Signed	Positive	MAC, MACL	Addition	$7FFFFFFF
		MSAC, MSACL	Subtraction	$80000000
	Negative	MAC, MACL	Addition	$80000000
		MSAC, MSACL	Subtraction	$7FFFFFFF
Unsigned	-	MAC, MACL	Addition	$FFFFFFFF
		MSAC, MSACL	Subtraction	$00000000

The overflow value will remain in the ACC until the V bit in the MACSR is cleared by loading a new value into the ACC or MACSR.

6.5 MAC INSTRUCTION SET SUMMARY

The MAC unit in the ColdFire2M enhances the multiply operations that are currently supported by the ColdFire2 architecture, adds new multiply-accumulate instructions, and adds register instructions for accessing the new MAC registers. Table 6-3 below summarizes the MAC unit instruction set. Refer to **Appendix B New MAC Instructions** for details.

Table 6-3. MAC Instruction Set Summary

INSTRUCTION	OPERAND SYNTAX	OPERAND SIZE	OPERATION
MULS	<ea>,Dx	$16 \times 16 \to 32$ $32 \times 32 \to 32$	$<ea> \times Dx \to Dx$ Note: signed operation
MULU	<ea>,Dx	$16 \times 16 \to 32$ $32 \times 32 \to 32$	$<ea> \times Dx \to Dx$ Note: unsigned operation
MAC	Ry,Rx,<shift>	$16 \times 16 + 32 \to 32$ $32 \times 32 + 32 \to 32$	$ACC + (Ry \times Rx)\{<<1 \,|\, >>1\} \to ACC$
MACL	Ry,Rx,<shift>,<ea>,Rw	$16 \times 16 + 32 \to 32$ $32 \times 32 + 32 \to 32$	$ACC + (Ry \times Rx)\{<<1 \,|\, >>1\} \to ACC; (<ea>\{\&MASK\}) \to Rw$
MSAC	Ry,Rx,<shift>	$32 - 16 \times 16 \to 32$ $32 - 32 \times 32 \to 32$	$ACC - (Ry \times Rx)\{<<1 \,|\, >>1\}SF \to ACC$
MSACL	Ry,Rx,<shift>,<ea>,Rw	$32 - 16 \times 16 \to 32$ $32 - 32 \times 32 \to 32$	$ACC - (Ry \times Rx)\{<<1 \,|\, >>1\}SF \to ACC; (<ea>\{\&MASK\}) \to Rw$
MOVE from ACC	ACC,Rx	32	$ACC \to Rx$
MOVE from MACSR	MACSR,Rx MACSR,CCR	32 8	$MACSR \to Rx$ $MACSR \to CCR$
MOVE from MASK	MASK,Rx	32	$MASK \to Rx$
MOVE to ACC	Ry,ACC #<data>,ACC	32 32	$Ry \to ACC$ $\#<data> \to ACC$
MOVE to CCR	Dy,CCR, #<data>,CCR	16	$Dy \to CCR$ $\#<data> \to CCR$
MOVE to MACSR	Ry,MACSR #<data>,MACSR	32	$Ry \to MACSR$ $\#<data> \to MACSR$
MOVE to MASK	Ry,MASK #<data>,MASK	32 32	$Ry \to MASK$ $\#<data> \to MASK$

SECTION 7
DEBUG SUPPORT

This section details the hardware debug support functions within the ColdFire2/2M. The general topic of debug support has been divided into three separate areas:

- Real-Time Trace Support
- Background Debug Mode (BDM)
- Real-Time Debug Support

Each is addressed in detail in the following sections.

The logic required to support these three areas is contained in a debug module, which is shown in the system block diagram in Figure 7-1.
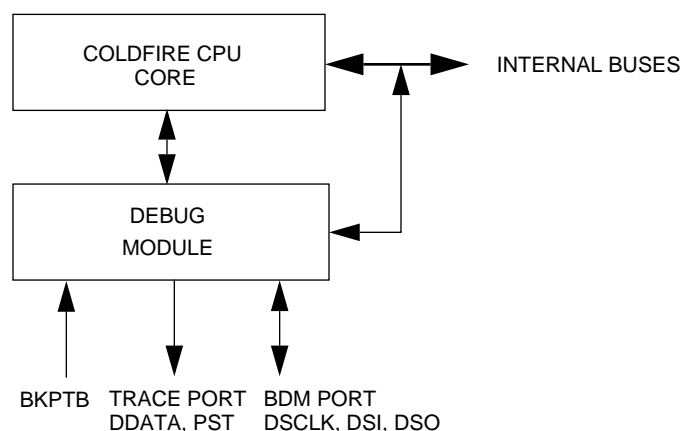
Figure 7-1. Processor/Debug Module Interface

7.1 SIGNAL DESCRIPTION

This section describes the ColdFire2/2M signals associated with the debug module. All ColdFire2/2M signals are unidirectional and synchronous.

7.1.1 Break Point (BKPTB)

This active-low, unidirectional input signal is used to request a manual break point. It will cause the processor to enter a halted state after the completion of the current instruction. This status will be reflected on the processor status (PST) outputs.

7.1.2 Debug Data (DDATA[3:0])

These output signals display the captured processor status and break point status.

7.1.3 Development Serial Clock (DSCLK)

This input signal is used as the development serial clock for the serial interface to the Debug Module.The maximum frequency is 1/2 the clock (CLK) frequency.

7.1.4 Development Serial Input (DSI)

This input signal provides the single-bit communication for the debug module commands.

7.1.5 Development Serial Output (DSO)

This output signal provides single-bit communication for the debug module responses.

7.1.6 Processor Status (PST[3:0])

These output signals report the processor status. Table 7-1 shows the encoding of these signals. These signals indicate the current status of the processor pipeline and, as a result, are not related to the current bus transfer.

Table 7-1. Processor Status Encoding

PST[3:0]		DEFINITION
(HEX)	(BINARY)	
$0	0000	Continue execution
$1	0001	Begin execution of an instruction
$2	0010	Reserved
$3	0011	Entry into user-mode
$4	0100	Begin execution of **PU**LSE orWDDATA instruction
$5	0101	Begin execution of taken branch
$6	0110	Reserved
$7	0111	Begin execution of **RTE** instruction
$8	1000	Begin 1-byte transfer on DDATA
$9	1001	Begin 2-byte transfer on DDATA
$A	1010	Begin 3-byte transfer on DDATA
$B	1011	Begin 4-byte transfer on DDATA
$C	1100	Exception processing†
$D	1101	Emulator-mode entry exception processing†
$E	1110	Processor is stopped, waiting for interrupt†
$F	1111	Processor is halted †

NOTE: †These encodings are asserted for multiple cycles.

7.2 REAL-TIME TRACE

In the area of debug functions, one fundamental requirement is support for real-time trace functionality, i.e., definition of the dynamic execution path. The ColdFire2/2M solution is to include a parallel output port providing encoded processor status and data to an external development system. This port is partitioned into two nibbles (4 bits): one nibble allows the

processor to transmit information concerning the execution status of the core (processor status, PST[3:0]), while the other nibble allows operand data to be displayed (debug data, DDATA[3:0]). The processor status timing is synchronous with the processor clock (CLK) and may not be related to the current bus transfer. Table 7-1 shows the encoding of these signals.

The processor status (PST) outputs can be used with an external image of the program to completely track the dynamic execution path of the machine. The tracking of this dynamic path is complicated by any change-of-flow operation. This is especially evident when the branch target address is calculated based on the contents of a program visible register (variant addressing.) For this reason, the debug data (DDATA) outputs will display the target address of a taken branch instruction. Because the DDATA bus is only 4 bits wide, the address is displayed a nibble at a time across multiple clock cycles.

The debug module includes two 32-bit storage elements for capturing the internal ColdFire2/2M bus information. These two elements effectively form a FIFO buffer connecting the internal bus to the external development system through the DDATA signals. The FIFO buffer captures branch target addresses along with certain operand read/write data for eventual display on the DDATA output port one nibble at a time. The execution speed of the ColdFire2/2M is affected only when both storage elements contain valid data waiting to be dumped onto the DDATA port. In this case, the processor core is stalled until one FIFO entry is available. In all other cases, data output on the DDATA port does not impact execution speed.

7.2.1 Processor Status Signal Encoding

The processor status (PST) signals are encoded to indicate a variety of conditions that are not always visible outside of the ColdFire2/2M.

7.2.1.1 CONTINUE EXECUTION (PST = $0). Most instructions complete in a single cycle. If an instruction requires more clock cycles, the subsequent clock cycles are indicated by driving the PST outputs with this encoding.

7.2.1.2 BEGIN EXECUTION OF AN INSTRUCTION (PST = $1). For most instructions, this encoding signals the first cycle of an instruction's execution. Some instructions generate a different unique encoding.

7.2.1.3 ENTRY INTO USER MODE (PST = $3). This encoding indicates the ColdFire2/2M has entered user mode. This encoding is signaled after the instruction which causes the user mode to be entered is executed (signaled with the appropriate encoding.)

7.2.1.4 BEGIN EXECUTION OF PULSE OR WDDATA INSTRUCTIONS (PST = $4). The ColdFire2/2M instruction set architecture includes a PULSE opcode. This opcode generates a unique PST encoding, $4, when executed. This instruction can define logic analyzer triggers for debug and/or performance analysis. Additionally, a WDDATA instruction is supported that allows the processor core to write any operand (byte, word, long) directly to the DDATA port, independent of any debug module configuration. This opcode also generates the special PST encoding, $4, when executed, but a data transfer on DDATA will

Freescale Semiconductor, Inc.

also be indicated. The size of the transfer will depend on the format of the WDDATA instruction.

7.2.1.5 BEGIN EXECUTION OF TAKEN BRANCH (PST = $5). This encoding is generated whenever a taken branch is executed. The branch target may be optionally displayed on DDATA depending on the control parameters contained in the configuration/ status register (CSR). The number of bytes of the address to be displayed is also controlled in the CSR and indicated during the data transfer on the following clock cycle.

The bytes are always displayed in a least-significant to most-significant order. The processor captures only those target addresses associated with taken branches using a variant addressing mode, i.e. all JMP and JSR instructions using address register indirect or indexed addressing modes, all RTE and RTS instructions as well as 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 these types of change-of-flow operations, the ColdFire2/2M processor uses the debug pins to output a sequence of information on successive clock cycles

1. Identify a taken branch has been executed using the PST pins ($5).

2. Using the PST pins, optionally signal the target address is to be displayed on the DDATA pins. The encoding ($9, $A, $B) identifies the number of bytes that are displayed.

3. The new target address is optionally available on subsequent cycles using the nibble-wide DDATA port. The number of bytes of the target address displayed on this port is a configurable parameter (2, 3, or 4 bytes).

Another example of a variant branch instruction would be a JMP (A0) instruction. If the CSR was programmed to display the lower two bytes of an address, the output of the PST and DDATA signals when this instruction executed are shown in Figure 7-2.
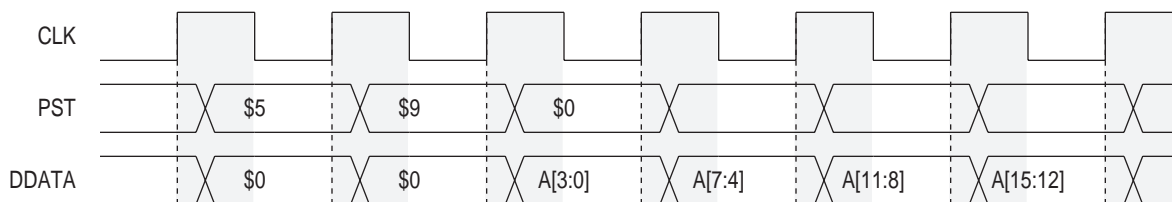
Figure 7-2. Example PST Diagram

In the first cycle, PST is driven with a $5 indicating a taken branch with a variant address. In the second cycle, PST is driven with a $9 indicating a two-byte address will be displayed four bits at a time on the DDATA signals over the next four clock cycles. The remaining four clock cycles display the lower two-bytes of the address (A0), least significant nibble to most significant nibble. The output of the PST signals after the branch instruction completes will

be dependent on the next instruction in the pipeline. The PST can continue with the next instruction before the address has completely displayed on the DDATA because of the DDATA FIFO. If the FIFO is full and the next instruction needs to display something on DDATA, the pipeline will stall (PST = $0) until space is available in the FIFO.

7.2.1.6 BEGIN EXECUTION OF RTE INSTRUCTION (PST = $7). The unique encoding is generated whenever the return-from-exception instruction is executed.

7.2.1.7 BEGIN DATA TRANSFER (PST = $8 - $A). These encodings indicate the number of bytes to be displayed on the DDATA port on subsequent clock cycles. This encoding is driven onto the PST port one machine cycle before the actual data is displayed on DDATA.

7.2.1.8 EXCEPTION PROCESSING (PST = $C). This encoding is displayed during normal exception processing. Exceptions which enter emulation mode (debug interrupt, or optional trace) generate a different encoding. Because this encoding defines a multicycle mode, the PST outputs are driven with this value until exception processing is completed.

7.2.1.9 EMULATOR-MODE EXCEPTION PROCESSING (PST = $D). Exceptions which enter emulation mode (debug interrupt, or optional trace) generate a different this encoding. Because this encoding defines a multicycle mode, the PST outputs are driven with this value until exception processing is completed.

7.2.1.10 PROCESSOR STOPPED (PST = $E). This encoding is generated as a result of the STOP instruction. The ColdFire2/2M remains in the stopped state until an interrupt occurs. Because this encoding defines a multicycle mode, the PST outputs are driven with this value until the stopped mode is exited.

7.2.1.11 PROCESSOR HALTED (PST = $F). This encoding is generated as when the ColdFire2/2M is halted (see **Section 7.3.1 CPU Halt**.) Because this encoding defines a multicycle mode, the PST outputs are driven with this value until the halted mode is exited.

7.3 BACKGROUND DEBUG MODE (BDM)

The ColdFire2/2M supports a modified version of the background debug mode (BDM) functionality found on Motorola's CPU32 family of parts. BDM implements a low-level system debugger in the microprocessor hardware. Communication with the development system is handled via a dedicated, high-speed serial command interface (BDM port).

Unless noted otherwise, the BDM functionality provided by the ColdFire2/2M is a proper subset of the CPU32 functionality. The main differences include the following:

- ColdFire2/2M implements the BDM controller in a dedicated hardware module. Although some BDM operations do require the CPU to be halted (e.g. CPU register accesses), other BDM commands such as memory accesses can be executed while the processor is running.

- DSCLK, DSI, and DSO are treated as synchronous signals, where the inputs, DSCLK and DSI, must meet the required input setup and hold timings, and the output, DSO, is specified as a delay relative to the rising edge of the processor clock.

- On CPU32 parts, the DSO signal can inform hardware that a serial transfer can start. ColdFire clocking schemes restrict the use of this bit. Because DSO changes only when DSCLK is high, DSO cannot be used to indicate the start of a serial transfer. The development system should use either a free-running DSCLK or count the number of clocks in any given transfer.

- The read/write system register commands, RSREG and WSREG, have been replaced by read/write control register commands, RCREG and WCREG. These commands use the register coding scheme from the MOVEC instruction.

- The read/write debug module register commands, RDMREG and WDMREG, have been added to support debug module register accesses.

- CALL and RST commands are not supported and will generate an illegal command response.

- Illegal command responses can be returned using the FILL and DUMP commands.

- For any command performing a byte-sized memory read operation, the upper 8 bits of the response data are undefined. The referenced data is returned in the lower 8 bits of the response.

- The debug module forces alignment for memory-referencing operations: long accesses are forced to a 0-modulo-4 address; word accesses are forced to a 0-modulo-2 address. An address error response can no longer be returned.

7.3.1 CPU Halt

Although some BDM operations can occur in parallel with CPU operation, unrestricted BDM operation requires the CPU to be halted. A number of sources can cause the CPU to halt, including the following as shown in order of priority:

1. The occurrence of the catastrophic fault-on-fault condition automatically halts the processor. The halt status, $F, is posted on the PST port.

2. The occurrence of a hardware breakpoint can be configured to generate a pending halt condition in a manner similar to the assertion of the BKPTB signal. In all cases, the occurrence of this type of breakpoint halts the processor in an imprecise manner. Once the hardware breakpoint is asserted, the processor halts at the next sample point. See **Section 7.4.1 Theory of Operation** for more detail.

3. The execution of the HALT**,** also known as BGND on the 683xx devices, instruction immediately suspends execution and posts the halt status ($F) on the PST outputs. By default, this is a supervisor instruction and attempted execution while in user mode generates a privilege-violation exception. A User Halt Enable (UHE) control bit is provided in the Configuration/Status Register (CSR) to allow execution of HALT in user mode.

4. The assertion of the BKPTB input pin is treated as a pseudo-interrupt, i.e., the halt condition is made pending 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 halt status, $F, is reflected in the PST outputs.

There are two special cases involving the assertion of the BKPTB pin to be considered.

After the master reset signal (MRSTB) is negated, the processor waits for 16 clock cycles before beginning reset exception processing. If the BKPTB input pin is asserted within the first eight cycles after MRSTB is negated, the processor will enter the halt state, signaling that halt status, $F, on the PST outputs. While in this state, all resources accessible via the debug module can be referenced. This is the only opportunity to force the ColdFire2/2M into emulation mode via the EMU bit in the configuration/status register (CSR). Once the system initialization is complete, the processor response to a BDM GO command is dependent on the set of BDM commands performed while breakpointed. Specifically, if the processor's PC register was loaded, then the GO command simply causes the processor to exit the halted state and pass control to the instruction address contained in the PC. Note in this case, the normal reset exception processing is bypassed. Conversely, if the PC register was not loaded, then the GO command causes the processor to exit the halted state and continue with reset exception processing.

The ColdFire2/2M also handles a special case with the assertion of BKPTB 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. Once halted, the standard BDM commands may be exercised. When the processor is restarted, it continues with the execution of the next sequential instruction, i.e., the instruction following the STOP opcode.

The halt source is indicated in CSR[27:24]; for simultaneous halt conditions, the highest priority source is indicated.

7.3.2 BDM Serial Interface

Once the CPU is halted and the halt status reflected on the PST outputs (PST[3:0] = $F), the development system may send unrestricted commands to the debug module. The debug module implements a synchronous protocol using a three-pin interface: development serial clock (DSCLK), development serial input (DSI), and development serial output (DSO). The development system serves as the serial communication channel master and is responsible for generation of the clock (DSCLK). The operating range of the serial channel is DC to 1/2 of the processor frequency. The channel uses a full duplex mode, where data is transmitted 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 seen in Figure 7-3, data is exchanged on the positive edge of CLK when DSCLK is high (i.e. DSI is sampled and DSO is driven.) The DSCLK signal must also be sampled low (on a positive edge of CLK) between each bit exchange. The MSB is transferred first.
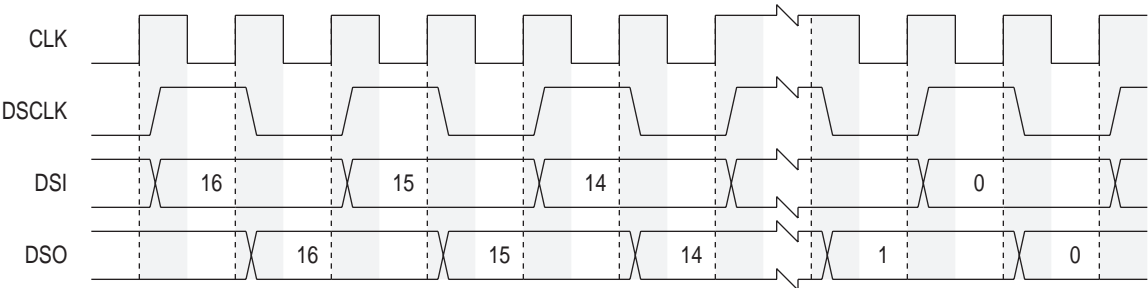
Figure 7-3. BDM Serial Transfer

Both DSCLK and DSI are synchronous inputs and must meet input setup and hold times with respect to CLK. The DSCLK signal essentially acts as a pseudo "clock enable" and is sampled on the rising edge of CLK. If the setup time of DSCLK is met, then the internal logic transitions on the rising edge of CLK, and DSI is sampled on the same CLK rising edge. The DSO output is specified as a delay from the DSCLK-enabled CLK rising edge. All events in the debug module's serial state machine are based on the rising edge of the microprocessor clock (see Figure 7-4 below). Also refer to the Electrical Characteristics section of this manual.

Figure 7-4. BDM Signal Sampling

7.3.2.1 RECEIVE PACKET FORMAT. The basic receive packet of information is 17 bits long,16 data bits plus a status bit, as shown below in Figure 7-5.

16	15															0
S	DATA FIELD [15:0]															

Figure 7-5. Receive BDM Packet

Status[16]

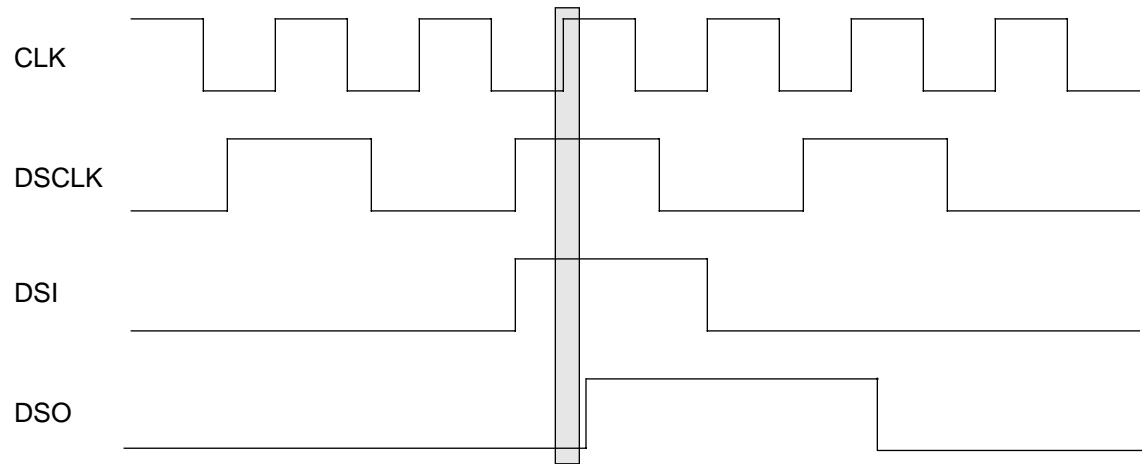The status bit indicates the status of CPU-generated messages (always single word), with the data field encoded as listed in Table 7-2.

Table 7-2. CPU-Generated Message Encoding

S BIT	DATA	MESSAGE TYPE
0	xxxx	Valid data transfer
0	$FFFF	Command complete; status OK
1	$0000	Not ready with response; come again
1	$0001	TEA-terminated bus cycle; data invalid
1	$FFFF	Illegal command

Data Field[15:0]

The data field contains the message data to be communicated from the debug module to the development system. The response message is always a single word, with the data field encoded as shown in Table 7-2.

7.3.2.2 TRANSMIT PACKET FORMAT. The basic transmit packet of information is 17 bits long,16 data bits plus a control bit, as shown below in Figure 7-6.

16	15	0
C	DATA FIELD [15:0]	

Figure 7-6. Transmit BDM Packet

Control[16]

The control bit is not used but is reserved by Motorola for future use. Command and data transfers initiated by the development system should clear bit 16.

Data Field[15:0]

The data field contains the message data to be communicated from the development system to the debug module.

7.3.3 BDM Command Set

The ColdFire2/2M supports a subset of BDM instructions from the current 683xx parts, as well as extensions to provide access to new hardware features. The BDM commands should not be issued whenever the ColdFire2/2M is accessing the debug module registers using the WDEBUG instruction.

7.3.3.1 BDM COMMAND SET SUMMARY. The BDM command set is summarized in Figure 7-6. Subsequent paragraphs contain detailed descriptions of each command.

Table 7-3. BDM Command Summary

COMMAND	MNEMONIC	DESCRIPTION	CPU IMPACT[1]	PAGE
READ A/D REGISTER	RAREG/RDREG	Read the selected address or data register and return the results via the serial interface.	HALTED	7-13
WRITE A/D REGISTER	WAREG/WDREG	The data operand is written to the specified address or data register.	HALTED	7-13
READ MEMORY LOCATION	READ	Read the data at the memory location specified by the longword address.	CYCLE STEAL	7-14
WRITE MEMORY LOCATION	WRITE	Write the operand data to the memory location specified by the longword address.	CYCLE STEAL	7-16
DUMP MEMORY BLOCK	DUMP	Used in conjunction with the READ command 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. Subsequent operands are retrieved with the DUMP command.	CYCLE STEAL	7-17
FILL MEMORY BLOCK	FILL	Used in conjunction with the WRITE command 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. Subsequent operands are written with the FILL command.	CYCLE STEAL	7-19
RESUME EXECUTION	GO	The pipeline is flushed and refilled before resuming instruction execution at the current PC.	HALTED	7-21
NO OPERATION	NOP	NOP performs no operation and may be used as a null command.	PARALLEL	7-21
READ CONTROL REGISTER	RCREG	Read the system control register.	HALTED	7-22
WRITE CONTROL REGISTER	WCREG	Write the operand data to the system control register.	HALTED	7-23
READ DEBUG MODULE REGISTER	RDMREG	Read the Debug Module register.	PARALLEL	7-24
WRITE DEBUG MODULE REGISTER	WDMREG	Write the operand data to the Debug Module register.	PARALLEL	7-25

NOTE: 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 which can be interleaved with CPU accesses

Parallel - Command is executed in parallel with CPU activity

Refer to command summaries for detailed operation descriptions.

7.3.3.2 COLDFIRE BDM COMMANDS. All ColdFire Family BDM commands include a 16-bit operation word followed by an optional set of one or more extension words.

15		10	9	8	7	6	5	4	3	2	0
OPERATION			0	R/$\overline{\text{W}}$	OP SIZE		0	0	A/$\overline{\text{D}}$	REGISTER	
EXTENSION WORD(S)											

BDM Command Format

Operation Field

The operation field specifies the command.

R/$\overline{\text{W}}$ Field

The R/$\overline{\text{W}}$ field specifies the direction of operand transfer. When the bit is set, the transfer is from the CPU to the development system. When the bit is cleared, data is written to the CPU or to memory from the development system.

Operand Size

For sized operations, this field specifies the operand data size. All addresses are expressed as 32-bit absolute values. The size field is encoded as listed in Table 7-4.

Table 7-4. BDM Size Field Encoding

ENCODING	OPERAND SIZE
00	Byte
01	Word
10	Long
11	Reserved

Address / $\overline{\text{Data}}$ (A/$\overline{\text{D}}$) Field

The A/D field is used in commands that operate on address and data registers in the processor. It determines whether the register field specifies a data or address register. A one indicates an address register; zero, a data register.

Register Field

In commands that operate on processor registers, this field specifies which register is selected. The field value contains the register number.

Extension Word(s) (as required):

Certain commands require extension words for addresses and/or immediate data. Addresses require two extension words because only absolute long addressing is permitted. Immediate data can be either one or two words in length—byte and word data each require a single extension word; longword data requires two words. Both 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 the "Operand Data."

7.3.3.3 COMMAND SEQUENCE DIAGRAM. A command sequence diagram (see Figure 7-7) illustrates the serial bus traffic for each command. Each bubble in the diagram represents a single 17-bit transfer across the bus. The top half in each bubble corresponds to the data transmitted by the development system to the debug module; the bottom half corresponds to the data returned by the debug module in response to the development system commands. Command and result transactions are overlapped to minimize latency.

The cycle in which the command is issued contains the development system command mnemonic (in this example, "read memory location"). During the same cycle, the debug module responds with either the lowest order results of the previous command or with a command complete status (if no results were required).

Freescale Semiconductor, Inc.

During the second cycle, the development system supplies the high-order 16 bits of the memory address. The debug module returns a "not ready" response ($10000) unless the received command was decoded as unimplemented, in which case the response data is the illegal command ($1FFFF) encoding. If an illegal command response occurs, the development system should retransmit the command.

NOTE

> The response can be ignored unless a memory bus cycle is in progress. Otherwise, the debug module can accept a new serial transfer after eight system clock periods.

In the third cycle, the development system supplies the low-order 16 bits of a memory address. The debug module always returns the "not ready" response in this cycle. At the completion of the third cycle, the debug module initiates a memory read operation. Any serial transfers that begin while the memory access is in progress return the "not ready" response.

Results are returned in the two serial transfer cycles following the completion of the memory access. The data transmitted to the debug module during the final transfer is the opcode for the following command. Should a memory access generate a bus error, an error status ($10001) is returned in place of the result data.
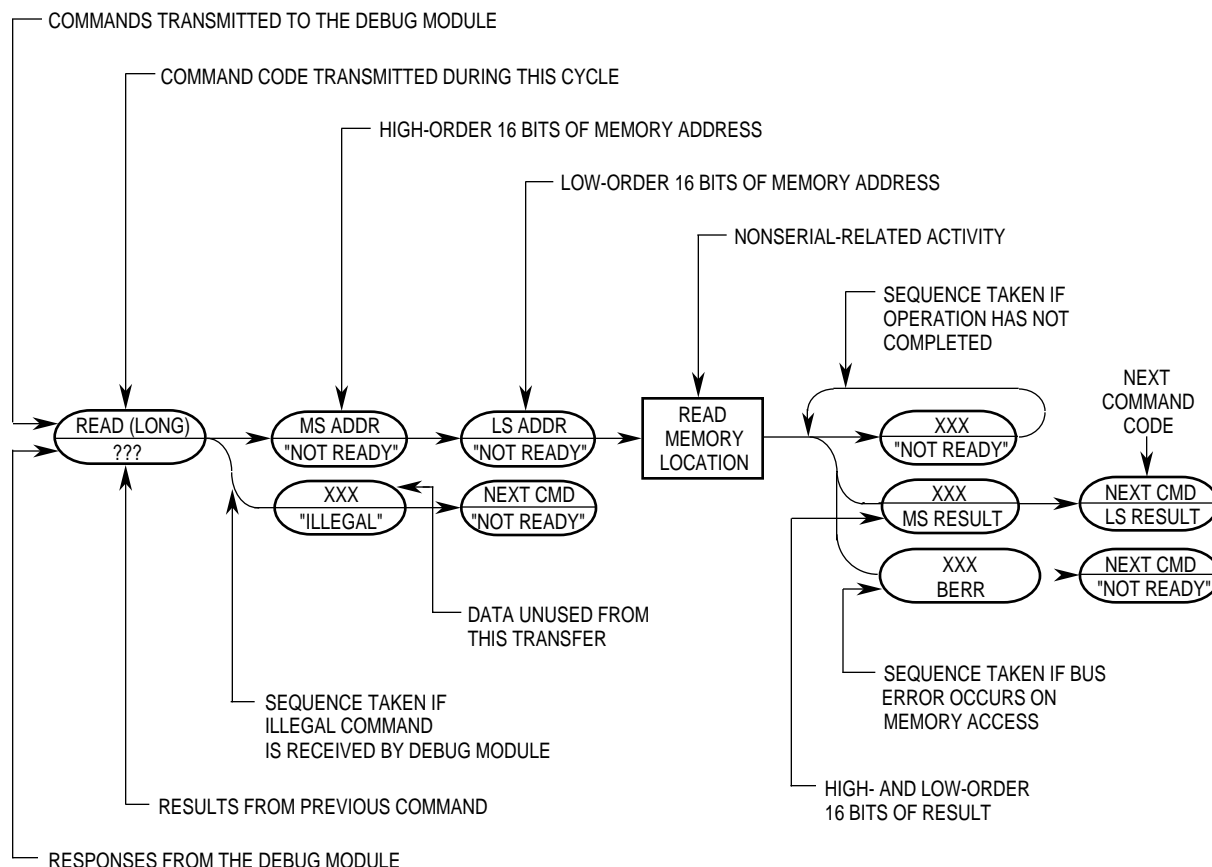
Figure 7-7. Command Sequence Diagram

7.3.3.4 COMMAND SET DESCRIPTIONS. The BDM command set is summarized in Table 7-3. Subsequent paragraphs contain detailed descriptions of each command.
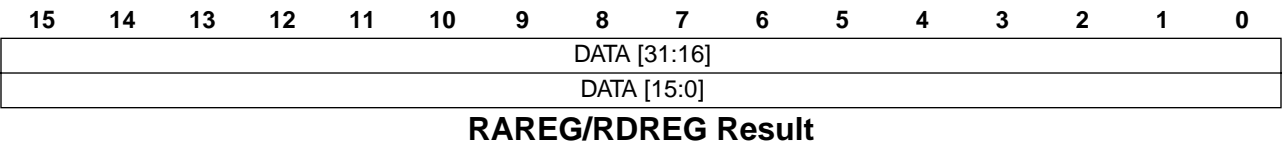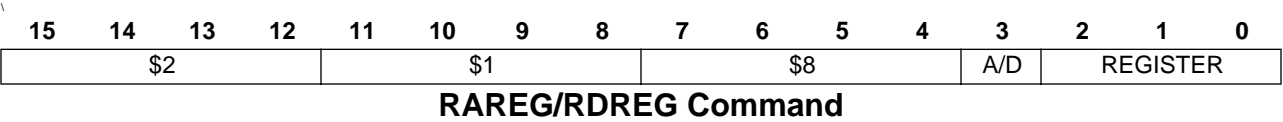
Note

> All the accompanying BDM commands assume the control bit (C) is zero. The BDM results are defined with the status bit (S) as zero. Refer to **Section 7.3.2 BDM Serial Interface** for information on the serial packet format
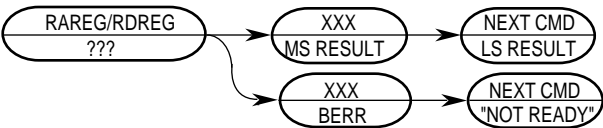
Unassigned command opcodes are reserved by Motorola for future expansion. All unused command formats within any revision level will perform a NOP and return the ILLEGAL command response.

7.3.3.4.1 Read A/$\overline{D}$ 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.

Formats:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
\$2				\$1				\$8				A/D	REGISTER		

RAREG/RDREG Command

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA [31:16]															
DATA [15:0]															

RAREG/RDREG Result

Command Sequence:

Operand Data:
None

Result Data:
The contents of the selected register are returned as a longword value. The data is returned most significant word first.
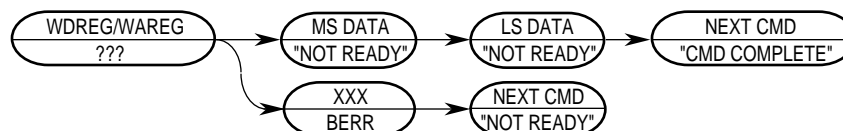
7.3.3.4.2 Write A/$\overline{D}$ Register (WAREG/WDREG). The operand data is written to the specified address or data register. All 32 register bits are altered by the write. A bus error response is returned if the CPU core is not halted.

Freescale Semiconductor, Inc.

Command Formats:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
\$2				\$0				\$8				A/D	REGISTER		
DATA [31:16]															
DATA [15:0]															

WAREG/WDREG Command

Command Sequence:

```
  ┌──────────────┐      ┌──────────────┐    ┌──────────────┐    ┌──────────────┐
  │  WDREG/WAREG │ ───▶ │   MS DATA    │──▶ │   LS DATA    │──▶ │   NEXT CMD   │
  │     ???      │      │ "NOT READY"  │    │ "NOT READY"  │    │"CMD COMPLETE"│
  └──────────────┘      └──────────────┘    └──────────────┘    └──────────────┘
           │            ┌──────────────┐    ┌──────────────┐
           └──────────▶ │     XXX      │──▶ │   NEXT CMD   │
                        │     BERR     │    │ "NOT READY"  │
                        └──────────────┘    └──────────────┘
```

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 will be indicated by returning the data \$FFFF (with the status bit cleared) when the register write is complete.

7.3.3.4.3 Read Memory Location (READ). Read the operand data from the memory location specified by the longword address. The address space is defined by the contents of the low-order 5 bits {TT, TM} of the address attribute register (AATR). The hardware forces the low-order bits of the address to zeros for word and longword accesses to ensure that operands are always accessed on natural boundaries: words on 0-modulo-2 addresses, longwords on 0-modulo-4 addresses.

Formats:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
\$1				\$9				\$0				\$0			
ADDRESS [31:16]															
ADDRESS [15:0]															

Byte READ Command

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

Byte READ Result

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
\$1				\$9				\$4				\$0			
ADDRESS [31:16]															
ADDRESS [15:0]															

Word READ Command

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

Word READ Result

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
\$1				\$9				\$8				\$0			
ADDRESS [31:16]															
ADDRESS [15:0]															

Long READ Command

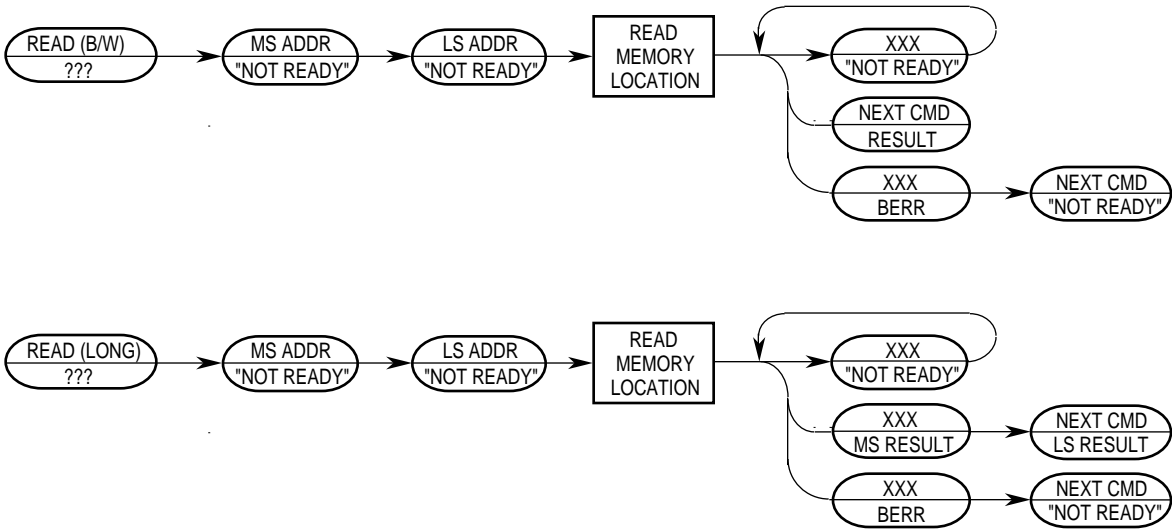| 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA [31:16]															
DATA [15:0]															
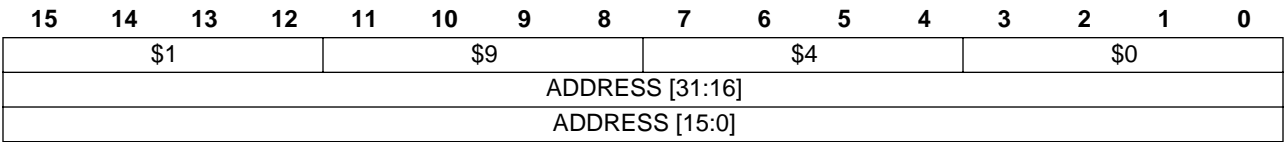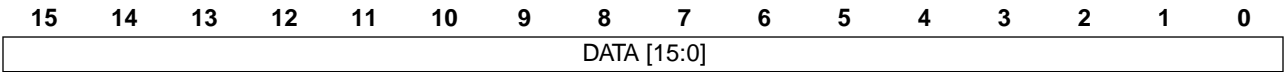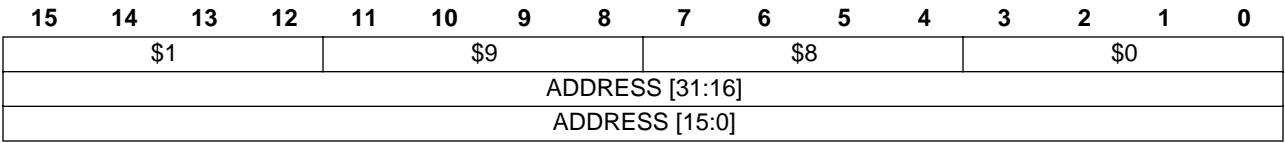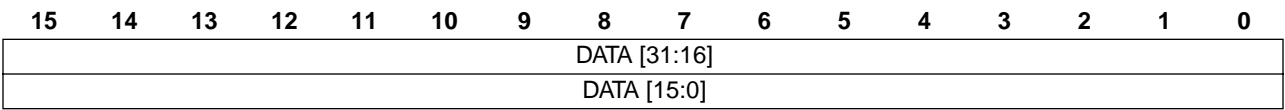
Long READ Result

Command Sequence:

Operand Data:

The single operand is the longword address of the requested memory location.

Freescale Semiconductor, Inc.

Result Data:

The requested data is returned as either a word or longword. Byte data is returned in the least significant byte of a word result, with the upper byte undefined. Word results return 16 bits of significant data; longword results return 32 bits. A value of $0001 (with the status bit set) will be returned if a bus error occurs.

7.3.3.4.4 Write Memory Location (WRITE). Write the operand data to the memory location specified by the longword address. The address space is defined by the contents of the low-order 5 bits {TT, TM} of the address attribute register (AATR). The hardware forces the low-order bits of the address to zeros for word and longword accesses to ensure that operands are always accessed on natural boundaries: words on 0-modulo-2 addresses, longwords on 0-modulo-4 addresses.

Formats:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	$1				$8				$0				$0		
ADDRESS [31:16]															
ADDRESS [15:0]															
X	X	X	X	X	X	X	X	DATA [7:0]							

Byte WRITE Command

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	$1				$8				$4				$0		
ADDRESS [31:16]															
ADDRESS [15:0]															
DATA [15:0]															

Word WRITE Command

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	$1				$8				$8				$0		
ADDRESS [31:16]															
ADDRESS [15:0]															
DATA [31:16]															
DATA [15:0]															

Long WRITE Command

Command Sequence:
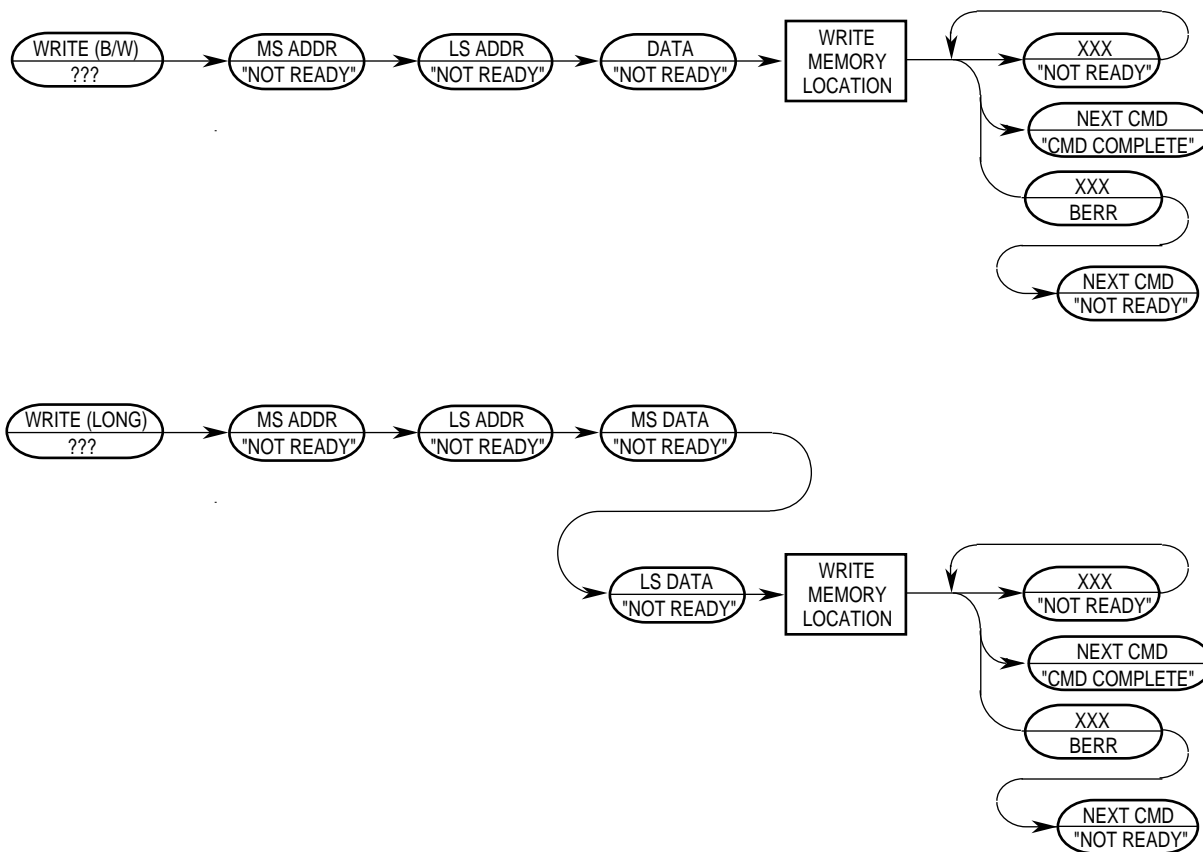
Operand Data:

Two operands are required for this instruction. The first operand is a longword absolute address that specifies a location to which the operand data is to be written. The second operand is the data. Byte data is transmitted as a 16-bit word, justified in the least significant byte; 16- and 32-bit operands are transmitted as 16 and 32 bits, respectively.

Result Data:

Successful write operations will be indicated by returning the data $FFFF (with the status bit cleared) when the register write is complete. A value of $0001 (with the status bit set) will be returned if a bus error occurs.

7.3.3.4.5 Dump Memory Block (DUMP). DUMP is used in conjunction with the READ command 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. 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 (address breakpoint high (ABHR)). Subsequent DUMP commands use this address, perform the memory read, increment it by the current operand size, and store the updated address in ABHR.

NOTE

The DUMP command does not check for a valid address in ABHR—DUMP is a valid command only when preceded by another DUMP, NOP or by a READ 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 DUMP command is given, allowing the operand size to be dynamically altered.

Command Formats:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	$1				$D				$0				$0		

Byte DUMP Command

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

Byte DUMP Result

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	$1				$D				$4				$0		

Word DUMP Command

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

Word DUMP Result

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	$1				$D				$8				$0		

Long DUMP Command

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
							DATA [31:16]								
							DATA [15:0]								

Long DUMP Result

Command Sequence:
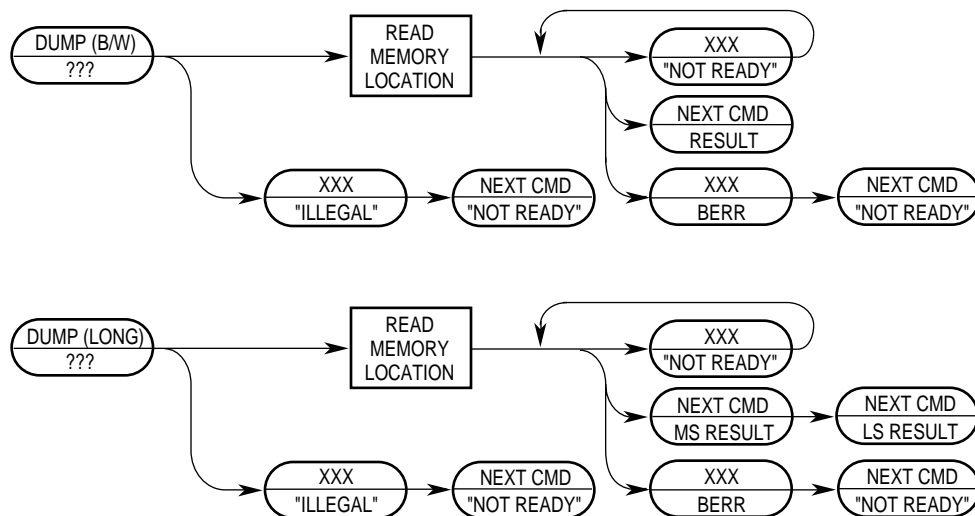
Operand Data:

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 $0001 (with the status bit set) will be returned if a bus error occurs.

7.3.3.4.6 Fill Memory Block (FILL). FILL is used in conjunction with the WRITE command 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. The FILL command writes subsequent operands. The initial address is incremented by the operand size (1, 2, or 4) and is saved in address breakpoint high (ABHR) 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 ABHR.

NOTE

The FILL command does not check for a valid address in ABHR—FILL is a valid command only when preceded by another FILL, NOP or by 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.

Freescale Semiconductor, Inc.

Formats:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
\$1				\$C				\$0				\$0			
X	X	X	X	X	X	X	X	DATA [7:0]							

Byte FILL Command

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
\$1				\$C				\$4				\$0			
DATA [15:0]															

Word FILL Command

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
\$1				\$C				\$8				\$0			
DATA [31:16]															
DATA [15:0]															

Long FILL Command

Command Sequence:
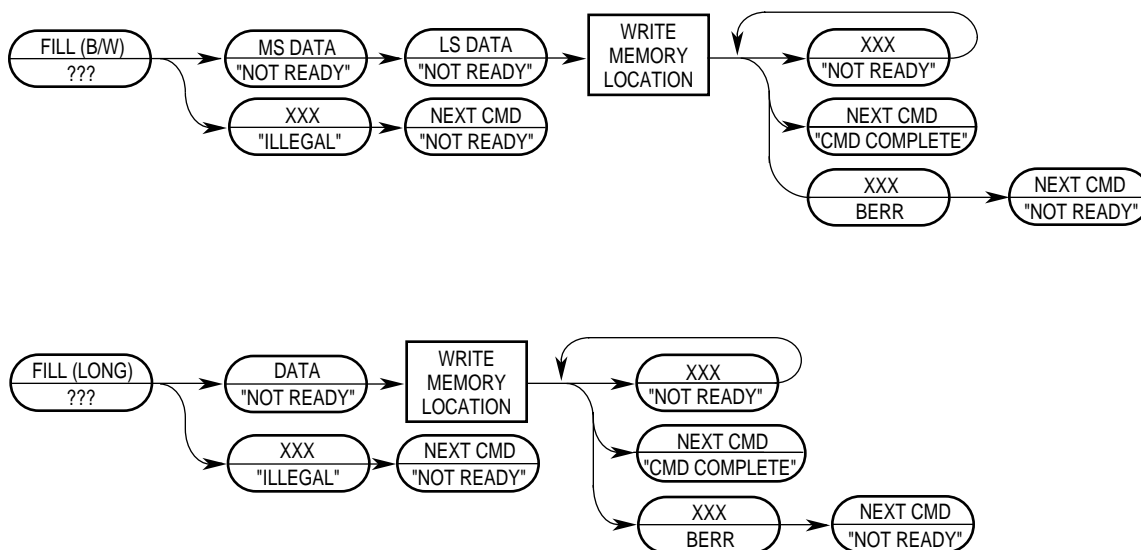
Operand Data:

A single operand is data to be written to the memory location. Byte data is transmitted as a 16-bit word, justified in the least significant byte; 16- and 32-bit operands are transmitted as 16 and 32 bits, respectively.

Result Data:

Command complete status will be indicated by returning the data $FFFF (with the status bit cleared) when the register write is complete. A value of $0001 (with the status bit set) will be returned if a bus error occurs.

7.3.3.4.7 Resume Execution (GO). The pipeline is flushed and refilled before resuming normal instruction execution. Prefetching begins at the current PC and current privilege level. If either the PC or SR is altered during BDM, the updated value of these registers is used when prefetching begins.

Formats:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$0				$C				$0				$0			
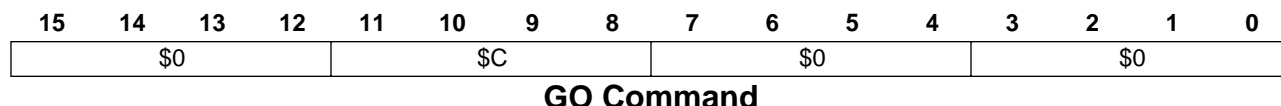
GO Command

Command Sequence:

Operand Data:
None

Result Data:

The "command complete" response ($0FFFF) is returned during the next shift operation.

7.3.3.4.8 No Operation (NOP). NOP performs no operation and may be used as a null command where required.

Formats:

15		12	11		8	7		4	3		0
$0			$0			$0			$0		

NOP Command
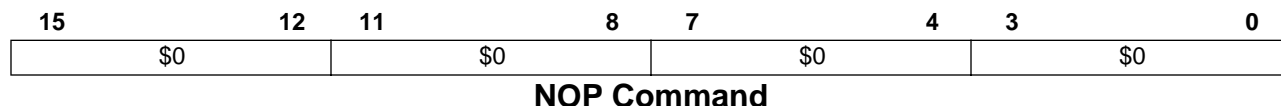
Command Sequence:
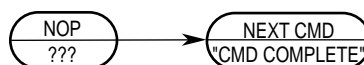
Operand Data:

None

Result Data:

The "command complete" response, $FFFF (with the status bit cleared), is returned during the next shift operation.

7.3.3.4.9 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 in size, regardless of the implemented register width. The second and third words of the command effectively form a 32-bit address used by the debug module 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.

Formats:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$2				$9				$8				$0			
$0				$0				$0				$0			
$0				Rc											

RCREG Command

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA [31:16]															
DATA [15:0]															

RCREG Result

Rc encoding:

Table 7-5. Control Register Map

Rc	REGISTER DEFINITION
$002	Cache Control Register (CACR)
$004	Access Control Register 0 (ACR0)
$005	Access Control Register 1 (ACR1)
$801	Vector BASE Register (VBR)
$804	MAC Status Register (MACSR)†
$805	MAC Mask Register (MASK)†
$806	MAC Accumulator (ACC)†
$80E	Status Register (SR)
$80F	Program Counter (PC)
$C00	ROM Base Address Register (ROMBAR0)
$C04	RAM Base Address Register (RAMBAR0)

NOTE: †Available on the ColdFire2M only.

Command Sequence:
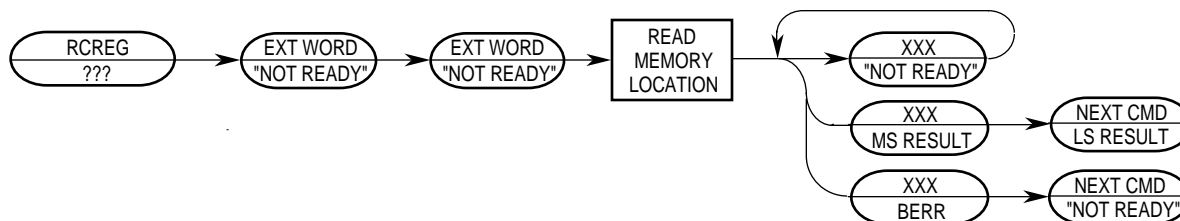
Operand Data:

The single operand is the 32-bit Rc control register select field.

Result Data:

The contents of the selected control register are returned as a longword value. The data is returned most-significant-word first. For those control registers with widths less than 32 bits, only the implemented portion of the register is guaranteed to be correct. The remaining bits of the longowrd are undefined. As an example, a read of the 16-bit SR will return the SR in the lower word and undefined data in the upper word.

7.3.3.4.10 Write Control Register (WCREG). The operand (longword) data is written to the specified control register. The write alters all 32 register bits.

Formats:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$2				$8				$8				$0			
$0				$0				$0				$0			
$0				Rc											
DATA [31:16]															
DATA [15:0]															

WCREG Command

Command Sequence:
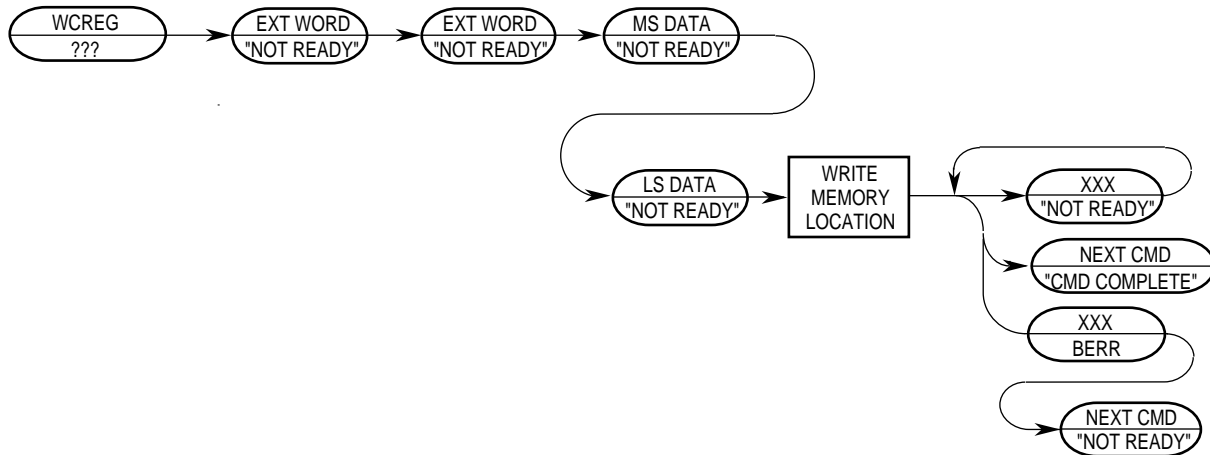
Operand Data:

Two operands are required for this instruction. The first long operand selects the register to which the operand data is to be written. The second operand is the data.

Result Data:

Successful write operations return $FFFF. Bus errors on the write cycle are indicated by the assertion of bit 16 in the status message and by a data pattern of $0001.

7.3.3.4.11 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 the CSR (DRc = $0).

Command Formats:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
\$2				\$D				\$8				DRc			

RDMREG BDM Command

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA [31:16]															
DATA [15:0]															

RDMREG BDM Result
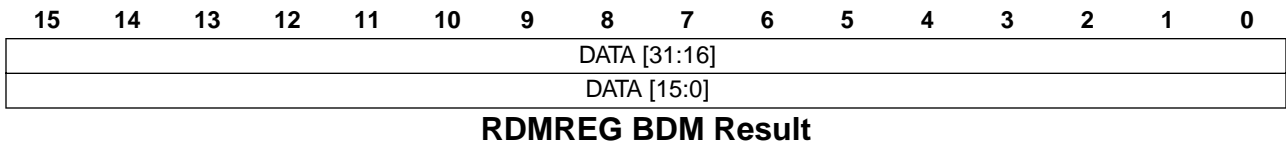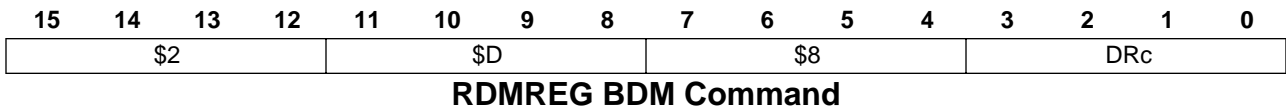
DRc encoding:

Table 7-6. Definition of DRc Encoding - Read

DRc[3:0]	DEBUG REGISTER DEFINITION	MNEMONIC	INITIAL STATE
\$0	Configuration/Status	CSR	\$0
\$1-\$F	Reserved	-	–

Command Sequence:
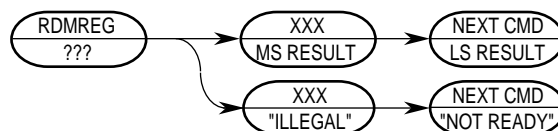
Operand Data:

None

Result Data:

The contents of the selected debug register are returned as a longword value. The data is returned most-significant-word first.

7.3.3.4.12 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. The DSCLK signal must be inactive while CPU accesses are being performed.

Command Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
\$2				\$C				\$8				DRc			
DATA [31:16]															
DATA [15:0]															

WDMREG BDM Command

DRc encoding:

Table 7-7. Definition of DRc Encoding - Write

DRc[3:0]	DEBUG REGISTER DEFINITION	MNEMONIC	INITIAL STATE
\$0	Configuration/Status	CSR	\$0
\$1-\$5	Reserved	-	–
\$6	Bus Attributes and Mask	AATR	\$0005
\$7	Trigger Definition	TDR	\$0
\$8	PC Breakpoint	PBR	–
\$9	PC Breakpoint Mask	PBMR	–
\$A-\$B	Reserved	–	–
\$C	Operand Address High Breakpoint	ABHR	–
\$D	Operand Address Low Breakpoint	ABLR	–
\$E	Data Breakpoint	DBR	–
\$F	Data Breakpoint Mask	DBMR	–

Command Sequence:
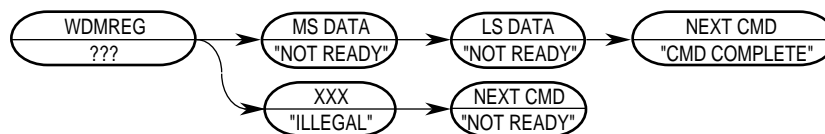
Operand Data:

Longword data is written into the specified debug register. The data is supplied most-significant-word first.

Result Data:

Command complete status ($0FFFF) is returned when register write is complete.

7.3.3.4.13 Unassigned Opcodes. Unassigned command opcodes are reserved by Motorola. All unused command formats within any revision level will perform a NOP and return the ILLEGAL command response*.*

7.4 REAL-TIME DEBUG SUPPORT

The ColdFire2/2M provides support for the debug of real-time applications. For these types of embedded systems, the processor cannot be halted during debug, but must continue to operate. The foundation of this area of debug support is that while the processor cannot be halted to allow debug, the system can tolerate small intrusions into the real-time operation.

As discussed in the previous section, the debug module provides a number of hardware resources to support various hardware breakpoint functions. Specifically, three types of breakpoints are supported: PC with mask, operand address range, and data with mask. These three basic breakpoints can be configured into one- or two-level triggers with the exact trigger response also programmable. The debug module programming model is accessible from either the external development system using the serial interface or from the processor's supervisor programming model using the WDEBUG instruction.

7.4.1 Theory of Operation

The breakpoint hardware can be configured to respond to triggers in several ways. The desired response is programmed into the Trigger Definition Register. In all situations where a breakpoint triggers, an indication is provided on the DDATA output port, when not displaying captured operands or branch addresses, as shown in Table 7-8.

Table 7-8. DDATA, CSR[31:28] Breakpoint Response

DDATA[3:0], CSR[31:28]	BREAKPOINT STATUS
$0	No breakpoints enabled
$1	Waiting for level 1 breakpoint
$2	Level 1 breakpoint triggered
$3-4	Reserved
$5	Waiting for level 2 breakpoint
$6	Level 2 breakpoint triggered
$7-$F	Reserved

The breakpoint status is also posted in the CSR.

The new BDM instructions load and configure the desired breakpoints using the appropriate registers. As the system operates, a breakpoint trigger generates a response as defined in the TDR. If the system can tolerate the processor being halted, a BDM entry can be used. With the TRC bits of the TDR = $1, the breakpoint trigger causes the core to halt as reflected in the status of PST = $F. For PC breakpoints, the halt occurs before the targeted instruction is executed. For address and data breakpoints, the processor may have executed several additional instructions. As a result, trigger reporting is considered imprecise.

If the processor core cannot be halted, the special debug interrupt can be used. With this configuration, TRC bits of the TDR **= $2**, the breakpoint trigger is converted into a debug interrupt to the processor (see **Section 4.2.8 Debug Interrupt**.) This interrupt is treated as higher than the nonmaskable level 7 interrupt request. As with all interrupts, it is made pending when the processor samples, once per instruction. Again, the hardware forces the PC breakpoint to occur immediately (before the execution of the targeted instruction). This is possible because the PC breakpoint comparison is enabled at the same time the interrupt sampling occurs. For the address and data breakpoints, the reporting is considered imprecise because several additional instructions may be executed after the triggering address or data is seen.

Once the debug interrupt is recognized, the processor aborts execution and initiates exception processing. At the initiation of the exception processing, the core enters emulator mode. After the standard 8-byte exception stack is created, the processor fetches a unique exception vector, $C, from the vector table.

Execution continues at the instruction address contained in this exception vector. All interrupts are ignored while in emulator mode. Users can program the debug-interrupt handler to perform the necessary context saves using the supervisor instruction set. As an example, this handler may save the state of all the program-visible registers as well as the current context into a reserved memory area.

Once the required operations are complete, the return-from-exception (RTE) instruction is executed and the processor exits emulator mode. Once the debug interrupt handler has completed its execution, the external development system can then access the reserved memory locations using the BDM commands to read memory.

Freescale Semiconductor, Inc.

7.4.1.1 EMULATOR MODE. Emulator mode is used to facilitate non-intrusive emulator functionality. This mode can be entered in three different ways:

- The EMU bit in the configuration/status register (CSR) may be programmed to force the ColdFire2/2M into emulation mode. This bit is examined only when MRSTB is negated and the processor begins reset exception processing. It may be set while the ColdFire2/2M is halted before the reset exception processing begins. Refer to **Section 7.3.1 CPU Halt**.

- A debug interrupt always enters emulation mode when the debug interrupt exception processing begins.

- The TCR bit in the CSR may be programmed to force the ColdFire2/2M into emulation mode when trace exception processing begins.

During emulation mode, the ColdFire2/2M's exhibits the following properties:

- All interrupts are ignored, including level seven.

- If the MAP bit in the CSR is set, all memory accesses are forced, including the exception stack frame writes and the vector fetch, into a specially mapped address space signalled by TT = $2, TM = $5 or $6.

- If the MAP bit in the CSR is set, all caching of memory accesses is disabled.

The return-from-exception (RTE) instruction exits emulation mode. The processor status output port provides a unique encoding for emulator mode entry ($D) and exit ($7).

7.4.1.2 REUSE OF DEBUG MODULE HARDWARE. The debug module implementation provides a common hardware structure for both BDM and breakpoint functionality. Several structures are used for both BDM and breakpoint purposes. Table 7-9 identifies the shared hardware structures.

Table 7-9. Shared BDM/Breakpoint Hardware

REGISTER	BDM FUNCTION	BREAKPOINT FUNCTION
AATR	Bus attributes for all memory commands	Attributes for address breakpoint
ABHR	Address for all memory commands	Address for address breakpoint
DBR	Data for All BDM write commands	Data for data breakpoint

The shared use of these hardware structures means the loading of the register to perform any specified function is destructive to the shared function. For example, if an operand address breakpoint is loaded into the debug module, a BDM command to access memory overwrites the breakpoint. If a data breakpoint is configured, a BDM write command overwrites the breakpoint contents.

7.4.2 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 nine registers to support the

required functionality. All of these registers are treated as 32-bit quantities, regardless of the actual number of bits in the implementation. The registers, known as the debug control registers, are accessed through the BDM port using two new BDM commands: WDMREG and RDMREG. These commands contain a 4-bit field, DRc, which specifies the particular register being accessed.

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 may be accessed by the external development system using the serial interface, or by the operating system running on the processor core. It is the responsibility of the software to guarantee that all accesses to these resources are serialized and 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 = 1). The BDM commands should not be issued whenever the ColdFire2/2M is accessing the debug module registers using the WDEBUG instruction.

Figure 7-8 illustrates the debug module programming model.
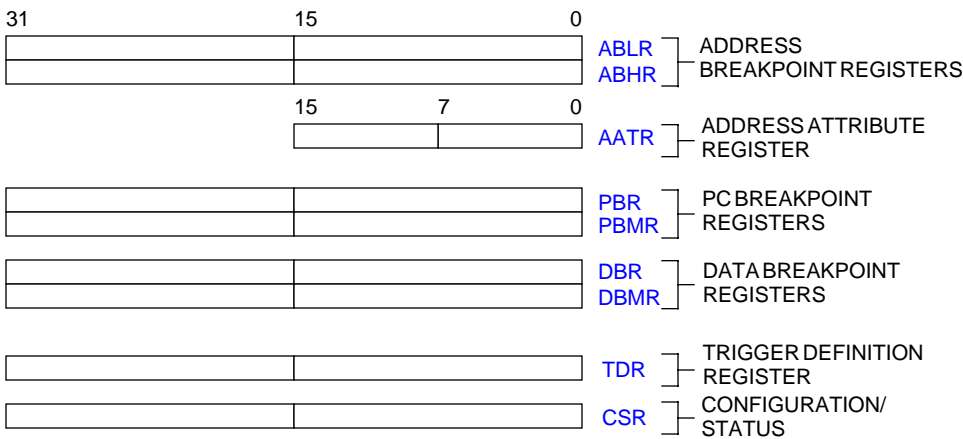
Figure 7-8. Debug Programming Model

7.4.2.1 ADDRESS BREAKPOINT REGISTERS (ABLR, ABHR). The address breakpoint registers define a region in the operand address space of the processor that can be used as part of the trigger. The full 32-bits of the ABLR and ABHR values are compared with the internal address signals of the ColdFire2/2M. The trigger definition register (TDR) determines if the trigger is the inclusive range bound by ABLR and ABHR, all addresses outside this range, or the address in ABLR only. The ABHR is accessible in supervisor mode as debug control register $C using the WDEBUG instruction and via the BDM port using the RDMREG and WDMREG commands. The ABLR is accessible in supervisor mode as debug control register $D using the WDEBUG instruction and via the BDM port using the WDMREG commands. The ABHR is overwritten by the BDM hardware when accessing memory as described in **Section 7.4.1.2 Reuse of Debug Module Hardware**.

BITS	31	0
FIELD	ADDRESS	
RESET	-	
R/W	W	

Address Breakpoint Low Register (ABLR)

Field Definition:

ADDRESS[31:0]–Low Address

This field contains the 32-bit address which marks the lower bound of the address breakpoint range.

BITS	31	0
FIELD	ADDRESS	
RESET	-	
R/W	W	

Address Breakpoint High Register (ABHR)

Field Definition:

ADDRESS[31:0]–High Address

This field contains the 32-bit address which marks the upper bound of the address breakpoint range.

7.4.2.2 ADDRESS ATTRIBUTE REGISTER (AATR). The AATR defines the address attributes and a mask to be matched in the trigger. The AATR value is compared with the internal address attribute signals of the ColdFire2/2M, as defined by the setting of the TDR. The AATR is accessible in supervisor mode as debug control register $6 using the WDEBUG instruction and via the BDM port using the WDMREG command. The lower five bits of the AATR are also used for BDM command definition to define the address space for memory references as described in **Section 7.4.1.2 Reuse of Debug Module Hardware**.

BITS	15	14 13	12 11	10 8	7	6 5	4 3	2 0
FIELD	RM	SZM	TTM	TMM	R	SZ	TT	TM
RESET	0	0	0	0	0	0	0	101
R/W	W	W	W	W	W	W	W	W

Address Attribute Register (AATR)

Field Definitions:

RM[15]–Read/Write Mask

This field corresponds to the R-field. Setting this bit causes R to be ignored in address comparisons.

SZM[14:13]–Size Mask

This field corresponds to the SZ field. Setting a bit in this field causes the corresponding bit in SZ to be ignored in address comparisons.

TTM[12:11]–Transfer Type Mask

This field corresponds to the TT field. Setting a bit in this field causes the corresponding bit in TT to be ignored in address comparisons.

TMM[10:8]–Transfer Modifier Mask

This field corresponds to the TM field. Setting a bit in this field causes the corresponding bit in TM to be ignored in address comparisons.

R[7]–Read/Write

This field is compared with the internal R/W signal of the ColdFire2/2M. A high level indicates a read cycle and a low level indicates a write cycle.

SZ[6:5]—Size

This field is compared to the internal size signals of the ColdFire2/2M. These signals indicate the data size for the bus transfer.

    00 = Longword
    01 = Byte
    10 = Word
    11 = Reserved

TT[4:3]—Transfer Type

This field is compared with the internal TT signals of the ColdFire2/2M. These signals indicate the transfer type for the bus transfer. These signals are always encoded as if the ColdFire2/2M is in the ColdFire IACK mode (see Section 3.1.15 Master Transfer Type (MTT[1:0]).)

    00 = ColdFire2/2M Access
    01 = Reserved
    10 = Emulator Mode Access
    11 = Acknowledge/CPU Space Access

These bits also define the TT encoding for BDM memory commands. In this case, the 01 encoding generates an alternate master access.

TM[2:0]—Transfer Modifier

This field is compared with the internal TM signals of the ColdFire2/2M. These signals provide supplemental information for each transfer type. These signals are always encoded as if the ColdFire2/2M is in the ColdFire IACK mode (see Section 3.1.13 Master Transfer Modifier (MTM[2:0]).) The encoding for normal ColdFire2/2M transfers is:

    000 = Reserved
    001 = User Data Access
    010 = User Code Access
    011 = Reserved
    100 = Reserved
    101 = Supervisor Data Access
    110 = Supervisor Code Access
    111 = Reserved

The encoding for emulator mode transfers is:

    0xx = Reserved
    100 = Reserved
    101 = Emulator Mode Data Access
    110 = Emulator Mode Code Access
    111 = Reserved

The encoding for acknowledge/CPU space transfers is:

    000 = CPU Space Access
    001 = Interrupt Acknowledge Level 1
    010 = Interrupt Acknowledge Level 2
    011 = Interrupt Acknowledge Level 3
    100 = Interrupt Acknowledge Level 4
    101 = Interrupt Acknowledge Level 5
    110 = Interrupt Acknowledge Level 6
    111 = Interrupt Acknowledge Level 7

These bits also define the TM encoding for BDM memory commands.

7.4.2.3 PROGRAM COUNTER BREAKPOINT REGISTER (PBR, PBMR). The PC breakpoint registers define a region in the code address space of the processor that can be used as part of the trigger. The PBR value is masked by the PBMR value, allowing only those bits in PBR that have a corresponding zero in PBMR to be compared with the processor's program counter register, as defined in the TDR. The PBR is accessible in supervisor mode as debug control register $8 using the WDEBUG instruction and via the BDM port using the RDMREG and WDMREG commands. The PBMR is accessible in supervisor mode as debug control register $9 using the WDEBUG instruction and via the BDM port using the WDMREG command.

BITS	31	0
FIELD	ADDRESS	
RESET	-	
R/W	W	

Program Counter Breakpoint Register (PBR)

Field Definition:

ADDRESS[31:0]–PC Breakpoint Address

This field contains the 32-bit address to be compared with the PC as a breakpoint trigger.

BITS	31	0
FIELD	MASK	
RESET	-	
R/W	W	

Program Counter Breakpoint Mask Register (PBMR)

Field Definition:

MASK[31:0]–PC Breakpoint Mask

This field contains the 32-bit mask for the PC breakpoint trigger. A zero in a bit position causes the corresponding bit in the PBR to be compared to the appropriate bit of the PC. A one causes that bit to be ignored.

7.4.2.4 DATA BREAKPOINT REGISTER (DBR, DBMR). The data breakpoint registers define a specific data pattern that can be used as part of the trigger into debug mode.The DBR value is masked by the DBMR value, allowing only those bits in DBR that have a corresponding zero in DBMR to be compared with internal data bus of the ColdFire2/2M, as defined in the TDR. The DBR is accessible in supervisor mode as debug control register $E using the WDEBUG instruction and via the BDM port using the RDMREG and WDMREG commands. The DBMR is accessible in supervisor mode as debug control register $F using the WDEBUG instruction and via the BDM port using the WDMREG command. The DBR is overwritten by the BDM hardware when accessing memory as described in **Section 7.4.1.2 Reuse of Debug Module Hardware**.

BITS	31	0
FIELD	ADDRESS	
RESET	-	
R/W	W	

Data Breakpoint Register (DBR)

Field Definition:

ADDRESS[31:0]–Data Breakpoint Value

This field contains the 32-bit value to be compared with the internal data bus as a breakpoint trigger.

BITS	31	0
FIELD	MASK	
RESET	-	
R/W	W	

Data Breakpoint Mask Register (DBMR)

Field Definition:

MASK[31:0]–Data Breakpoint Mask

This field contains the 32-bit mask for the data breakpoint trigger. A zero in a bit position causes the corresponding bit in the DBR to be compared to the appropriate bit of the internal data bus. A one causes that bit to be ignored.

The data breakpoint register supports both aligned and misaligned operand references. The relationship between the processor address, the access size, and the corresponding location within the 32-bit data bus is shown in Table 7-10.

Table 7-10. Misaligned Data Operand References

ADDRESS[1:0]	ACCESS SIZE	OPERAND LOCATION
00	Byte	Data[31:24]
01	Byte	Data[23:16]
10	Byte	Data[15:8]
11	Byte	Data[7:0]
00	Word	Data[31:16]
10	Word	Data[15:0]
00	Long	Data[31:0]

7.4.2.5 TRIGGER DEFINITION REGISTER (TDR). The TDR configures the operation of the hardware breakpoint logic within the debug module and controls the actions taken under the defined conditions. The breakpoint logic may be configured as a one- or two-level trigger, where bits [31:16] of the TDR define the 2nd level trigger and bits [15:0] define the first level trigger. The TDR is accessible in supervisor mode as debug control register $7 using the WDEBUG instruction and via the BDM port using the WDMREG command.

BITS	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	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	W		W	W	W	W	W	W	W	W	W	W	W	W	W	W

BITS	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	-		0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	-		W	W	W	W	W	W	W	W	W	W	W	W	W	W

Trigger Definition Register (TDR)

Field Definitions:

TRC–Trigger Response Control

The trigger response control determines how the processor is to respond to a completed trigger condition. The trigger response is always displayed on the DDATA pins.

> 00 = display on DDATA only
> 01 = processor halt
> 10 = debug interrupt
> 11 = reserved

EBL–Enable Breakpoint Level

If set, this bit serves as the global enable for the breakpoint trigger. If cleared, all breakpoints are disabled.

EDLW–Enable Data Breakpoint for the Data Longword

If set, this bit enables the data breakpoint based on the entire internal data bus. The assertion of any of the ED bits enables the data breakpoint. If all bits are cleared, the data breakpoint is disabled.

EDWL–Enable Data Breakpoint for the Lower Data Word

If set, this bit enables the data breakpoint based on the low-order word of the internal data bus.

EDWU–Enable Data Breakpoint for the Upper Data Word

If set, this bit enables the data breakpoint trigger based on the high-order word of the internal data bus.

EDLL–Enable Data Breakpoint for the Lower Lower Data Byte

If set, this bit enables the data breakpoint trigger based on the low-order byte of the low-order word of the internal data bus.

EDLM–Enable Data Breakpoint for the Lower Middle Data Byte

If set, this bit enables the data breakpoint trigger based on the high-order byte of the low-order word of the internal data bus.

EDUM–Enable Data Breakpoint for the Upper Middle Data Byte

If set, this bit enables the data breakpoint trigger on the low-order byte of the high-order word of the internal data bus.

EDUU–Enable Data Breakpoint for the Upper Upper Data Byte

If set, this bit enables the data breakpoint trigger on the high-order byte of the high-order word of the internal data bus.

DI–Data Breakpoint Invert

This bit provides a mechanism to invert the logical sense of all the data breakpoint comparators. This can develop a trigger based on the occurrence of a data value not equal to the one programmed into the DBR.

EAI–Enable Address Breakpoint Inverted

If set, this bit enables the address breakpoint based outside the range defined by ABLR and ABHR. The assertion of any of the EA bits enables the address breakpoint. If all three bits are cleared, this breakpoint is disabled.

EAR–Enable Address Breakpoint Range

If set, this bit enables the address breakpoint based on the inclusive range defined by ABLR and ABHR.

EAL–Enable Address Breakpoint Low

If set, this bit enables the address breakpoint based on the address contained in the ABLR.

EPC–Enable PC Breakpoint

If set, this bit enables the PC breakpoint.

PCI–PC Breakpoint Invert

If set, this bit allows execution outside a given region as defined by PBR and PBMR to enable a trigger. If cleared, the PC breakpoint is defined within the region defined by PBR and PBMR.

7.4.2.6 CONFIGURATION/STATUS REGISTER (CSR). The CSR defines the operating configuration for the processor and memory subsystem. In addition to defining the microprocessor configuration, this register also contains status information from the breakpoint logic. The CSR is cleared during system reset. The CSR can be read and written to by the external development system and written to by the supervisor programming model.

The CSR is accessible in supervisor mode as debug control register $0 using the WDEBUG instruction and via the BDM port using the RDMREG and WDMREG commands.

BITS	31 28	27	26	25	24	23 17	16
FIELD	STATUS	FOF	TRG	HALT	BKPT	-	IPW
RESET	0	0	0	0	0	-	0
R/W[1]	R	R	R	R	R	-	R/W

BITS	15	14	13	12 11	10	9 8	7	6	5	4	3 0
FIELD	MAP	TRC	EMU	DDC	UHE	BTB	-	NPL	IPI	SSM	-
RESET	0	0	0	0	0	0	0	0	0	0	-
R/W†	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	R/W	R/W	-

NOTE: †The CSR is a write only register from the programming model. It can be read from and written to via the BDM port.

Configuration/Status Register (CSR)

Field Definitions:

STATUS[31:28]–Breakpoint Status

This 4-bit field provides read-only status information concerning the hardware breakpoints. This field is defined as follows:

    000x = no breakpoints enabled
    001x = waiting for level 1 breakpoint
    010x = level 1 breakpoint triggered
    101x = waiting for level 2 breakpoint
    110x = level 2 breakpoint triggered

The breakpoint status is also output on the DDATA port when not busy displaying other ColdFire2/2M data. A write to the TDR resets this field.

FOF[27]–Fault-on-Fault

If this read-only status bit is set, a catastrophic halt has occurred and forced entry into BDM. This bit is cleared on a read from the CSR.

TRG[26]–Hardware Breakpoint Trigger

If this read-only status bit is set, a hardware breakpoint has halted the processor core and forced entry into BDM. This bit is cleared by reading CSR, or when the processor is restarted.

HALT[25]–Processor Halt

If this read-only status bit is set, the processor has executed the HALT instruction and forced entry into BDM. This bit is cleared by reading the CSR, or when the processor is restarted.

BKPT[24]–Breakpoint Assert

If this read-only status bit is set, the BKPTB signal was asserted, forcing the processor into BDM. This bit is cleared on a read from the CSR, or when the processor is restarted.

IPW[16]–Inhibit Processor Writes to Debug Registers

If set, this bit inhibits any processor-initiated writes to the debug module's programming model registers. This bit can only be modified by commands from the external development system.

MAP[15]–Force Processor References in Emulator Mode

If set, this bit forces the processor to map all references while in emulator mode to a special address space, TT = \$2, TM = \$5 (data) or \$6 (text). If cleared, all emulator-mode references are mapped into supervisor code and data spaces.

TRC[14]–Force Emulation Mode on Trace Exception

If set, this bit forces the processor to enter emulator mode when a trace exception occurs.

EMU[13]–Force Emulation Mode

If set, this bit forces the processor to begin execution in emulator mode when a trace exception occurs. Refer to **Section 7.4.1.1 Emulator Mode**.

DDC[12:11]–Debug Data Control

This 2-bit field provides configuration control for capturing operand data for display on the DDATA port. The encoding is:

    00 = no operand data is displayed
    01 = capture all M-Bus write data
    10 = capture all M-Bus read data
    11 = capture all M-Bus read and write data

In all cases, the DDATA port displays the number of bytes defined by the operand reference size, i.e., byte displays 8 bits, word displays 16 bits, and long displays 32 bits (one nibble at a time across multiple clock cycles.) Refer to **Section 7.2.1.7 Begin Data Transfer (PST = \$8 - \$A)**.

UHE[10]-User Halt Enable

This bit selects the CPU privilege level required to execute the HALT instruction.

    0 = HALT is a privileged, supervisor-only instruction
    1 = HALT is a non-privileged, supervisor/user instruction

BTB[8:9]–Branch Target Bytes

This 2-bit field defines the number of bytes of branch target address to be displayed on the DDATA outputs. The encoding is

> 00 = 0 bytes
> 01 = lower two bytes of the target address
> 10 = lower three bytes of the target address
> 11 = entire four-byte target address

Refer to **Section 7.2.1.5 Begin Execution of Taken Branch (PST = $5)**.

NPL[6]–Non-pipelined Mode

If set, this bit forces the processor core to operate in a nonpipeline mode of operation. In this mode, the processor effectively executes a single instruction at a time with no overlap.

IPI[5]–Ignore Pending Interrupts

If set, this bit forces the processor core to ignore any pending interrupt requests signalled while executing in single-instruction-step mode.

SSM[4]–Single-Step Mode

If set, this bit forces the processor core to operate in a single-instruction-step mode. While in this mode, the processor executes a single instruction and then halts. While halted, any of the BDM commands may be executed. On receipt of the GO command, the processor executes the next instruction and then halts again. This process continues until the single-instruction-step mode is disabled.

Reserved - All bits labelled "Reserved" or "0" are currently unused and are reserved for future use. These bits should always be written as "0".

7.4.3 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 for the 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 ColdFire2/2M's internal bus. The processor responds by stalling the instruction fetch pipeline and then waiting until all current bus activity is complete. At that time, the processor relinquishes the internal bus to allow the debug module to perform the required operation. After the conclusion of the debug module bus cycle, the processor reclaims ownership of the bus. By implementing this scheduling mechanism, the processor can minimize the amount of intrusion caused by debug module requests.

Under certain conditions, the processor may never grant the processor's internal bus to the debug module causing the BDM command to never be performed. Specifically, the

processor's internal bus grant may be withheld from the debug module if the processor is executing a tight loop where the entire loop is contained within one aligned longword.

Examples include:

```
                align  4
label1:         nop
                bra.b  label1

                align  4
label2:         bra.w  label2
```

The solution to this scheduling problem is to force the loop to be aligned ACROSS a longword boundary. Given this alignment, the processor will always correctly grant the processor's internal bus to the debug module.

The development system must use caution in configuring the Breakpoint Registers if the processor is executing. The debug module does not contain any hardware interlocks, so Motorola recommends that the TDR be disabled while the Breakpoint Registers are being loaded. At the conclusion of this process, the TDR can be written to define the exact trigger. This approach guarantees that no spurious breakpoint triggers will occur.

Because there are no hardware interlocks in the debug unit, no BDM operations are allowed while the CPU is writing the debug registers (DSCLK must be inactive).

7.4.4 Motorola Recommended BDM Pinout

The ColdFire BDM connector is a 26-pin Berg Connector arranged 2x13, shown in Figure 7-9.

Freescale Semiconductor, Inc.
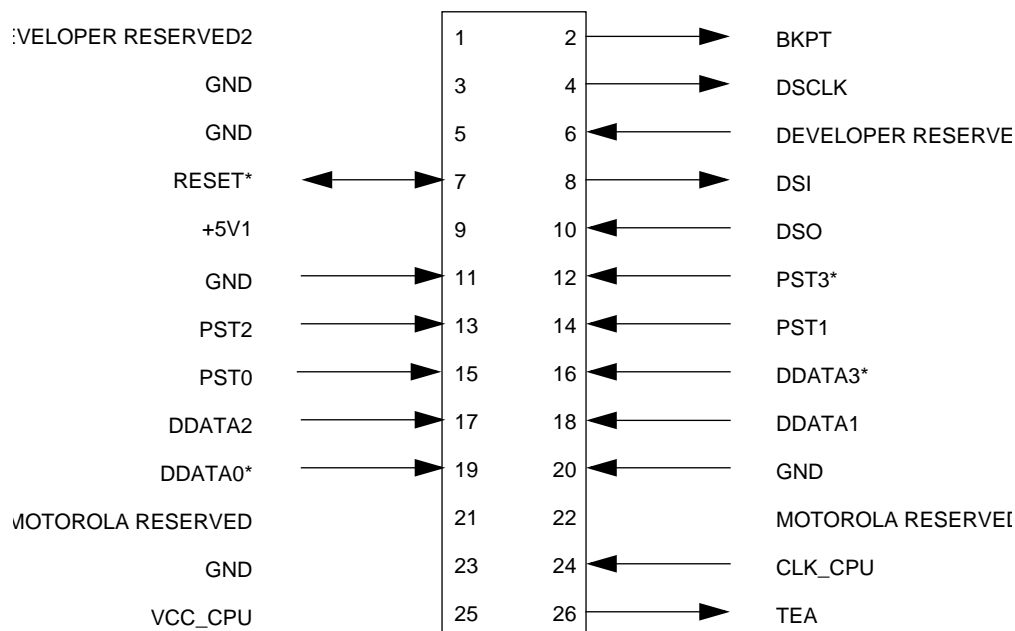
NOTES: 1.   Supplied by target

2.   Pins reserved for BDM developer use. Contact developer.

3.   * Denotes a vectored signal.

Figure 7-9. Recommended BDM Connector

Table 7-11 shows the correlation between the standard ColdFire connector and the ColdFire2/2M's signals.

Table 7-11. BDM Connector Correlation

CONNECTOR	COLDFIRE2/2M	CONNECTOR	COLDFIRE2/2M
+5V	+5V	DSO	DSO
BKPT	BKPTB	GND	GND
CLK_CPU	CLK	PST0	PST[0]
DDATA0	DDATA[0]	PST1	PST[1]
DDATA1	DDATA[1]	PST2	PST[2]
DDATA2	DDATA[2]	PST3	PST[3]
DDATA3	DDATA[3]	RESET	MRSTB
DSCLK	DSCLK	TEA	MTEAB
DSI	DSI	Vcc_CPU	V_{DD}

7.4.5 Differences Between the ColdFire2/2M BDM and CPU32 BDM

1. DSCLK, $\overline{\text{BKPT}}$, and DSI need to meet the setup and hold times relative to the rising edge of the processor clock to prevent the processor from propagating metastable states.

2. DSO transitions relative to the rising edge of DSCLK only. In the CPU32 BDM, DSO transitions between serial transfers to indicate to the development system that a command has successfully completed. The ColdFire BDM does not support this feature.

3. The development system must note that the DSO is not valid during the first rising edge of DSCLK. Instead, the first rising edge of DSCLK causes DSO to transmit the MSB of DSO. A serial transfer is illustrated in Figure 7-3.

SECTION 8
TEST OPERATION

The ColdFire2/2M test stategy includes:

1. At-speed parallel scan testing via 16 parallel scan chains to provide high fault coverage

2. The use of an I/O test ring to enable scan testing of the embedded CPU and the surrounding ASIC logic

3. A methodology to provide testing of the integrated memories connected to the CPU with a minimal required pinout.

The following sections describe the signals and methodologies for:

- Scan Testing
- Integrated Memory Testing

IDDQ testing can also be performed contingent upon the ability to constrain the design to avoid conditions that cause high leakage currents.

8.1 SIGNALS REQUIRED TO PERFORM SCAN TEST

This section describes the ColdFire2/2M signals dedicated to the scan testing of the ColdFire2/2M. These signals are required to be muxed out to package pins. All ColdFire2/2M signals are unidirectional and synchronous.

8.1.1 Scan Enable (SCAN_ENABLE)

This active-high input signal enables scan testing of the ColdFire2/2M. It forces all internal flip-flops to be linked together into sixteen parallel scan chains.

8.1.2 Scan Exercise Array (SCAN_XARRAY)

This active-high input signal is used to exercise the integrated memory arrays during scan testing. This signal causes random writes to the internal RAMs by strobing the write strobes while scanning. The array output data does not affect the scan via use of the SCAN_MODE pin.

8.1.3 Scan Input (SI[15:0])

These input signals are connected to the16 internal ColdFire2/2M scan chain inputs.

8.1.4 Scan Mode (SCAN_MODE).

This active-high, unidirection input signal gates off all memory array outputs during scan testing.SCAN_MODE must be asserted for the duration of scan testing.

8.1.5 Scan Output (SO[15:0])

These output signals are connected to the 16 internal ColdFire2/2M scan chain outputs.

8.1.6 I/O Test Ring Clock (TRCLK)

This input signal is the synchronous clock used to transition the test ring during scan testing. TR_CLK is connected to the clock input of all I/O test ring registers.

8.1.7 I/O Test Ring Core Mode Enable (CORE_TEST)

This active-high input signal enables the core mode of the test ring during scan testing. The test ring is in scan core mode if CORE_TEST is asserted and in scan ASIC mode if CORE_TEST is negated.

8.1.8 I/O Test Ring Data Input (TR_SDI[1:0])

These input signals are the serial data inputs for the I/O test ring chains.

8.1.9 I/O Test Ring Data Output (TR_SDO[1:0])

These output signals are the serial output data from the I/O test ring chains.

8.1.10 I/O Test Ring Enable (TR_SE)

This active-high input signal enables the test ring. TR_SE is connected to the scan enable input of all I/O test ring scannable registers.

8.1.11 I/O Test Ring Mode (TR_MODE)

This active-high input signal enables the scan test mode of the test ring. The test ring is in scan test mode if TR_MODE is asserted and in normal functional mode if negated. TR_MODE should be asserted for the duration of scan testing, and be held negated for the duration of memory testing and during functional operation of the device.

8.1.12 TEST WRITE INHIBIT (TEST_WR_INH).

Optional: Asserting this signal will prevent strobing; i.e. writing, of any of the integrated memories. However, as long as SCAN_MODE is asserted, the array outputs are gated off from the ColdFire2/2M and will not affect the scan vectors.

8.2 SCAN OPERATION

Motorola provides ATPG vectors for the ColdFire2/2M. The signals listed in the previous section must be brought out to package pins (muxed out) in order for Motorola supplied scan vectors to be applied. This section provides an understanding of the ColdFire2/2M scan implementation. The following diagram illustrates the ColdFire2/2M scan test implementation:
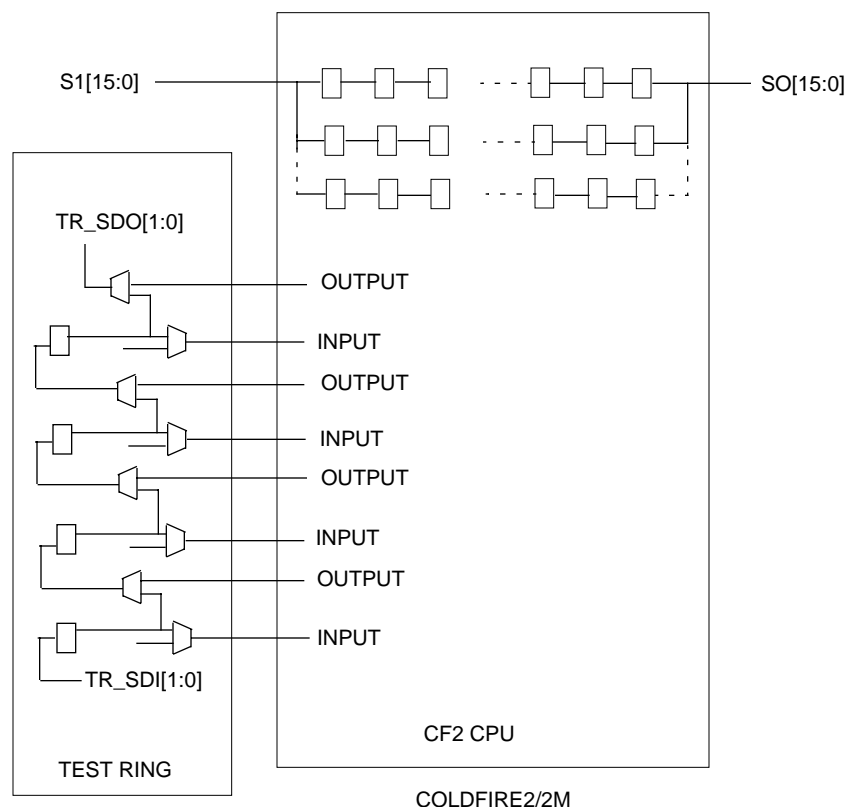
Figure 8-1. Scan Test Implementation Diagram

SI/SO refer to the 16 parallel scan chains that cover the registers in ColdFire2/
2M.SCAN_EN controls these chains. If these pins, plus all other inputs and outputs of
ColdFire2/2M were muxed to package pins, ATPG coverage of the core would be enabled.
The I/O test ring allows stimulus to be applied to embedded ColdFire2/2M inputs and allows
ColdFire2/2M embedded outputs to be observed with a minimum of pins muxed to the chip
periphery. SI[15:0] is used to load the ColdFire2/2M parallel scan chains while TR_SDI[1:0]
set up the appropriate values on the ColdFire2/2M inputs for each scan vector. SO[15:0] and
TR_SDO[1:0] are used to observe and verify the scan behavior.

The I/O test ring isolates the ASIC logic from the ColdFire2/2M enabling separate scan
testing of each. It is comprised of two scan chains, each containing 44 registers (half
scannable, half not; see Table 8-3, Table 8-4 and Figure 8-2). The heads of the chains are
TR_SDI[0] and TR_SDI[1]. The tails of the chains are TR_SDO[0] and TR_SDO[1]. The I/O
test ring also enables input and output spec testing as follows: the clocks (TR_CLK and
CLK) can be skewed to verify that a transition launched from the scan ring is captured at the
input register in a specified period of time. This is then verified by capturing outputs of the
ColdFire2/2M and shifting the scan data out to compare to expected values. This is also how
output specs are verified: the clocks can be skewed a specified amount, and an output
transition can be captured by the scan ring. Again, the data is shifted out and values
compared to expected values. See Table 8-1 for scan chain I/O and length information.

Freescale Semiconductor, Inc.

In functional mode, the I/O test ring adds a 2:1 multiplexor delay to the inputs and outputs of the core. Each cell in the I/O test ring pairs an input and an output which allows for the sharing of registers.
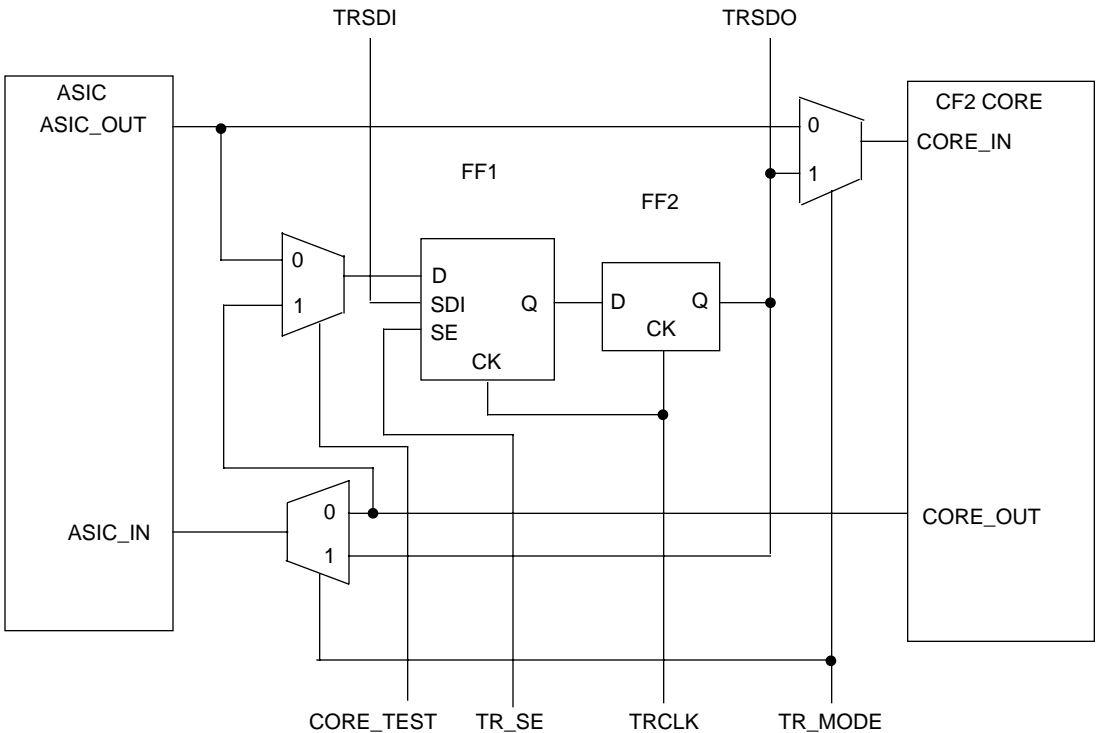
Figure 8-2. I/O Test Ring Cell

Table 8-1. Parallel Scan Chain Information

SCAN CHAIN DESIGNATOR	SCAN CHAIN INPUT	SCAN CHAIN OUTPUT	LENGTH
0	SI[0]	SO[15]	99
1	SI[1]	SO[14]	99
2	SI[2]	SO[13]	99
3	SI[3]	SO[12]	99
4	SI[4]	SO[11]	99
5	SI[5]	SO[10]	99
6	SI[6]	SO[9]	99
7	SI[7]	SO[8]	99
8	SI[8]	SO[7]	99
9	SI[9]	SO[6]	99
10	SI[10]	SO[5]	99
11	SI[11]	SO[4]	99
12	SI[12]	SO[3]	99
13	SI[13]	SO[2]	99
14	SI[14]	SO[1]	99
15	SI[15]	SO[0]	84
I/O TEST RING 0	TRSDI[0]	TRSDO[0]	88
I/O TEST RING 1	TRSDI[1]	TRSDO[1]	88

The I/O test ring behaves in the same manner for the ASIC logic as it does for ColdFire2/2M. Table 8-2 lists the I/O test ring control signal configurations for the different modes of testing. Table 8-3 and Table 8-4 list each signal, which I/O test ring chain it is part of, and how deep (how many cells from TRSDI[1:0] it is in the chain. Note that an X means that the signal needs to be driven, however high or low is optional. The actual value is a don't care.

Table 8-2. I/O TEST RING MODE CONFIGURATIoNS

	LOAD	SCAN ColdFire2/2M	SCAN ASIC	SHIFT OUT	FUNCTIONAL (NON-SCAN)
CORE_TEST	*X	1	0	*X	*X
TR_SE	1	0	0	1	*X
TRCLK	ON	ON	ON	ON	*X
TR_MODE	*X	1	1	*X	0
TR_SDI[1:0]	APPLY DATA	*X	*X	*X	*X

* Needs to driven high or low. The actual value is a don't care.

Table 8-3. I/O TEST RING CHAIN 0 (TRSDI[0]/TRSDO[0])

CORE INPUT	CORE OUTPUT	CELL # (2 REGS/ CELL)	CORE INPUT	CORE OUTPUT	CELL # (2 REGS/ CELL)
MRDATA_IE	MWDATA_OE	HEAD OF I/O TEST RING CHAIN 0	MRDATA[21]	MWDATA[21]	23
MRDATA[0]	MWDATA[0]	2	MRDATA[22]	MWDATA[22]	24
MRDATA[1]	MWDATA[1]	3	MRDATA[23]	MWDATA[23]	25
MRDATA[2]	MWDATA[2]	4	MRDATA[24]	MWDATA[24]	26
MRDATA[3]	MWDATA[3]	5	MRDATA[25]	MWDATA[25]	27
MRDATA[4]	MWDATA[4]	6	MRDATA[26]	MWDATA[26]	28
MRDATA[5]	MWDATA[5]	7	MRDATA[27]	MWDATA[27]	29
MRDATA[6]	MWDATA[6]	8	MRDATA[28]	MWDATA[28]	30
MRDATA[7]	MWDATA[7]	9	MRDATA[29]	MWDATA[29]	31
MRDATA[8]	MWDATA[8]	10	MRDATA[30]	MWDATA[30]	32
MRDATA[9]	MWDATA[9]	11	MRDATA[31]	MWDATA[31]	33
MRDATA[10]	MWDATA[10]	12	MTEAB	MKILLB	34
MRDATA[11]	MWDATA[11]	13	IPLB[0]	MTM[0]	35
MRDATA[12]	MWDATA[12]	14	IPLB[1]	MTM[1]	36
MRDATA[13]	MWDATA[13]	15	IPLB2]	MTM[2]	37
MRDATA[14]	MWDATA[14]	16	ICH_SZ[0]	PST[0]	38
MRDATA[15]	MWDATA[15]	17	ICH_SZ[1]	PST[1]	39
MRDATA[16]	MWDATA[16]	18	ICH_SZ[2]	PST[2]	40
MRDATA[17]	MWDATA[17]	19	SRAM_SZ[0]	DDATA[0]	41
MRDATA[18]	MWDATA[18]	20	SRAM_SZ[1]	DDATA[1]	42
MRDATA[19]	MWDATA[19]	21	SRAM_SZ[2]	DDATA[2]	43
MRDATA[20]	MWDATA[20]	22	ROM_SZ[2]	TEST_RHIT	TAIL OF I/O TEST RING CHAIN 0

Table 8-4. I/O TEST RING CHAIN 1 (TRSDI[1]/TRSDO[1])

CORE INPUT	CORE OUTPUT	CELL # (2 REGS/ CELL)	CORE INPUT	CORE OUTPUT	CELL # (W REGS/ CELL)
NC	mfrzb	HEAD OF I/O TEST RING CHAIN 1	TEST_CTRL	MADDR[18]	23
MRDATA[0]	MARBC[0]	2	TEST_RD	MADDR[19]	24
ROM_SZ[0]	MARBC[1]	3	TEST_DATA_RD	MADDR[20]	25
ROM_SZ[1]	PST[3]	4	TEST_RAM_RD	MADDR[21]	26
MRSTB	DDATA[3]	5	TEST_ROM_RD	MADDR[22]	27
DSCLK	DSO	6	ROM_VLD	MADDR[23]	28
IACK_68k	MRW	7	TEST_IVLD_INH	MADDR[24]	29
TEST_MODE	MTT[0]	8	NC	MADDR[25]	30
TEST_KTA	MTT[1]	9	NC	MADDR[26]	31
TEST_ADDR[2]	MADDR[5]	10	TEST_ITAG_WRT	MADDR[27]	32
TEST_ADDR[3]	MADDR[6]	11	TEST_DATA_WRT	MADDR[28]	33
TEST_ADDR[4]	MADDR[7]	12	TEST_RAM_WRT	MADDR[29]	34
TEST_ADDR[5]	MADDR[8]	13	MTAB	MADDR[30]	35
TEST_ADDR[4]	MADDR[9]	14	BKPTB	MADDR[31]	36
TEST_ADDR[7]	MADDR10]	15	NC	MADDR[0]	37
TEST_ADDR[8]	MADDR[11]	16	NC	MADDR[1]	38
TEST_ADDR[9]	MADDR[12]	17	NC	MADDR[2]	39
TEST_ADDR[10]	MADDR[13]	18	NC	MADDR[3]	40
TEST_ADDR[11]	MADDR[14]	19	NC	MADDR[4]	41
TEST_ADDR[12]	MADDR[15]	20	NC	MSIZ[0]	42
TEST_ADDR[13]	MADDR[16]	21	NC	MSIZ[1]	43
TEST_ADDR[14]	MADDR[17]	22	NC	MTSB	TAIL OF I/O TEST RING CHAIN 1

8.3 INTEGRATED MEMORY TESTING

A test mode is provided to test embedded SRAM, ROM, and/or ICACHE arrays that connect directly to the ColdFire2/2M. All integrated memories are tested by placing the ColdFire2/ 2M into test mode (TEST_MODE = 1) and performing reads and writes via the test bus and system-bus. These busses allow external access to the integrated memories via functional paths. The path from the test/system bus to the memory arrays is a pipelined path. The test bus input signals serve as multiplexor select signals to enable the correct path from MRDATA and TEST_ADDR to a particular array, and from the particular array out through MWDATA. During test mode, ColdFire2/2M is idle and does not execute code.

The following section details the signal descriptions, test methodology, and data transfer mechanisms.

8.3.1 Test Bus Signal Description

This section describes the ColdFire2/2M signals dedicated to testing the integrated memories. Additional signals are required to be either asserted or negated during memory test and are described here also. If all three types of integrated memories are used; ie.

SRAM, ROM, and ICACHE, all signals listed here must be brought out to package pins via muxing. If only one or two types of integrated memories are used, a subset of signals listed here must be brought out to package pins. I.E. some of the test input control pins could be negated instead of needing to be muxed out. Also, depending upon array size, not all of TEST_ADDR may be necessary. All ColdFire2/2M signals are unidirectional and synchronous.

8.3.1.1 TEST ADDRESS BUS (TEST_ADDR[14:2]). These input signals are used to specify an address when testing the integrated memories.

8.3.1.2 TEST CONTROL (TEST_CTRL). This active-high input signal indicates the test address bus (TEST_ADDR[14:2]) will be latched on the next positive clock edge.

8.3.1.3 TEST IDATA READ (TEST_IDATA_RD). This active-high input signal is used to test the instruction cache data memory read operation.

8.3.1.4 TEST IDATA WRITE (TEST_IDATA_WRT). This active-high input signal is used to test the instruction cache data memory write operation.

8.3.1.5 TEST INSTRUCTION CACHE READ HIT (TEST_RHIT). This active-high output signal indicates a hit has occurred when accessing the instruction cache during memory array testing.

8.3.1.6 TEST INVALIDATE INHIBIT (TEST_IVLD_INH). This active-high input signal inhibits the invalidate operation when testing the instruction cache.

8.3.1.7 TEST ITAG WRITE (TEST_ITAG_WRT). This active-high input signal is used to test the instruction cache tag memory write operation.

8.3.1.8 TEST KTA MODE ENABLE (TEST_KTA). This active-high input signal allows the instruction cache tag and data arrays to be read in parallel, mimicking the functional operation. This allows testing of the speed path from the tag and data arrays to the core.

8.3.1.9 TEST MODE ENABLE (TEST_MODE). This active-high input signal is used to enable all of the integrated memory test signals. TEST_MODE should be asserted for the duration of memory testing.

8.3.1.10 TEST SRAM READ (TEST_SRAM_RD). This active-high input signal is used to test the integrated SRAM memory read operation.

8.3.1.11 TEST SRAM WRITE (TEST_SRAM_WRT). This active-high input signal is used to test the integrated SRAM memory write operation.

8.3.1.12 TEST READ (TEST_RD). This active-high input signal is used to test read operations on all of the integrated memories.

8.3.1.13 TEST ROM READ (TEST_ROM_RD). This active-high input signal is used to test the integrated ROM memory read operation.

8.3.1.14 TEST WRITE INHIBIT (TEST_WR_INH). This active-high input signal disables the write strobes to the SRAM and instruction cache compiled RAMS. TEST_WR_INH should be negated for the duration of memory test.

8.3.1.15 MIE. This active-high input signal enables the capturing of MRDATA into the memory arrays. MIE should be asserted for the duration of memory testing.

8.3.1.16 TR_MODE. This signal needs to be negated to keep the I/O test ring in functional mode. TR_MODE should be negated for the duration of memory testing.

8.3.1.17 MWDATA[31:0]. The M-Bus write data bus is used to observe array data during array reads.

8.3.1.18 MRDATA[31:0]. The M-Bus read data bus is used to apply write data during array writes.

8.3.1.19 SCAN_MODE. This signal must be negated to allow array output data into the ColdFire2/2M.

8.3.1.20 SCAN_SE. This signal must be negated to allow functional operation of the ColdFire2/2M.

8.3.1.21 SCAN_XARRAY. This signal should be negated to prevent continuous strobing of the ColdFire2/2M integrated memories.

8.3.2 Memory Test Theory of Operation

A write consists of 4 cycles; i.e. it is not until the fourth cycle after applying TEST_ADDR and MRDATA, with the array's write control signal active, that data is strobed into the array. Data and address can be updated continuously every cycle to fill the array via the pipeline.

A read consists of 6 cycles; i.e. it is not until the sixth cycle after applying TEST_ADDR, with the array's read control signal active, that data appears on MWDATA. Address can be updated continuously every cycle to read the array via the pipeline.

Both the SRAM and ICACHE DATA arrays are accessed as noted above. The ROM is read as noted above. Two test modes exist for the ICACHE TAG array. Essentially the TAG and DATA array are written. Leaving TEST_MODE asserted, TEST_KTA is then asserted and address applied. TEST_RHIT will be asserted 4 cycles later, ICACHE DATA array data will subsequently appear on MWDATA 2 cycles later.

The processor must be placed in reset in accordance with the system reset specification. The sequence (MRSTB asserted 6 cycles, stall 8cycles) must be executed both upon entering test mode at power-up, as well as when switching into test mode from any other mode. It is this sequence that directs the CPU to go into an idle mode to allow integrated memory testing.

8.3.3 Test Mode

The test mode is entered by asserting TEST_MODEand MIE, negating TEST_CTRL, TEST_WR_INH, and TR_MODE, and resetting the ColdFire2/2M as described in **Section 3.9 Reset Operation.** MRSTB must be asserted (low) >= 6 cycles. The first test sequence should not begin until at least eight clock cycles after MRSTB is negated. TEST_MODE and MIE should remain asserted during all test sequences. TEST_WR_INH and TR_MODE should remain negated during all test sequences.

8.3.4 Instruction Cache Tag RAM Testing

The instruction cache tag RAM consists of up to 2K addresses that can be accessed individually through the test bus. Testing the compiled tag RAM is accomplished by first writing into the tag RAM to set the valid bit, and reading the tag RAM to check for a cache hit.

8.3.4.1 INSTRUCTION CACHE TAG RAM WRITE FUNCTION. Writing to the instruction cache tag RAM is performed though the test bus and MRDATA[31:0] after entering test mode. The address and control signals are input on the test bus and the data to be written to the tag RAM is input on MRDATA[31:0].

Writes to the tag RAM are performed in a pipelined fashion as shown in Figure 8-3. All input signals are latched on the positive edge of CLK and all outputs transition on the positive edge of CLK.
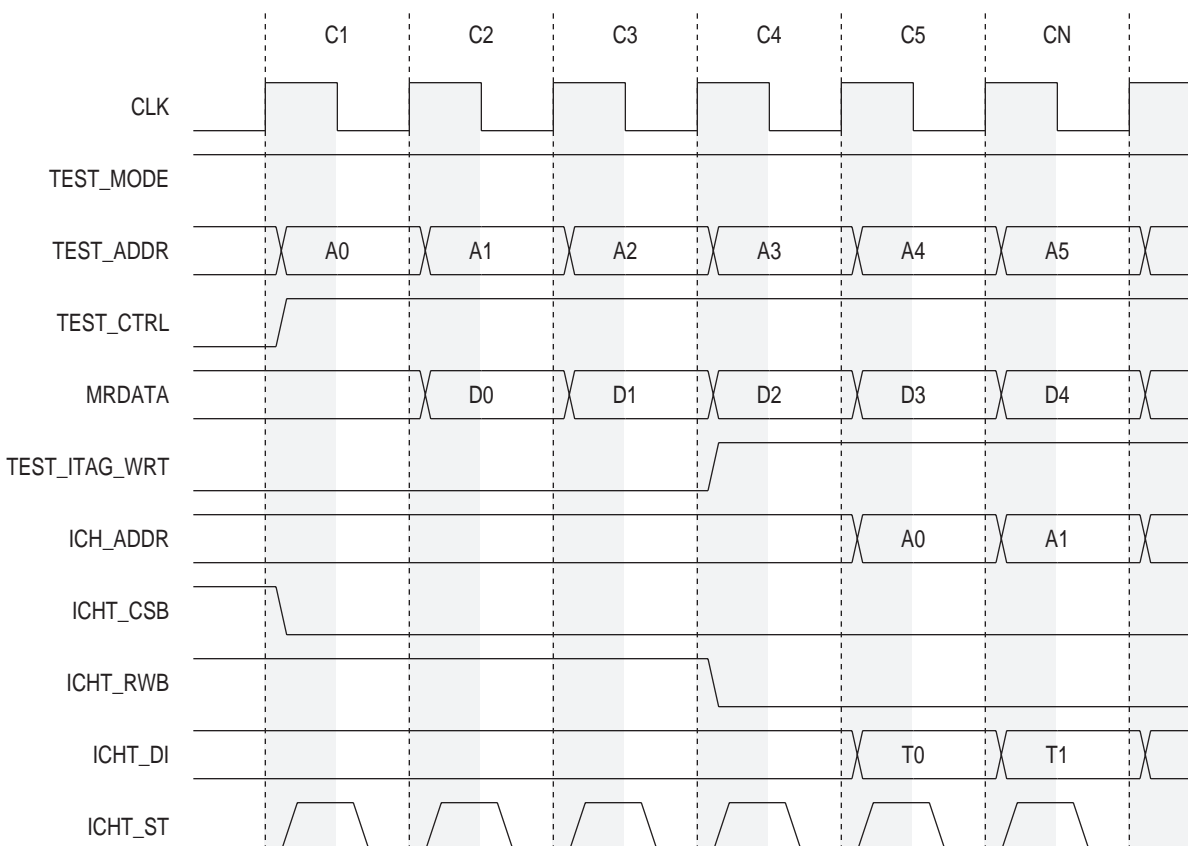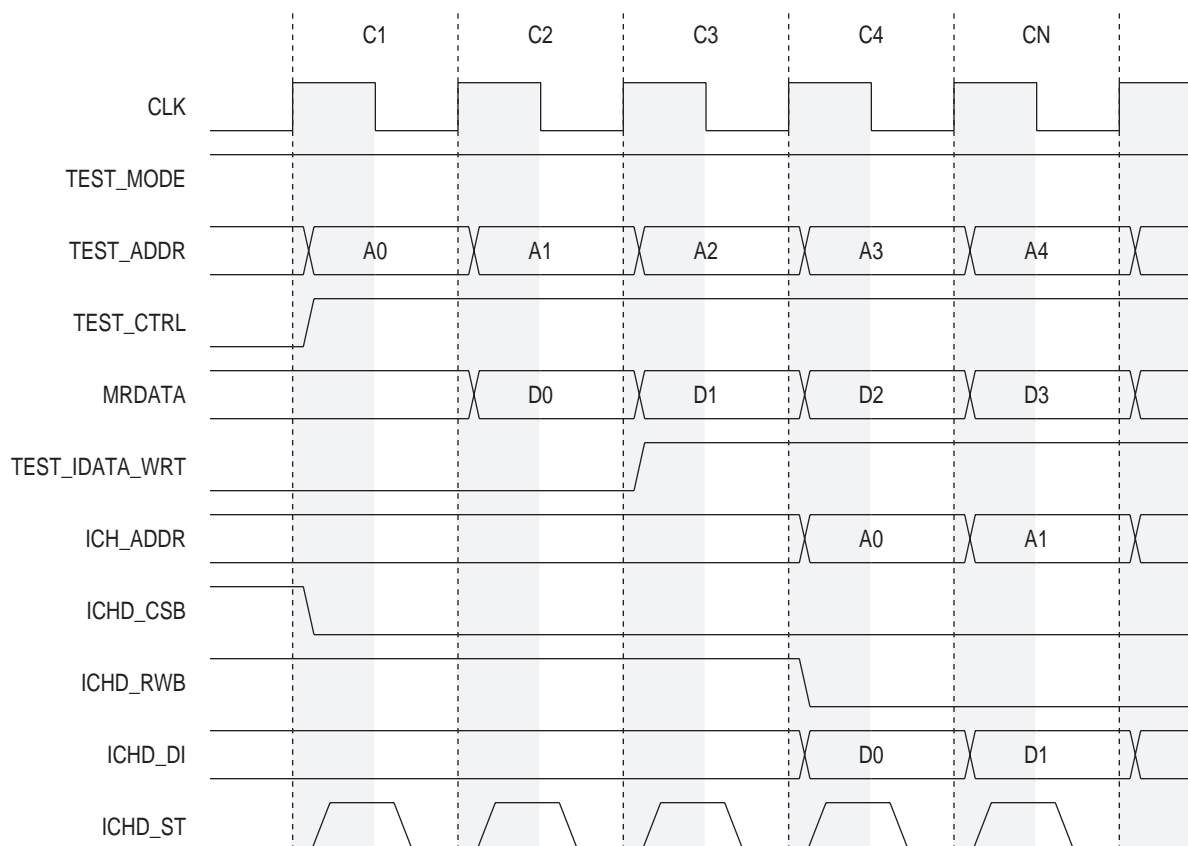
Figure 8-3. Test Instruction Cache Tag Write Cycles

Clock 1 (C1)

On the first clock cycle of the tag RAM write sequence, the first tag RAM address should be driven onto TEST_ADDR[14:2] and TEST_CTRL should be asserted.

Clock 2 (C2)

On the second cycle, the first data value associated with the address asserted in C1 should be driven onto MRDATA[31:0] and the next address should be driven onto TEST_ADDR[14:2].

Clock 3 (C3)

On the third cycle, the next address and data values should be driven.

Clock 4 (C4)

On the fourth cycle, the next address and data values should be driven and TEST_ITAG_WRT should be asserted. The remaining addresses and data are driven each successive clock cycle and TEST_ITAG_WRT should remain asserted until the last data value has been latched.

Clock 5 (C5)

The fifth clock cycle is identical to C4. The first write to the tag RAM occurs.

Clock n (Cn)

The remaining clock cycles in the test are identical to C5.

Throughout the entire sequence, the data value being driven is associated with the address driven in the previous clock cycle. The format of the data value written to the tag RAM depends on the cache configuration. The data value consists of the significant upper bits latched on MRDATA with the valid bit (bit eight) set. See **Section 5.1 Instruction Cache** for more information.

8.3.4.2 INSTRUCTION CACHE TAG RAM READ FUNCTION. Reading from the instruction cache tag RAM is not performed as a typical input address, read data operation. Instead, reading the instruction cache tag RAM is performed though the test bus and MRDATA[31:0] after entering test mode. The address and control signals are input on the test bus and the expected value of the data from the tag RAM is input on MRDATA[31:0].

Reads from the tag RAM are performed in a pipelined fashion as shown in Figure 8-4. All input signals are latched on the positive edge of CLK and all outputs transition on the positive edge of CLK.
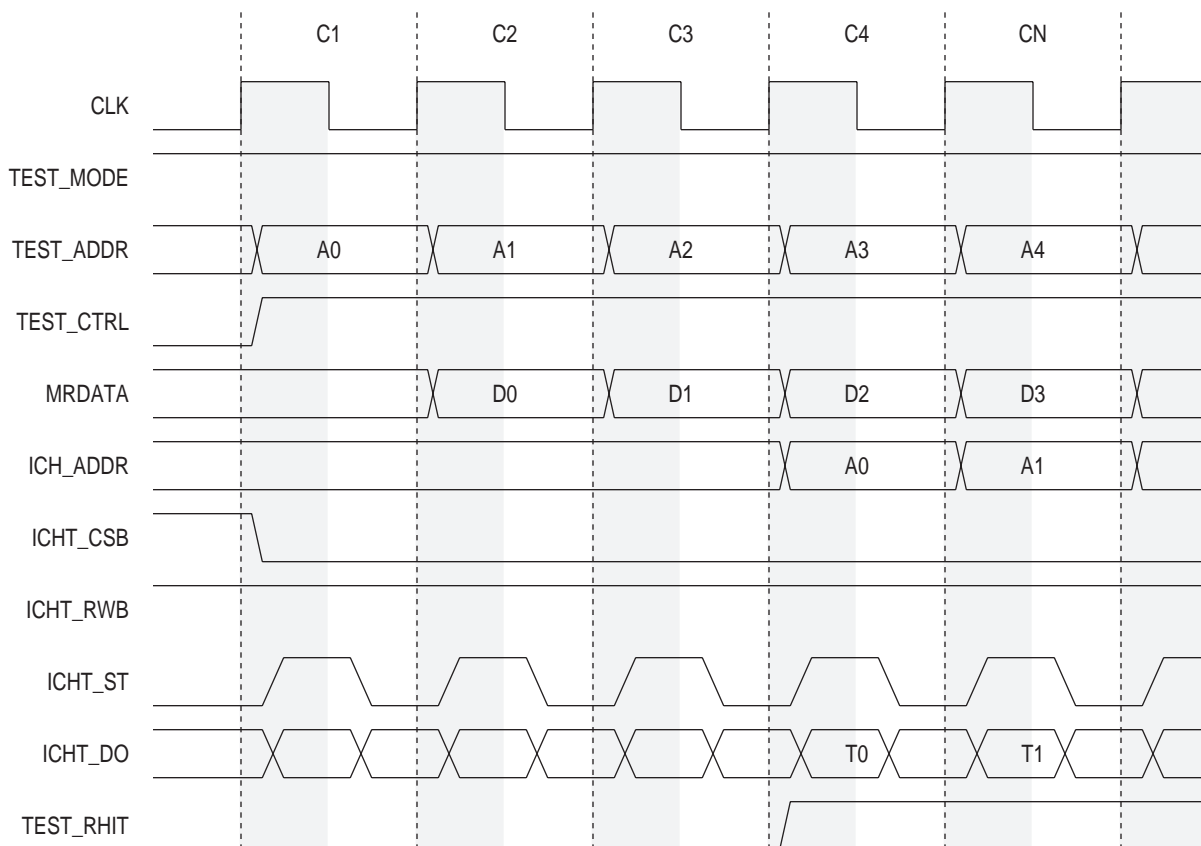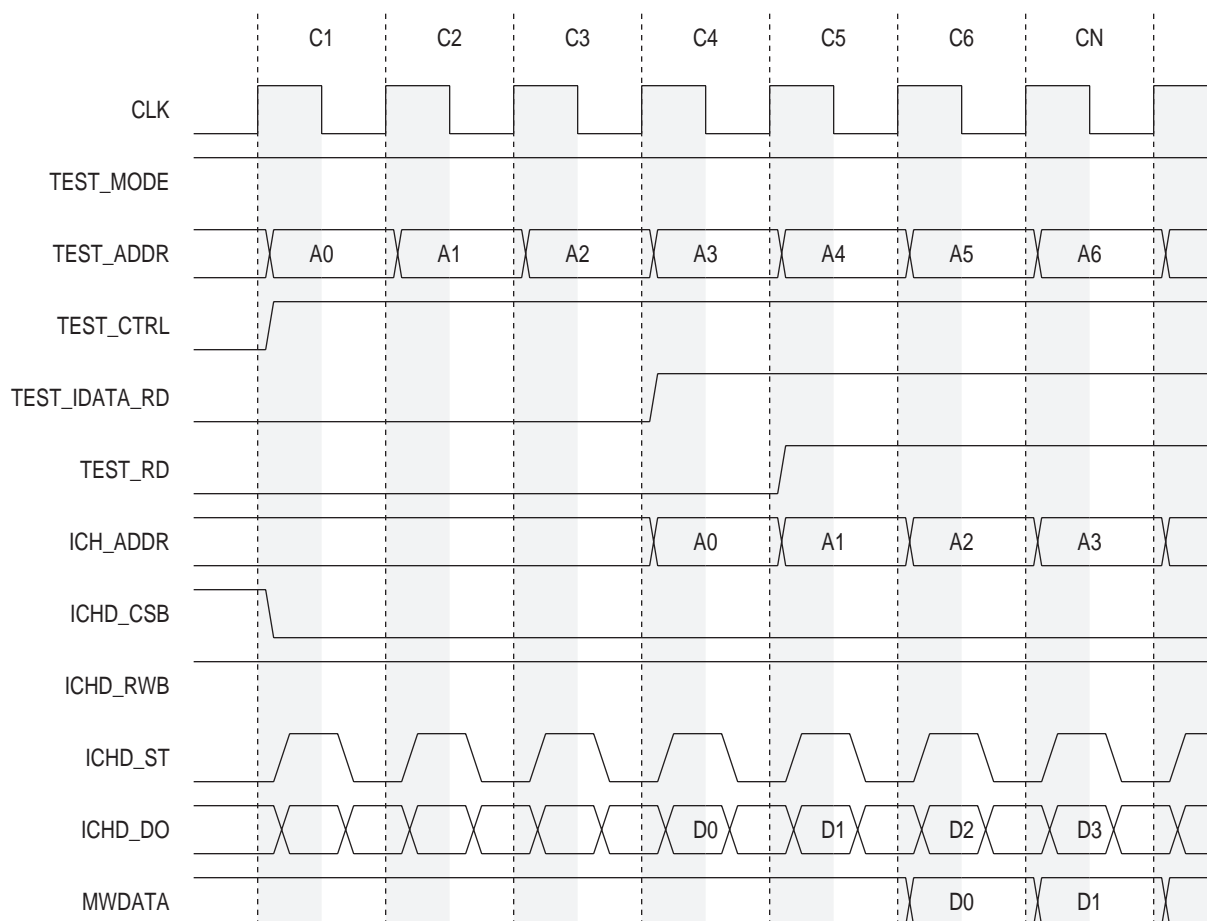

Figure 8-4. Test Instruction Cache Tag Read Cycles

Clock 1 (C1)

On the first clock cycle of the tag RAM read sequence, the first data RAM address should be driven onto TEST_ADDR[14:2] and TEST_CTRL should be asserted.

Clock 2 (C2)

On the second cycle, the first expected data value associated with the address asserted in C1 should be driven onto MRDATA[31:0] and the next address should be driven onto TEST_ADDR[14:2]. The remaining addresses and expected data are driven each successive clock cycle.

Clock 3 (C3)

The third clock cycle is identical to C2.

Clock 4 (C4)

The fourth clock cycle is identical to C3. The first read from the tag RAM occurs in C4, three cycles after the first address was driven. If the value read from the tag RAM is equal to the associated value latched on MRDATA[31:0] and the valid bit is set, TEST_RHIT will be asserted.

Clock n (Cn)

The remaining clock cycles in the test are identical to C4.

Throughout the entire sequence, TEST_RHIT is associated with the address driven three clock cycles earlier. For this reason, three stall cycles (TEST_CTRL negated) should occur after the last address is driven before the next sequence to wait for the last TEST_RHIT.

8.3.4.3 INSTRUCTION CACHE TAG RAM WRITE FOLLOWED BY READ FUNCTION.

The following timing diagram illustrates the minimum number of cycles necessary to perform a write followed by read of the instruction cache tag RAM. A certain sequence is neccessary because of the pipelined datapath to the arrays from MRDATA and MWDATA.
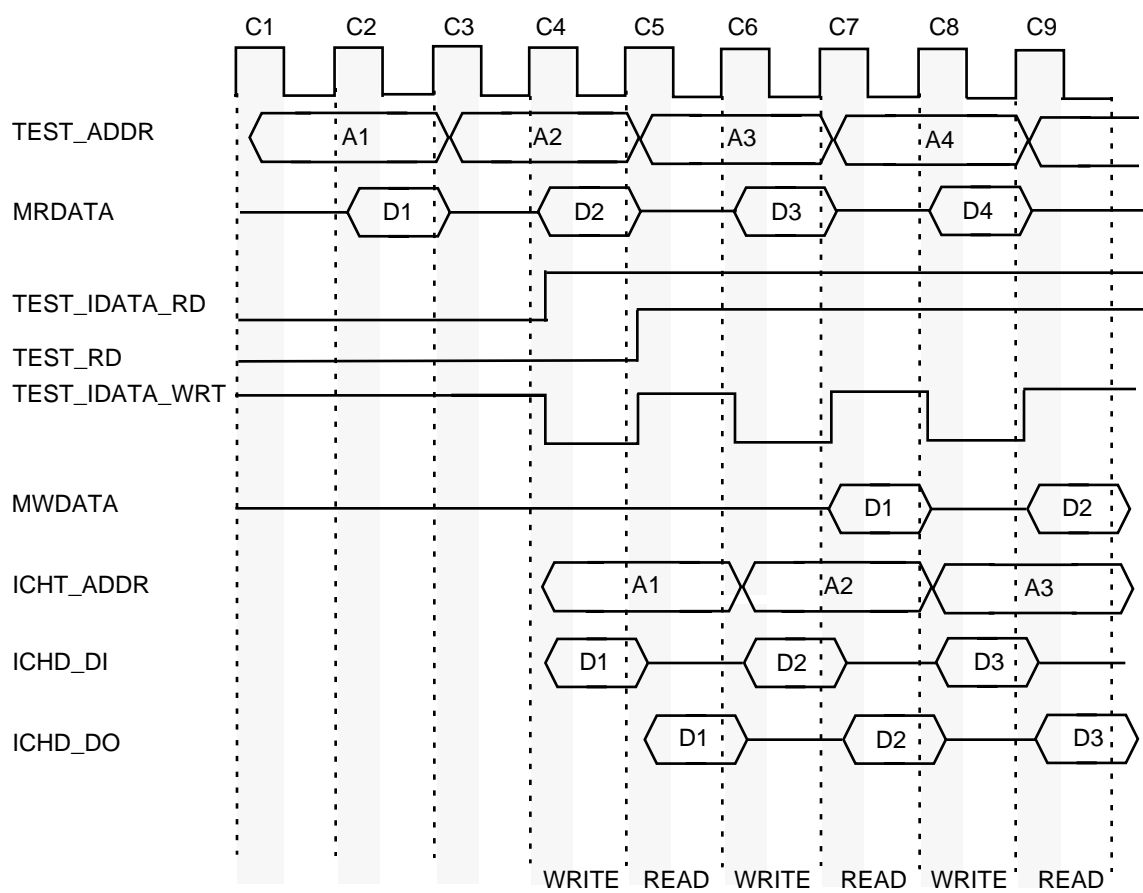
.
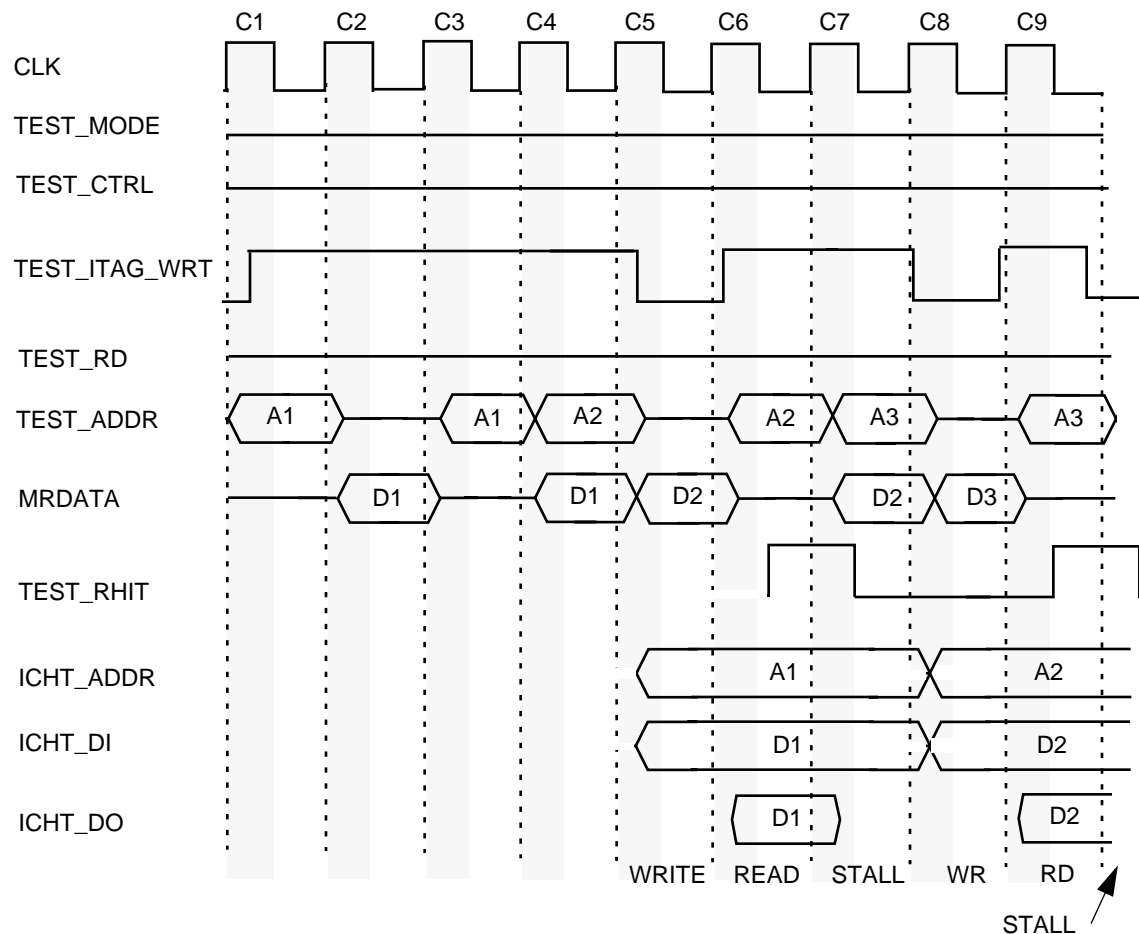
Figure 8-5. I-Cache Tag Write Followed by Read

Clock 1 (C1)

On the first clock cycle , the data RAM address A1 should be driven onto TEST_ADDR[14:2] and TEST_ITAG_WRT should be asserted.

Clock 2 (C2)

On the second cycle, the D1, data value associated with the address asserted in C1, should be driven onto MRDATA[31:0]. The address value driven in C1, A1, can optionally be driven in C2. TEST_ITAG_WRT remains asserted.

Clock 3 (C3)

The third clock cycle is a stall cycle necessary because of the pipelined data path. The address driven in C1, A1, must be driven during C3.TEST_ITAG_WRT remains asserted.

Clock 4 (C4)

The next write address, A2, should be driven onto TEST_ADDR[14:2] during C4. The data driven in C2, D1, must be driven in C4.TEST_ITAG_WRT remains asserted.

Clock 5(C5)

TEST_ITAG_WRT must be negated during C5. A1 and D1, associated with C1/C2, are at the ICACHE tag array during C5; i.e. the actual WRITE to the array occurs in this cycle. The data value associated with the address asserted in C4, D2, is driven onto MRDATA[31:0] during this cycle as well. You have the option of driving A2 during this cycle.

Clock 6 (C6)

The TEST_RHIT signal will assert during this cycle if the ICACHE tag array data for A1 matches D1, the access in Cycles C1 and C2; i.e. it is in this cycle that the actual READ occurs. A2 must be driven onto TEST_ADDR[14:2] during this cycle. TEST_ITAG_WRT asserts and D2 can optionally be driven during this cycle.

Clock 7 (C7)

On C7, D2, must be driven onto MRDATA[31:0] and the next data RAM address, A3, should be driven onto TEST_ADDR[14:2]. This is a stall cycle. TEST_ITAG_WRT remains asserted during this cycle.

Clock 8 (C8)

During C8, TEST_ITAG_WRT must be negated. A3 can optionally be driven during this cycle; D3 must be driven during this cycle. The actual write of A2/D2 occurs during this cycle.

This periodic 3-cycle sequence continues; ie. C4-C6 are repeated during C7-C9, C10-C12, etc. These three cycles are STALL, WRITE, then READ. TEST_ITAG_WRT always negates during the "C5" cycle or the WRITE cycle and asserts during the "C6-C7" cycles.. TEST_RHIT always asserts if no error during "C6" or the READ cycle. The address is asserted in the "C4" cycle, optionally asserted in the "C5" cycle and again asserted during the "C6" cycle. The corresponding data to that address is driven in "C5", optionally driven in "C6" and then driven again in "C7".

8.3.5 Instruction Cache Data RAM Testing

The instruction cache data RAM consists of up to 8K long words that can be accesses individually through the test bus. Testing the compiled data RAM is accomplished by first writing test patterns into the data RAM, reading the data RAM, and verifying the results.

8.3.5.1 INSTRUCTION CACHE DATA RAM WRITE FUNCTION. Writing to the instruction cache data RAM is performed though the test bus and MRDATA[31:0] after entering test mode. The address and control signals are input on the test bus and the data to be written to the data RAM is input on MRDATA[31:0].

Writes to the data RAM are performed in a pipelined fashion as shown in Figure 8-6. All input signals are latched on the positive edge of CLK and all outputs transition on the positive edge of CLK.

Figure 8-6. Test Instruction Cache Data Write Cycles

Clock 1 (C1)

On the first clock cycle of the data RAM write sequence, the first data RAM address should be driven onto TEST_ADDR[14:2] and TEST_CTRL should be asserted.

Clock 2 (C2)

On the second cycle, the first data value associated with the address asserted in C1 should be driven onto MRDATA[31:0] and the next address should be driven onto TEST_ADDR[14:2].

Clock 3 (C3)

On the third cycle, the next address and data values should be driven and TEST_IDATA_WRT should be asserted.

Clock 4 (C4)

The fourth clock cycle is identical to C3. The remaining addresses and data are driven each successive clock cycle and TEST_IDATA_WRT should remain asserted until the last data value has been latched. The first write to the data RAM occurs.

Clock n (Cn)

The remaining clock cycles in the test are identical to C4.

Throughout the entire sequence, the data value being driven is associated with the address driven in the previous clock cycle.

8.3.5.2 INSTRUCTION CACHE DATA RAM READ FUNCTION. Reading from the instruction cache data RAM is performed though the test bus and MWDATA[31:0] after entering test mode. The address and control signals are input on the test bus and the data from the data RAM is output on MWDATA[31:0].

Reads from the data RAM are performed in a pipelined fashion as shown in Figure 8-7. All input signals are latched on the positive edge of CLK and all outputs transition on the positive edge of CLK.
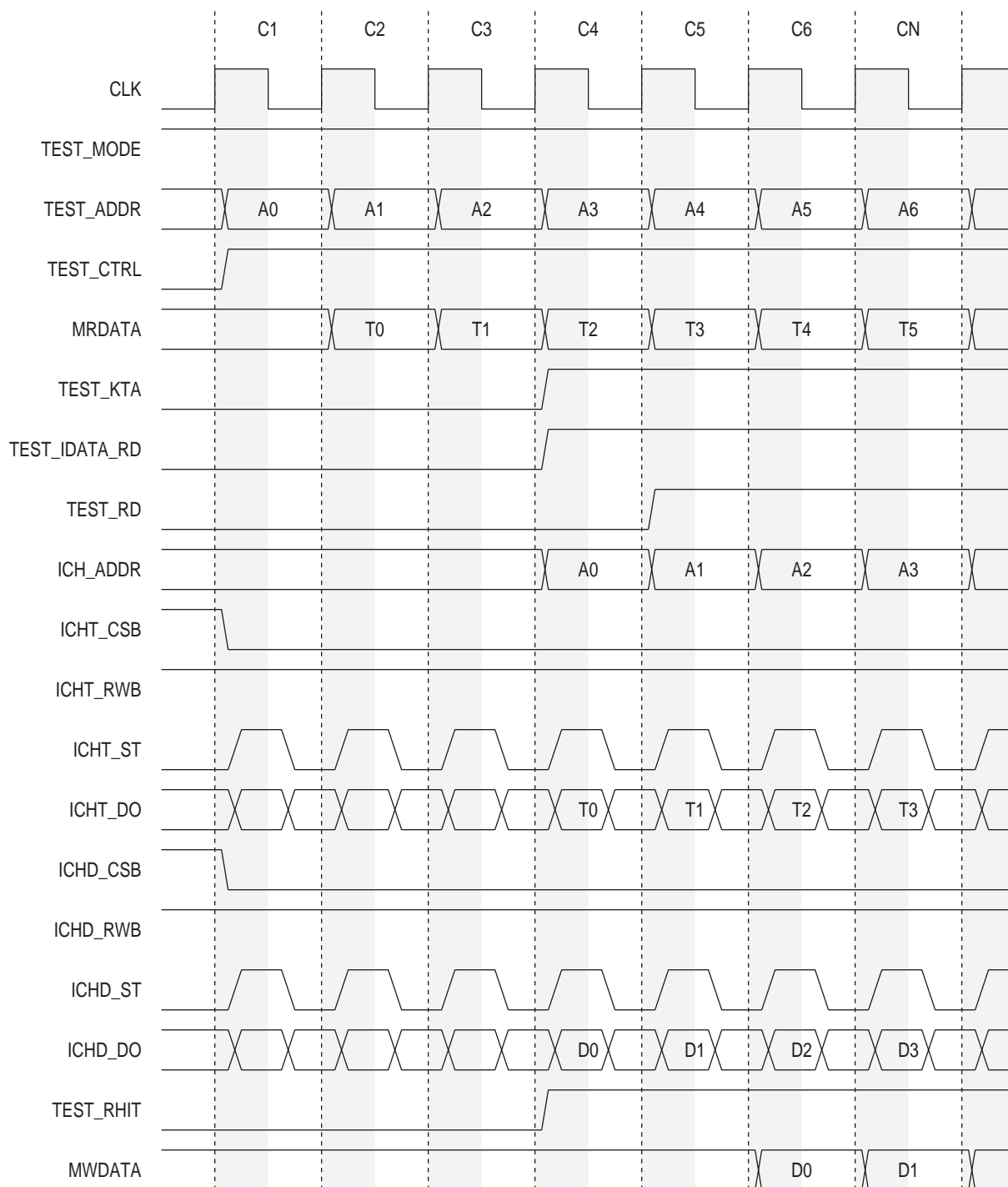

Figure 8-7. Test Instruction Cache Data Read Cycles

Clock 1 (C1)

On the first clock cycle of the data RAM read sequence, the first data RAM address should be driven onto TEST_ADDR[14:2] and TEST_CTRL should be asserted.

Clock 2 (C2)

On the second cycle, the next addresses should be driven onto TEST_ADDR[14:2].

Clock 3 (C3)

The third cycle is identical to C2.

Clock 4 (C4)

On the fourth cycle, the next address should be driven and TEST_IDATA_RD should be asserted. The first read from the data RAM occurs in C4.

Clock 5 (C5)

On the fifth cycle, the next address should be driven and TEST_RD should be asserted. The remaining addresses are driven each successive clock cycle and TEST_IDATA_RD should remain asserted until the last data value has been read from the data RAM.

Clock 6 (C6)

Cycle 6 is identical to C5. The MWDATA[31:0] signals are driven with the first data, five cycles after the associated address was driven. TEST_RD should remain asserted until the last data value has been driven onto MWDATA[31:0].

Clock n (Cn)

The remaining clock cycles in the test are identical to C6.

Throughout the entire sequence, the data value being driven is associated with the address driven five clock cycles earlier. For this reason, five stall cycles (TEST_CTRL negated) should occur after the last address is driven before the next sequence to wait for the last data to be driven.

8.3.5.3 INSTRUCTION CACHE DATA RAM WRITE FOLLOWED BY READ FUNCTION.
The following timing diagram illustrates the minimum number of cycles necessary to perform a write followed by read of the instruction cache data RAM. A certain sequence is neccessary because of the pipelined datapath to the arrays from MRDATA and MWDATA.

Figure 8-8. I-Cache Data Write/Read

Clock 1 (C1)

On the first clock cycle , A1 (data RAM address) should be driven onto TEST_ADDR[14:2] and TEST_IDATA_WRT should be asserted.

Clock 2 (C2)

On the second cycle, D1 (the data value associated with A1, the address asserted in C1) should be driven onto MRDATA[31:0]. A1 must continue to be driven during this cycle. TEST_IDATA_WRT continues to be asserted.

Clock 3 (C3)

On the third cycle, the next data RAM address, A2, is driven onto TEST_ADDR[14:2]. D1 can optionally be driven during this cycle. TEST_IDATA_WRT remains asserted during this cycle.

Clock 4 (C4)

D2 should be driven onto MRDATA[31:0] during this cycle. TEST_IDATA_RD must be asserted here for the length of this test. TEST_IDATA_WRT is negated during this cycle. The actual WRITE to the ICACHE data array of the C1/C2 (A1/D1) access occurs during this cycle. A2 must remain driven during this cycle.

Clock 5(C5)

The next data RAM address, A3, should be driven onto TEST_ADDR[14:2] during this cycle. TEST_RD is asserted here for the length of this test. TEST_IDATA_WRT is asserted during this cycle. The acutal READ of the ICACHE data array C1/C2 (A1/D1) access occurs during this cycle. D2 can optionally be driven during this cycle.

Clock 6 (C6)

D3, the data associated with the C5 address, A3, is driven onto MRDATA[31:0] during this cycle. TEST_IDATA_WRT is negated during this cycle .The actual WRITE of the ICACHE data access C3/C4 (A2/D2) occurs during this cycle. A3 must remain driven in this cycle.

Clock 7 (C7)

A4 should be driven onto TEST_ADDR[14:2] during this cycle. TEST_IDATA_WRT is asserted. The C1/C2 access read data, D1, will be driven onto MWDATA[31:0] during this cycle. The actual READ of the ICACHE data array C3/C4 access (A2/D2) occurs during this cycle. D3 can optionally be driven in this cycle.

Clock 8 (C8)

D4 must be driven onto MRDATA[31:0] during this cycle. A4 must remain driven. TEST_IDATA_WRT is negated. The actual WRITE of the ICACHE data array C3/C4 access, A3/D, occurs here.

Clock 9 (C9)

A5 must be driven onto TEST_ADDR[14:2] during this cycle. TEST_IDATA_WRT is asserted. D4 can optionally be driven in this cycle. The C5/C6 access read data, D2, will be driven onto MWDATA[31:0] during this cycle. The actual READ of the ICACHE data array C5/C6 access (A3/D3) occurs during this cycle.

This periodic 2-cycle sequence continues; ie. C6-C7 are repeated during C8-C9, C10-C11, etc. The "C6" cycle is the actual WRITE cyle and the "C7" cycle is the actual READ cycle where the data from the previous actual WRITE/READ sequence is displayed on MWDATA.

8.3.6 Instruction Cache KTA Mode Testing

The KTA mode tests the read data path for both of the instruction cache (tag and data) RAMs. This test checks the valid bit and performs a compare between the upper bits of the appropriate tag and the upper bits of the address to determine a cache hit. If the read is a hit, the value from the data RAM is returned. This mimicks the reading of the instruction cache during normal operation. Both RAMs are first written using instruction cache data and tag write cycles.

Both the MRDATA[31:15] signals and part of the TEST_ADDR[14:2] signals are used to drive data into the array and compare against during the read.

Freescale Semiconductor, Inc.

A 512 byte ICACHE array uses an ICACHE tag array sized 32x24; thus having 5 address bits. TEST_ADDR[8:4] is used to address the tag array, while TEST_ADDR[8:2] addresses the ICACHE data array (128x32). {MRDATA[31:15], TEST_ADDR[14:9]} is used to compare data to the tag array. These 23 bits compare to the 23 bits of address in the tag array, the 24th bit is the valid bit.

Table 8-5. TEST KTA CONFIGURATIoNS

| ICACHE SZ	TAG ARRAY SZ	TAG ARRAY	TEST_ADDR	MRDATA & TEST_ADDR
BYTES	LOCATIONS X # OF DATA BITS	ADDR BITS	USED TO ADDR	USED FOR DATA WRITES/CMPRS
512	32x24	5	TEST_ADDR[8:4]	{MRDATA[31:15], TEST_ADDR[14:9]}
1K	64x23	6	TEST_ADDR[9:4]	{MRDATA[31:15], TEST_ADDR[14:10]}
2K	128x22	7	TEST_ADDR[10:4]	{MRDATA[31:15], TEST_ADDR[14:11]}
4K	256x21	8	TEST_ADDR[11:4]	{MRDATA[31:15], TEST_ADDR[14:12]}
8K	512x20	9	TEST_ADDR[12:4]	{MRDATA[31:15], TEST_ADDR[14:13]}
16K	1Kx19	10	TEST_ADDR[13:4]	{MRDATA[31:15], TEST_ADDR[14]}
32K	2Kx18	11	TEST_ADDR[14:4]	MRDATA[31:15]

Figure 8-9. KTA Mode Cycles

Clock 1 (C1)

On the first clock cycle of the KTA mode sequence, the first data RAM address should be driven onto TEST_ADDR[14:2] and TEST_CTRL should be asserted.

Clock 2 (C2)

On the second cycle, the next data RAM addresses should be driven onto TEST_ADDR[14:2]. The MRDATA[31:8] signals should be driven with the tag information expected from the tag RAM for the line associated with the address driven in C1.

Clock 3 (C3)

The third cycle is identical to C2.

Clock 4 (C4)

On the fourth cycle, the next address should be driven, TEST_KTA should be asserted, and TEST_IDATA_RD should be asserted. The first read from the cache RAMs occur in C4. If the value from the tag RAM is equal to the tag value driven in C2, TEST_RHIT will be asserted.

Clock 5 (C5)

On the fifth cycle, the next address should be driven and TEST_RD should be asserted. The remaining addresses are driven each successive clock cycle and TEST_KTA and TEST_IDATA_RD should remain asserted until the last data value has been read from the data RAM.

Clock 6 (C6)

Cycle 6 is identical to C5. The MWDATA[31:0] signals are driven with the first data, five cycles after the associated address was driven. TEST_RD should remain asserted until the last data value has been driven onto MWDATA[31:0].

Clock n (Cn)

The remaining clock cycles in the test are identical to C6.

8.3.7 ROM Testing

The ROM consists of up to 8K long words that can be accesses individually through the test bus. Testing the compiled ROM is accomplished by reading the ROM and verifying the results.

8.3.7.1 ROM READ FUNCTION. Reading from the ROM is performed though the test bus and MWDATA[31:0] after entering test mode. The address and control signals are input on the test bus and the data from the ROM is output on MWDATA[31:0].

Reads from the ROM are performed in a pipelined fashion as shown in Figure 8-10. All input signals are latched on the positive edge of CLK and all outputs transition on the positive edge of CLK.
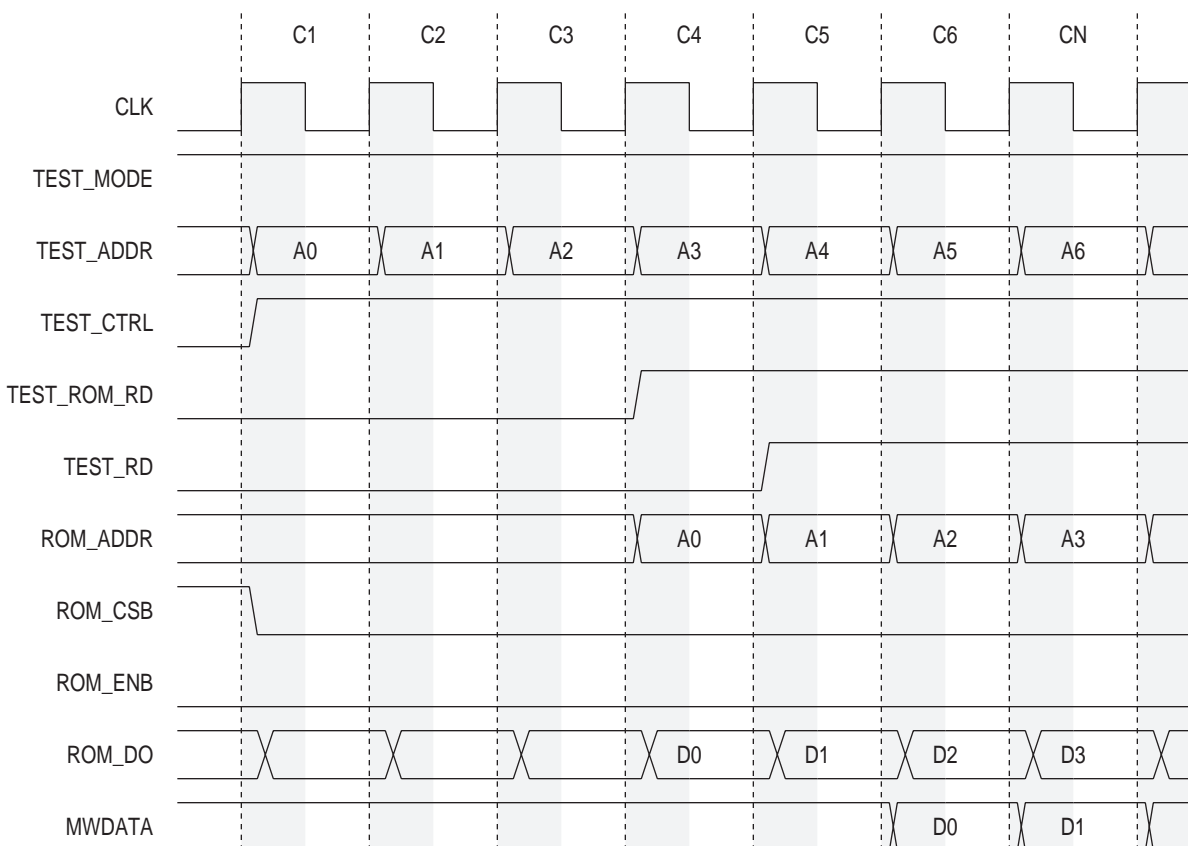
Figure 8-10. Test ROM Read Cycles

Clock 1 (C1)

On the first clock cycle of the ROM read sequence, the first ROM address should be driven onto TEST_ADDR[14:2] and TEST_CTRL should be asserted.

Clock 2 (C2)

On the second cycle, the next addresses should be driven onto TEST_ADDR[14:2].

Clock 3 (C3)

The third cycle is identical to C2.

Clock 4 (C4)

On the fourth cycle, the next address should be driven and TEST_ROM_RD should be asserted. The first read from the ROM does not occur until C4.

Clock 5 (C5)

On the fifth cycle, the next address should be driven and TEST_RD should be asserted. The remaining addresses are driven each successive clock cycle and TEST_ROM_RD should remain asserted until the last data value has been read from the ROM.

Clock 6 (C6)

Cycle six is identical to C5. The MWDATA[31:0] signals are driven with the first data, five cycles after the associated address was driven. TEST_RD should remain asserted until the last data value has been driven onto MWDATA[31:0].

Clock n (Cn)

The remaining clock cycles in the test are identical to C6.

Throughout the entire sequence, the data value being driven is associated with the address driven five clock cycles earlier. For this reason, five stall cycles (TEST_CTRL negated) should occur after the last address is driven before the next sequence to wait for the last data to be driven.

8.3.8 SRAM Testing

The SRAM consists of up to 8K longwords that can be accesses individually through the test bus. Testing the compiled SRAM is accomplished by first writing test patterns into the SRAM, reading the SRAM, and verifying the results.

8.3.8.1 SRAM WRITE FUNCTION. Writing to the SRAM is performed though the test bus and MRDATA[31:0] after entering test mode. The address and control signals are input on the test bus and the data to be written to the SRAM is input on MRDATA[31:0].

Writes to the SRAM are performed in a pipelined fashion as shown in Figure 8-11. All input signals are latched on the positive edge of CLK and all outputs transition on the positive edge of CLK.
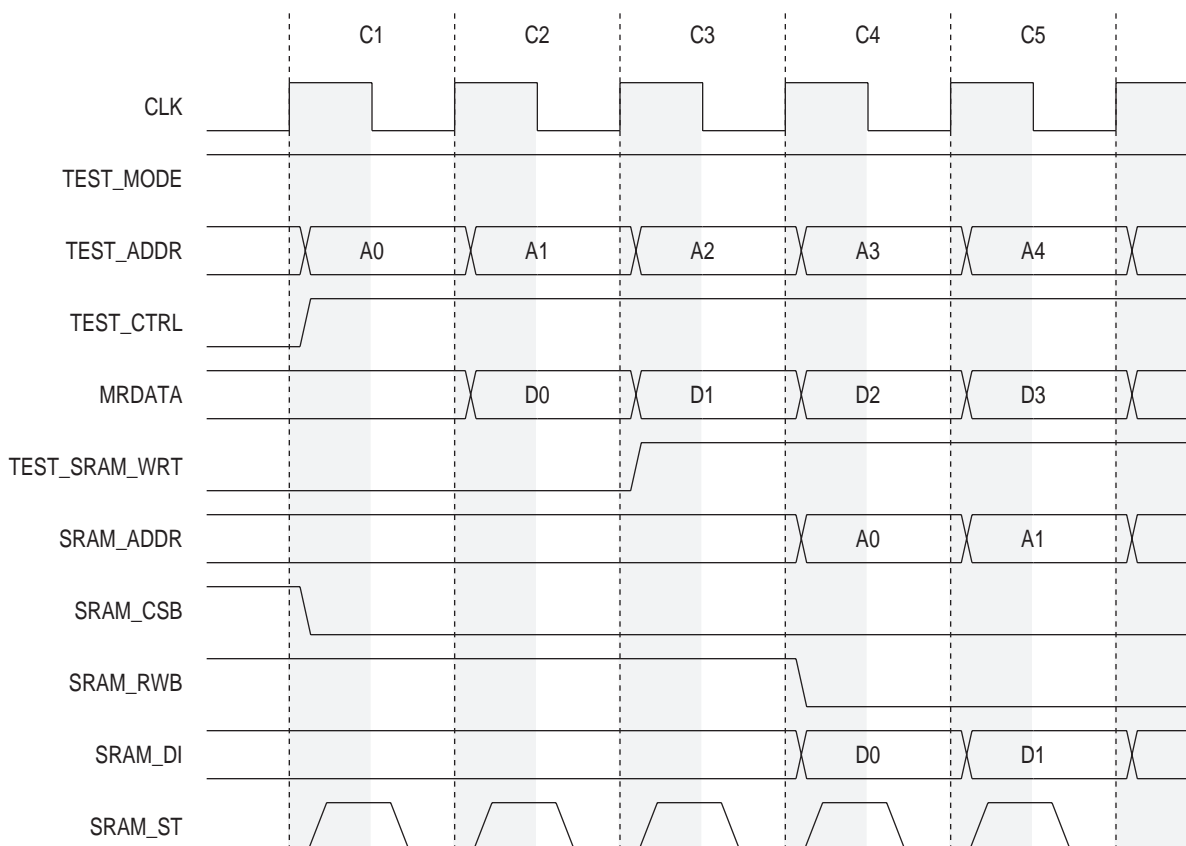
Figure 8-11. Test SRAM Write Cycles

Clock 1 (C1)

On the first clock cycle of the SRAM write sequence, the first SRAM address should be driven onto TEST_ADDR[14:2] and TEST_CTRL should be asserted.

Clock 2 (C2)

On the second cycle, the first data value associated with the address asserted in C1 should be driven onto MRDATA[31:0] and the next address should be driven onto TEST_ADDR[14:2].

Clock 3 (C3)

On the third cycle, the next address and data values should be driven and TEST_SRAM_WRT should be asserted. The remaining addresses and data are driven each successive clock cycle and TEST_SRAM_WRT should remain asserted until the last data value has been latched.

Clock 4 (C4)

On the fourth cycle, the next address and data values should be driven. The first write to the SRAMs occurs in cycle four, three cycles after the first address was driven.

Freescale Semiconductor, Inc.

Clock n (Cn)

The remaining clock cycles in the test are identical to C4.

Throughout the entire sequence, the data value being driven is associated with the address driven in the previous clock cycle.

8.3.8.2 SRAM READ FUNCTION. Reading from the SRAM is performed though the test bus and the MWDATA[31:0] after entering test mode. The address and control signals are input on the test bus and the data from the SRAM is output on MWDATA[31:0].

Reads from the SRAM are performed in a pipelined fashion as shown in Figure 8-12. All input signals are latched on the positive edge of CLK and all outputs transition on the positive edge of CLK.
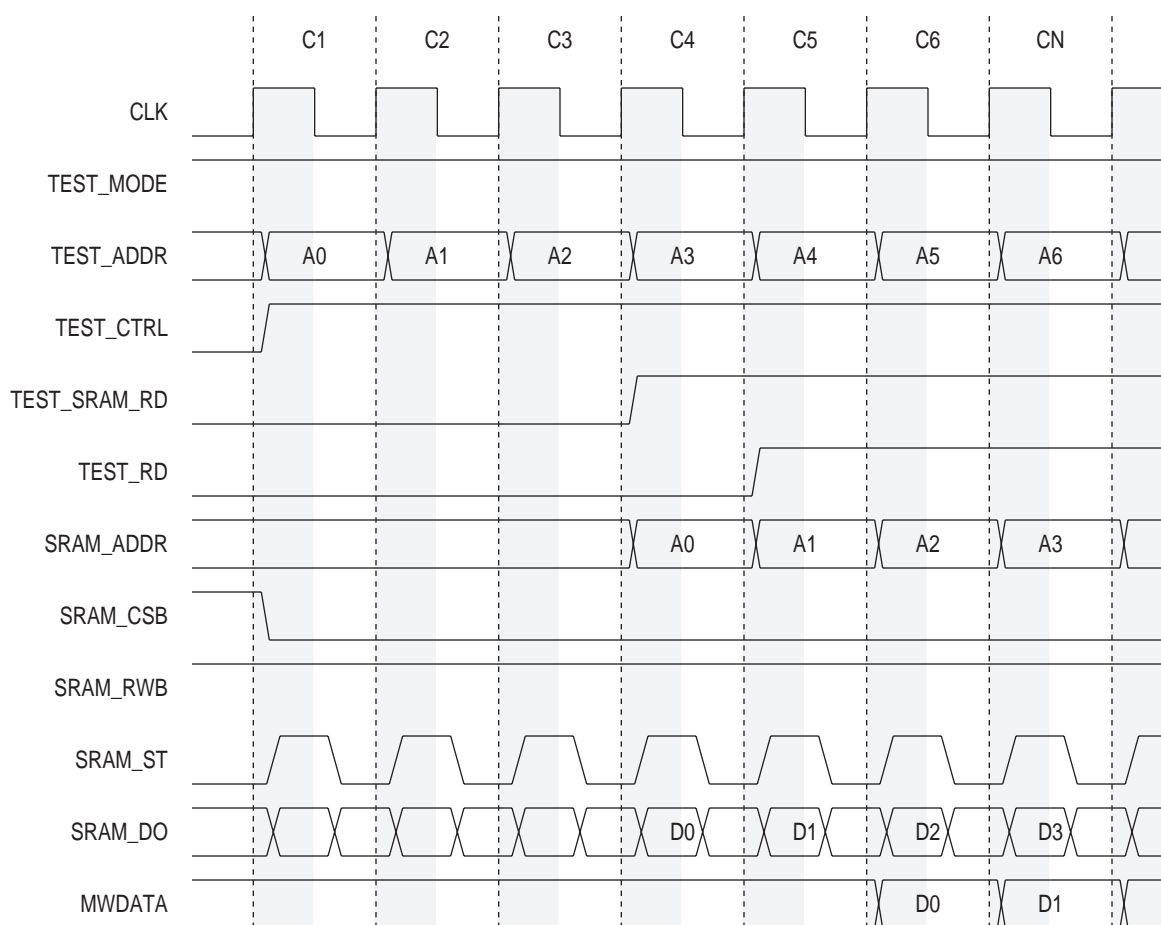
Figure 8-12. Test SRAM Read Cycles

Clock 1 (C1)

On the first clock cycle of the SRAM read sequence, the first SRAM address should be driven onto TEST_ADDR[14:2] and TEST_CTRL should be asserted.

Clock 2 (C2)

On the second cycle, the next addresses should be driven onto TEST_ADDR[14:2].

Clock 3 (C3)

The third cycle is identical to C2.

Clock 4 (C4)

On the fourth cycle, the next address should be driven and TEST_SRAM_RD should be asserted. The first read to the SRAM does not occur until C4, three cycles after the first address was driven.

Clock 5 (C5)

On the fifth cycle, the next address should be driven and TEST_RD should be asserted. The remaining addresses are driven each successive clock cycle and TEST_SRAM_RD should remain asserted until the last data value has been read from the SRAM.

Clock 6 (C6)

Cycle six is identical to C5. The MWDATA[31:0] signals are driven with the first data, five cycles after the associated address was driven. TEST_RD should remain asserted until the last data value has been driven onto MWDATA[31:0].

Clock n (Cn)

The remaining clock cycles in the test are identical to C6.

Throughout the entire sequence, the data value being driven is associated with the address driven five clock cycles earlier. For this reason, five stall cycles (TEST_CTRL negated) should occur after the last address is driven before the next sequence to wait for the last data to be driven.

8.3.8.3 SRAM WRITE FOLLOWED BY READ FUNCTION. The following timing diagram illustrates the minimum number of cycles necessary to perform a write followed by read of the SRAM. A certain sequence is neccessary because of the pipelined datapath to the arrays from MRDATA and MWDATA.
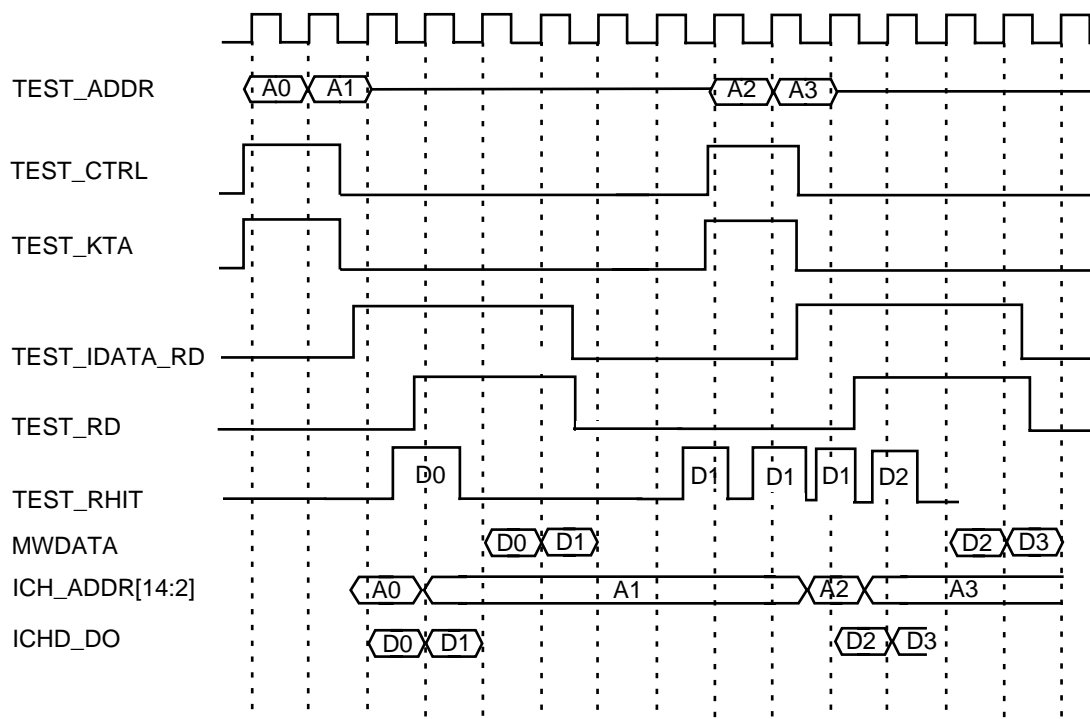
If 8K cache, TEST_ADDR[14:13] are used as the lower two bits of data to be compared for the hit signal.

Figure 8-13. SRAM Write Followed by Read

Clock 1 (C1)

On the first clock cycle , A1, SRAM address, should be driven onto TEST_ADDR[14:2] and TEST_SRAM_WRT should be asserted.

Clock 2 (C2)

On the second cycle, D1 (the data value associated with A1, the address asserted in C1) should be driven onto MRDATA[31:0]. A1 must continue to be driven during this cycle. TEST_SRAM_WRT remains asserted.

Clock 3 (C3)

C3 is a stall cycle. TEST_SRAM_WRT must remain asserted

Clock 4 (C4)

C4 is a stall cycle. TEST_SRAM_WRT must negate during this cycle which does the actual WRITE of A1/D1 at the array. TEST_SRAM_RD must assert during this cycle.

Freescale Semiconductor, Inc.

Clock 5 (C5)

The next SRAM address, A2, should be driven onto TEST_ADDR[14:2] during this cycle. TEST_RD is asserted here; it will remain asserted during the next cycle C5, and then repeat 2 cycles negated, 2 cycles asserted for the duration of this type of test. TEST_IDATA_WRT remains negated during this cycle; it will repeat two cycles asserted, two cyles negated. TEST_SRAM_RD remains asserted during this cycle. The acutal READ of the ICACHE data array C1/C2 (A1/D1) access occurs during this cycle.

Clock 6 (C6)

D2, associated with A2, is driven onto MRDATA[31:0] during this cycle. A2 remains driven. TEST_RD remains asserted, TEST_SRAM_RD negates, and TEST_SRAM_WRT asserts during this cycle.

Clock 7 (C7)

D1 is driven onto MWDATA[31:0]. This is a stall cycle; TEST_SRAM_RD remains negated, TEST_RD negates, and TEST_SRAM_WRT remains asserted during this cycle.

Clock 8 (C8)

The repetition of C4-C7 begins on C8. TEST_SRAM_RD is asserted, TEST_RD negated, and TEST_SRAM_WRT negated during this cycle. The actual WRITE of A2/D2 at the array occurs during this cycle.

Clock 9 (C9)

C9 mimmicks C5: TEST_SRAM_RD remains asserted, TEST_RD asserts, and TEST_SRAM_WRT remains negated. A3 is driven onto TEST_ADDR[14:2]. The actual READ of A2/D2 at the array occurs during this cycle.

Clock 10 (C10)

C10 mimmicks C6: A3 remains driven on TEST_ADDR[14:2]; D3 asserts on MRDATA[31:0]. TEST_SRAM_RD negates, TEST_RD remains asserted, and TEST_SRAM_WRT asserts during this cycle.

Clock 11 (C11)

C11 mimmicks C7: D2 is driven onto MWDATA[31:0]. This is a stall cycle; TEST_SRAM_RD remains negated, TEST_RD negates, and TEST_SRAM_WRT remains asserted during this cycle.

This periodic 4-cycle sequence continues; ie. C4-C7 are repeated during C8-C11, C12-C15, etc. The "C4" cycle is the actual WRITE cyle and the "C5" cycle is the actual READ cycle. During the "C7' cycle the data from the previous actual WRITE/READ sequence at the array is displayed on MWDATA.

SECTION 9
INSTRUCTION EXECUTION TIMING

This section presents ColdFire2/2M instruction execution times in terms of clock cycles. The number of operand references for each instruction is also included, enclosed in parentheses following the number of clock cycles. Each timing entry is presented as **C**(r/w) where:

- **C** - The number of ColdFire2/2M clock cycles, including all applicable operand fetches and writes, as well as all internal cycles required to complete the instruction execution.

- r/w - The number of operand reads (r) and writes (w) required by the instruction. An operation performing a read-modify-write function is denoted as (1/1).

This section includes assumptions concerning the timing values and the execution time details and applies to both the ColdFire2 and ColdFire2M unless otherwise noted.

9.1 TIMING ASSUMPTIONS

The timing data presented in this section have the following assumptions:

1. The operand execution pipeline (OEP) is loaded with the opword and all required extension words at the beginning of each instruction execution. This implies that the OEP doesn't wait for the instruction fetch pipeline (IFP) to supply opwords and/or extension words.

2. The OEP does not experience any sequence-related pipeline stalls. For the ColdFire2/2M, the most common example of this type of stall involves consecutive STORE operations, excluding the MOVEM instruction. For all STORE operations (except MOVEM), certain hardware resources within the ColdFire2/2M are marked as 'busy' for two clock cycles after the final DSOC cycle of the STORE instruction. If a subsequent STORE instruction is encountered within this 2-cycle window, it will be stalled until the resource again becomes available. Thus, the maximum pipeline stall involving consecutive STORE operations is 2 cycles. The MOVEM instruction uses a different set of resources and this stall does not apply.

3. The OEP completes all memory accesses without any stall conditions caused by the memory itself. Thus, the timing details provided in this section assume an infinite zero-wait state memory is attached to the ColdFire2/2M.

4. All operand data accesses are 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.

If the operand alignment fails these guidelines, the misalignment unit is used. The processor core decomposes the misaligned operand reference into a series of aligned accesses as shown in Table 9-1.

Table 9-1. Misaligned Operand References

ADDRESS[1:0]	SIZE	KBUS OPERATIONS	ADDITIONAL C(R/W)
x1	word	byte, byte	2(1/0) if read 1(0/1) if write
x1	long	byte, word, byte	3(2/0) if read 2(0/2) if write
10	long	word, word	2(1/0) if read 1(0/1) if write

9.2 MOVE INSTRUCTION EXECUTION TIMES

The execution times for the MOVE.{B,W} instructions are shown in Table 9-2, while Table 9-3 provides the timing for MOVE.L.

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

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

Table 9-2. Move Byte and Word Execution Times

SOURCE	DESTINATION						
	RX	(AX)	(AX)+	-(AX)	(D16,AX)	(D8,AX,XN*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)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)
(Ay)+	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)
-(Ay)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)
(d16,Ay)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	—	—
(d8,Ay,Xn*SF)	4(1/0)	4(1/1)	4(1/1)	4(1/1)	—	—	—
xxx.w	3(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
xxx.l	3(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
(d16,PC)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	—	—
(d8,PC,Xn*SF)	4(1/0)	4(1/1)	4(1/1)	4(1/1)	—	—	—
#xxx	1(0/0)	3(0/1)	3(0/1)	3(0/1)	—	—	—

Table 9-3. Move Long Execution Times

SOURCE	DESTINATION						
	RX	(AX)	(AX)+	-(AX)	(D16,AX)	(D8,AX,XN*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)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
(Ay)+	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
-(Ay)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	3(1/1)	2(1/1)
(d16,Ay)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—
(d8,Ay,Xn*SF)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
xxx.w	2(1/0)	2(1/1)	2(1/1)	2(1/1)	—	—	—
xxx.l	2(1/0)	2(1/1)	2(1/1)	2(1/1)	—	—	—
(d16,PC)	2(1/0)	2(1/1)	2(1/1)	2(1/1)	2(1/1)	—	—
(d8,PC,Xn*SF)	3(1/0)	3(1/1)	3(1/1)	3(1/1)	—	—	—
#xxx	1(0/0)	2(0/1)	2(0/1)	2(0/1)	—	—	—

Freescale Semiconductor, Inc.

9.3 STANDARD ONE OPERAND INSTRUCTION EXECUTION TIMES

Table 9-4. One Operand Instruction Execution Times

OPCODE	<EA>	EFFECTIVE ADDRESS							
		RN	(AN)	(AN)+	-(AN)	(D16,AN)	(D8,AN,XN*SF)	XXX.WL	#XXX
dr.b	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
dr.w	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
dr.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)	—	—	—	—	—	—	—
scc	Dx	1(0/0)	—	—	—	—	—	—	—
swap	Dx	1(0/0)	—	—	—	—	—	—	—
tst.b	<ea>	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
tst.w	<ea>	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
tst.l	<ea>	1(0/0)	2(1/0)	2(1/0)	2(1/0)	2(1/0)	3(1/0)	2(1/0)	1(0/0)

9.4 STANDARD TWO OPERAND INSTRUCTION EXECUTION TIMES

Table 9-5. Two Operand Instruction Execution Times

OPCODE	<EA>	EFFECTIVE ADDRESS							
		RN	(AN)	(AN)+	-(AN)	(D16,AN) (D16,PC)	(D8,AN,XN*SF) (D8,PC,XN*SF)	XXX.WL	#XXX
add.l	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
add.l	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
addi.l	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
addq.l	#imm,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
addx.l	Dy,Dx	1(0/0)	—	—	—	—	—	—	—
and.l	<ea>,Dn	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
and.l	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
and.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)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
bchg	#imm,<ea>	2(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	—	—	—
bclr	Dy,<ea>	2(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
bclr	#imm,<ea>	2(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	—	—	—
bset	Dy,<ea>	2(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
bset	#imm,<ea>	2(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	—	—	—
btst	Dy,<ea>	2(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
btst	#imm,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	—	—	1(0/0)
cmp.l	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
cmpi.l	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
eor.l	Dy,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
eori.l	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
lea	<ea>,Ax	—	1(0/0)	—	—	1(0/0)	2(0/0)	1(0/0)	—
lsl.l	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
lsr.l	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
moveq	#imm,Dx	—	—	—	—	—	—	—	1(0/0)
muls.w	<ea>,Dx	9(0/0)[1]	11(1/0)[1]	11(1/0)[1]	11(1/0)[1]	11(1/0)[1]	12(1/0)[1]	11(1/0)[1]	9(0/0)[1]
		3(0/0)[2]	5(1/0)[2]	5(1/0)[2]	5(1/0)[2]	5(1/0)[2]	6(1/0)[2]	5(1/0)[2]	3(0/0)[2]
mulu.w	<ea>,Dx	9(0/0)[1]	11(1/0)[1]	11(1/0)[1]	11(1/0)[1]	11(1/0)[1]	12(1/0)[1]	11(1/0)[1]	9(0/0)[1]
		3(0/0)[2]	5(1/0)[2]	5(1/0)[2]	5(1/0)[2]	5(1/0)[2]	6(1/0)[2]	5(1/0)[2]	3(0/0)[2]
muls.l	<ea>,Dx	18(0/0)[1]	20(1/0)[1]	20(1/0)[1]	20(1/0)[1]	20(1/0)[1]	-	-	-
		5(0/0)[2]	7(1/0)[2]	7(1/0)[2]	7(1/0)[2]	7(1/0)[2]	-	-	-
mulu.l	<ea>,Dx	18(0/0)[1]	20(1/0)[1]	20(1/0)[1]	20(1/0)[1]	20(1/0)[1]	-	-	-
		5(0/0)[2]	7(1/0)[2]	7(1/0)[2]	7(1/0)[2]	7(1/0)[2]	-	-	-
or.l	<ea>,Dn	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
or.l	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
or.l	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
sub.l	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
sub.l	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
subi.l	#imm,Dx	1(0/0)	—	—	—	—	—	—	—

NOTES: 1.   Applies to the ColdFire2 only.

2.   Applies to the ColdFire2M only.

Freescale Semiconductor, Inc.

Table 9-5. Two Operand Instruction Execution Times (Continued)

OPCODE	<EA>	EFFECTIVE ADDRESS							
		RN	(AN)	(AN)+	-(AN)	(D16,AN) (D16,PC)	(D8,AN,XN*SF) (D8,PC,XN*SF)	XXX.WL	#XXX
subq.l	#imm,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
subx.l	Dy,Dx	1(0/0)	—	—	—	—	—	—	—

NOTES: 1.   Applies to the ColdFire2 only.

       2.   Applies to the ColdFire2M only.

ColdFire2/2M User's Manual MOTOROLA

9.5 MISCELLANEOUS INSTRUCTION EXECUTION TIMES

Table 9-6. Miscellaneous Instruction Execution Times

OPCODE	<EA>	EFFECTIVE ADDRESS							
		RN	(AN)	(AN)+	-(AN)	(D16,AN)	(D8,AN,XN*SF)	XXX.WL	#XXX
cpush	—	—	9(0/1)	—	—	—	—	—	—
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	7(0/0)	—	—	—	—	—	—	7(0/0) [2]
move.l[1]	<ea>,ACC	1(0/0)	-	-	-	-	-	-	1(0/0)
move.l[1]	<ea>,MACSR	1(0/0)	-	-	-	-	-	-	1(0/0)
move.l[1]	<ea>,MASK	1(0/0)	-	-	-	-	-	-	1(0/0)
move.l[1]	ACC,<ea>	2(0/0)	-	-	-	-	-	-	-
move.l[1]	MACSR,<ea>	2(0/0)	-	-	-	-	-	-	-
move.l[1]	MASK,<ea>	2(0/0)	-	-	-	-	-	-	-
movec	Ry,Rc	9(0/1)	—	—	—	—	—	—	—
movem.l	<ea>,&list	—	1+n(n/0)[3]	—	—	1+n(n/0)[3]	—	—	—
movem.l	&list,<ea>	—	1+n(0/n)[3]	—	—	1+n(0/n)[3]	—	—	—
nop	—	3(0/0)	—	—	—	—	—	—	—
pea	<ea>	—	2(0/1)	—	—	2(0/1) [4]	3(0/1)[5]	2(0/1)	—
pulse		1(0/0)	—	—	—	—	—	—	—
stop	#imm	—	—	—	—	—	—	—	3(0/0) [6]
trap	#imm	—	—	—	—	—	—	—	15(1/2)
tpf		1(0/0)	—	—	—	—	—	—	—
tpf.w	#imm	1(0/0)	—	—	—	—	—	—	—
tpf.l	#imm	1(0/0)	—	—	—	—	—	—	—
unlk	Ax	2(1/0)	—	—	—	—	—	—	—
wddata	<ea>	—	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	3(1/0)
wdebug	<ea>	—	5(2/0)	—	—	5(2/0)	—	—	—

NOTES: 1. Applies to the ColdFire2M only.

2. If a MOVE.W #imm,SR instruction is executed and imm[13] = 1, the execution time is 1(0/0).

3. n is the number of registers moved by the movem opcode.

4. PEA execution times are the same for (d16,PC)

5. PEA execution times are the same for (d8,PC,Xn*SF)

6. The execution time for STOP is the time required until the ColdFire2/2M begins sampling continuously for interrupts.

Freescale Semiconductor, Inc.

9.6 MAC INSTRUCTION EXECUTION TIMING

These instructions are supported on the ColdFire2M only.

Table 9-7. MAC Instruction Execution Times

OPCODE	<EA>	EFFECTIVE ADDRESS							
		RN	(AN)	(AN)+	-(AN)	(D16,AN) (D16,PC)	(D8,AN,XN*SF) (D8,PC,XN*SF)	XXX.WL	#XXX
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)	-	-	-	-	-	-	-
macl.w	Ry,Rx,<ea>,Rw	-	2(1/0)	2(1/0)	2(1/0)	2(1/0)†	-	-	-
macl.l	Ry,Rx,<ea>,Rw	-	4(1/0)	4(1/0)	4(1/0)	4(1/0)†	-	-	-
msacl.w	Ry,Rx,<ea>,Rw	-	2(1/0)	2(1/0)	2(1/0)	2(1/0)†	-	-	-
msacl.l	Ry,Rx,<ea>,Rw	-	4(1/0)	4(1/0)	4(1/0)	4(1/0)†	-	-	-

NOTE: †Effective address of (d16,PC) not supported

9.7 BRANCH INSTRUCTION EXECUTION TIMES

Table 9-8. General Branch Instruction Execution Times

OPCODE	<EA>	EFFECTIVE ADDRESS							
		RN	(AN)	(AN)+	-(AN)	(D16,AN)	(D8,AN,XI*SF)	XXX.WL	#XXX
bsr		—	—	—	—	3(0/1)	—	—	—
jmp	<ea>	—	3(0/0)	—	—	3(0/0)	4(0/0)	3(0/0)	—
jsr	<ea>	—	3(0/1)	—	—	3(0/1)	4(0/1)	3(0/1)	—
rte		—	—	8(2/0)	—	—	—	—	—
rts		—	—	5(1/0)	—	—	—	—	—

Table 9-9. BRA, Bcc Instruction Execution Times

OPCODE	FORWARD TAKEN	FORWARD NOT TAKEN	BACKWARD TAKEN	BACKWARD NOT TAKEN
bra	2(0/0)	—	2(0/0)	—
Bcc	3(0/0)	1(0/0)	2(0/0)	3(0/0)

APPENDIX A
REGISTER SUMMARY

A.1 REGISTER ACCESS METHODS

All of the ColdFire2/2M registers are accessed via special instructions or addressing modes. None of them are mapped into the user data address space. Table A-1 lists the ColdFire2/2M registers and their control register addresses.

Table A-1. Register Summary

REGISTER	ACRONYM	PROGRAM ACCESS[1]		DEBUG ACCESS[2]	
		INSTRUCTION	RC	COMMAND	DRC
Address Attribute Register	AATR	WDEBUG	$6	WDMREG	$6
Address Breakpoint Registers (High)	ABHR	WDEBUG	$C	WDMREG	$C
Address Breakpoint Registers (Low)	ABLR	WDEBUG	$D	WDMREG	$D
Access Control Register 0	ACR0	MOVEC	$004	RCREG,WCREG	$004
Access Control Register 1	ACR1	MOVEC	$005	RCREG,WCREG	$005
Accumulator[3]	ACC	MOVE to/from ACC	-	RCREG,WCREG	$806
Address Registers	A0-A6	MOVE	-	RAREG,WAREG	-
Cache Control Register	CACR	MOVEC	$002	RCREG,WCREG	$002
Condition Code Register	CCR	MOVE to/from CCR	-	RCREG,WCREG	$80E
Configuration/Status Register	CSR	WDEBUG	$0	RDMREG,WDMREG	$0
Data Breakpoint Mask Register	DBMR	WDEBUG	$F	WDMREG	$F
Data Breakpoint Register	DBR	WDEBUG	$E	WDMREG	$E
Data Registers	D0-D7	MOVE	-	RDREG,WDREG	-
MAC Status Register[3]	MACSR	MOVE to/from MACSR	-	RCREG,WCREG	$804
Mask Register[3]	MASK	MOVE to/from MASK	-	RCREG,WCREG	$805
Program Counter	PC	-	-	RCREG,WCREG	$80F
Program Counter Breakpoint Mask Register	PBMR	WDEBUG	$9	WDMREG	$9
Program Counter Breakpoint Register	PBR	WDEBUG	$8	WDMREG	$8
RAM Base Address Register	RAMBAR0	MOVEC	$C04	RCREG,WCREG	$C04
ROM Base Address Register	ROMBAR0	MOVEC	$C00	RCREG,WCREG	$C00

NOTES: 1. Refer to the *ColdFire Programmer's Reference Manual* (MCF5200PRM/AD).

2. Refer to **Section 7.3.3.1 BDM Command Set Summary**.

3. ColdFire2M only.

Freescale Semiconductor, Inc.

Table A-1. Register Summary (Continued)

REGISTER	ACRONYM	PROGRAM ACCESS[1]		DEBUG ACCESS[2]	
		INSTRUCTION	RC	COMMAND	DRC
Stack Pointer	A7,SP	MOVE	-	RAREG/WAREG	-
Status Register	SR	MOVE to/from SR	-	RCREG,WCREG	$80E
Trigger Definition Register	TDR	WDEBUG	$7	WDMREG	$7
Vector Base Register	VBR	MOVEC	$801	RCREG,WCREG	$801

NOTES: 1. Refer to the *ColdFire Programmer's Reference Manual* (MCF5200PRM/AD).

2. Refer to **Section 7.3.3.1 BDM Command Set Summary**.

3. ColdFire2M only.

A.1 REGISTER FORMATS

BITS	15	14 13	12 11	10 8	7	6 5	4 3	2 0
FIELD	RM	SZM	TTM	TMM	R	SZ	TT	TM
RESET	0	0	0	0	0	0	0	101
R/W	W	W	W	W	W	W	W	W

Figure A-2. Address Attribute Register (AATR)

BITS	31 0
FIELD	ADDRESS
RESET	-
R/W	W

Figure A-3. Address Breakpoint High Register (ABHR)

BITS	31 0
FIELD	ADDRESS
RESET	-
R/W	W

Figure A-4. Address Breakpoint Low Register (ABLR)

BITS	31		24	23		16
FIELD		AB			AM	
RESET		0			0	
R/W		W			W	

BITS	15	14	13	12		8	7	6	5	4	3	2	1	0
FIELD	EN	SM			-		ENIB	CM	BUFW		-	WP		-
RESET	0	0			-		0	0	0		-	0		-
R/W	W	W			-		W	W	W		-	W		-

Figure A-5. Access Control Register (ACR0, ACR1)

BITS	31	30	29	28	27	26	25	24	23		18	17	16
FIELD	CENB		-	CPSH	CFRZ		-	CINV		-		PARK	
RESET	0		-	0	0		-	0		-		0	
R/W	W		-	W	W		-	W		-		W	

BITS	15		11	10	9	8	7	6	5	4		2	1	0
FIELD		-		CBEN	CMOD	CBUF		-	CWRP		-		CLNF	
RESET		-		0	0	0		-	0		-		0	
R/W		-		W	W	W		-	W		-		W	

Figure A-6. Cache Control Register (CACR)

BITS	7	6	5	4	3	2	1	0
FIELD	-	-	-	X	N	Z	V	C
RESET	0	0	0	0	0	0	0	0
R/W	R	R	R	R/W	R/W	R/W	R/W	R/W

Figure A-7. Condition Code Register (CCR)

Freescale Semiconductor, Inc.

BITS	31		28	27	26	25	24	23		17	16
FIELD	STATUS			FOF	TRG	HALT	BKPT	-			IPW
RESET	0			0	0	0	0	-			0
R/W[1]	R			R	R	R	R	-			R/W

BITS	15	14	13	12	11	10	9	8	7	6	5	4	3		0
FIELD	MAP	TRC	EMU	DDC		UHE	BTB		-	NPL	IPI	SSM	-		
RESET	0	0	0	0		0	0		0	0	0	0	-		
R/W†	R/W	R/W	R/W	R/W		R/W	R/W		R	R/W	R/W	R/W	-		

NOTE: †The CSR is a write only register from the programming model. It can be read from and written to via the BDM port.

Figure A-8. Configuration/Status Register (CSR)

BITS	31	0
FIELD	MASK	
RESET	-	
R/W	W	

Figure 1-9. Data Breakpoint Mask Register (DBMR)

BITS	31	0
FIELD	ADDRESS	
RESET	-	
R/W	W	

Figure 1-10. Data Breakpoint Register (DBR)

BITS	7	6	5	4	3	2	1	0
FIELD	OMC	S/U	-	-	N	Z	V	C
RESET	0	0	0	0	-	-	-	0
R/W	R/W	R/W	R	R	R/W	R/W	R/W	R

Figure 1-11. MAC Status Register (MACSR)

BITS	15	0
FIELD	MASK	
RESET	-	
R/W	R/W	

Figure 1-12. MAC Mask Register (MASK)

BITS	31	0
FIELD	MASK	
RESET	-	
R/W	W	

Figure 1-13. Program Counter Breakpoint Mask Register (PBMR)

BITS	31	0
FIELD	ADDRESS	
RESET	-	
R/W	W	

Figure 1-14. Program Counter Breakpoint Register (PBR)

BITS	31	16
FIELD	BA	
RESET	-	
R/W	W	

BITS	15	9	8	7	6	5	4	3	2	1	0
FIELD	BA		WP	-		CSM	SCM	SDM	UCM	UDM	V
RESET	-		-	-	-	-	-	-	-	-	0
R/W	W		W	-		W	W	W	W	W	W

Figure A-15. SRAM Base Address Register (RAMBAR0)

BITS	31														16
FIELD	BA														
RESET	- (0)														
R/W	W														

BITS	15						9	8	7	6	5	4	3	2	1	0
FIELD	BA							WP	-		CSM	SCM	SDM	UCM	UDM	V
RESET	- (0)							- (1)	-		- (1)	- (0)	- (0)	- (0)	- (0)	0 (1)
R/W	W							W	-		W	W	W	W	W	W

NOTE: The reset values in parenthesis are valid if the ROM_VLD signal is asserted during reset.

Figure A-16. ROM Base Address Register (ROMBAR0)

BITS	15	14	13	12	11	10		8	7		5	4	3	2	1	0
FIELD	T	-	S	M	-	I			-			X	N	Z	V	C
RESET	0	0	1	1	0	7			0			0	0	0	0	0
R/W	R/W	R	R/W	R/W	R	R/W			R			R/W	R/W	R/W	R/W	R/W

Figure A-17. Status Register (SR)

BITS	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	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	W		W	W	W	W	W	W	W	W	W	W	W	W	W	W

BITS	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	-		0	0	0	0	0	0	0	0	0	0	0	0	0	0
R/W	-		W	W	W	W	W	W	W	W	W	W	W	W	W	W

Figure A-18. Trigger Definition Register (TDR)

APPENDIX B
NEW MAC INSTRUCTIONS

B.1 ENHANCED INTEGER-MULTIPLY INSTRUCTIONS

Opcodes for the MULS and MULU instructions are described in the *MCF5200 Family Programmer's Reference Manual*. The only change to these opcodes is the improved execution efficiency.

B.2 NEW MAC INSTRUCTIONS

This section describes the new MAC instructions for the ColdFire2M. Details of each instruction description **are** arranged in alphabetical order by instruction mnemonic. For notational conventions, refer to Table 1-4 in Section 1.8 Instruction Set Summary.

Freescale Semiconductor, Inc.

MAC MAC

<div style="text-align:center">**Multiply and Accumulate**</div>

Operation: ACC + ((Ry $\times$ Rx){<< 1 | >> 1}) $\rightarrow$ ACC

**Assembler
Syntax:**   MAC.<size> Ry.<ul>,Rx.

                 MAC.<size> Ry.<ul>,Rx.<ul>,<shift>

Attributes: size = (Word, Long)

                ul = (Upper, Lower)

                shift = (<<, >>)

Description: Multiplies two 16- or 32-bit numbers to produce a 32-bit result, then adds this product, optionally shifted left or right one bit, to the accumulator (ACC). The result is stored back into the accumulator. If 16-bit operands are used, the upper or lower word of each register must be specified.

MAC Status Register:

OMC	S/U	-	-	N	Z	V	C
-	-	0	0	*	*	*	0

OMC	-	not affected.
S/U	-	not affected.
N	-	set if the most significant bit of the result is set, otherwise cleared.
Z	-	set if the result is zero, otherwise cleared.
V	-	set if an overflow is generated, otherwise unchanged.
C	-	always cleared.

Processor Condition Codes:

     Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0		RW		0	0	RW	0	0			RX	
	-			SZ	SF		0	ULW	ULX			-			

Instruction Fields:

Ry - Source Y field

Specifies a source register operand, where $0 is D0,..., $7 is D7, $8 is A0,..., $F is A7. Note that bit ordering is 6, 11, 10, 9 (MSB to LSB).

Rx - Source X field

Specifies a source register operand, where $0 is D0,..., $7 is D7, $8 is A0,..., $F is A7. Note that bit ordering is 3, 2, 1, 0 (MSB to LSB).

SZ - Size field

> 0 = word-sized input operands
> 1 = long-sized input operands

SF - Scale Factor field

> 00 = none
> 01 = product << 1
> 10 = reserved
> 11 = product >> 1

U/Ly -Source YWord Select field

This bit determines which 16-bit operand of the source W register is used in the operation for word-sized operations only.

> 0 = lower word
> 1 = upper word

U/Lx - Source X Word Select field

This bit determines which 16-bit operand of the source X register is used in the operation for word-sized operations only.

> 0 = lower word
> 1 = upper word

MACL   Multiply and Accumulate   MACL
<div align="center">with Register Load</div>

Operation: $ACC + ((Ry \times Rx)\{<< 1 \mid >> 1\}) \rightarrow ACC$

                 $(<ea>\{\& \ MASK\}) \rightarrow Ry$

Assembler
Syntax: MACL.<size> Ry.<ul>,Rx.<ul>,<ea>,Rw

             MACL.<size> Ry.<ul>,Rx.<ul>,<shift>,<ea>,Rw

             MACL.<size> Ry.<ul>,Rx.<ul>,<shift>,<ea>&,Rw

Attributes: size = (Word, Long)

                ul = (Upper, Lower)

                shift = (<<, >>)

                ea = Effective Address

Description: Multiplies two **16-** or 32-bit numbers to produce a 32-bit result, then **adds** this product, optionally shifted left or right one bit, to the accumulator (ACC). The result is stored back into the accumulator. If 16-bit operands are used, the upper or lower word of each register must be specified. In parallel with this operation, a 32-bit operand is fetched from the memory location defined by <ea> and loaded into the destination register, Rw. If the mask addressing mode is used, the low-order word of <ea> is ANDed with the mask register. Refer to **Section 6.2.3 Mask Register (MASK)**.

MAC Status Register:

OMC	S/U	-	-	N	Z	V	C
-	-	0	0	*	*	*	0

OMC -   not affected.

S/U   -   not affected.

N   -   set if the most significant bit of the result is set, otherwise cleared.

Z   -   set if the result is zero, otherwise cleared.

V   -   set if an overflow is generated, otherwise unchanged.

C   -   always cleared.

Processor Condition Codes:

    Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0		RY		0	1	RY			<EA>			
											MODE		REG		
	RW			SZ		SF	0	U/LY	U/LX	MAM	0		RX		

Instruction Fields:

Ry -Source Yfield

Specifies a source register operand, where $0 is D0,..., $7 is D7, $8 is A0,..., $F is A7. Note that bit ordering is 15, 14, 13, 12 (MSB to LSB).

Rx - Source X field

Specifies a source register operand, where $0 is D0,..., $7 is D7, $8 is A0,..., $F is A7. Note that bit ordering is 3, 2, 1, 0 (MSB to LSB).

Rw - Destination field

Specifies a destination register operand, where $0 is D0,..., $7 is D7, $8 is A0,..., $F is A7. Note that bit ordering is 6, 11, 10, 9 (MSB to LSB).

<ea> - Effective Address of Memory Operand field

ADDRESSING MODE	MODE	REGISTER	ADDRESSING MODE	MODE	REGISTER
Dn	-	-	(xxx).W	-	-
An	-	-	(xxx).L	-	-
(An)	010	reg.num:An	#<data>	-	-
(An)+	011	reg.num:An			
-(An)	100	reg.num:An			
(d16,An)	101	reg.num:An	(d16,PC)	-	-
(d8,An,Xn)	-	-	(d8,PC,Xn)	-	-

SZ - Size field

    0 = word-sized input operands
    1 = long-sized input operands

SF - Scale Factor field

    00 = none
    01 = product << 1
    10 = reserved
    11 = product >> 1

U/Ly -Source YWord Select field

This bit determines which 16-bit operand of the source Yregister is used in the operation for word-sized operations only.

> 0 = lower word
> 1 = upper word

U/Lx - Source X Word Select field

This bit determines which 16-bit operand of the source X register is used in the operation for word-sized operations only.

> 0 = lower word
> 1 = upper word

MAM - Mask Addressing Mode Modifier

This bit determines if the mask addressing mode should be used. Refer to **Section 6.2.3 Mask Register (MASK)**.

> 0 = normal addressing mode
> 1 = mask addressing mode

MSAC **Multiply and Subtract** MSAC

Operation: ACC - ((Ry × Rx){<< 1 | >> 1}) → ACC

Assembler Syntax: MSAC.<size> Ry.<ul>,Rx.
MSAC.<size> Ry.<ul>,Rx.<ul>,<shift>

Attributes: size = (Word, Long)
ul = (Upper, Lower)
shift = (<<, >>)

Description: Multiplies two **16-** or 32-bit numbers to produce a 32-bit result, then **subtracts** this product, optionally shifted left or right one bit, from the accumulator (ACC). The result is stored back into the accumulator. If 16-bit operands are used, the upper or lower word of each register must be specified.

MAC Status Register:

OMC	S/U	-	-	N	Z	V	C
-	-	0	0	*	*	*	0

OMC	-	not affected.
S/U	-	not affected.
N	-	set if the most significant bit of the result is set, otherwise cleared.
Z	-	set if the result is zero, otherwise cleared.
V	-	set if an overflow is generated, otherwise unchanged.
C	-	always cleared.

Processor Condition Codes:

Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0		RY		0	0	RY	0	0		RX		
	-			SZ	SF		1	ULY	ULX				-		

Instruction Fields:

Ry - Operand Yfield

Specifies a source register operand, where $0 is D0,..., $7 is D7, $8 is A0,..., $F is A7. Note that bit ordering is 6, 11, 10, 9 (MSB to LSB).

Rx - Operand X field

Specifies a source register operand, where $0 is D0,..., $7 is D7, $8 is A0,..., $F is A7. Note that bit ordering is 3, 2, 1, 0 (MSB to LSB).

SZ - Size field

    0 = word-sized input operands
    1 = long-sized input operands

SF - Scale Factor field

    00 = none
    01 = product << 1
    10 = reserved
    11 = product >> 1

U/Ly -Source YWord Select field

This bit determines which 16-bit operand of the source Yregister is used in the operation for word-sized operations only.

    0 = lower word
    1 = upper word

U/Lx - Source X Word Select field

This bit determines which 16-bit operand of the source X register is used in the operation for word-sized operations only.

    0 = lower word
    1 = upper word

MSACL

MSACL   Multiply and Subtract   **MSACL**
with Register Load

Operation: ACC - ((Ry $\times$ Rx){<< 1 | >> 1}) $\rightarrow$ ACC
(<ea>{& MASK}) $\rightarrow$ Rw

Assembler
Syntax: MSACL.<size> Ry.<ul>,Rx.<ul>,<ea>,Rw
MSACL.<size> Ry.<ul>,Rx.<ul>,<shift>,<ea>,Rw
MSACL.<size> Ry.<ul>,Rx.<ul>,<shift>,<ea>&,Rw

Attributes: size = (Word, Long)
ul = (Upper, Lower)
shift = (<<, >>)
ea = Effective Address

Description: Multiplies two 16- or 32-bit numbers to produce a 32-bit result, then subtracts this product, optionally shifted left or right one bit, from the accumulator (ACC). The result is stored back into the accumulator. If 16-bit operands are used, the upper or lower word of each register must be specified. In parallel with this operation, a 32-bit operand is fetched from the memory location defined by <ea> and loaded into the destination register, Rw. If the mask addressing mode is used, the low-order word of <ea> is ANDed with the mask register. Refer to **Section 6.2.3 Mask Register (MASK)**.

MAC Status Register:

OMC	S/U	-	-	N	Z	V	C
-	-	0	0	*	*	*	0

OMC   -   not affected.
S/U   -   not affected.
N   -   set if the most significant bit of the result is set, otherwise cleared.
Z   -   set if the result is zero, otherwise cleared.
V   -   set if an overflow is generated, otherwise unchanged.
C   -   always cleared.

Processor Condition Codes:

Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0		RY		0	1	RY		MODE	<EA>		REG	
RW				SZ	SF		1	U/LY	U/LX	MAM	0	RX			

Instruction Fields:

Ry -Source Yfield

Specifies a source register operand, where $0 is D0,..., $7 is D7, $8 is A0,..., $F is A7. Note that bit ordering is 15, 14, 13, 12 (MSB to LSB).

Rx - Source X field

Specifies a source register operand, where $0 is D0,..., $7 is D7, $8 is A0,..., $F is A7. Note that bit ordering is 3, 2, 1, 0 (MSB to LSB).

Rw - Destination field

Specifies a destination register operand, where $0 is D0,..., $7 is D7, $8 is A0,..., $F is A7. Note that bit ordering is 6, 11, 10, 9 (MSB to LSB).

<ea> - Effective Address of Memory Operand field

ADDRESSING MODE	MODE	REGISTER		ADDRESSING MODE	MODE	REGISTER
Dn	-	-		(xxx).W	-	-
An	-	-		(xxx).L	-	-
(An)	010	reg.num:An		#<data>	-	-
(An)+	011	reg.num:An				
-(An)	100	reg.num:An				
(d16,An)	101	reg.num:An		(d16,PC)	-	-
(d8,An,Xn)	-	-		(d8,PC,Xn)	-	-

SZ - Size field

    0 = word-sized input operands
    1 = long-sized input operands

SF - Scale Factor field

    00 = none
    01 = product << 1
    10 = reserved
    11 = product >> 1

U/Ly -Source YWord Select field

This bit determines which 16-bit operand of the source Yregister is used in the operation for word-sized operations only.

0 = lower word
1 = upper word

U/Lx - Source X Word Select field

This bit determines which 16-bit operand of the source X register is used in the operation for word-sized operations only.

0 = lower word
1 = upper word

MAM - Mask Addressing Mode Modifier

This bit determines if the mask addressing mode should be used. Refer to **Section 6.2.3 Mask Register (MASK)**.

0 = normal addressing mode
1 = mask addressing mode

B.3 NEW REGISTER INSTRUCTIONS

This section describes the new register instructions for the ColdFire2M. **Details** of each instruction description **are** arranged in alphabetical order by instruction mnemonic. For notational conventions, refer to Table 1-4 in Section 1.8 Instruction Set Summary.

MOVE
from ACC

Move from Accumulator

MOVE
from ACC

Operation: ACC $\rightarrow$ Rx

**Assembler
Syntax:**   MOVE.<size> ACC, Rx

Attributes: size = Long

Description: **Moves** a 32-bit value from the accumulator (ACC) to a register. The size of the operation must be specified as long.

MAC Status Register:

Not affected

Processor Condition Codes:

Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3			0
1	0	1	0	0	0	0	1	1	0	0	0		RX		

Instruction Fields:

Rx[3:0] specifies the destination register, where $0 is D0,..., $7 is D7, $8 is A0,..., $F is A7.

MOVE
from MACSR   Move from MAC Status Reg

MOVE
from MACSR

Operation: MACSR $\rightarrow$ Rx[7:0]

0 $\rightarrow$ Rx[31:8]

**Assembler
Syntax:**   MOVE.<size> MACSR, Rx

Attributes: size = Long

Description: Moves the contents of the MAC status register (MACSR), zero-extended to long size, into a general-purpose register, Rx. The size of the operation must be specified as long.

MAC Status Register:

Not affected

Processor Condition Codes:

Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3			0
1	0	1	0	1	0	0	1	1	0	0	0	RX			

Instruction Fields:

Rx[3:0] specifies the destination register, where $0 is D0,..., $7 is D7, $8 is A0,..., $F is A7.

MOVE
from MASK

Move from Mask

MOVE
from MASK

Operation:	MASK $\rightarrow$ Rx[15:0]
	0xFFFF $\rightarrow$ Rx[31:16]

**Assembler
Syntax:**   MOVE.\<size\> MASK, Rx

Attributes: size = Long

Description: Moves a 32-bit value from the mask register (MASK), one-extended to long
size, to a register. The size of the operation must be specified as long.

MAC Status Register:

Not affected

Processor Condition Codes:

Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3			0
1	0	1	0	1	1	0	1	1	0	0	0			RX	

Instruction Fields:

Rx[3:0] specifies the destination register, where $0 is D0,..., $7 is D7, $8 is A0,..., $F is A7.

MOVE to ACC

Move to Accumulator

Operation: Source → ACC

Assembler Syntax: MOVE.<size> <ea>, ACC

Attributes: size = Long

Description: Moves a 32-bit value from a register or an immediate value into the accumulator (ACC). The size of the operation must be specified as long.

MAC Status Register:

OMC	S/U	-	-	N	Z	V	C
-	-	0	0	*	*	0	0

OMC - not affected.
S/U - not affected.
N - set if the most significant bit of the result is set, otherwise cleared.
Z - set if the result is zero, otherwise cleared.
V - always cleared.
C - always cleared.

Processor Condition Codes:

Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5		0
1	0	1	0	0	0	0	1	0	0	\<EA\>		
										MODE	REG	

Freescale Semiconductor, Inc.

Instruction Fields:

<ea> - Effective Address

ADDRESSING MODE	MODE	REGISTER	ADDRESSING MODE	MODE	REGISTER
Dn	000	reg.num:Dn	(xxx).W	-	-
An	001	reg.num:An	(xxx).L	-	-
(An)	-	-	#<data>	111	100
(An)+	-	-			
-(An)	-	-			
(d16,An)	-	-	(d16,PC)	-	-
(d8,An,Xn)	-	-	(d8,PC,Xn)	-	-

MOVE
to CCR

Move to Condition Code Register

Operation: MACSR[4:0] $\rightarrow$ CCR[4:0]

**Assembler
Syntax:**   MOVE.<size> MACSR,CCR

Attributes: size = Long

Description: Moves the indicator flags of the MAC status register (MACSR) into the processor's condition code cegister (CCR). The size of the operation must be specified as long.

MAC Status Register:

Not affected

Processor Condition Codes:

X	N	Z	V	C
-	*	*	*	*

X	-	not affected.
N	-	set to the value of MACSR bit 3, N.
Z	-	set to the value of MACSR bit 2, Z.
V	-	set to the value of MACSR bit 1, V.
C	-	set to the value of MACSR bit 0, C.

Instruction Format

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

MOVE
to MACSR

Move to MAC Status Register

Operation: Source → MACSR

**Assembler
Syntax:**   MOVE.<size> <ea>, MACSR

Attributes: size = Long

Description: Moves the low-order byte of a 32-bit value from a register or an immediate value into the MAC status register (MACSR). The size of the operation must be specified as long.

MAC Status Register:

OMC	S/U	-	-	N	Z	V	C
*	*	0	0	*	*	*	0

OMC -   set to the value of bit 7 of the source operand.
S/U -   set to the value of bit 6 of the source operand.
N   -   set to the value of bit 3 of the source operand.
Z   -   set to the value of bit 2 of the source operand.
V   -   set to the value of bit 1 of the source operand.
C   -   always cleared.

Processor Condition Codes:

Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5		0
											<EA>	
1	0	1	0	1	0	0	1	0	0	MODE		REG

Instruction Fields:

<ea> - Effective Address

ADDRESSING MODE	MODE	REGISTER		ADDRESSING MODE	MODE	REGISTER
Dn	000	reg.num:Dn		(xxx).W	-	-
An	001	reg.num:An		(xxx).L	-	-
(An)	-	-		#<data>	111	100
(An)+	-	-				
-(An)	-	-				
(d16,An)	-	-		(d16,PC)	-	-
(d8,An,Xn)	-	-		(d8,PC,Xn)	-	-

MOVE
to MASK

Move to Modulus Register

MOVE
to MASK

Operation: Source → MASK

Assembler
Syntax: MOVE.<size> <ea>, MASK

Attributes: size = Long

Description: Moves the low-order word of a 32-bit value from a register or an immediate value into the mask register (MASK). The size of the operation must be specified as long.

MAC Status Register:

   Not affected

Processor Condition Codes:

   Not affected

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5			0
1	0	1	0	1	1	0	1	0	0		<EA>		
										MODE		REG	

Instruction Fields:

<ea> - Effective Address

ADDRESSING MODE	MODE	REGISTER		ADDRESSING MODE	MODE	REGISTER
Dn	000	reg.num:Dn		(xxx).W	-	-
An	001	reg.num:An		(xxx).L	-	-
(An)	-	-		#<data>	111	100
(An)+	-	-				
-(An)	-	-				
(d16,An)	-	-		(d16,PC)	-	-
(d8,An,Xn)	-	-		(d8,PC,Xn)	-	-

B.4 OPERATION CODE MAP

All MAC instructions are mapped into line A, i.e. bits 15-12 of the instruction are 1010 ($A).

1. MAC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0		RY		0	0	RY	0	0			RX	
	-			SZ	SF		0	ULY	ULX			-			

2. MSAC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0		RY		0	0	RY	0	0			RX	
	-			SZ	SF		1	ULY	ULX			-			

3. MACL

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0		RY		0	1	RY		MODE		REG <EA>		
	RY			SZ	SF		0	U/LY	U/LX	MAM	0			RX	

4. MSACL

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0		RW		0	1	RY		MODE		REG <EA>		
	RY			SZ	SF		1	U/LY	U/LX	MAM	0			RX	

5. MOVE to ACC

15	14	13	12	11	10	9	8	7	6	5					0
1	0	1	0	0	0	0	1	0	0		MODE		REG <EA>		

6. MOVE to MACSR

15	14	13	12	11	10	9	8	7	6	5					0
1	0	1	0	1	0	0	1	0	0		MODE		REG <EA>		

7. MOVE to MASK

15	14	13	12	11	10	9	8	7	6	5					0
1	0	1	0	1	1	0	1	0	0		MODE		REG <EA>		

8. MOVE from ACC

15	14	13	12	11	10	9	8	7	6	5	4	3			0
1	0	1	0	0	0	0	1	1	0	0	0			RX	

Freescale Semiconductor, Inc.

9. MOVE from MACSR

15	14	13	12	11	10	9	8	7	6	5	4	3			0
1	0	1	0	1	0	0	1	1	0	0	0			RX	

10. MOVE from MASK

15	14	13	12	11	10	9	8	7	6	5	4	3			0
1	0	1	0	1	1	0	1	1	0	0	0			RX	

11. MOVE to CCR

15	14	13	12	11	10	9	8	7	6	5			0
											<EA>		
1	0	1	0	0	0	0	1	0	0	MODE		REG	

ColdFire2/2M User's Manual

INDEX

For More Information On This Product,
Go to: www.freescale.com