MOTOROLA

MCF5202

ColdFire™

Integrated Microprocessor

User's Manual

68K FAX-IT

Documentation Comments

FAX 512-891-8593—Documentation Comments Only

The Motorola High-Performance Embedded Systems Technical Communications Department provides a fax number for you to submit any questions or comments about this document or how to order other documents. We welcome your suggestions for improving our documentation. Please do not fax technical questions.

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.

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

For Internet Access:

Telnet: pirs.aus.sps.mot.com (Login: pirs)
WWW: http: / / pirs.aus.sps.mot.com/aesop/hmpg.html
Query By Email: aesop_query@pirs.aus.sps.mot.com
(Type ‘‘HELP’’ in text body.)

For Dial-Up:

Phone: +1-512-891-3650
Phone (US or Canada): 1-800-843-3451
Connection Settings: N/8/1/F
Data Rate: < 14,400 bps
Terminal Emulation: VT100
Login: pirs

For AESOP Questions:

FAX: +1-512-891-8775
EMAIL: aesop_sysop@pirs.aus.sps.mot.com

For Hotline Questions:

FAX (US or Canada): 1-800-248-8567
EMAIL: aesop_support@pirs.aus.sps.mot.com

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, Rosevllle	(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 Whltely	(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

PREFACE

The *MCF5202 ColdFire Integrated Microprocessor User's Manual* describes the programming, capabilities, and operation of the MCF5202 device. Refer to the *MCF5200 ColdFire Family Programmer's Reference Manual* for information on the ColdFire Family of microprocessors.

TRADEMARKS

ACRONYMS AND ABBREVIATIONS

The following acronyms and abbreviations are used throughout this manual:

ACR1, ACR2: Access Control Register 1; Access Control Register 2

BDM: Background Debug Mode

CACR: Cache Control Register

DS0: development serial ouput

DS1: development serial input

DSCLK: development serial clock

DRc: Debug Control Register

FIFO: first-in-first-out

IFP: instruction fetch pipeline

JTAG: Joint Test Action Group

LSB: least significant bit

MSB: most significant bit

OEP: operand execution pipeline

PC: program counter

SBC: system bus controller

SIM: system integration module

SR: status register

TAP: test access port

VBR: vector base register

TABLE OF CONTENTS

Section 1
Introduction

Section 2
Signal Description

Section 3
ColdFire Core

Section 4
Cache

Section 5
Bus Operations

Section 6
Debug Support

Section 7
JTAG Specification

Section 8
Porting from M68K Architecture

Section 9
Electrical Characteristics

Section 10
Mechanical Data

LIST OF FIGURES

Section 1
Introduction

Section 2
Signal Description

Section 3
ColdFire Core

Section 4
Cache

Section 5
Bus Operations

Section 6
Debug Support

Section 7
JTAG Specification

Section 8
Porting from M68K Architecture

Section 9
Electrical Characteristics

Section 10
Mechanical Data

LIST OF TABLES

Section 1
Introduction

Section 2
Signal Description

Section 3
ColdFire Core

Section 4
Cache

Section 5
Bus Operations

Section 6
Debug Support

Section 7
JTAG Specification

Section 8
Porting from M68K Architecture

Section 9
Electrical Characteristics

Section 10
Mechanical Data

Date: 7-28-98
Revision Number: 0.1
Pages affected: See change bars

SECTION 1
INTRODUCTION

ColdFire™ represents a revolutionary new microprocessor architecture that has been optimized for embedded processing applications. It brings new levels of price and performance to cost-sensitive high-volume markets. Based on the concept of variable-length RISC technology, ColdFire combines the architectural simplicity of conventional 32-bit RISC with a memory-saving, variable-length instruction set.

Employing a variable-length instruction set architecture, ColdFire RISC processors are tuned to offer embedded processor designers significant system-level advantages over conventional fixed-length RISC architectures. Softword code for ColdFire processors is denser and therefore takes up less memory than for any fixed-length instruction set RISC processor. This improved code density results in systems that require less memory for a given application and also allows the use of slower and less costly memory to achieve a given performance level. Denser code improves cache hit ratios and improves performance for a given size cache.

The MCF5202 processor is a ColdFire Family member that has been optimized for cost-effective performance in deeply embedded applications**.** The MCF5202 processor can operate on a 32-, 16-, or 8-bit external data bus.

1.1 FEATURES

The primary features of the MCF5202 processor include the following:

- Best-in-Class Code Density
  - — Requires less memory than fixed length RISC equivalents
  - — Allows use of slower memory for a given performance level than fixed-length RISCs
  - — Improves cache effectiveness
- Dynamic Bus Sizing
  - — 32-, 16-, and 8-bit bus support on the MCF5202 processor
- 2 kbyte On-Chip Unified Cache
  - — High performance 4-way set associative, non-blocking cache implementation
- Simple Instruction Set Architecture
  - — Optimized for high-level language constructs
  - — Requires minimal silicon area to implement processor core
  - — 16 user-visible 32-bit wide general-purpose registers
  - — Supervisor / user modes for system protection
  - — Vector base register to relocate exception-vector table
- Debug Module Including Background Debug and Real Time Debug Support
- Low Interrupt Latency
- Full Static Design Allows Operation Down to DC for Minimizing Power Consumption
- Three-State Pin
- JTAG IEEE 1149.1 Test Interface
- Single Clock Input

1.2 FUNCTIONAL BLOCKS

Figure 1-1 is a simplified block diagram of the MCF5202 processor. The MCF5202 device consists of a pipelined instruction execution unit, a two-kbyte unified cache, a debug module, and an external bus controller that supports the IEEE 1149.1 JTAG interface. The instruction execution unit is comprised of two separate pipelines that are decoupled by an instruction buffer. The instruction fetch pipeline (IFP) is responsible for instruction address generation and instruction fetch. The instruction buffer is a first-in-first-out (FIFO) buffer that holds prefetched instructions waiting for execution in the operand execution pipeline (OEP). The OEP includes two pipeline stages. The first decodes instructions and selects operands; the second calculates operand effective addresses, if needed, and performs instruction execution.

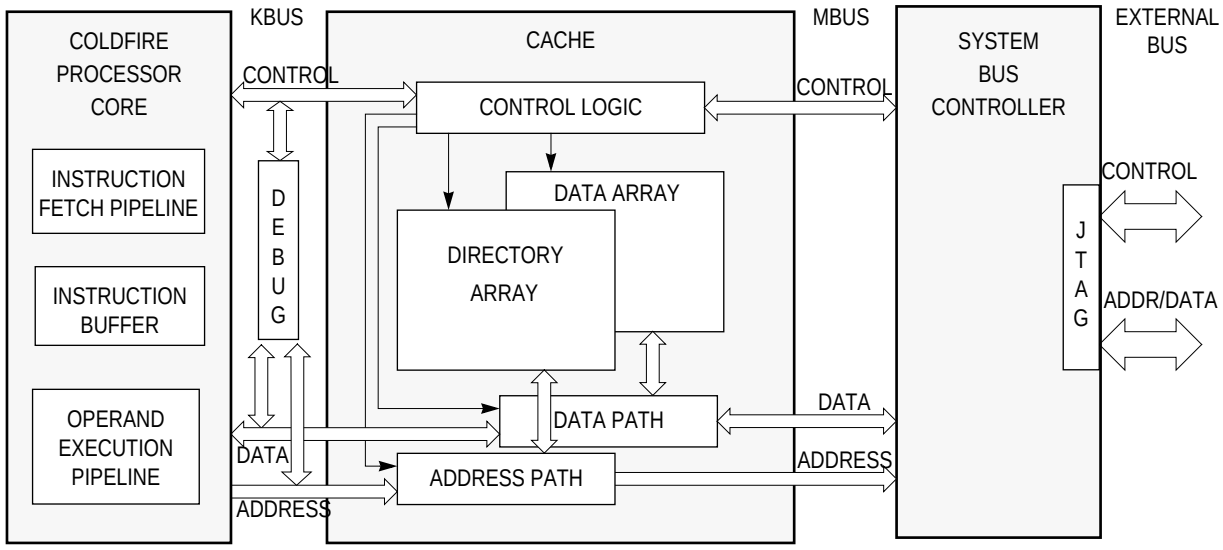


Figure 1-1. Block Diagram

The MCF5202 processor uses a unified data/instruction cache to improve overall system performance. The primary improvement is because of the availability of the most recently used instructions and data in a memory that can be accessed by the processor core in a single cycle. A second improvement is the increased external bus bandwidth available for alternate bus masters in the system. The nonblocking cache is organized as 4-way set associative and is physically mapped, thereby reducing software support for multitasking operating systems.

The bus controller performs bus transfers on the external bus. The MCF5202 bus controller supports a high-speed, multiplexed, synchronous, external bus interface. This interface in turn supports burst accesses for both reads and writes to provide high data transfer rates to and from the internal cache. The bus controller also supports the industry-standard IEEE 1149.1 JTAG interface.

1.3 PROCESSOR STATES

The processor is always in one of four states: normal processing, exception processing, stopped, or halted. It is in the normal processing state when executing instructions, fetching instructions and operands, and storing instruction results.

Exception processing is the transition from program processing to system, interrupt, and exception handling. Exception processing includes fetching the exception vector, stacking operations, and refilling the instruction fetch pipe after an exception. The processor enters exception processing when an exceptional internal condition arises such as tracing an instruction, an instruction resulting in a trap, or executing specific instructions. External conditions, such as interrupts and access errors, also cause exceptions. Exception processing ends when the first instruction of the exception handler enters the operand execution pipeline.

Stopped mode is a reduced power mode of operation that causes the processor to remain quiescent until either a reset or nonmasked interrupt occurs. The STOP instruction is used to enter this operation mode.

The processor halts when it receives an access error or generates an address error while in the exception processing state. For example, if during exception processing of one access error another access error occurs, the MCF5202 processor cannot complete the transition to normal processing nor can it save the internal machine state. The processor assumes that the system is not operational and halts. Only an external reset can restart a halted processor. When the processor executes a STOP instruction, it is in a special type of normal processing state, e.g., one without bus cycles. The processor stops but it does not halt.

The processor can also halt in a restart mode because of Background Debug Mode events.

1.4 PROGRAMMING MODEL

The ColdFire programming model is separated into two privilege modes: supervisor and user. The S-bit in the status register (SR) indicates the current privilege mode. The processor identifies a logical address by accessing either the supervisor or user address space, which differentiates between supervisor and user modes.

Programs access registers based on the indicated mode. User programs can access only registers specific to the user mode. System software executing in the supervisor mode can access all registers using the control registers to perform supervisory functions. User programs are thus restricted from accessing privileged information. The operating system performs management and service tasks for user programs by coordinating their activities. This difference allows the supervisor mode to protect system resources from uncontrolled accesses.

Most instructions execute in either mode but some instructions that have important system effects are privileged and can only execute in the supervisor mode. For instance, user programs cannot execute the STOP instructions. To prevent a program executing in user mode from entering the supervisor mode, instructions that can alter the S-bit in the SR are privileged. The TRAP instructions provide controlled access to operating system services

for user programs.

The processor employs the user mode and the user programming model when it is in normal processing. During exception processing, the processor changes from user to supervisor mode. Exception processing saves the current SR value on the stack and then sets the S-bit, forcing the processor into the supervisor mode. To return to the user mode, a system routine must execute a MOVE to SR, or an RTE, which operate in the supervisor mode, modifying the S-bit of the SR. After these instructions execute, the instruction fetch pipeline flushes and is refilled from the appropriate address space.

The registers depicted in the programming model (see Figure 1-2) provide operand storage and control for the ColdFire processor core. The registers are partitioned into two levels of privilege modes: user and supervisor. The user programming model consists of 16 general-purpose 32-bit registers and two control registers. The supervisor model consists of five more registers that can be accessed only by code running in supervisor mode.

Only system programmers can use the supervisor programming model to implement operating system functions and I/O control. This supervisor/user distinction allows for the coding of application software that will run without modification on any ColdFire Family processor. The supervisor programming model contains the control features that system designers would not want user code to erroneously access as this might effect normal system operation. Furthermore, the supervisor programming model may need to change slightly from ColdFire generation to generation to add features or improve performance as the architecture evolves.

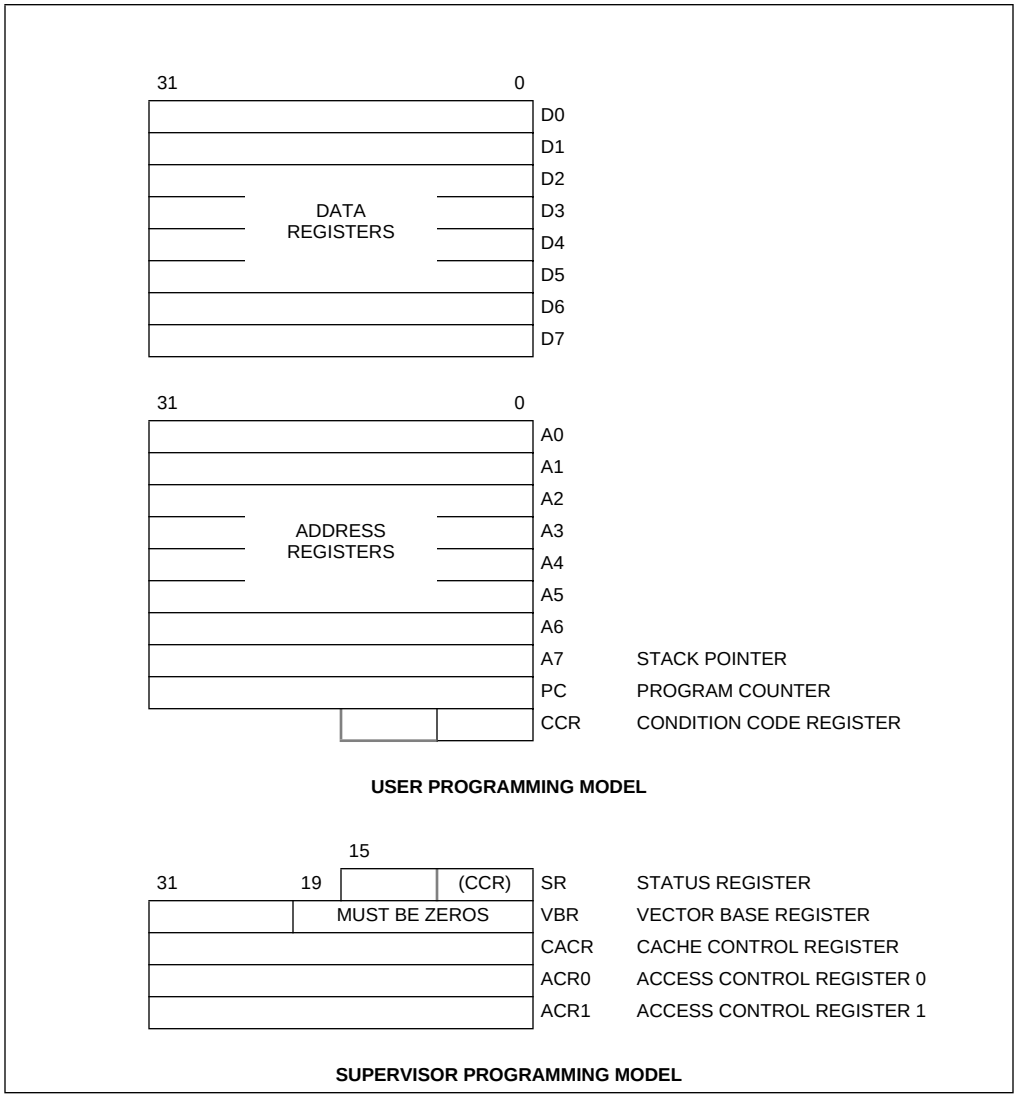


Figure 1-2. Programming Model

The user programming model includes eight data registers, seven address registers, and a stack pointer register. The address registers and stack pointer can be used as base address registers or software stack pointers, and any of the 16 registers can be used as index registers. Two control registers are available in the user mode—the program counter (PC), which contains the address of the instruction that the MCF5202 device is executing, and the lower byte of the SR, which is accessible as the Condition Code Register (CCR). The CCR contains the condition codes that reflect the results of a previous operation and can be used for conditional instruction execution in a program.

The supervisor programming model includes the upper byte of the SR, which contains operation control information. The Vector Base Register (VBR) contains the upper 12 bits of the base address of the exception vector table, which is used in exception processing. The

lower 20 bits of the VBR are forced to zero, allowing the vector table to reside on any 1 Mbyte memory boundary.

The Cache Control Register (CACR) controls enabling of the on-chip cache of the MCF5202 processor. There are two access control registers (ACR1, ACR0) that allow portions of the address space to be mapped as noncacheable. See Sections 4.3 and 4.4 for more details on these registers.

1.5 DATA FORMAT SUMMARY

The processor performs all arithmetic using 2's complement, but operands may be signed or unsigned. Registers, memory, or instructions themselves can contain operands. The operand size for each instruction is either explicitly encoded in the instruction or implicitly defined by the instruction operation. Table1-1 lists a summary of the MCF5202 data formats.

Table 1-1. ColdFire MCF5202 Data Formats

OPERAND DATA FORMAT	SIZE
Bit	1 Bit
Byte	8 Bits
Word	16 Bits
Longword	32 Bits

1.6 ADDRESSING CAPABILITIES SUMMARY

The MCF5202 processor supports seven addressing modes. The register indirect addressing modes support postincrement, predecrement, offset, and indexing, which are particularly useful for handling data structures common to sophisticated embedded applications and high-level languages. The program counter indirect mode also has indexing and offset capabilities. This addressing mode is typically required to support position-independent software. Besides these addressing modes, the MCF5202 processor provides index scaling features.

An instruction's addressing mode can specify the value of an operand or a register containing the operand. It can also specify how to derive the effective address of an operand in memory. Each addressing mode has an assembler syntax. Some instructions imply the addressing mode for an operand. These instructions include the appropriate fields for operands that use only one addressing mode. Table 1-2 lists a summary of the effective addressing modes of ColdFire processors.

Table 1-2. ColdFire Effective Addressing Modes

ADDRESSING MODES	SYNTAX
Register Direct	
Data	Dn
Address	An
Register Indirect	
Address	(An)
Address with Postincrement	(An)+
Address with Predecrement	–(An)
Address with Displacement	(d16,An)
Address Register Indirect with Index 8-Bit Displacement	(d_8,An,Xn)
Program Counter Indirect with Displacement	(d_{16},PC)
Program Counter Indirect with Index 8-Bit Displacement	(d_8,PC,Xn)
Absolute Data Addressing	
Short	(xxx).W
Long	(xxx).L
Immediate	#<xxx>

Table 1-3. Specific Effective Addressing Modes

ADDRESSING VARIANT	ALLOWABLE MODES
<ea-1>	Dn (An) (An)+ -(An) (d_{16},An)
<ea-2>	(An) (d_{16},An)

Table 1-4. MOVE Specific Effective Addressing Modes

SOURCE <EA>	DESTINATION <EA>
Dn	All
An	All
(An)	All
(An)+	All

Table 1-4. MOVE Specific Effective Addressing Modes (Continued)

SOURCE <EA>	DESTINATION <EA>
-(An)	All
(d_{16},An) (d_{16},PC)	Dn An (An) (An)+ -(An) (d_{16},An)
(d_8,An,Xn) (d_8,PC,Xn)	Dn An (An) (An)+ -(An)
(xxx).W (xxx).L	Dn An (An) (An)+ -(An)
#<xxx>	Dn An (An) (An)+ -(An)

1.7 NOTATIONAL CONVENTIONS

Table 1-5 lists the notation conventions used throughout this manual, unless otherwise specified.

Table 1-5. 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 destination register
Rc	Any Control Register (example VBR is the vector base register)
REGISTER/PORT NAMES	
DDATA	Debug Data Port
CCR	Condition Code Register (lower byte of status register)
PC	Program Counter
PST	Processor Status Port
SR	Status Register
MISCELLANEOUS OPERANDS	
#<data>	Immediate data following the instruction word(s)
<ea>	Effective Address
<ea>y,<ea>x	Source and Destination Effective Addresses, respectively
<label>	Assembly Program Label
<list>	List of registers (example: D3–D0)
<size>	Operand data size: Byte (B), Word (W), Longword (L)
OPERATIONS	
+	Arithmetic addition or postincrement indicator
–	Arithmetic subtraction or predecrement indicator
x	Arithmetic multiplication
/	Arithmetic division
~	Invert; operand is logically complemented
&	Logical AND
|	Logical OR
~	Logical exclusive OR
<<	Shift left (example: D0 << 3 is shift D0 left 3 bits)
>>	Shift right (example: D0 >> 3 is shift D0 right 3 bits)
→	Source operand is moved to destination operand
←→	Two operands are exchanged
sign-extended	All bits of the upper portion are made equal to the high-order bit of the lower portion
If <condition> then <operations> else <operations>	Test the condition. If true, the operations after ‘then’ are performed. If the condition is false and the optional ‘else’ clause is present, the operations after ‘else’ are performed. If the condition is false and else is omitted, the instruction performs no operation. Refer to the Bcc instruction description as an example.

SUBFIELDS AND QUALIFIERS	
{}	Optional Operation
()	Identifies an indirect address
d_n	Displacement Value, n-Bits Wide (example: d_{16} is a 16-bit displacement)
Address	Calculated Effective Address (pointer)
Bit	Bit Selection (example: Bit 3 of D0)
LSB	Least Significant Bit (example: MSB of D0)
LSW	Least Significant Word
MSB	Most Significant Bit
MSW	Most Significant Word
CONDITION CODE REGISTER BIT NAMES	
P	Branch Prediction Bit in CCR
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

1.8 INSTRUCTION SET OVERVIEW

The ColdFire instruction set supports high-level languages and is optimized for those instructions embedded code most commonly executes. Table 1-6Table 1-6 and Table 1-7 provide an alphabetized listing of the ColdFire instruction set opcode, operation, and syntax. Refer to Table 1-5 for notations used in Table 1-4 and Table 1-7. The left operand in the syntax is always the source operand and the right operand is the destination operand.

Table 1-6. 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 & Destination → Destination
ANDI	#<data>,Dx	32	Immediate Data & Destination → Destination
ASL	Dx,Dy #<data>,Dx	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; next sequential 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	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>	none	Address of <ea> → PC
JSR	<ea>	32	SP– 4 → SP; next sequential PC → (SP); <ea> → PC
LEA	<ea>y,Ax	32	<ea> → Ax
LINK	Ax,#<data>	16	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

INSTRUCTION	OPERAND SYNTAX	OPERAND SIZE	OPERATION
MOVE	<ea>y,<ea>x	8,16,32	<ea>y → <ea>x
MOVE from CCR	Dx	16	CCR → Dx
MOVE from SR	Dx	16	SR → Dx
MOVE to CCR	Dy,CCR #<data>,CCR	8	Dy → CCR #<data> → CCR
MOVE to SR	Dy,SR #<data>,SR	16	Source → SR
MOVEA	<ea>y,Ax	16,32 → 32	Source → Destination
MOVEC	Ry,Rc	32	Ry → 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
MULS	<ea>y,Dx	16 x 16 → 32 32 x 32 → 32	Source × Destination → Destination Signed operation
MULU	<ea>y,Dx	16 x 16 → 32 32 x 32 → 32	Source × Destination → Destination Unsigned operation
NEG	<ea>x	32	0 – Destination → Destination
NEGX	<ea>x	32	0 – Destination– X → Destination
NOP	none	none	PC + 2 → PC; Synchronize Pipelines
NOT	<ea>	32	~ Destination → Destination
OR	Dy,<ea>x <ea>y,Dx	32	Source | Destination → Destination
ORI	#<data>,Dx	32	Immediate Data | Destination → Destination
PEA	<ea>	32	SP – 4 → SP; Address of <ea> → (SP)
PULSE	none	none	Set PST= $4
RTE	none	none	(SP+2) → SR; SP+4 → SP; (SP) → PC; SP + FormatField → SP
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>y,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	Dn	16	MSW of Dn ←→ LSW of Dn
TRAP	none	none	SP – 4 → SP;PC → (SP); SP – 2 → SP;SR → (SP); SP – 2 → SP; Format → (SP); Vector Address → PC
TRAPF	none #<data>	none 16 32	PC + 2 → PC PC + 4 → PC PC + 6 → PC
TST	<ea>y	8,16,32	Set 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	2 x 32	<ea>y → Debug Module

SECTION 2
SIGNAL DESCRIPTION

2.1 INTRODUCTION

This section describes the specification for the external signals that will be used on the MCF5202 series of integrated circuits. A signal block diagram is shown in Figure 2-1 below.

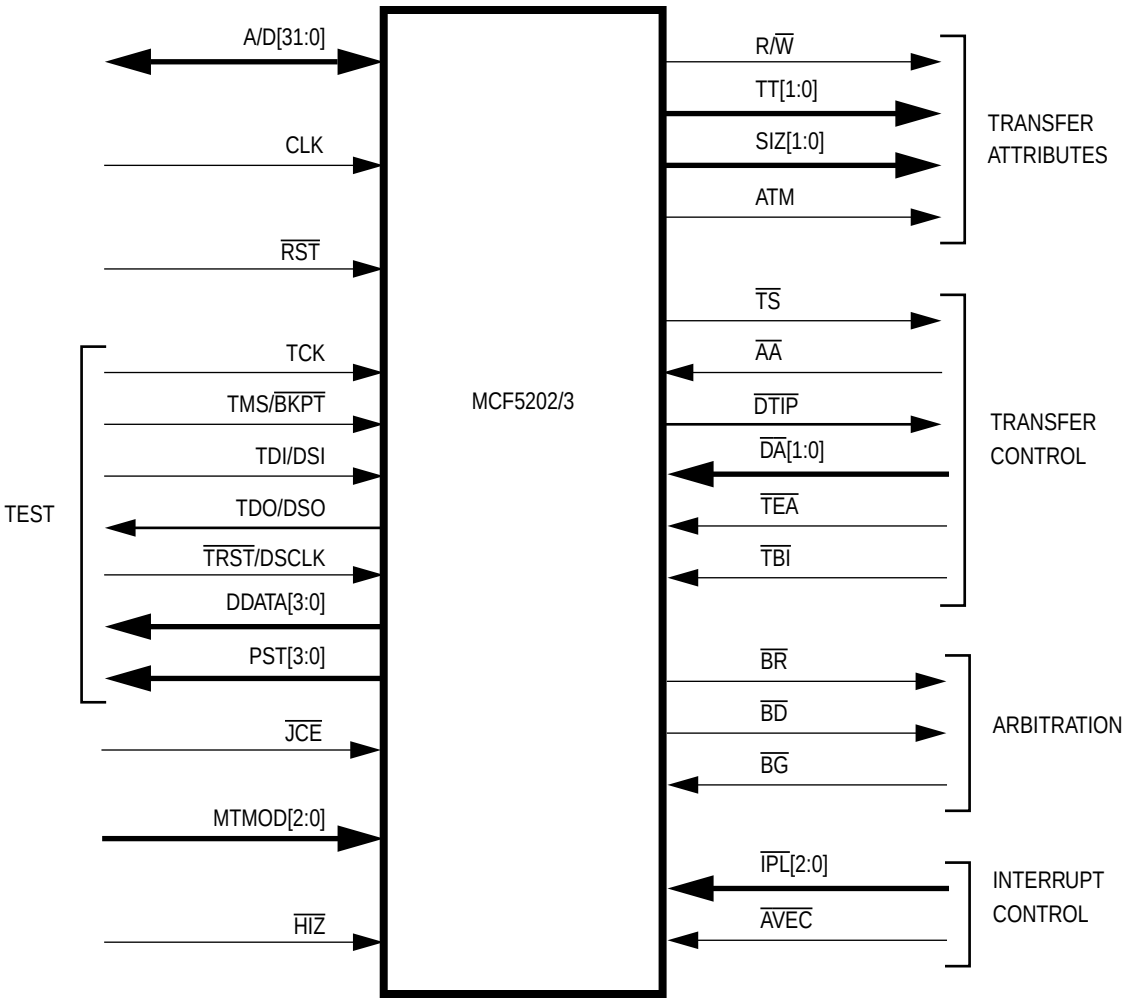


Figure 2-1. MCF5202 Block Diagram

Table 2-1: MCF5202 Signal Index

SIGNAL NAME	MNEMONIC	FUNCTION
Clock Input	CLK	Input used to clock internal logic
Reset	$\overline{\text{RST}}$	Processor Reset
Address, Data Lines	A/D[31:0]	Address/Data bus, time multiplexed providing access to 4Gbytes of memory
Read/Write	R/$\overline{\text{W}}$	Identifies read and write transfers
Size	SIZ[1:0]	Indicates the data transfer size
Transfer Type	TT[1:0]	Indicates the transfer type, normal, CPU space or emulator mode
Access/Mode	ATM	Multiplexed output signal indicating access type (instruction or data) during address phase and access mode (supervisor or user) during data phase
Transfer Start	$\overline{\text{TS}}$	Indicates the beginning of a bus transfer
Address Acknowledge	$\overline{\text{AA}}$	Assertion terminates address phase of transfer
Data Transfer in Progress	$\overline{\text{DTIP}}$	Assertion indicates access is in data phase
Data Acknowledge	$\overline{\text{DA}}$[1:0]	Indicates an acknowledge of a data transfer from either a 32-bit, 16-bit or 8-bit data port
Transfer Error Acknowledge	$\overline{\text{TEA}}$	Indicates an error condition for a bus transfer
Transfer Burst Inhibit	$\overline{\text{TBI}}$	Assertion indicates slave cannot handle a burst access
Bus Request	$\overline{\text{BR}}$	Indicates processor requires bus mastership
Bus Grant	$\overline{\text{BG}}$	Asserted by arbiter to grant mastership to processor
Bus Driven	$\overline{\text{BD}}$	Indicates processor is currently driving the bus
Interrupt Priority Level	$\overline{\text{IPL}}$[2:0]	Provides encoded interrupt level to processor
Autovector	$\overline{\text{AVEC}}$	Asserted during interrupt acknowledge cycle to request internal generation of vector number
Processor Status	PST[3:0]	Indicates internal processor status
Debug Data	DDATA[3:0]	Displays captured processor data and break-point status
Motorola Test Mode	MTMOD[2:0]	Test Mode signals tied to ground for normal usage
High ImpedAnce	$\overline{\text{HIZ}}$	Assertion three-states all output signal pins
Test Clock	TCK	Clock signal for IEEE 1149.1 Test Access Port (TAP)
Test Data Output/ Development Serial Output	TDO/DSO	Serial output for the TAP and debug module
Test Mode Select/ Break Point	TMS/$\overline{\text{BKPT}}$	TMS in JTAG mode and Hardware breakpoint in debug mode
Test Data Input / Development Serial Input	TDI/DSI	Serial input for the TAP and debug module
Test Reset/Development Serial Clock	$\overline{\text{TRST}}$/DSCLK	Asynchronous reset for TAP controller. Clock enable for debug module
JTAG Compliance Enable	$\overline{\text{JCE}}$	Test mode signal; should always be tied to a logic 0

2.2 ADDRESS AND CONTROL SIGNALS

2.2.1 Address/Data Lines - (A/D[31:0])

These bidirectional signals multiplex the address and data buses to external memory. During a bus cycle, the address is driven first, then data. When not the bus master, the address/data lines are three-stated. The MCF5203 processor will multiplex address and data on
A/D[31:16] and will maintain the valid address on A/D[15:0] for the duration of the bus cycle. During burst accesses, the MCF5203 device will increment A/D[3:0] to reflect the proper address for the current data acknowledge cycle. During interrupt-acknowledge cycles, the acknowledged level will be placed on A/D[4:2]; A/D[31:5] is driven high and

A/D[1:0] is driven low. For acknowledge cycles that are not autovectored, the vector number will be placed on the most significant bits (MSBs), A/D[31:24].

2.2.2 Read/Write - (R/$\overline{W}$)

When the MCF5202 processor is the bus master, it drives the R/W signal to indicate the direction of subsequent data transfers. It is driven high during read bus cycles and driven low during write bus cycles. This signal is three-stated when the MCF5202 device is not the bus master.

2.2.3 Transfer Start - ($\overline{TS}$)

The MCF5202 processor asserts this signal during the first clock cycle when a valid address is driven on A/D[31:0] and is negated in the following clock cycle. $\overline{TS}$ is three-stated when the MCF5202 device is not the bus master.

2.2.4 Address Acknowledge - ($\overline{AA}$)

The external system drives this input signal to terminate the address phase of the bus transfer. The address bus will continue to be driven until this synchronous signal is asserted. When it is asserted, the A/D bus will be driven with data if the access is a write or three-stated if the access is a read. The $\overline{AA}$ signal may be tied low if additional address valid time is not needed.

2.2.5 Size - (SIZ[1:0])

When it is the bus master, the MCF5202 processor outputs these signals to indicate the requested data transfer size. Table 2-2 shows the definition of the bus request size encodings. When the MCF5202 device is not the bus master, these signals are three-stated.

Table 2-2. Bus Cycle Size Encodings

SIZ[1:0]	BYTES	DATA BUS PORT SIZE		
		32-BIT (5202 ONLY)	16-BIT	8-BIT
00	4	Longword	2-Word Burst	4-Byte Burst
01	1	Byte	Byte	Byte
10	2	Word	Word	2-Byte Burst
11	16	4 - Longword Burst	8 - Word Burst	16-Byte Burst

2.2.6 Transfer Type - (TT[1:0])

These signals are output from the MCF5202 processor when it is the bus master indicating the type of access for the current bus access. Table 2-3 shows the encoding definitions.

Table 2-3. Bus Cycle Transfer Type Encoding

TT[1:0]	TRANSFER TYPE
0 0	Normal Access
0 1	Reserved
1 0	Emulator Access
1 1	CPU Space or Interrupt Acknowledge

2.2.7 Access Type and Mode - (ATM)

This output-only signal is time-multiplexed during bus transfers. During the address phase, ATM indicates whether the transfer is an instruction or data access. During the data phase, ATM indicates whether the transfer is a supervisor- or user-mode access.

A write access will indicate a cache push, with SIZ = 11 and the address phase of ATM indicating an instruction access.

During an interrupt-acknowledge cycle, the ATM address phase will indicate an instruction access. The data phase will default to user mode.

During emulator accesses, the ATM address phase will indicate the appropriate instruction/ data access. The data phase will default to supervisor mode. Table 2-4 shows the encodings of this signal.

Table 2-4. Access/Mode Encodings

ATM	PHASE	INDICATION
1	Address	Instruction Access
0	Address	Data Access
1	Data	Supervisor Mode Access
0	Data	User Mode Access

2.2.8 Data Transfer in Progress - ($\overline{\text{DTIP}}$)

The MCF5202 device asserts this signal to indicate that the bus cycle data phase is in progress. It is asserted on the rising CLK edge where $\overline{\text{AA}}$ is asserted. $\overline{\text{DTIP}}$ is negated at the completion of the current transfer. $\overline{\text{DTIP}}$ may be negated at any time in the bus cycle to indicate that the transfer has been aborted. When the MCF5202 processor is not the bus master, $\overline{\text{DTIP}}$ is three-stated.

2.2.9 Data Acknowledge - ($\overline{\text{DA}}$[1:0])

Data acknowledge indicates the completion of a data transfer operation. The device being accessed can indicate the successful completion of the transfer as well as its port size using $\overline{\text{DA}}$[1:0]. Table 2-5 shows the encodings for data-acknowledge signals.

Table 2-5. External Data Acknowledge Encodings

$\overline{\text{DA}}$[1:0]	TRANSFER INDICATION
00	Cycle complete - Data port size is 32 bits*
01	Cycle complete - Data port size is 16 bits
10	Cycle complete - Data port size is 8 bits
11	Cycle not complete - Wait state

*MCF5203 will default this to a data port size of 16 bits.

The slave device uses data acknowledge $\overline{\text{DA}}$[1:0] to indicate the port size, not the size of data being transferred. A 32-bit port device should always respond $\overline{\text{DA}}$[1:0]=00 regardless of the data size indicated for the transfer. In addition, a 32-bit port must always reside on A/D[31:0], a 16-bit port on A/D[31:16] and an 8-bit port must reside on A/D[31:24], as shown in Figure 2-2. The MSB of a longword operand is byte 0, with byte 3 being the LSB. The first data-acknowledge determines the port size for all subsequent transfers if the port size is smaller than the requested data size. With the MCF5203 device, if a port responds with $\overline{\text{DA}}$[1:0]=00, it will be treated as a 16-bit port.

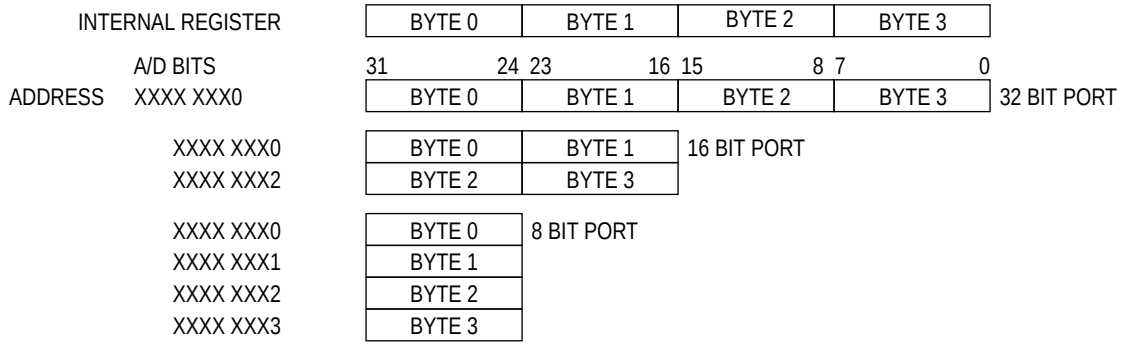


Figure 2-2. Data Bit Assignments to External Port Sizes

2.2.10 Transfer Error Acknowledge - ($\overline{\text{TEA}}$)

The external device being accessed asserts this input-only signal to indicate an error condition for the current bus transaction. Assertion of $\overline{\text{TEA}}$ during either the address phase or data phase aborts the access.

2.2.11 Transfer Burst Inhibit - ($\overline{\text{TBI}}$)

The device being accessed asserts this input-only signal to indicate it cannot support burst-mode accesses and that the requested burst transfer should be divided into individual transfers. $\overline{\text{TBI}}$ is sampled only on the first transfer of the burst cycle. If the first transfer of a burst cycle is terminated by asserting $\overline{\text{TBI}}$ with $\overline{\text{DA}}$[1] and/or $\overline{\text{DA}}$[0], the bus controller will terminate the burst cycle and access the remaining data as individual transfers.

2.3 BUS ARBITRATION

2.3.1 Bus Request - ($\overline{\text{BR}}$)

This output signal indicates to an external arbiter that the processor needs to become bus master for one or more bus cycles. $\overline{\text{BR}}$ is negated when the MCF5202 processor begins an access to the external bus with no other internal accesses pending, and $\overline{\text{BR}}$ remains negated until another internal request occurs.

2.3.2 Bus Grant - ($\overline{\text{BG}}$)

An external arbiter asserts this input signal to indicate the MCF5202 device can control the bus at the next rising edge of CLK. When the arbiter negates $\overline{\text{BG}}$, the MCF5202 processor must relinquish the bus as soon as the current transfer is complete. The external arbiter must not grant the bus to any other master until both $\overline{\text{BD}}$ and $\overline{\text{BG}}$ are negated.

2.3.3 Bus Driven - ($\overline{\text{BD}}$)

The MCF5202 device asserts this output signal to indicate that the MCF5202 is the current master. If the MCF5202 processor loses bus mastership during a bus transfer, it will complete the last transfer of the current access, negate $\overline{\text{BD}}$, and three-state all bus signals on the rising edge of CLK. $\overline{\text{DTIP}}$ will be driven to a high level and three-stated one clock after the other bus signals. If the MCF5202 device loses bus mastership during an idle clock cycle, it will three-state all bus signals on the rising edge of CLK.

2.4 INTERRUPT CONTROL

2.4.1 Interrupt Priority Level - ($\overline{\text{IPL}}$[2:0])

These input signals indicate the encoded priority level of a requested interrupt. $\overline{\text{IPL}}$[2:0] = 000 (Level 7) is the highest priority interrupt and cannot be internally masked. $\overline{\text{IPL}}$[2:0] = 111 (Level 0) indicates no interrupt is requested.

2.4.2 Autovector - ($\overline{\text{AVEC}}$)

$\overline{\text{AVEC}}$ asserted concurrently with $\overline{\text{DA}}$ during an interrupt-acknowledge bus cycle indicates a request for internal generation of the vector number. The data bus is not required to be

driven if $\overline{\text{AVEC}}$ is asserted. If autovectors are required for all interrupts, $\overline{\text{AVEC}}$ can be tied low.

2.5 CLOCK, RESET AND STATUS

2.5.1 Clock Input - (CLK)

CLK is the MCF5202 processor synchronous clock. CLK is used internally to clock or sequence the internal logic of the MCF5202 processor.

2.5.2 Reset ($\overline{\text{RST}}$)

Asserting $\overline{\text{RST}}$ will cause the MCF5202 processor to enter reset exception processing. When $\overline{\text{RST}}$ is recognized, the A/D bus, TT, SIZ, R/$\overline{\text{W}}$, $\overline{\text{DTIP}}$, ATM and $\overline{\text{TS}}$ will be three-stated. $\overline{\text{BR}}$ and $\overline{\text{BD}}$ will be negated.

2.5.3 Processor Status - (PST[3:0])

These outputs indicate the MCF5202 processor status. The timing is synchronous with the processor clock (CLK) and the status may not be related to the current bus transfer. Table 2-6 shows the encodings of these signals.

.

Table 2-6. MCF5202 Processor PST Definition

PST[3:0]	DEFINITION
0000	Continue execution
0001	Begin execution of an instruction
0010	Reserved
0011	Entry into user-mode
0100	Begin execution of **PULSE** instruction
0101	Begin execution of taken branch
0110	Reserved
0111	Begin execution of **RTE** instruction
1000	Begin 1-byte transfer on DDATA
1001	Begin 2-byte transfer on DDATA
1010	Begin 3-byte transfer on DDATA
1011	Begin 4-byte transfer on DDATA
1100	† Exception processing
1101	† Emulator-mode entry exception processing
1110	† Processor is stopped, waiting for interrupt
1111	† Processor is halted

† These encodings are asserted for multiple cycles

2.6 TEST

The MCF5202 supports JTAG and contains an internal debug module. Several of the control pins for these modes are multiplexed, so users should be careful when the system requires both modes.

2.6.1 Motorola Test Mode - (MTMOD[2:0])

These asynchronous signals determine the mode of operation for the MCF5202. Table 2-7 shows the encoding for these signals. For normal operations not using the debug module, tie these signals to a logic low.

Table 2-7. MTMOD Definition

MTMOD[2:0]	MODE
000	Functional mode - JTAG enabled
001	Functional mode - Debug enabled
010	Motorola reserved test modes
011	Motorola reserved test modes
100	Motorola reserved test modes
101	Motorola reserved test modes
110	Motorola reserved test modes
111	Motorola reserved test modes

2.6.2 Test Clock - (TCK)

TCK is the dedicated Test Access Port (TAP) clock that is independent of the MCF5202 processor clock. Various TAP operations occur on the rising or falling edge of TCK. The internal TAP controller logic is designed in such a way that holding TCK high or low for an indefinite period of time will not cause the TAP test logic to lose state information. If TCK is not used, it should be tied to ground.

2.6.3 Debug Data - DDATA[3:0]

This nibble-wide bus displays captured processor data and breakpoint status. See **Section 6 Debug Support** for additional information on this bus.

2.6.4 Test Reset/Development Serial Clock - ($\overline{\text{TRST}}$/DSCLK)

The MTMODE[2:0] signals determine the function of this dual-purpose pin. If MTMODE[2:0] = 000, the $\overline{\text{TRST}}$ function is selected. If MTMODE[2:0] = 001, the DSCLK function is selected. MTMODE[2:0] should not be changed while $\overline{\text{RST}}$ = 1. When used as $\overline{\text{TRST}}$, this pin will asynchronously reset the internal TAP controller to the test logic reset state, causing the TAP instruction register to choose the “bypass” command. When this occurs, all the TAP logic is benign and will not interfere with the normal functionality of the MCF5202 processor. Even with this asynchronous signal, Motorola recommends that $\overline{\text{TRST}}$ only make a 0 to 1 (asserted to negated) transition while TMS is held at a logic 1 value. $\overline{\text{TRST}}$ has an internal pullup so that if it is not driven low its value will default to a logic level of 1. However, if $\overline{\text{TRST}}$ will not be used, it can either be tied to ground or, if TCK is clocked, it can be tied to VDD. The former connection will place the TAP controller in the test logic reset state immediately, while the later connection will cause the TAP controller (if TMS is a logic 1) to eventually end up in the test logic reset state after 5 clocks of TCK.

This pin is also used as the development serial clock (DSCLK) for the serial interface to the debug module.The maximum frequency for the DSCLK signal is 1/2 the CLK frequency. See **Section 6 Debug Support** for additional information on this signal.

2.6.5 Test Mode Select/ Break Point (TMS/$\overline{\text{BKPT}}$)

The MTMODE[2:0] signals determine the function of this dual-purpose pin. If MTMODE[2:0] = 000, then the TMS function is selected. If MTMODE[2:0] = 001, the $\overline{\text{BKPT}}$ function is selected. MTMODE[2:0] should not change while $\overline{\text{RST}}$ = 1. When used as TMS, this input signal provides information to the TAP controller for determining which mode of test operation should be performed. The value of TMS and current state of the internal 16-state TAP controller state machine at the rising edge of TCK determine whether the TAP controller holds its current state or advances to a next state. This directly controls whether TAP data or instruction operations occur. TMS has an internal pullup so that if it is not driven low, its value will default to a logic level of 1. However, if TMS will not be used, it should be tied to VDD.

This pin also signals a hardware breakpoint to the processor when in the debug mode. See **Section 6 Debug Support** for additional information on this signal.

2.6.6 Test Data Input/Development Serial Input - (TDI/DSI)

This is a dual-function pin. If MTMODE[2:0] = 000, then TDI is selected. If MTMODE[2:0] = 001, then DSI is selected. When used as TDI, this input signal provides the serial data port for loading the various JTAG shift registers composed of the Boundary Scan Register, the Bypass Register, and the Instruction Register. Shifting in of data depends on the state of the TAP controller state machine and the instruction currently in the Instruction Register, and occurs on the rising edge of TCK. TDI also has an internal pullup so that if it is not driven low, its value will default to a logic level of 1. However, if TDI will not be used, it should be tied to VDD.

This pin also provides the single-bit communication for the debug module commands. See **Section 6 Debug Support** for additional information on this signal.

2.6.7 Test Data Output/Development Serial Output - (TDO/DSO)

This is a dual-function pin. When MTMODE[2:0] = 000, TDO is selected. When MTMODE[2:0] = 001, then DSO is selected. When used as TDO, this output signal provides the serial data port for outputting data from the TAP logic. Shifting out of data depends on the state of the TAP controller state machine and the instruction currently in the Instruction Register, and occurs on the falling edge of TCK. When TDO is not outputting test data, it is three-stated. TDO can also be placed in three-state mode to allow bussed or parallel connections to other devices that have a tap.

This signal also provides single-bit communication for the debug module responses. See **Section 6 Debug Support** for additional information on this signal.

2.6.8 High Impedance - ($\overline{\text{HIZ}}$)

The assertion of $\overline{\text{HIZ}}$ will force all output drivers to be in a high-impedance state. The timing on $\overline{\text{HIZ}}$ is independent of the clock. $\overline{\text{HIZ}}$ does not override the JTAG operation. TDO/DSO can be forced to high impedance by asserting $\overline{\text{TRST}}$.

2.6.9 JTAG Compliance Enable - ($\overline{\text{JCE}}$)

This Motorola test pin should be tied to a logic 0 at all times in a system.

Date: 7-28-98
Document revision Number: 0.1
Pages affected: see change bars

SECTION 3
COLDFIRE CORE

This section describes the organization of the ColdFire 5200 processor core and presents a brief description of the program-visible registers. For detailed information on instructions, see the ColdFire Programmer's Reference Manual.

3.1 PROCESSOR PIPELINES

Figure 3-1 is a block diagram showing the processor pipelines of a ColdFire 5200 core.

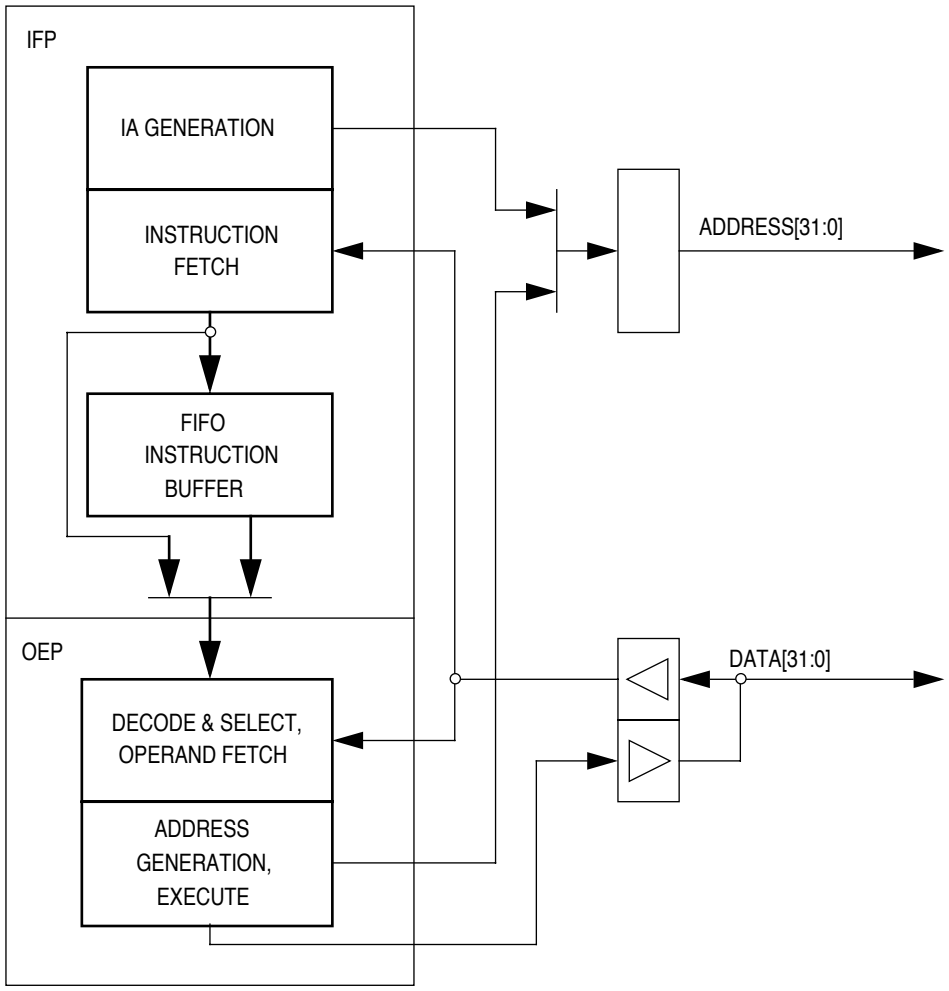


Figure 3-1. ColdFire Processor Core Pipelines

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

3.2 PROCESSOR REGISTER DESCRIPTION

The following paragraphs describe the processor registers in the user and supervisor programming models. The appropriate programming model is selected based on the privilege level of the processor as defined by the S-bit of the status register, i.e., user mode or supervisor mode.

3.2.1 User Programming Model

Figure 3-2 illustrates the user programming model. The model is the same as for M68000 Family microprocessors, consisting of the following registers:

- 16 general-purpose 32-bit registers (D0–D7, A0–A7)
- 32-bit program counter (PC)
- 8-bit condition code register (CCR)

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

3.2.1.2 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 longword operations.

3.2.1.3 STACK POINTER (A7). ColdFire supports a single hardware stack pointer (A7) for explicit references or implicit ones during stacking for subroutine calls and returns, and exception handling. The initial value of A7 is loaded from the reset exception vector, address $0. The same register is used for both user and supervisor mode and can be used for word and longword operations.

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

3.2.1.4 PROGRAM COUNTER. 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.

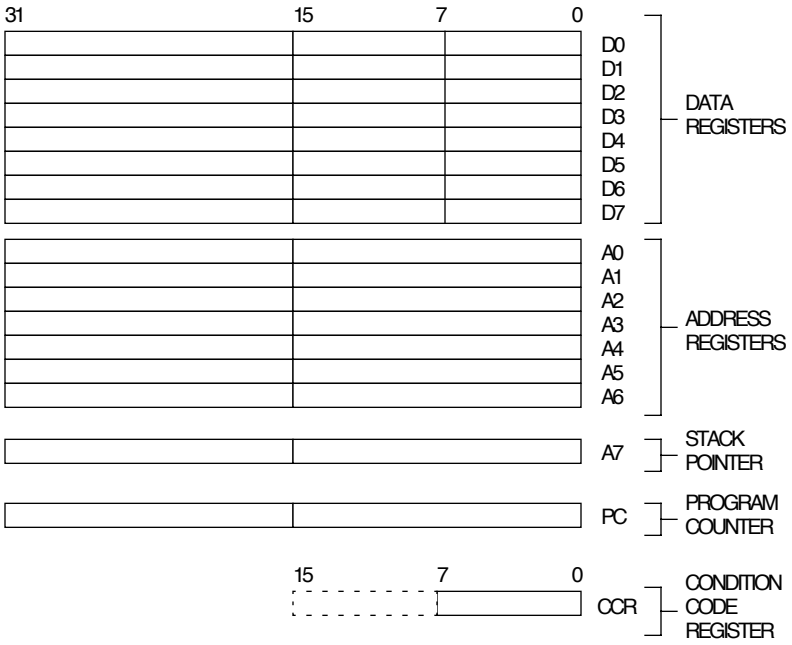


Figure 3-2. User Programming Model

3.2.1.5 CONDITION CODE REGISTER . The CCR is the least significant byte of the processor status register (SR), as shown. 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.

4	3	2	1	0
X	N	Z	V	C

Set to the value of the C-bit for arithmetic operations; otherwise not affected.

X— extend condition code bit

N– negative condition code bit

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

Z– zero condition code bit

Set if the result equals zero; otherwise cleared

V– overflow condition code bit

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

C– carry condition code bit

Set if a carryout of the operand MSB occurs for an addition, or if a borrow occurs in a subtraction; otherwise cleared

3.2.2 Supervisor Programming Model

Only system programmers use the supervisor programming model (see Figure 3-3) to implement sensitive operating system functions, I/O control, and memory management. All accesses that affect the control features of ColdFire 5200 processors are in the supervisor programming model, which consists of the registers available to users as well as the following control registers:

- 16-bit status register (SR)
- 32-bit vector base register (VBR)

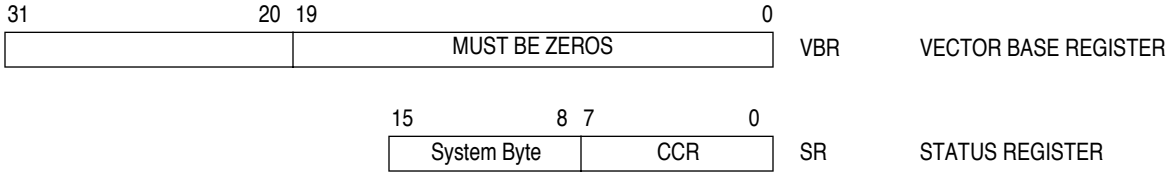


Figure 3-3. Supervisor Programming Model

Additional registers may be supported on a part basis.

The following paragraphs describe the supervisor programming model registers.

3.2.2.1 STATUS REGISTER. The SR (see Figure 3-4) stores the processor status and includes the CCR, the interrupt priority mask, and other control bits. In the supervisor mode, software can access the entire SR. In user mode, only the lower 8 bits are accessible (CCR).

The control bits indicate the following states for the processor: trace mode (T-bit), supervisor or user mode (S-bit), and master or interrupt state (M).

SYSTEM BYTE								CONDITION CODE REGISTER (CCR)							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T	0	S	M	0	I[2:0]			0	0	0	X	N	Z	V	C

Figure 3-4. Status Register

T– trace enable

When set, the processor will perform a trace exception after every instruction.

S– supervisor / user state

Denotes whether the processor is in supervisor mode (S=1) or user mode (S=0).

M– master / interrupt state

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

I[2:0]– interrupt priority mask

Defines the current interrupt priority. Interrupt requests are inhibited for all priority levels less than or equal to the current priority, except the edge-sensitive level 7 request, which cannot be masked.

3.2.2.2 VECTOR BASE REGISTER (VBR). The Vector Base Register in the ColdFire architecture is a 32-bit address register with only the upper 12 bits physically implemented in hardware. The low-order 20 bits are forced to zero when the CPU uses the VBR to calculate the exception vector address, effectively placing the vector table on a 0-modulo-1 MByte address.

The VBR may be written using the MOVEC instruction from the CPU, or from a BDM serial command. The register may be read from the BDM only. When a BDM read of the VBR is performed, the contents of the register are returned in the upper12 bits of the 32-bit result, with the low-order 20 bits being UNDEFINED.

The ColdFire 5200 processors provide a simplified exception processing model. The next section details the model.

3.3 EXCEPTION PROCESSING OVERVIEW

Exception processing for ColdFire processors is streamlined for performance. Differences from previous 68000 Family processors include:

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

ColdFire 5200 processors use an instruction restart exception model but do require more software support to recover from certain access errors. See **3.5.1 Access Error Exception** for details.

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 handler instruction has been initiated.

First, the processor makes an internal copy of the SR and then enters supervisor mode by asserting the S-bit and disabling trace mode by negating the T-bit. The occurrence of an interrupt exception also forces the M-bit to be cleared and the interrupt priority mask to be set to the level of the current interrupt request

Second, the processor determines the exception vector number. For all faults except interrupts, the processor performs this calculation based on the exception type. For interrupts, the processor performs an interrupt-acknowledge (IACK) bus cycle to obtain the vector number from a peripheral device. The IACK cycle is mapped to a special acknowledge address space with the interrupt level encoded in the address.

Third, the processor saves the current context by creating an exception stack frame on the system stack. ColdFire 5200 processors support a single stack pointer in the A7 address register; therefore, 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 processor uses a simplified fixed-length stack frame for all exceptions. The exception type determines whether the program counter placed in the exception stack frame defines the location of the faulting instruction (fault) or the address of the next instruction to be executed (next).

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

ColdFire 5200 processors support a 1024-byte vector table aligned on any 1 Mbyte address boundary (see Table 3-1). The table contains 256 exception vectors where the first 64 are defined by Motorola and the remaining 192 are user-defined interrupt vectors.

Table 3-1. Exception Vector Assignments

VECTOR NUMBER(S)	VECTOR OFFSET (HEX)	STACKED PROGRAM COUNTER	ASSIGNMENT
0	$000	-	Initial stack pointer
1	$004	-	Initial program counter
2	$008	Fault	Access error
3	$00C	Fault	Address error
4	$010	Fault	Illegal instruction
5-7	$014-$01C	-	Reserved
8	$020	Fault	Privilege violation
9	$024	Next	Trace
10	$028	Fault	Unimplemented line-a opcode
11	$02C	Fault	Unimplemented line-f opcode
12	$030	Next	Debug interrupt
13	$034	-	Reserved
14	$038	Fault	Format error
15	$03C	Next	Uninitialized interrupt
16-23	$040-$05C	-	Reserved
24	$060	Next	Spurious interrupt
25-31	$064-$07C	Next	Level 1-7 autovectored interrupts
32-47	$080-$0BC	Next	Trap # 0-15 instructions
48-63	$0C0-$0FC	-	Reserved
64-255	$100-$3FC	Next	User-defined interrupts

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

ColdFire 5200 processors inhibit sampling for interrupts during the first instruction of all exception handlers. This allows any handler to effectively disable interrupts, if necessary, by raising the interrupt mask level contained in the status register.

3.4 EXCEPTION STACK FRAME DEFINITION

The exception stack frame is shown in Figure 3-5. The first longword of the exception stack frame contains the 16-bit format/vector word (F/V) and the 16-bit status register, and the second longword contains the 32-bit program counter address.

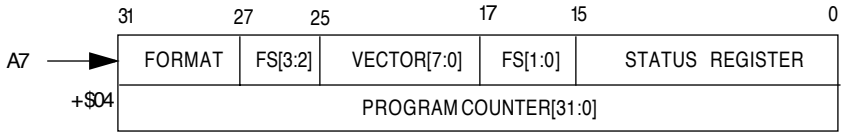


Figure 3-5. Exception Stack Frame Form

The 16-bit format/vector word contains 3 unique fields:

- A 4-bit format field at the top of the system stack is always written with a value of {4,5,6,7} by the processor indicating a two-longword frame format. See Table 3-2.

Table 3-2. Format Field Encodings

ORIGINAL A7 @ TIME OF EXCEPTION, BITS 1:0	A7 @ 1ST INSTRUCTION OF HANDLER	FORMAT FIELD
00	Original A7 - 8	4
01	Original A7 - 9	5
10	Original A7 - 10	6
11	Original A7 - 11	7

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

Table 3-3. Fault Status Encodings

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

- The 8-bit vector number, vector[7:0], defines the exception type and is calculated by the processor for all internal faults and represents the value supplied by the peripheral in the case of an interrupt. Refer to Table 3-1.

3.5 PROCESSOR EXCEPTIONS

3.5.1 Access Error Exception

The exact processor response to an access error depends on the type of memory reference being performed. For an instruction fetch, the processor postpones the error reporting until the faulted reference is needed by an instruction for execution. Therefore, faults that occur during instruction prefetches that are then followed by a change of instruction flow will not generate an exception. When the processor 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 processor immediately aborts the current instruction's execution and initiates exception processing. In this situation, any address register updates attributable to the auto-addressing modes, {e.g., (An)+,-(An)}, will already

have been performed. So, the programming model contains the updated An value. In addition, if an access error occurs during the execution of a MOVEM instruction loading from memory, any registers already updated before the fault occurs will contain the operands from memory.

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

3.5.2 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.

Any attempted use of a word-sized index register (Xn.w) or a scale factor of 8 on an indexed effective addressing mode generates an address error as does an attempted execution of a full-format indexed addressing mode.

3.5.3 Illegal Instruction Exception

The attempted execution of the $0000 and the $4AFC opwords generates an illegal instruction exception. Additionally, the attempted execution of any line A and most line F opcode generates their unique exception types, vector numbers 10 and 11 respectively. ColdFire 5200 processors do not provide illegal instruction detection on the extension words on any instruction, including MOVEC. If any other nonsupported opcode is executed, the resulting operation is undefined.

3.5.4 Privilege Violation

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

3.5.5 Trace Exception

To aid in program development, the ColdFire 5200 processors provide an instruction-by-instruction tracing capability. While in trace mode, indicated by the assertion of the T-bit in the status register (SR[15] = 1), the completion of an instruction execution signals a trace exception. This functionality allows a debugger to monitor program execution.

The single exception to this definition is the STOP instruction. When the STOP opcode is executed, the processor core waits until an unmasked interrupt request is asserted, then aborts the pipeline and initiates interrupt exception processing.

Because ColdFire processors do not support any hardware stacking of multiple exceptions, it is the responsibility of the operating system to check for trace mode after processing other exception types. As an example, consider the execution of a TRAP instruction while in trace mode. The processor will initiate the TRAP exception and then pass control to the corresponding handler. If the system requires that a trace exception be processed, it is the responsibility of the TRAP exception handler to check for this condition (SR[15] in the exception stack frame asserted) and pass control to the trace handler before returning from the original exception.

3.5.6 Debug Interrupt

This special type of program interrupt is discussed in detail in **Section 6, Debug Support**. This exception is generated in response to a hardware breakpoint register trigger. The processor does not generate an IACK cycle, but rather calculates the vector number internally (vector number 12).

3.5.7 RTE and Format Error Exceptions

When an RTE instruction is executed, the processor first examines the 4-bit format field to validate the frame type. For a ColdFire 5200 processor, any attempted execution of an RTE where the format is not equal to {4,5,6,7} generates a format error. The exception stack frame for the format error is created without disturbing the original RTE frame and the stacked PC pointing to the RTE instruction.

The selection of the format value provides some limited debug support for porting code from 68000 applications. On 680x0 Family processors, the SR was located at the top of the stack. On those processors, bit[30] of the longword addressed by the system stack pointer is typically zero. Thus, if an RTE is attempted using this “old” format, it generates a format error on a ColdFire 5200 processor.

If the format field defines a valid type, the processor: (1) reloads the SR operand, (2) fetches the second longword operand, (3) adjusts the stack pointer by adding the format value to the auto-incremented address after the fetch of the first longword, and then (4) transfers control to the instruction address defined by the second longword operand within the stack frame.

3.5.8 TRAP Instruction Exceptions

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

3.5.9 Interrupt Exception

The interrupt exception processing, with interrupt recognition and vector fetching, includes uninitialized and spurious interrupts as well as those where the requesting device supplies the 8-bit interrupt vector. Autovectoring may optionally be supported through the System Integration Module (SIM). Refer to the SIM section to see if this is supported on this device.

3.5.10 Fault-on-Fault Halt

If a ColdFire 5200 processor encounters any type of fault during the exception processing of another fault, the processor immediately halts execution with the catastrophic “fault-on-fault” condition. A reset is required to force the processor to exit this halted state.

3.5.11 Reset Exception

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

The reset exception places the processor in the supervisor mode by setting the S-bit and disables tracing by clearing the T-bit in the SR. This exception also clears the M-bit and sets the processor’s interrupt priority mask in the SR to the highest level (level 7). Next, the VBR is initialized to zero ($00000000). The control registers specifying the operation of any memories (e.g., cache and/or RAM modules) connected directly to the processor are disabled.

Note

Other implementation-specific supervisor registers are also affected. Refer to the specific user’s manual for details.

Once the processor 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. The first longword at address 0 is loaded into the stack pointer and the second longword at address 4 is loaded into the program counter. After the initial instruction is fetched from memory, program execution begins at the address in the PC. If an access error or address error occurs before the first instruction is executed, the processor enters the fault-on-fault halted state.

3.6 INSTRUCTION EXECUTION TIMING

This section presents ColdFire 5200 Family processor instruction execution times in terms of processor core clock cycles. The number of operand references for each instruction is enclosed in parentheses following the number of clock cycles. Each timing entry is presented as **C**(r/w) where:

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

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

3.6.1 Timing Assumptions

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

1. The operand execution pipeline (OEP) is loaded with the opword and all required extension words at the beginning of each instruction execution. This implies that the OEP does not wait for the instruction fetch pipeline (IFP) to supply opwords and/or extension words.
2. The OEP does not experience any sequence-related pipeline stalls. For ColdFire 5200 processors, the most common example of this type of stall involves consecutive store operations, excluding the MOVEM instruction. For all STORE operations (except MOVEM), certain hardware resources within the processor are marked as “busy” for two clock cycles after the final DSOC cycle of the store instruction. If a subsequent STORE instruction is encountered within this 2-cycle window, it will be stalled until the resource again becomes available. Thus, the maximum pipeline stall involving consecutive STORE operations is 2 cycles. The MOVEM instruction uses a different set of resources and this stall does not apply.
3. The OEP completes all memory accesses without any stall conditions caused by the memory itself. Thus, the timing details provided in this section assume that an infinite zero-wait state memory is attached to the processor core.
4. All operand data accesses are aligned on the same byte boundary as the operand size, i.e., 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, it is misaligned. The processor core decomposes the misaligned operand reference into a series of aligned accesses as shown in Table 3-4.

Table 3-4. Misaligned Operand References

ADDRESS[1:0]	SIZE	KBUS OPERATIONS	ADDITIONAL C(R/W)
X1	Word	Byte, Byte	2(1/0) if read 1(0/1) if write
X1	Long	Byte, Word, Byte	3(2/0) if read 2(0/2) if write
10	Long	Word, Word	2(1/0) if read 1(0/1) if write

3.6.2 MOVE Instruction Execution Times

The execution times for the MOVE.{B,W} instructions are shown in Table 3-5, while Table 3-6 provides the timing for MOVE.L.

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

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

Table 3-5. 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 3-6. 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)	—	—	—

3.7 STANDARD ONE OPERAND INSTRUCTION EXECUTION TIMES

Table 3-7. One Operand Instruction Execution Times

OPCODE	<EA>	EFFECTIVE ADDRESS							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
CLR.B	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
CLR.W	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
CLR.L	<ea>	1(0/0)	1(0/1)	1(0/1)	1(0/1)	1(0/1)	2(0/1)	1(0/1)	—
EXT.W	Dx	1(0/0)	—	—	—	—	—	—	—
EXT.L	Dx	1(0/0)	—	—	—	—	—	—	—
EXTB.L	Dx	1(0/0)	—	—	—	—	—	—	—
NEG.L	Dx	1(0/0)	—	—	—	—	—	—	—
NEGX.L	Dx	1(0/0)	—	—	—	—	—	—	—
NOT.L	Dx	1(0/0)	—	—	—	—	—	—	—
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)

3.8 STANDARD TWO OPERAND INSTRUCTION EXECUTION TIMES

Table 3-8. 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>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
AND.L	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
ANDI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
ASL.L	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
ASR.L	<ea>,Dx	1(0/0)	—	—	—	—	—	—	1(0/0)
BCHG	Dy,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1)	4(1/1)	—
BCHG	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	—	—	—
BCLR	Dy,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	5(1/1)	4(1/1)	—
BCLR	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	—	—	—
BSET	Dy,<ea>	2(0/0)	4(1/1)	41/1)	4(1/1)	4(1/1)	5(1/1)	4(1/1)	—
BSET	#imm,<ea>	2(0/0)	4(1/1)	4(1/1)	4(1/1)	4(1/1)	—	—	—
BTST	Dy,<ea>	2(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
BTST	#imm,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	—	—	1(0/0)
CMP.L	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
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)	11(1/0)	11(1/0)	11(1/0)	11(1/0)	12(1/0)	11(1/0)	9(0/0)
MULU.W	<ea>,Dx	9(0/0)	11(1/0)	11(1/0)	11(1/0)	11(1/0)	12(1/0)	11(1/0)	9(0/0)
MULS.L	<ea>,Dx	£ 18(0/0)	£ 20(1/0)	£ 20(1/0)	£ 20(1/0)	£ 20(1/0)	—	—	—
MULU.L	<ea>,Dx	£ 18(0/0)	£ 20(1/0)	£ 20(1/0)	£ 20(1/0)	£ 20(1/0)	—	—	—
OR.L	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
OR.L	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
ORI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
SUB.L	<ea>,Rx	1(0/0)	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	1(0/0)
SUB.L	Dy,<ea>	—	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
SUBI.L	#imm,Dx	1(0/0)	—	—	—	—	—	—	—
SUBQ.L	#imm,<ea>	1(0/0)	3(1/1)	3(1/1)	3(1/1)	3(1/1)	4(1/1)	3(1/1)	—
SUBX.L	Dy,Dx	1(0/0)	—	—	—	—	—	—	—

3.9 MISCELLANEOUS INSTRUCTION EXECUTION TIMES

Table 3-9. Miscellaneous Instruction Execution Times

OPCODE	<EA>	EFFECTIVE ADDRESS							
		Rn	(An)	(An)+	-(An)	(d16,An)	(d8,An,Xn*SF)	xxx.wl	#xxx
LINK.W	Ay,#imm	2(0/1)	–	–	–	–	–	–	–
MOVE.W	CCR,Dx	1(0/0)	–	–	–	–	–	–	–
MOVE.W	<ea>,CCR	1(0/0)	–	–	–	–	–	–	1(0/0)
MOVE.W	SR,Dx	1(0/0)	–	–	–	–	–	–	–
MOVE.W	<ea>,SR	7(0/0)	–	–	–	–	–	–	7(0/0) [1]
MOVEC	Ry,Rc	9(0/1)	–	–	–	–	–	–	–
MOVEM.L	<ea>,&list	–	1+n(n/0)	–	–	1+n(n/0)	–	–	–
MOVEM.L	&list,<ea>	–	1+n(0/n)	–	–	1+n(0/n)	–	–	–
NOP		3(0/0)	–	–	–	–	–	–	–
PEA	<ea>	–	2(0/1)	–	–	2(0/1) [3]	3(0/1) [4]	2(0/1)	–
PULSE		1(0/0)	–	–	–	–	–	–	–
STOP	#imm	–	–	–	–	–	–	–	3(0/0) [2]
TRAP	#imm	–	–	–	–	–	–	–	15(1/2)
TRAPF		1(0/0)	–	–	–	–	–	–	–
TRAPF.W		1(0/0)	–	–	–	–	–	–	–
TRAPF.L		1(0/0)	–	–	–	–	–	–	–
UNLK	Ax	2(1/0)	–	–	–	–	–	–	–
WDDATA	<ea>	–	3(1/0)	3(1/0)	3(1/0)	3(1/0)	4(1/0)	3(1/0)	3(1/0)
WDEBUG	<ea>	–	5(2/0)	–	–	5(2/0)	–	–	–

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

[1] If a MOVE.W #imm,SR instruction is executed and imm[13] = 1, the execution time is 1(0/0).

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

[3] PEA execution times are the same for (d16,PC)

[4] PEA execution times are the same for (d8,PC,Xn*SF)

3.10 BRANCH INSTRUCTION EXECUTION TIMES

Table 3-10. General Branch Instruction Execution Times

OPCODE	<EA>	EFFECTIVE ADDRESS							
		Rn	(An)	(An)+	-(An)	(d16,An) (d16,PC)	(d8,An,Xi*SF) (d8,PC,Xi*SF)	xxx.wl	#xxx
BSR		—	—	—	—	3(0/1)	—	—	—
JMP	<ea>	—	3(0/0)	—	—	3(0/0)	4(0/0)	3(0/0)	—
JSR	<ea>	—	3(0/1)	—	—	3(0/1)	4(0/1)	3(0/1)	—
RTE		—	—	10(2/0)	—	—	—	—	—
RTS		—	—	5(1/0)	—	—	—	—	—

Table 3-11. 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)

SECTION 4
CACHE

The MCF5202 contains a nonblocking, 2-kbyte, 4-way set-associative, unified (instruction and data) cache with a 16-byte line size. The cache improves system performance by providing low latency data to the MCF5202 instruction and data pipes. This decouples processor performance from system memory performance and increases bus availability for alternate bus masters.

The MCF5202 nonblocking cache services read hits or write hits from the processor while a fill (caused by a cache allocation) is in progress.

As shown in Figure 4-1, both instruction and data accesses are performed using a single bus connected to the cache. All addresses from the processor to the cache are physical addresses. If the address matches one of the cache entries, the access hits in the cache. For a read operation, the cache supplies the data to the processor, and for a write operation, the data from the processor updates the cache. If the access does not match one of the cache entries (misses in the cache) or a write access must be written through to memory, the cache performs a bus cycle on the MBUS and correspondingly on the external bus by way of the system bus controller (SBC). Throughout this chapter, all cache accesses on the MBUS have a corresponding access on the external bus by way of the SBC.

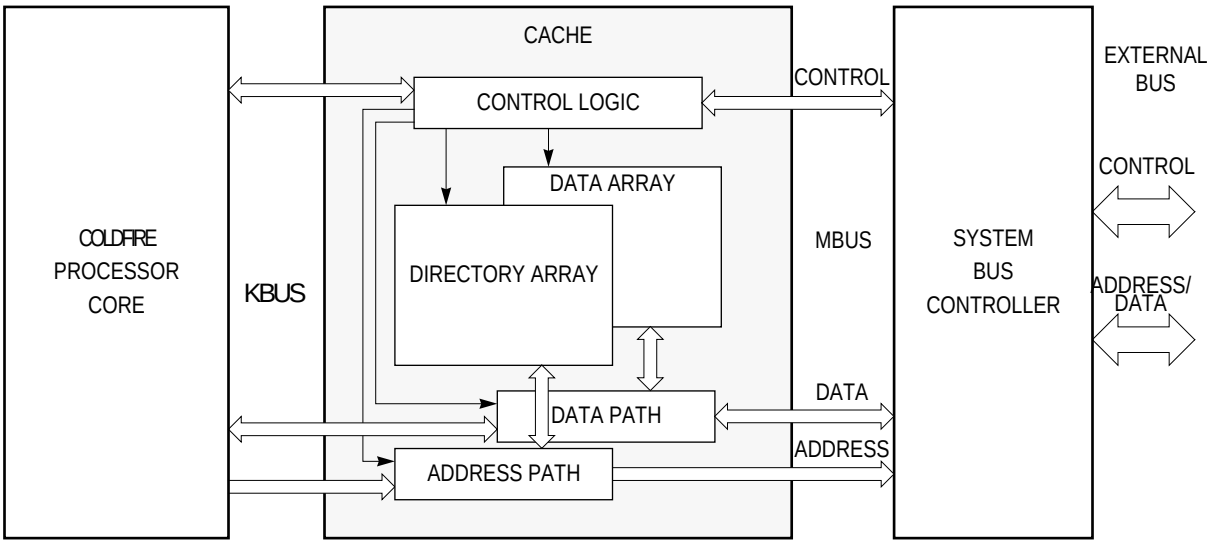


Figure 4-1. MCF5202 Unified Cache

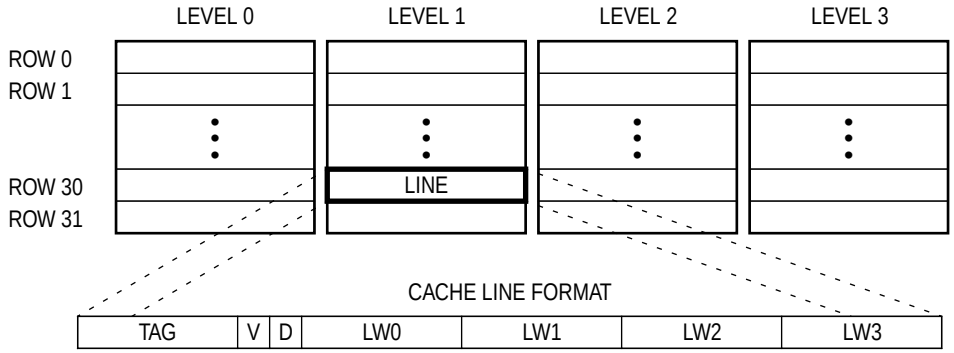
The MCF5202 does not implement a bus snooper. Users must maintain cache coherency with other possible bus masters via software.

4.1 CACHE ORGANIZATION

The 4-way set associative cache is organized as four levels of 32 lines each with each line containing 16 bytes of storage. Figure 4-2 illustrates the cache organization (as well as the terminology used) along with the cache line format.

Address bits A8–A4 provide an index to select a row. Levels are selected according to the rules of set association (discussed under **4.2 Cache Operation**).

Each line consists of an address tag (upper 23 bits of the address), two status bits and four longwords of data. The two status bits consist of a valid bit and a dirty bit for the line. Address bits A3 and A2 select the longword within the line.



WHERE:
TAG—23-BIT ADDRESS TAG
V—VALID BIT FOR LINE
D—DIRTY BIT FOR LINE
LWn—LONG WORD n (32-BIT) DATA ENTRY

Figure 4-2. Cache Organization and Line Format

4.2 CACHE OPERATION

The cache stores an entire line, thereby providing validity on a line-by-line basis. For burst-mode accesses, only those that successfully read four longwords can be cached.

A cache line is always in one of three states: invalid, valid, or dirty. For invalid lines, the V-bit is clear, causing the cache line to be ignored during lookups. Valid lines have their V-bit set and D-bit cleared, indicating the line contains valid data consistent with memory. Dirty cache lines have the V-bit and D-bit set, indicating that the line has valid entries that have not been written to memory. A cache line changes states from valid or dirty to invalid if the execution of the CPUSHL instruction explicitly invalidates the cache line**.** The cache should be explicitly cleared by setting the CINVA bit of the CACR after a hardware reset of the processor because reset does not invalidate the cache lines.

Figure 4-3 illustrates the general flow of a caching operation. To determine if the address is already allocated in the cache, the lower address bits 8–4 index into the cache and select one of 32 rows. A row is defined as the grouping of four lines, one from each level, corresponding to the same index into the cache array. Address bits 31–9 are used as a tag reference or to update the cache line tag field. The four tags from the selected cache row are compared with the tag reference. If any one of the four tags matches the tag reference and the tag status is either valid or dirty, then a cache hit has occurred. A cache hit indicates that the data entries (LW0–LW3) in that cache line contain valid data (for a read access) or can be written with new data (for a write access).

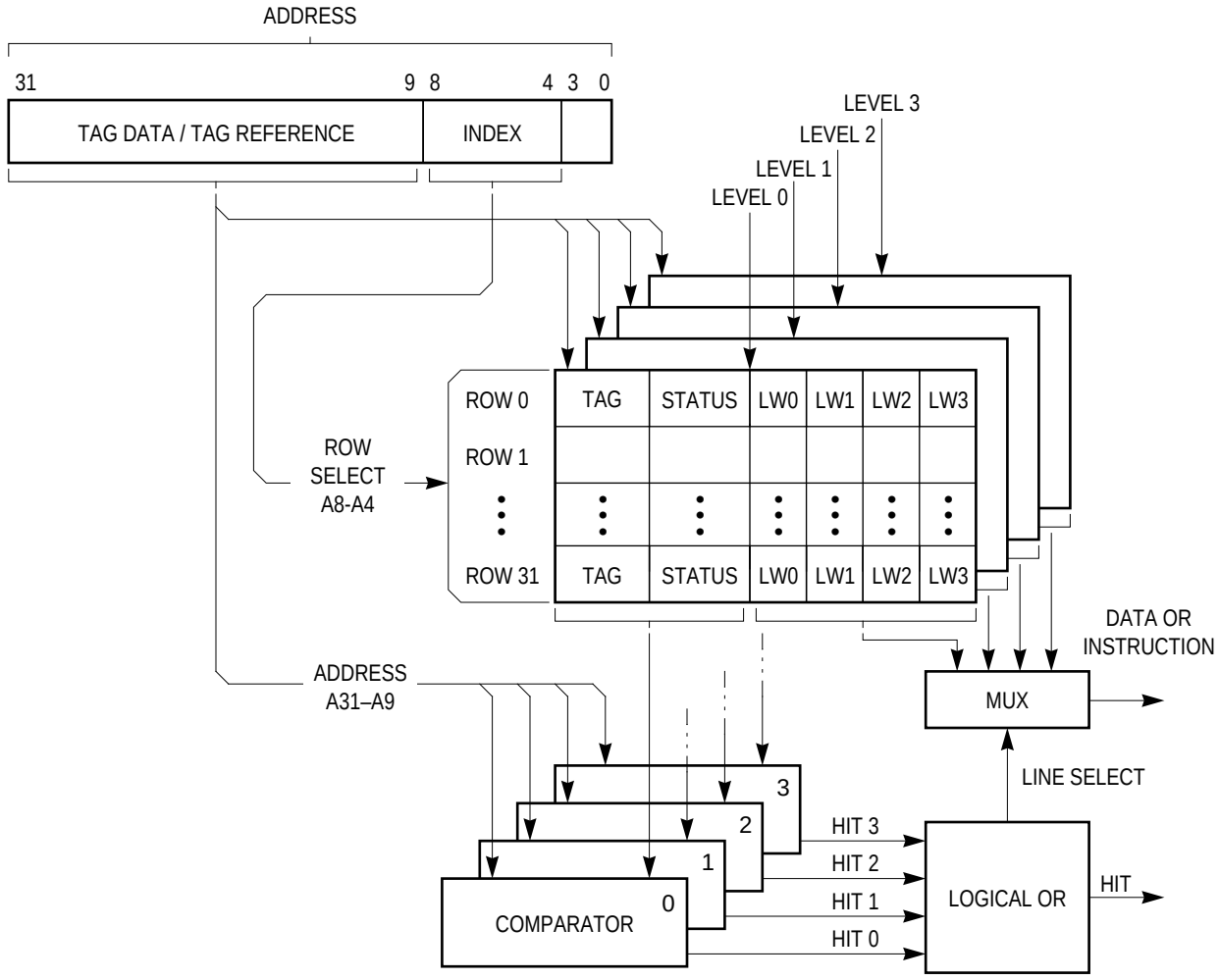


Figure 4-3. Caching Operation

To allocate an entry into the cache, the address bits 8–4 index into the cache and select one of the 32 rows. The status of each of the four cache lines for the selected row is examined. The cache control logic first looks for an invalid cache line to use for the new entry. If no invalid cache lines are available, a line from one of the four levels must be deallocated to host the new entry. The cache controller uses a pseudorandom replacement algorithm to determine which cache line will be deallocated and replaced. After a cache line is allocated,

the replacement pointer increments to point to the next level. During half-cache lock operation (HLCK equal to 1), the replacement pointer is forced to point to either level 2 or level 3.

In the process of deallocation, a cache line that is valid and not dirty is invalidated. A dirty cache line is placed in a push buffer (to do an external cache line push) before being invalidated. Once a cache line is invalidated, it can be replaced with a new entry.

When a cache line is selected to host a new cache entry, the new address bits 31–9 are written to the tag, the data bits LW3–LW0 are updated with the new memory data, and the cache line status changes to a valid state.

Read cycles that miss in the cache allocate normally as previously described. Write cycles that miss in the cache do not allocate on a cachable writethrough region, but do allocate for addresses in a cachable copyback region**.** A copyback byte, word, or longword write miss will cause the cache to initiate a line fill, allocate space for the new line, set the status bits to indicate valid and dirty, and write the data into the allocated space. No MBUS write to memory occurs. A copyback line write miss will not initiate a line fill, but will allocate space for the new line, set status bits to indicate valid and dirty, and write the data into the allocated space. No MBUS write to memory occurs and no MBUS line fill occurs.

Read hits do not change the status of the cache line and no deallocation or replacement occurs. Write hits in cacheable writethrough regions perform an MBUS write cycle; write hits in cacheable copyback regions do not perform an MBUS write cycle.

If the cache hits on a read access, data is driven back to the processor core. If the cache hits on a write access, the data is written to the appropriate portion of the accessed cache line. If the data access is misaligned, then the misalignment module breaks up the access into a sequence of smaller aligned cache accesses. Any misaligned operand reference generates at least 2 cache accesses. Because entry validity is provided only on a line basis, the entire line must be loaded from system memory on a cache miss for the cache to contain any valid information for that line address.

Noncacheable write accesses (i.e., those designated as cache-inhibited by the Cache Control Register (CACR) or Access Control Register (ACR)) bypass the cache and a corresponding MBUS write is performed. Normally, noncacheable read accesses bypass the cache and the read access is performed on the MBUS. The exception to this normal operation occurs when all of the following conditions are true during a noncacheable read:

- The appropriate noncacheable fill buffer bit (DNFB or NFB) is set
- Access is an instruction read
- Access is normal (i.e., transfer type (TT) equals 0)
- Access longword address is 0, 4, or 8 (i.e., the access is not referencing any of the last four bytes of a line)

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

Valid cache entries that match during noncachable address accesses are neither pushed nor invalidated. This scenario suggests that the associated cache mode for this address space was changed. Use the CPUSHL instruction to push and/or invalidate the cache entry, or set the CINVA bit of the CACR to invalidate the entire cache before switching cache modes.

4.3 CACHE CONTROL REGISTER (CACR)

The CACR is a 32-bit register that contains control information for the cache. The CACR can be written via the MOVEC register (register control field of the MOVEC instruction = $002). A hardware reset clears the CACR, disabling the cache; however, reset does not affect the tags, state information, and data within the cache. The CACR is illustrated in Figure 4-4.

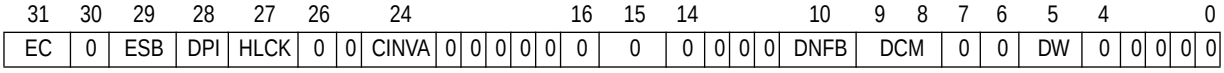


Figure 4-4. Cache Control Register (CACR)

EC—Enable Cache

0 = cache disabled
1 = cache enabled

Bit 30—Reserved

ESB — Enable Store Buffer

0 = all writes to writethrough or noncachable imprecise space will bypass the store buffer and generate bus cycles directly.
1 = the 4 entry first-in-first-out (FIFO) store buffer is enabled; this buffer defers pending writes to writethrough or cache-inhibited imprecise regions to maximize performance

Accesses to cache-inhibited precise space always bypass the store buffer.

DPI—Disable CPUSHL Invalidation

0 = each cache line is invalidated as it is pushed
1 = CPUSHLed lines remain valid in the cache

HLCK—1/2 Cache Lock Mode

0 = cache operates in normal, full cache mode
1 = cache operates in one-half cache lock mode

When this mode is enabled, levels 0 and 1 of the cache are locked such that their lines will never be displaced. Invalid entries in levels 0 and 1 can still be allocated. This implementation allows maximum use of the available cache memory and also provides

the flexibility of asserting the HLCK bit before, during, or after the needed allocations occur.

Bits 26–25—Reserved

CINVA—Cache Invalidate All

0 = no invalidation is performed
1 = initiate an invalidation of the entire cache

Writing a 1 to this bit will initiate entire cache invalidation. Once invalidation is complete, this bit will automatically return to 0 (i.e., users do not have to set it back to 0). This bit is always read as a 0.

Bits 23–11—Reserved

DNFB—Default Noncacheable Fill Buffer

0 = fill buffer is not used to store noncacheable accesses
1 = fill buffer is used to store noncacheable accesses
—fill buffer used only for normal (TT = 0) instruction reads of a noncacheable region from longword addresses of 0, 4, or 8
—the instructions are loaded into the fill buffer via a burst access (same as a line fill)

Note

It is possible that this feature can cause a coherency problem for self-modifying code. If enabled and a noncacheable access occurs that uses the fill buffer, the instructions remain valid in the fill buffer until a MOVEC, another noncacheable burst, or any miss that initiates a fill occurs. If a write occurs to the line in the fill buffer, the write will go to the MBUS without updating or invalidating the fill buffer. Any subsequent reads of that written data will be serviced by the fill buffer and receive stale information.

DCM—Default Cache Mode

This field selects the default cache mode and access precision as follows:

00 = cacheable, writethrough
01 = cacheable, copyback
10 = cache-inhibited, precise exception model
11 = cache-inhibited, imprecise exception model

Bits 7,6—Reserved

DW—Default Write Protect

This bit indicates the default write privilege.

0 = read and write accesses permitted
1 = write accesses not permitted

Bits 4–0—Reserved

4.4 ACCESS CONTROL REGISTERS

The 32-bit Access Control Registers (ACR0 and ACR1) assign access control attributes to specific regions of address space. The ACR registers can be written via the MOVEC instruction. (ACR0 has register control field of the MOVEC instruction = $004; ACR1 has register control field of the MOVEC instruction = $005). For overlapping regions, ACR0 takes priority. The control attributes are cache-mode and write-protection. Data transfers to and from these registers are longword transfers. Figure 4-5 illustrates ACR format. The paragraphs that follow describe the fields within the ACR. Bits 12–7, 4, 3, 1, and 0 always read as zero. At reset, all bits are reset to zero.

31 24	23 16	15	14 13	12	11	10	9	8	7	6 5	4	3	2	1	0
ADDRESS BASE	ADDRESS MASK	E	S-FIELD	0	0	0	0	0	NFB	CM	0	0	W	0	0

Figure 4-5. Access Control Register Format (ACR0, ACR1)

Bits 31–24— Address Base

This 8-bit field is compared with address bits A31–A24. Addresses that match in this comparison (and are otherwise eligible) are assigned the access control attributes of this register.

Bits 23–16— Address Mask

Because this 8-bit field contains a mask for the address base field, setting a bit in this field causes the corresponding bit in the address base field to be ignored. Regions of memory larger than 16 Mbytes can be assigned the access control attributes of this register by setting some of the address mask bits to ones (1’s). The low-order bits of this field can be set to define contiguous regions larger than 16 Mbytes. The mask can define multiple noncontiguous regions of memory.

E—Enable

This bit enables or disables the access control attributes of the region defined by this register:

0 = access control attributes disabled
1 = access control attributes enabled

S—Supervisor Mode

This field specifies the way FC2 matches an address:

00 = match only if FC2 = 0 (user mode access)
01 = match only if FC2 = 1 (supervisor mode access)
1X = ignore FC2 when matching

Bits 12–8—Reserved by Motorola

NFB—Noncacheable Fill Buffer

0 = fill buffer is not used to store noncacheable accesses
1 = fill buffer is used to store noncacheable accesses

—fill buffer used only for normal (TT = 0) instruction reads of a noncacheable region from longword addresses of 0, 4, or 8
—the instructions are loaded into the fill buffer via a burst access (same as a line fill)

Note

It is possible this feature can cause a coherency problem for self-modifying code. If enabled and a noncacheable access occurs that uses the fill buffer, the instructions remain valid in the fill buffer until a MOVEC, another noncacheable burst, or any miss that initiates a fill occurs. If a write occurs to the line in the fill buffer, the write will go to the MBUS without updating or invalidating the fill buffer. Any subsequent reads of that written data will be serviced by the fill buffer and receive stale information.

CM—Cache Mode

This field selects the cache mode and access precision as follows:

00 = cachable, writethrough
01 = cachable, copyback
10 = cache-inhibited, precise exception model
11 = cache-inhibited, imprecise exception model

W—Write Protect

This bit indicates the write privilege of the ACR region.

0 = read and write accesses permitted
1 = write accesses not permitted

Bits 4,3,1,0—Reserved by Motorola

4.5 CACHE MANAGEMENT

The cache is enabled and configured by using the MOVEC instruction to access the CACR. A hardware reset clears the CACR, disabling the cache and removing all configuration information; however, reset does not affect the tags, state information, and data within the cache. Users must set the CINVA bit in the CACR to invalidate the cache before enabling it.

The CINVA bit of the CACR allows invalidation of the entire cache only. The privileged CPUSHL instruction supports cache management by selectively pushing and invalidating an individual cache line. The address register used with the CPUSHL instruction directly addresses the cache's directory array. The CPUSHL instruction will either push and invalidate a line, or push and leave the line valid, depending on the state of the DPI bit of the CACR. To push the entire cache, users must implement a software loop to index through all 32 rows and each of the 4 lines within each row (for a total of 128 lines). The state of the cache enable bit in the CACR does not affect the operation of CPUSHL instruction nor the CINVA bit of the CACR.

The CPUSHL instruction flushes the MCF5202 cache. The instruction format is shown below.

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

where An is an address register.

The contents of An used with the CPUSHL instruction specify cache row and line indexes. This differs from the MC68040 where An specifies a physical address. The format for An is

31 … 9	8	7	6	5	4	3	2	1	0
0	Row Index					Line Index			

Bits 8-4 specify a row index and bits 3-0 specify a line index. On the MCF5202, only cache lines 0, 1, 2, and 3 are valid.

The following code example flushes the entire MCF5202 unified cache:

```
_cache_flush:
       nop                     ; synchronize - flush store buffer
       moveq.l      #0,d0      ; disable cache
       dc.l         0x4e7b0002 ; movec d0, cacr

       moveq.l      #0, d0     ; zero line counter
       moveq.l      #0, d1     ; zero row counter
       move.l       d0, a0     ; initialize An

rowloop:
       dc.w         0xf4e8     ; cpushl a0
       add.l        #0x0010, a0 ; increment row index by 1
       addq.l       #1, d1     ; increment row counter
       cmpi.l       #32, d1    ; check if rows for current line are done
       bne          rowloop    ; more rows to flush

       moveq.l      #0, d1     ; zero row counter
       addq.l       #1, d0     ; increment line counter
       add.l        d0, d1     ; form row and line for An
       move.l       d1, a0     ; initialize An
       cmpi.l       #4, d0     ; check if lines are done
       bne          rowloop

       rts
```

4.6 CACHING MODES

Every cache access has an associated caching mode that determines how the cache handles the access. An access can be cacheable in either the writethrough or copyback modes, or it can be cache-inhibited in precise or imprecise modes. For normal accesses, the CM field (from the ACR) corresponding to the address of the access specifies one of these caching modes. When the access address does not match either of the ACRs, the default caching mode defined by the DCM field of the CACR is used.

Addresses matching an ACR can also be write-protected using the W bit of that ACR. Address that do not match either of the ACRs can be write-protected using the DW bit of the CACR.

Reset disables the cache and places 0's in all CACR and ACR bits. Consequently, after reset, the defaults are writethrough cache mode and no addresses are write-protected. Note that users—and not reset—invalidate cache entries.

The ACRs allow the defaults to be overridden. In addition, some instructions (e.g., CPUSHL) and processor core operations perform accesses that have an implicit caching mode associated with them. The following paragraphs discuss the different caching accesses and their related cache modes.

4.6.1 Cacheable Accesses

If the CM field of an ACR or the default field of the CACR indicates writethrough or copyback, then the access is cacheable. A read access to a writethrough or copyback region is read from the cache if matching data is found. Otherwise, the data is read from memory and updates the cache. When a line is being read from memory, for both a writethrough read miss and a copyback read miss, the longword within the line that contains the core-requested data is fetched first and the requested data is given immediately to the processor. This releases the processor while the remaining three longwords of the line are read from memory and stored in the cache.

The following paragraphs describe the writethrough and copyback modes in detail.

4.6.1.1 WRITETHROUGH MODE. Write accesses to regions specified as writethrough are always passed on to the MBUS, although the cycle can be buffered (depending on the state of the ESB bit in the CACR). Writes in writethrough mode are handled with a no-write-allocate policy—i.e., writes that miss in the cache are written to the MBUS, but do not cause the corresponding line in memory to be loaded into the cache. Write accesses that hit always write through to memory and update matching cache lines. The cache supplies data to instruction or data-read accesses that hit in the cache; read misses cause a new cache line to be loaded into the cache.

4.6.1.2 COPYBACK MODE. Copyback regions are typically used for local data structures or stacks to minimize external bus use and reduce write-access latency. Write accesses to regions specified as copyback that hit in the cache update the cache line and set the corresponding D-bit without an MBUS bus access. The dirty cache data is written to memory only if the line is replaced because of a miss or a CPUSHL instruction pushes the line. If a byte, word, or longword write access misses in the cache, then the required cache line is read from memory, thereby updating the cache. If a line write access misses in the cache, the cache line will be completely sourced by the core and thus a cache line read from memory is avoided. When a miss causes a dirty cache line to be selected for replacement, the current cache data moves to the push buffer. The replacement line is read into the cache and the push buffer contents are then written to memory.

4.6.2 Cache-Inhibited Accesses

Address space regions containing targets such as I/O devices and shared data structures in multiprocessing systems can be designated as cache-inhibited. If the corresponding CM field (of the ACR) or DCM field (of the CACR) indicate precise or imprecise, then the access is cache-inhibited. The caching operation is identical for both cache-inhibited modes. The difference between these inhibited cache modes has to do with recovery from an external bus error.

Noncacheable write accesses bypass the cache and a corresponding MBUS write is performed. Normally, noncacheable read accesses bypass the cache and the read access is performed on the MBUS. The exception to this normal operation occurs when all of the following conditions are true during a noncacheable read:

- The appropriate noncacheable fill buffer bit (DNFB or NFB) is set
- Access is an instruction read
- Access is normal (i.e., transfer type (TT) equals 0)
- Access longword address is 0, 4, or 8 (i.e., the access is not referencing any of the last four bytes of a line)

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

If the CM field indicates either noncacheable precise or noncacheable imprecise modes, the cache controller bypasses the cache and performs an MBUS transfer. If a cache line matching the current address is already resident in the cache and the cache mode for that region is cache-inhibited, the cache does not automatically push the line if it is dirty, nor does it invalidate the line if it is valid. Users should first execute a CPUSHL instruction or set the CINVA bit of the CACR (to invalidate the entire cache) prior to switching the cache mode.

If the CM field indicates precise mode, then the sequence of read and write accesses to the region is guaranteed to match the sequence of the instruction order. In imprecise mode, the processor core allows read accesses that hit in the cache to occur before completion of a pending write from a previous instruction. Writes will not be deferred past operand-read accesses that miss in the cache (i.e., that must be read from the bus). Precise operation forces operand-read accesses for an instruction to occur only once by preventing the instruction from being interrupted after the operand-fetch stage. Otherwise, *if not in precise mode* and an exception occurs, the instruction is aborted and the operand may be accessed again when the instruction is restarted. These guarantees apply only when the CM field indicates the precise mode and the accesses are aligned.

All CPU space-register accesses (e.g. MOVEC) are always treated as noncacheable precise.

4.7 CACHE PROTOCOL

The following paragraphs describe the cache protocol for processor accesses and assumes that the data is cacheable (i.e., writethrough or copyback).

4.7.1 Read Miss

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

4.7.2 Write Miss

The cache controller handles processor writes that miss in the cache differently for writethrough and copyback regions. Byte, word, or longword write misses to copyback regions cause an MBUS line read to load the cache line. Line write misses to copyback regions do not cause an MBUS line read to load the cache line. The line is completely sourced by the core, avoiding the line read from memory. The new cache line is then updated with write data and the D-bit for the line is set, leaving the cache line in the dirty state. Write misses to writethrough regions write directly to memory without loading the corresponding cache line into the cache.

4.7.3 Read Hit

On a read hit, the cache provides the data to the processor core. No MBUS transaction is performed and the state of the cache line remains unchanged. If the cache mode changes for a specific region of address space, lines in the cache corresponding to that region containing dirty data will not be pushed out to memory when a read hit occurs within that line. Users should first execute a CPUSHL instruction or set the CINVA bit of the CACR (to invalidate the entire cache) before switching the cache mode.

4.7.4 Write Hit

The cache controller handles processor writes that hit in the cache differently for writethrough and copyback regions. For write hits to a writethrough region, the portions of the cache line(s) corresponding to the size of the access are updated with the data. The data is also written to the MBUS. The cache line state remains unchanged. If the access is copyback, the cache controller updates the cache line and sets the D-bit for the line. An MBUS write is not performed and the cache line state changes to, or remains in, the dirty state.

4.8 CACHE COHERENCY

The MCF5202 provides limited support for maintaining cache coherency in multimaster environments. Both writethrough and copyback memory update techniques are supported to maintain coherency between the cache and memory.

The MCF5202 cache does not support snooping (i.e., cache coherency is not supported while alternate masters are using the bus).

4.9 MEMORY ACCESSES FOR CACHE MAINTENANCE

The cache controller performs all maintenance activities that supply data from the cache to the processor core. The activities include requesting accesses to the SBC for reading new cache lines and writing dirty cache lines to memory. The following paragraphs describe the

memory accesses resulting from cache-fill and push operations. Refer to **Section 5 Bus Operation** for detailed information about the bus cycles required.

4.9.1 Cache Filling

When a new cache line is required, the cache controller requests a line read from the SBC. The SBC requests a burst read transfer by indicating a line access with the size signals (SIZ[1:0]).

The responding device supplies 4 longwords of data in sequence. If the responding device does not support the burst mode, it should assert the $\overline{\text{TBI}}$ signal for the first longword of the line access. The SBC responds by terminating the line access and completes the remainder of the line read as 3 sequential longword reads.

SBC line accesses implicitly request burst-mode operations from memory. For more information regarding burst mode accesses on the external bus, see **Section 5 Bus Operation**.

When a cache line read is initiated, the first cycle attempts to load the longword entry corresponding to the address requested by the processor core. Subsequent transfers are for the remaining longword entries in the cache line.

A bus error occurring during a burst operation aborts the operation. If the bus error occurs during the first cycle of a burst, the data from the bus is ignored and the line is not cached. If the access is a data cycle, exception processing proceeds immediately. If the cycle is for an instruction prefetch, a bus-error exception is not taken immediately, but will be taken if the instruction flow subsequently causes an attempt of the instruction. Refer to **Section 5 Bus Operation** for more information about this operation.

When a bus error occurs on the second cycle or later, the burst operation aborts and the line is not cached. The processor may or may not take an exception, depending on the status of the pending data request. If the bus-error cycle contains a portion of a data operand that the processor is specifically waiting for (e.g., the second half of a misaligned operand), the processor immediately takes an exception. Otherwise, no exception occurs and the cache line fill is repeated the next time data within the line is required.

4.9.2 Cache Pushes

When the cache controller selects a dirty cache line for replacement, memory must be updated with the dirty data before the line is replaced. Cache pushes occur for line replacement and as required for the execution of the CPUSHL instruction. To reduce the requested data’s latency in the new line, the dirty line being replaced is temporarily placed in the push buffer while the new line is fetched from memory. After the bus transfer for the new line completes, the dirty cache line is written back to memory and the push buffer is invalidated.

4.10 PUSH AND STORE BUFFERS

The push buffer reduces latency for requested new data on a cache miss by temporarily placing displaced dirty data into the push buffer while the new data is fetched from memory. The push buffer contains 16 bytes of storage (one displaced cache line).

If a cache miss displaces a dirty line, the miss reference is immediately placed on the MBUS. While waiting for the response, the current contents of the cache location are loaded into the push buffer. Once the bus transaction (burst read) completes, the cache controller can generate the appropriate line-write bus transaction to write the contents of the push buffer into memory.

The store buffer implements a FIFO buffer that can defer pending writes to imprecise regions in order to maximize performance. The store buffer can support as many as 4 entries (16 bytes maximum) for this purpose.

For operand writes destined for the store buffer, the processor core incurs no stalls. The store buffer effectively provides a measure of decoupling between the pipeline's ability to generate writes (1 write per cycle maximum) and the ability of the MBUS to retire those writes. When writing to imprecise regions, a stall will occur only in the event of a full store buffer and there is a write operation on the KBUS. The KBUS write cycle is held, stalling the operand execution pipeline.

If the store buffer is not used (i.e., store buffer disabled or cache- inhibited precise mode), MBUS cycles are generated directly for each pipeline write operation. The instruction is held in the EX cycle of the operand execution pipeline (OEP) until external bus transfer termination is received. This means each write operation is stalled for 3 cycles, making the minimum write time equal to 4 cycles when the store buffer is not used.

The store buffer enable bit (bit ESB of the CACR) controls the enabling of the store buffer. This bit can be set and cleared via the MOVEC instruction. At reset, this bit is cleared and all writes are precise. The ACR CM field or CACR DCM field generates the mode used when this bit is set. The cacheable writethrough and the cache-inhibited imprecise modes use the store buffer.

The store buffer can queue data up to 4 bytes wide per entry. Each entry matches a corresponding bus cycle it will generate; therefore, a misaligned longword write to a writethrough region will create 2 entries if the address is to an odd-word boundary, 3 entries if to an odd-byte boundary—1 per bus cycle.

4.11 PUSH AND STORE BUFFER BUS OPERATION

Once the push or store buffer has valid data, the cache controller uses the next available MBUS cycle to generate the appropriate write cycles. In the event that another cache fill is required (e.g., cache miss to process) during the continued instruction execution by the processor pipeline, the pipeline will stall until the push and store buffers are empty before generating the required MBUS transaction.

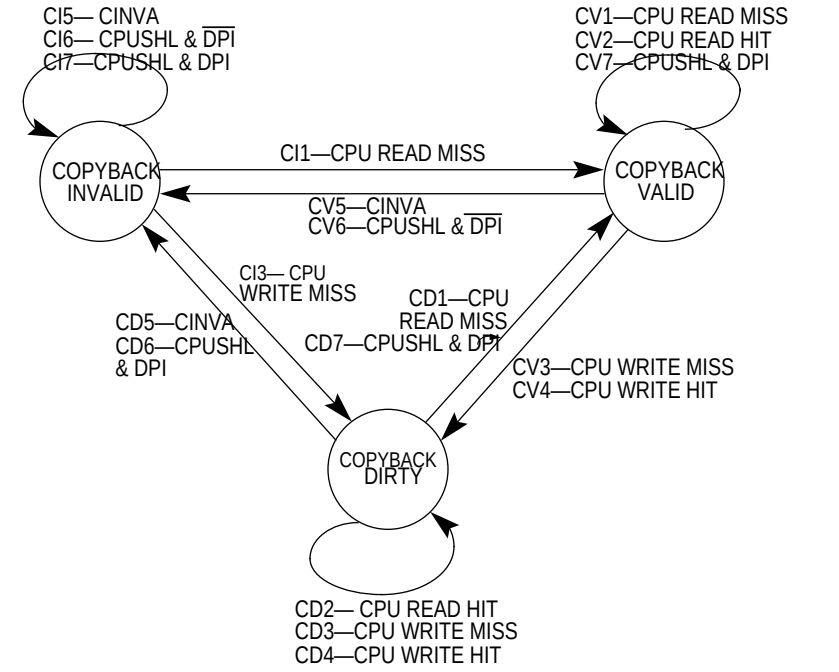
Certain instructions and exception processing that synchronize the processor core guarantee the push and store buffers are empty before proceeding.

4.12 CACHE OPERATION SUMMARY

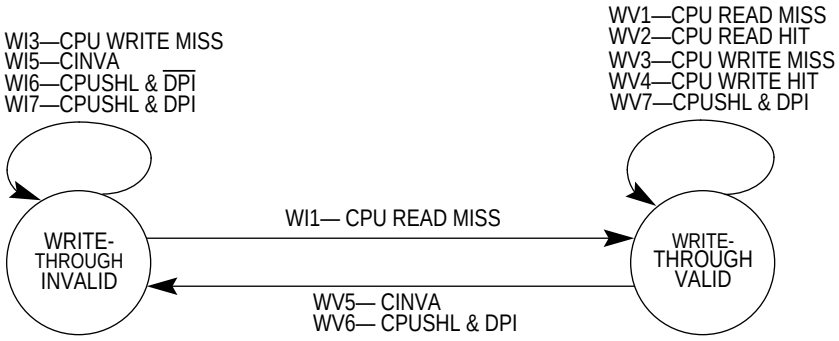
The following paragraphs discuss the operational details for the cache and present state diagrams depicting the cache line state transitions.

The cache supports a line-based protocol allowing individual cache lines to be in one of three states: invalid, valid, or dirty. To maintain coherency with memory, the cache supports both writethrough and copyback modes, specified by the CM field for the matched ACR or the DCM field of the CACR if no ACR matches.

Read misses and write misses to copyback regions cause the cache controller to read a new cache line from memory into the cache. If available, an invalid line in the selected row is updated with the tag and data from memory. The line state then changes from invalid to valid by setting the V-bit for the line. If all lines in the row are already valid or dirty, the pseudo random replacement algorithm selects 1 of the 4 lines and replaces the tag and data contents of the line with the new line information. Before replacement, dirty lines are temporarily buffered and later copied back to memory after the new line has been read from memory. Figure 4-6 illustrates the 3 possible states for a cache line, with the possible transitions caused by the processor. Transitions are labeled with a capital letter, indicating the previous state, followed by a number indicating the specific case listed in Table 4-1.



COPYBACK CACHING MODE



WRITETHROUGH CACHING MODE

Figure 4-6. Cache Line State Diagrams

Table 4-1. Cache Line State Transitions

CACHE OPERATION	CURRENT STATE					
	INVALID CASES		VALID CASES		DIRTY CASES	
READ MISS	(C,W)I1	Read line from memory and update cache; Supply data to processor; Go to valid state.	(C,W)V1	Read new line from memory and update cache; supply data to processor; Remain in current state.	CD1	Push dirty cache line to push buffer; Read new line from memory and update cache; Supply data to processor; Write push buffer contents to memory; Go to valid state.
READ HIT	(C,W)I2	Not possible.	(C,W)V2	Supply data to processor; Remain in current state.	CD2	Supply data to processor; Remain in current state.
WRITE MISS (COPYBACK MODE)	CI3	Read line from memory and update cache; Write data to cache; Go to dirty state.	CV3	Read new line from memory and update cache; Write data to cache; Go to dirty state.	CD3	Push dirty cache line to push buffer; Read new line from memory and update cache; Write push buffer contents to memory; Remain in current state.

Table 4-1. Cache Line State Transitions (Continued)

CACHE OPERATION	CURRENT STATE					
	INVALID CASES		VALID CASES		DIRTY CASES	
WRITE MISS (WRITE-THROUGH MODE)	WI3	Write data to memory; Remain in current state.	WV3	Write data to memory; Remain in current state.	WD3	Write data to memory; Remain in current state.
WRITE HIT (COPYBACK MODE)	CI4	Not possible.	CV4	Write data to cache; Go to dirty state.	CD4	Write data to cache; Remain in current state.
WRITE HIT (WRITE-THROUGH MODE)	WI4	Not possible.	WV4	Write data to memory and to cache; Remain in current state.	WD4	Write data to memory and to cache; Go to valid state.
CACHE INVALIDATE	(C,W)I5	No action; Remain in current state.	(C,W)V5	No action; Go to invalid state.	CD5	No action (dirty data lost); Go to invalid state.
CACHE PUSH	(C,W)I6	No action; Remain in current state.	(C,W)V6	No action; Go to invalid state.	CD6	Push dirty cache line to memory; Go to invalid state.
CACHE PUSH	(C,W)I7	No action; Remain in current state.	(C,W)V7	No action; Remain in current state.	CD7	Push dirty cache line to memory; Go to valid state

Note

The shaded areas indicate that the cache mode has changed for the region corresponding to this cache line. In writethrough mode, a cache line should never be dirty.

To avoid these states,

1. First execute a CPUSHL instruction, or

2. Set the CINVA bit of the CACR (to invalidate the entire cache) before switching the cache mode.

Date: 7-28-98
Revision Number: 0.1
Pages affected: See change bars

SECTION 5
BUS OPERATIONS

The MCF5202 bus interface supports synchronous, dynamic bus-size and bursted data transfers between the processor and other devices in the system.

This section provides a functional description of the bus, the signals that control the bus, and the bus cycles provided for data transfer operations. The waveforms in this document show the bus activity for an MCF5202 processor. Descriptions of bus arbitration and the reset operation are also included.

5.1 BUS CHARACTERISTICS

The MCF5202 processor uses a single clock signal CLK to generate its internal clocks as well as the bus clock. Therefore, the external bus operates at the same speed as the processor's internal clock rate, where all bus operations are synchronous to the rising edge of CLK. Figure 5-1 illustrates the general relationship between CLK and most input and output signals.

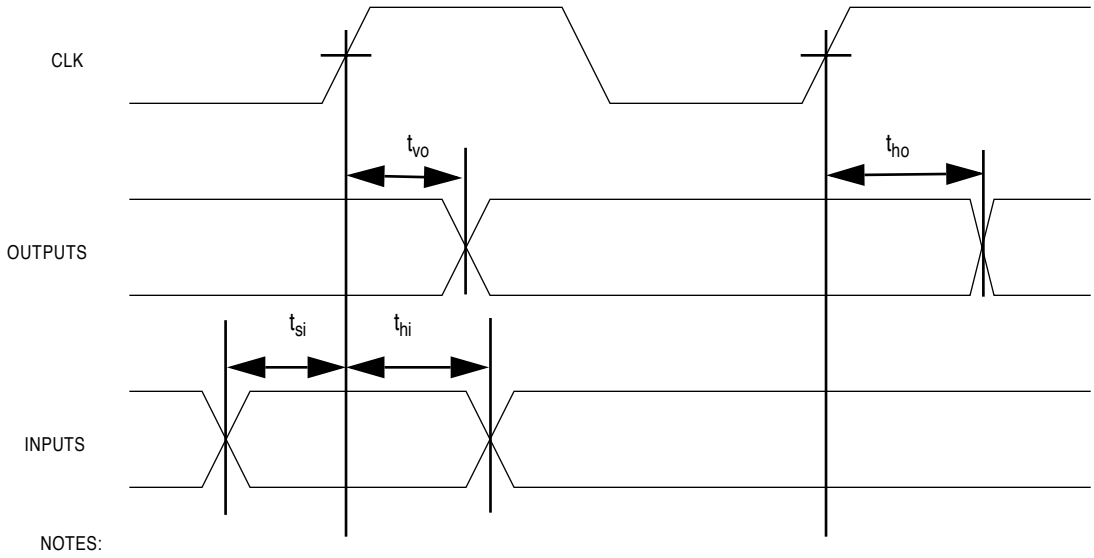


Figure 5-1. Signal Relationships to CLK

The synchronously sampled MCF5202 processor inputs (other than $\overline{IPL}$[2:0], $\overline{BKPT}$, and $\overline{RST}$ signals) must be stable during the sample window defined by t_{si} and t_{hi} (see Figure 5-1) to guarantee proper operation. The internally synchronized $\overline{IPL}$[2:0], $\overline{BKPT}$, and $\overline{RST}$ signals resolve the input to a valid level before being used.

The MCF5202 outputs begin to transition on the rising edge of CLK.

5.2 DATA TRANSFERS

Data transfer between the processor and other devices involves the address/data bus, bus attributes and control signals. The MCF5202 device will multiplex only addresses and data on A/D[31:16]. A/D[15:0] will drive the current address for the duration of the transfer, incrementing A/D[3:0] on burst accesses. Figure 5-2 displays a simple example of the MCF5202 processor executing a longword write followed by a longword read terminated with a bus error. The assertion of $\overline{TEA}$ during any portion of the transfer will abort the transfer. All figures display logical waveforms and do not indicate any timing relationships unless noted.

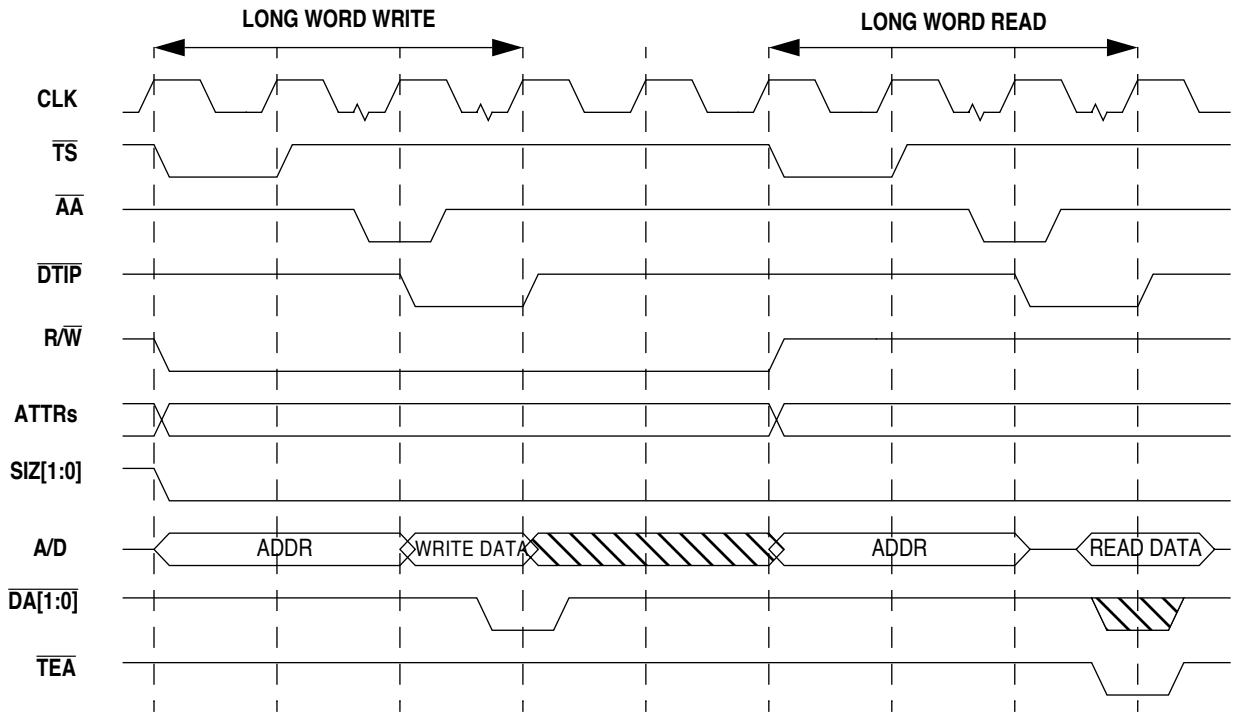


Figure 5-2. Simple Transfer Followed By Transfer Containing Bus Error

The MCF5202 device bursts all accesses to a port smaller than the transfer size accesses unless $\overline{TBI}$ is asserted during the first data acknowledge cycle of the transfer. The level of $\overline{TBI}$ during the remainder of the access has no affect on the transfer. Figures 5-3 through 5-6 show examples of various types of burst read/write cycles to dynamically sized ports, including burst-inhibited cycles. In these figures, $\overline{AA}$ is tied low.

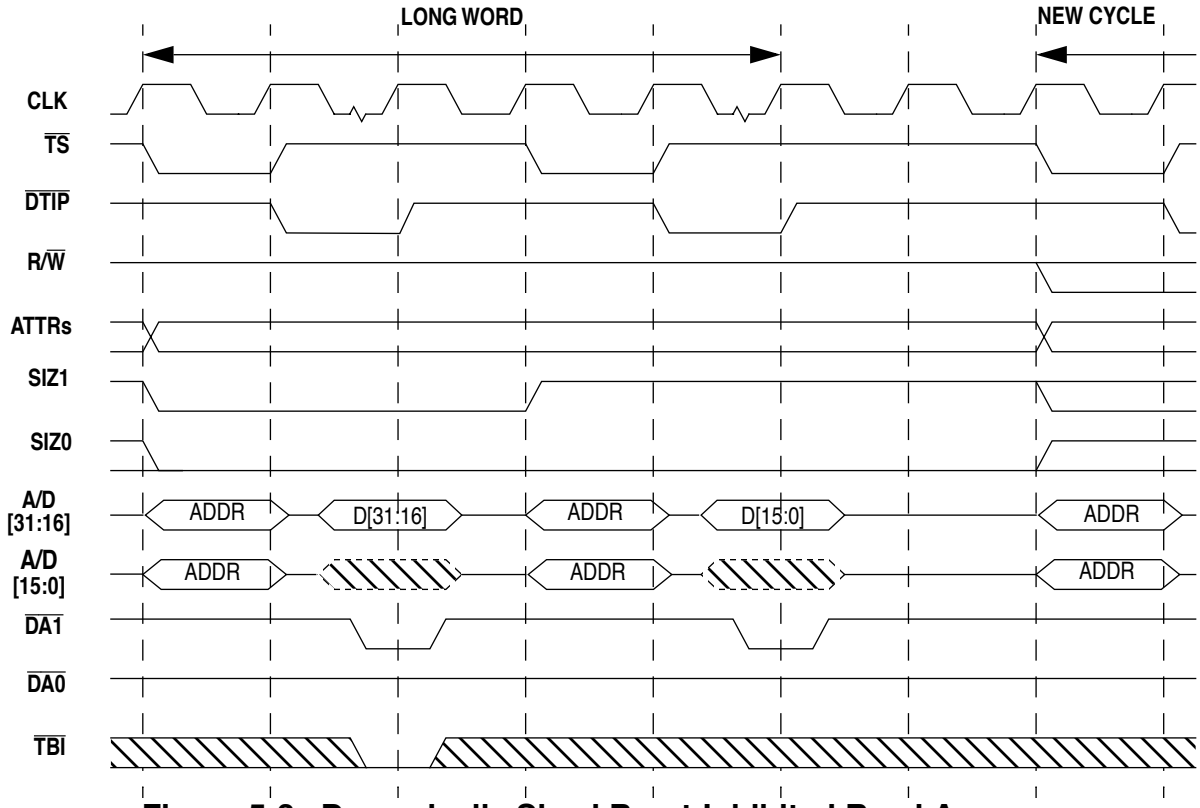


Figure 5-3. Dynamically Sized Burst-Inhibited Read Access

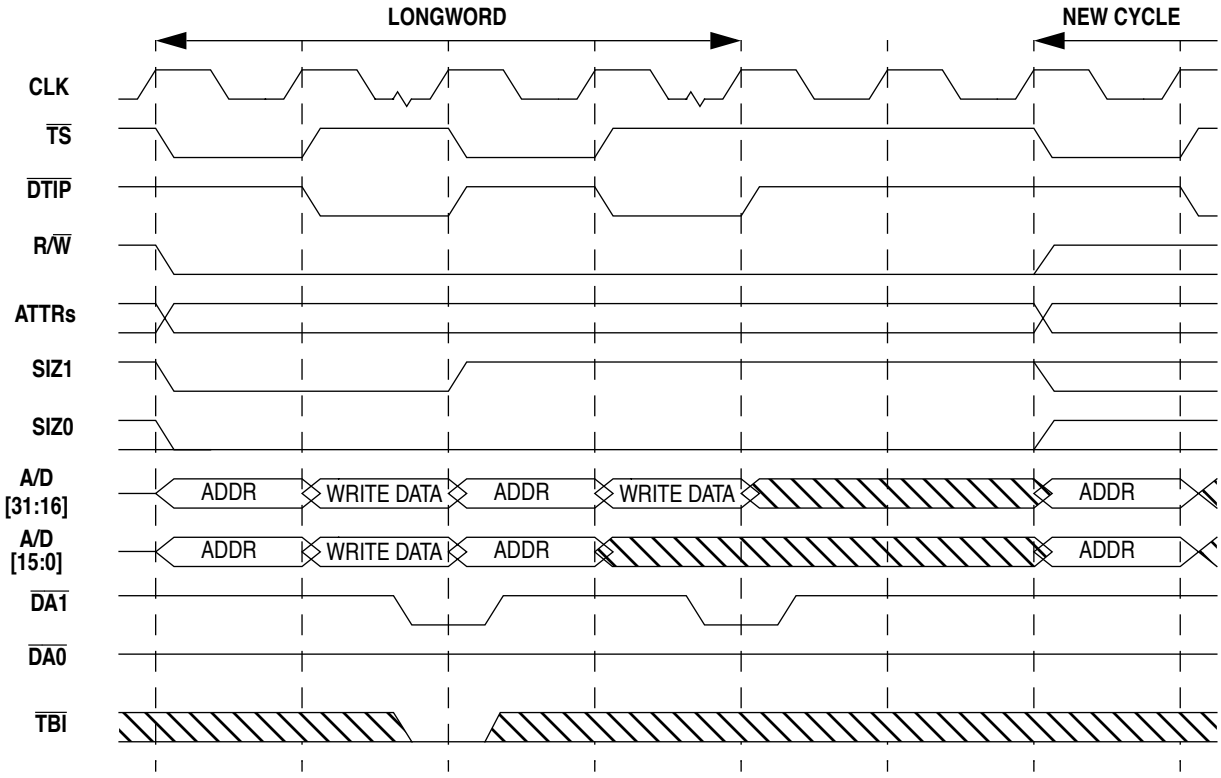


Figure 5-4. Dynamically Sized Burst-Inhibited Write Access

* Recovery states are not necessary on write cycles because the processor never gives up control of the bus.

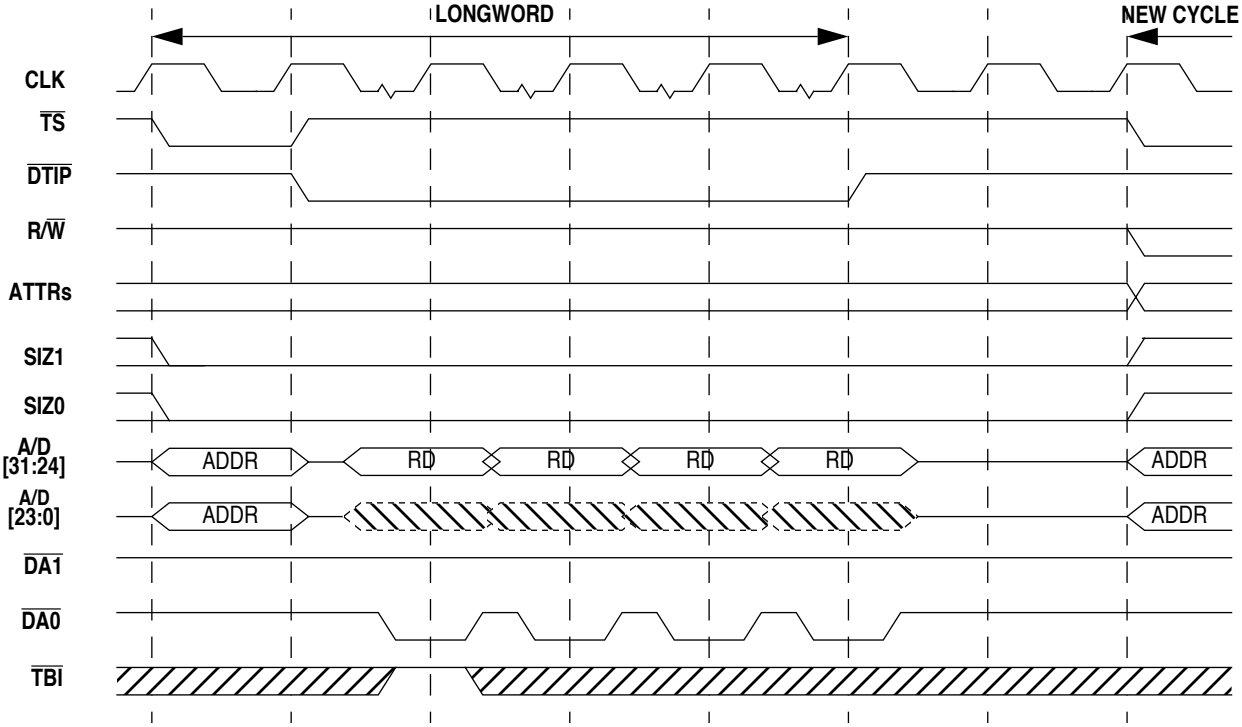


Figure 5-5. Dynamically Sized Burst Read

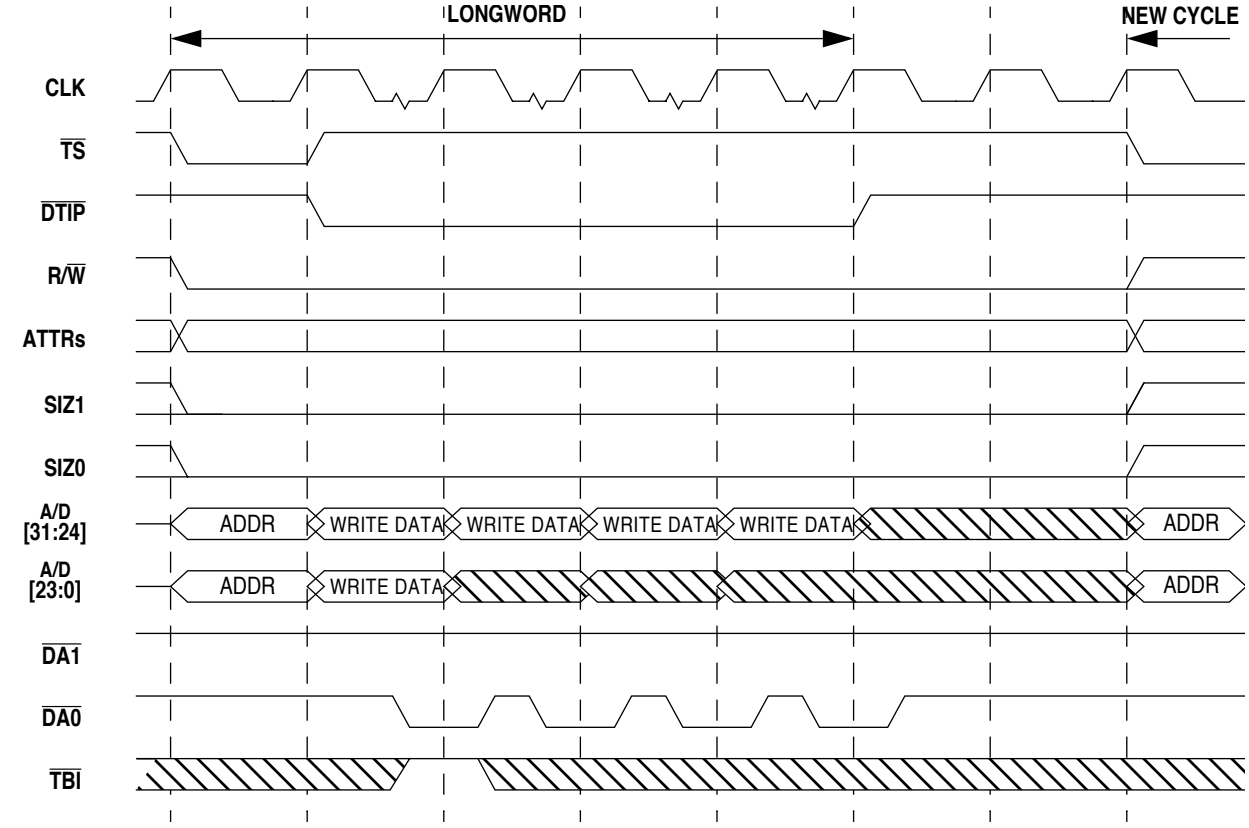


Figure 5-6. Dynamically Sized Burst Write

If the external port is 32-bits: for reads with 8 bit transfer:
Addr 0: AD[31:24]: Byte 0
Addr 1: AD[23:16]: Byte 1
Addr 2: AD[15:8]: Byte 2
Addr 3: AD[7:0]: Byte 3

For 16 bit transfers:
Addr 0: AD[31:16]: Byte 0, Byte 1
Addr 2: AD[15:0]: Byte 2, Byte 3

For writes if the external port is 32 bits:

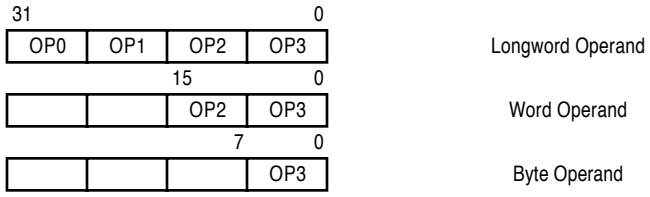


Table 5-1. Transfer Size Chart

Transfer Size	Size		Address		External Data Bus Connection			
	SIZ1	SIZ2	A1	A0	D[31:24]	D[23:16]	D[15:8]	D[7:0]
Byte	0	1	0	0	OP3	X	X	X
	0	1	0	1	OP3	OP3	X	X
	0	1	1	0	OP3	X	OP3	X
	0	1	1	1	OP3	OP3	X	OP3
Word	1	0	0	0	OP2	OP3	X	X
	1	0	0	1	OP3	X	X	X
	1	0	1	0	OP2	OP3	OP2	OP3
	1	0	1	1	OP3	X	X	X
Longword	0	0	0	0	OP0	OP1	OP2	OP3
		0	0	1	OP1	X	X	X
	0	0	1	0	OP2	OP3	X	X
	0	0	1	1	OP3	X	X	X
Line	1	1	0	0	OP0	OP1	OP2	OP3
	1	1	0	1	OP1	X	X	X
	1	1	1	0	OP2	OP3	X	X
	1	1	1	1	OP3	X	X	X

NOTE:

The 5202 does not perform line bursts across a line boundary nor does it stop at the line boundary and continue on a new line. The 5202 fills the entire line within the boundaries of that particular line. If the line transfer starts on a mis-aligned line boundary it wraps around the same line to complete the transfer.

NOTE:

$\overline{DA}$[1:0], $\overline{TEA}$, and $\overline{TBI}$ can be synchronous as long as they are negated before the 5202 samples them for the next bus cycle. Please refer to figure 9-3 on page 9-6 of the Electrical Characteristics section of the user's manual.

5.3 ACKNOWLEDGE BUS CYCLES

The MCF5202 processor supports both interrupt- and autovector-acknowledge cycles. In either case, if the transfer is terminated with the assertion of $\overline{TEA}$, a spurious interrupt exception is taken. Figure 5-7 displays an interrupt-acknowledge cycle. The cycle is similar to a byte read, except the access is in CPU space, TT = 11. The interrupt level being acknowledged is driven onto A/D[4:2], while A/D[31:5] are driven high and A/D[1:0] are driven low. The system drives the interrupt vector onto A/D[31:24], asserting $\overline{DA}$, to terminate the access. During the acknowledge cycle, $\overline{AVEC}$ can be asserted, forcing internal generation of the vector number. If $\overline{AVEC}$ is asserted, vector data is not required to be driven. The $\overline{IPL}$ signals should remain valid until the acknowledge cycle is complete.

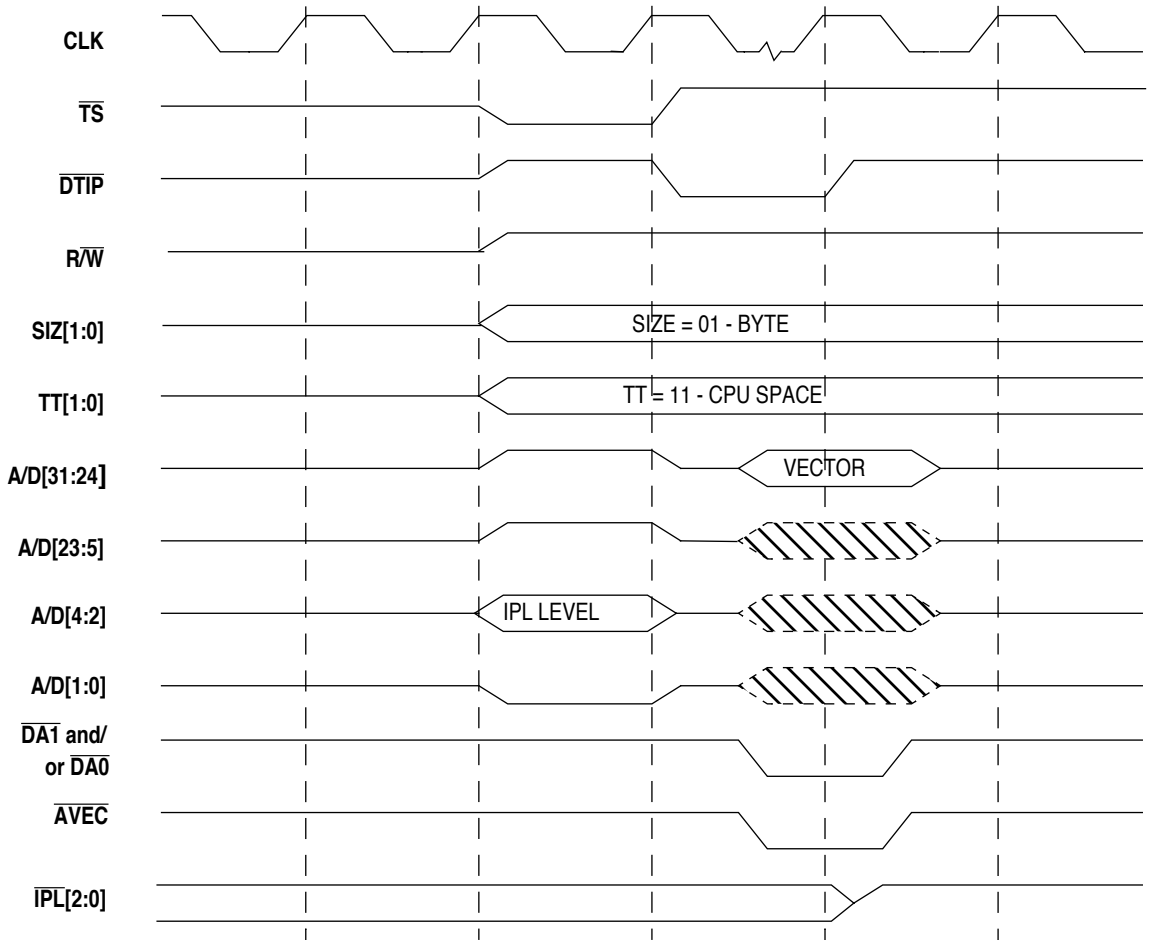
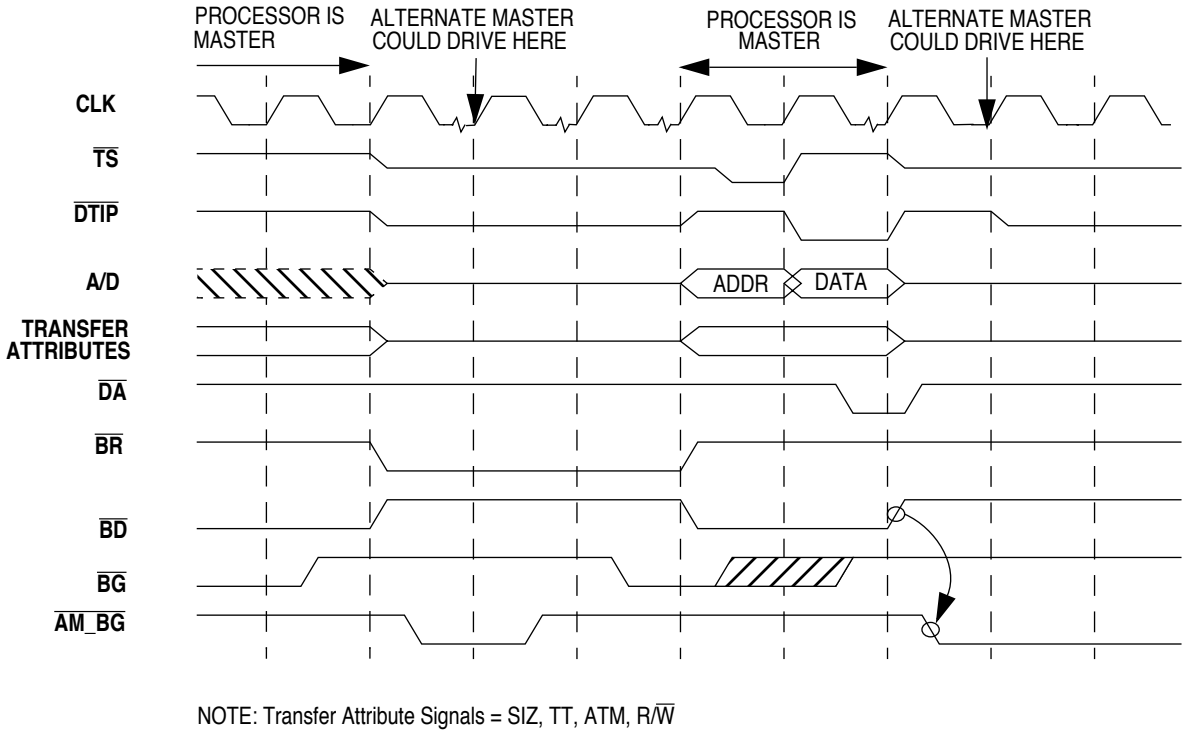


Figure 5-7. Interrupt-Acknowledge Operation

5.4 BUS ARBITRATION

The MCF5202 processor supports multimaster system designs by requesting the bus from

an external arbiter. The arbiter must monitor the signals bus request ($\overline{BR}$) and bus driven ($\overline{BD}$) to properly arbitrate the bus. There is a minimum one-clock dead time to arbitrate the bus from the MCF5202 device to another master. The external arbiter may offer the bus to another master any time both $\overline{BD}$ and $\overline{BG}$ are negated. The MCF5202 monitors its bus grant ($\overline{BG}$) signal to determine when it should become bus master and will start to drive the bus at the rising edge of the clock in which $\overline{BG}$ is recognized asserted. Figure 5-8 shows two scenarios of $\overline{BG}$ being negated. In the first, the MCF5202 is bus master and the bus is idle. On the rising CLK edge where $\overline{BG}$ is recognized negated, the MCF5202 device will drive the bus to a high-impedance level. In the second case, $\overline{BG}$ is negated within a bus cycle. The MCF5202 processor will complete the bus access by driving $\overline{BD}$ negated and driving all bus signals to a high-impedance level. If $\overline{DTIP}$ is tied to another master, there may be some period of time when this signal may be driven high by 2 separate drivers with both pins driving high before the previous master places its pin in a three-state mode.



NOTE: Transfer Attribute Signals = SIZ, TT, ATM, R/$\overline{W}$

Figure 5-8. Bus Arbitration Operation

5.5 RESET OPERATION

An external device asserts the $\overline{RST}$ signal to reset the processor. When power is applied to the system, external circuitry should assert $\overline{RST}$ for a minimum of 6 CLK cycles after VCC is within tolerance. CLK is required to be stable by the time VCC reaches the minimum operating specification. CLK should start oscillating as VCC is ramped up to resolve contention internal to the part caused by the random manner in which internal flip-flops power-up. $\overline{RST}$ is internally synchronized for 2 CLK cycles before being used, and must meet the specified setup and hold times to CLK only if recognition by a specific CLK rising edge is required.

Figure 5-9 shows the general relationship between VCC, $\overline{RST}$, and the bus signals during the power-on reset operation. Processor resets during normal operation must follow the same requirements as those for power-on reset.

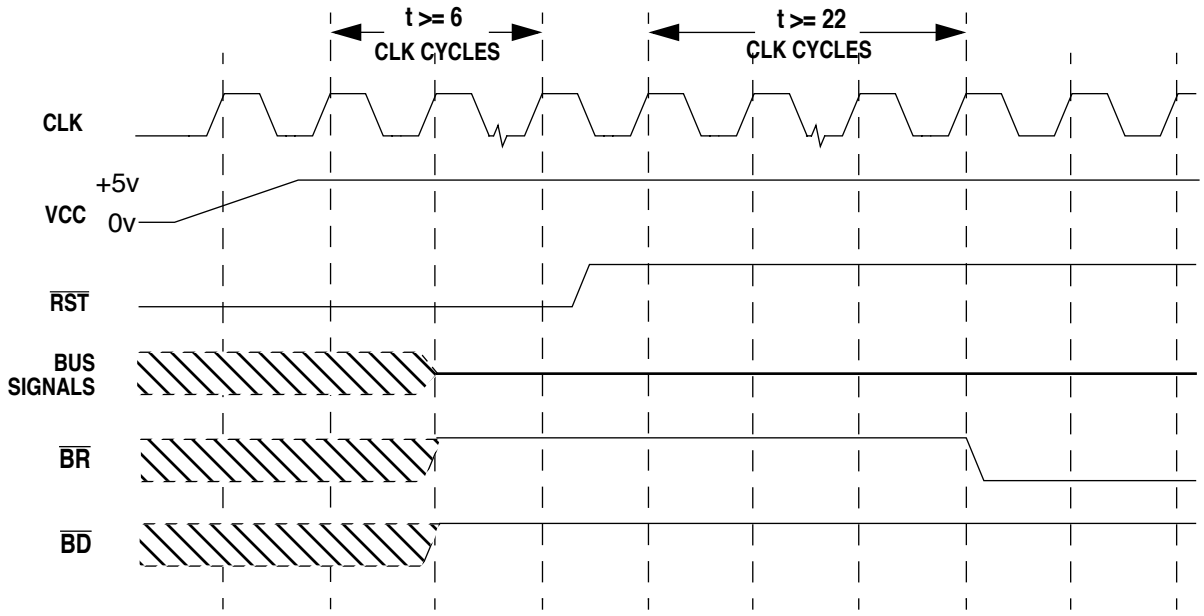


Figure 5-9. Reset Operation

SECTION 6
DEBUG SUPPORT

This section details the hardware debug support functions within the ColdFire 5200 Family of processors.

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 of the three areas 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 6-1.

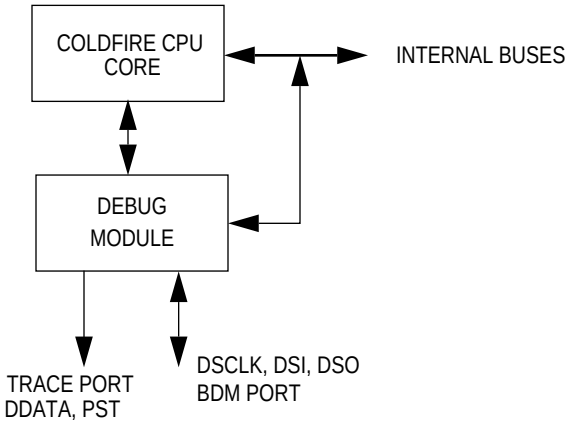


Figure 6-1. Processor/Debug Module Interface

6.1 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 ColdFire solution is to include a parallel output port providing encoded processor status and data to an external development system. This port is partitioned into two 4-bit nibbles: one nibble allows the processor to transmit information concerning the execution status of the core (processor status, PST), while the other nibble allows operand data to be displayed (debug data, DDATA).

The processor status 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 naturally complicated by any change-of-flow operation. Within the ColdFire instruction set architecture, most branch instructions are implemented using PC-relative addressing. Accordingly, the external program image can determine branch target addresses. Additionally, there are a number of instructions that use some type of variant addressing, i.e., the calculation of the target instruction address is not PC-relative or absolute, but involves the use of a program-visible register.

The processor status timing is synchronous with the processor clock (CLK) and the status may not be related to the current bus transfer. Table 6-1 below shows the encodings of these signals.

Table 6-1. Processor PST Definition

PST[3:0]	DEFINITION
0000	Continue execution
0001	Begin execution of an instruction
0010	Reserved
0011	Entry into user-mode
0100	Begin execution of **PULSE** instruction
0101	Begin execution of taken branch
0110	Reserved
0111	Begin execution of **RTE** instruction
1000	Begin 1-byte transfer on ddata
1001	Begin 2-byte transfer on ddata
1010	Begin 3-byte transfer on ddata
1011	Begin 4-byte transfer on ddata
1100	† Exception processing
1101	† Emulator-mode entry exception processing
1110	† Processor is stopped, waiting for interrupt
1111	† Processor is halted

† These encodings are asserted for multiple cycles.

The simplest example of a branch instruction using a variant addressing mode 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 ColdFire processor uses the debug pins to output a sequence of information.

1. Identify a taken branch has been executed using the PST pins.
2. Using the PST pins, signal the target address is to be displayed on the DDATA pins. The encoding identifies the number of bytes that are displayed and is optional.
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).

The nibble-wide DDATA port includes two 32-bit storage elements for capturing the CPU core bus information. These two elements effectively form a FIFO buffer connecting the core bus to the external development system. The FIFO buffer captures variant branch target addresses along with certain operand read/write data for eventual display on the DDATA output port. The execution speed of the ColdFire processor 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.

From the processor core perspective, the PST outputs signal the first AGEX cycle of an instruction's execution. Most single-cycle instructions begin and complete their execution within a given machine cycle.

Because the status values of $C, $D, $E and $F define a multicycle mode or a special operation, the PST outputs are driven with these values until the mode is exited or the operation completed. All the remaining fields specify information that is updated each machine cycle.

The status values of $8, $9, $A and $B qualify the contents of the DDATA output bus. These encodings are driven onto the PST port one machine cycle before the actual data is displayed on DDATA.

Figure 6-2 shows the execution of an indirect JMP instruction with the lower 16 bits of the target address being displayed on the DDATA output. In this diagram, the indirect JMP branches to address "target." The processor internally forms the PST marker ($9) one cycle before the address begins to appear on the DDATA port. The target address is displayed on DDATA for four consecutive clocks, starting with the least-significant nibble. The processor continues execution, unaffected by the DDATA bus activity.

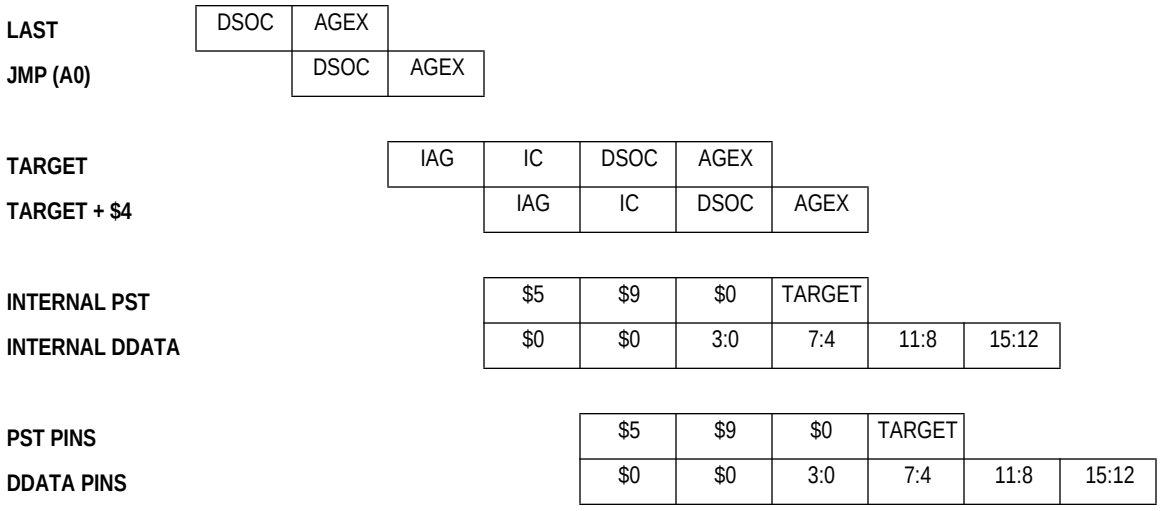


Figure 6-2. Pipeline Timing Example - Debug Output

The ColdFire instruction set architecture includes a PULSE opcode. This opcode generates a unique PST encoding when executed (PST = $4). 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 = $4 encoding when executed.

6.2 BACKGROUND DEBUG MODE (BDM)

ColdFire 5200 processors support 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.

Unless noted otherwise, the BDM functionality provided by ColdFire 5200 processors is a proper subset of the CPU32 functionality. The main differences include the following:

- ColdFire 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, DSO could signal 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/WSREG) have been replaced by Read/Write Control Register commands (RCREG/WCREG). These commands use the register coding scheme from the MOVEC instruction.
- Read/Write Debug Module Register commands (RDMREG/WDMREG) have been added to support debug module register accesses.
- CALL and RST commands are not supported.
- 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.

6.2.1 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 is posted on the PST port ($F).
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 $\overline{\text{BKPT}}$ signal. In some 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 **6.3.2 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 $\overline{\text{BKPT}}$ input pin is treated as an 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.

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

There are two special cases involving the assertion of the $\overline{\text{BKPT}}$ pin to be considered.

After $\overline{\text{RSTI}}$ is negated, the processor waits for 16 clock cycles before beginning reset exception processing. If the $\overline{\text{BKPT}}$ input pin is asserted within the first eight cycles after $\overline{\text{RSTI}}$ is negated, the processor will enter the halt state, signaling that status on the PST outputs ($F). While in this state, all resources accessible via the debug module can be referenced. Once the system initialization is complete, the processor response to a BDM GO command depends on the set of BDM commands performed while breakpointed. Specifically, if the processor's PC register was loaded, the GO command causes the processor to exit the halt state and pass control to the instruction address contained in the PC. In this case, the normal reset exception processing is bypassed. Conversely, if the PC register was not loaded, the GO BDM command causes the processor to exit the halt state and continue with reset exception processing.

ColdFire 5200 processors also handle a special case with the assertion of $\overline{\text{BKPT}}$ 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 debug module CSR register maintains status defining the condition that caused the CPU to halt.

6.2.2 BDM Serial Interface

Once the CPU is halted and the halt status reflected on the PST outputs, 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.

Both DSCLK and DSI are synchronous inputs and must meet input setup and hold times with respect to CLK. DSCLK 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 6-3 below).

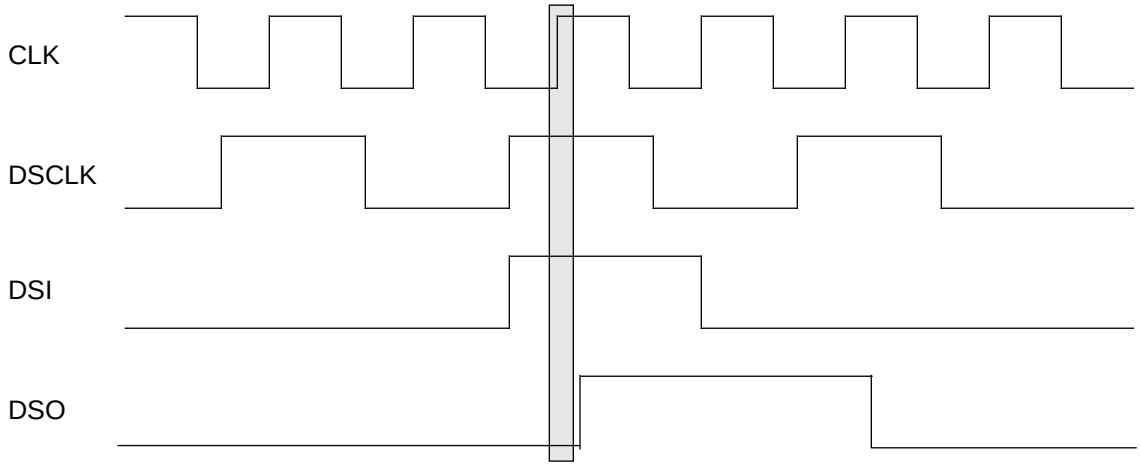


Figure 6-3. BDM Signal Sampling

The basic packet of information is a 17-bit word (16 data bits plus a status/control bit), as shown below.

Status/Control

The status/control bit indicates the status of CPU-generated messages as listed in Table 6-2. Command and data transfers initiated by the development system should clear bit 16. The current implementation ignores this bit; however, Motorola reserves the right to use this bit for future enhancements.The response message is always a single word, with the

Table 6-2. CPU-Generated Message Encoding

S/C 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 encoded as shown in Table 6-2.

Data Field

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

6.2.3 BDM Command Set

ColdFire 5200 processors support a subset of BDM instructions from the current 683xx parts, as well as extensions to provide access to new hardware features.

6.2.3.1 BDM COMMAND SET SUMMARY. The BDM command set is summarized in Table 6-3. Subsequent paragraphs contain detailed descriptions of each command.

Table 6-3. 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 result via the serial interface	Halted	8-10
Write A/D Register	WAREG/WDREG	The data operand is written to the specified address or data register	Halted	8-11
Read Memory Location	READ	Read the sized data at the memory location specified by the longword address	Steal	8-12
Write Memory Location	WRITE	Write the operand data to the memory location specified by the longword address	Steal	8-14
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.	Steal	8-16
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.	Steal	8-18
Resume Execution	GO	The pipeline is flushed and refilled before resuming instruction execution at the current PC	Halted	8-20
No Operation	NOP	NOP performs no operation and may be used as a null command	Parallel	8-20
Read Control Register	RCREG	Read the system control register	Halted	8-21
Write Control Register	WCREG	Write the operand data to the system control register	Halted	8-22

Table 6-3. BDM Command Summary (Continued)

COMMAND	MNEMONIC	DESCRIPTION	CPU IMPACT[1]	PAGE
Read Debug Module Register	RDMREG	Read the Debug Module register	Halted	8-23
Write Debug Module Register	WDMREG	Write the operand data to the Debug Module register	Halted	8-23

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.

6.2.3.2 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/W	OP SIZE	0	0	A/D	REGISTER
EXTENSION WORD(S)							

Operation Field

The operation field specifies the command.

R/W Field

The R/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 6-4.

Table 6-4. BDM Size Field Encoding

ENCODING	OPERAND SIZE
00	Byte
01	Word
10	Long
11	Reserved

Address / Data (A/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 by most significant word first. In the following descriptions of the BDM command set, the optional set of extension words are defined as the ‘‘Operand Data.’’

6.2.3.3 Command Sequence Diagram. A command sequence diagram (see Figure 6-4) 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 diagram 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).

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 unless the received command was decoded as unimplemented, in which case the response data is the illegal command encoding. If an illegal command response occurs, the development system should retransmit the command.

NOTE

The “not ready” 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 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 is returned in place of the result data.

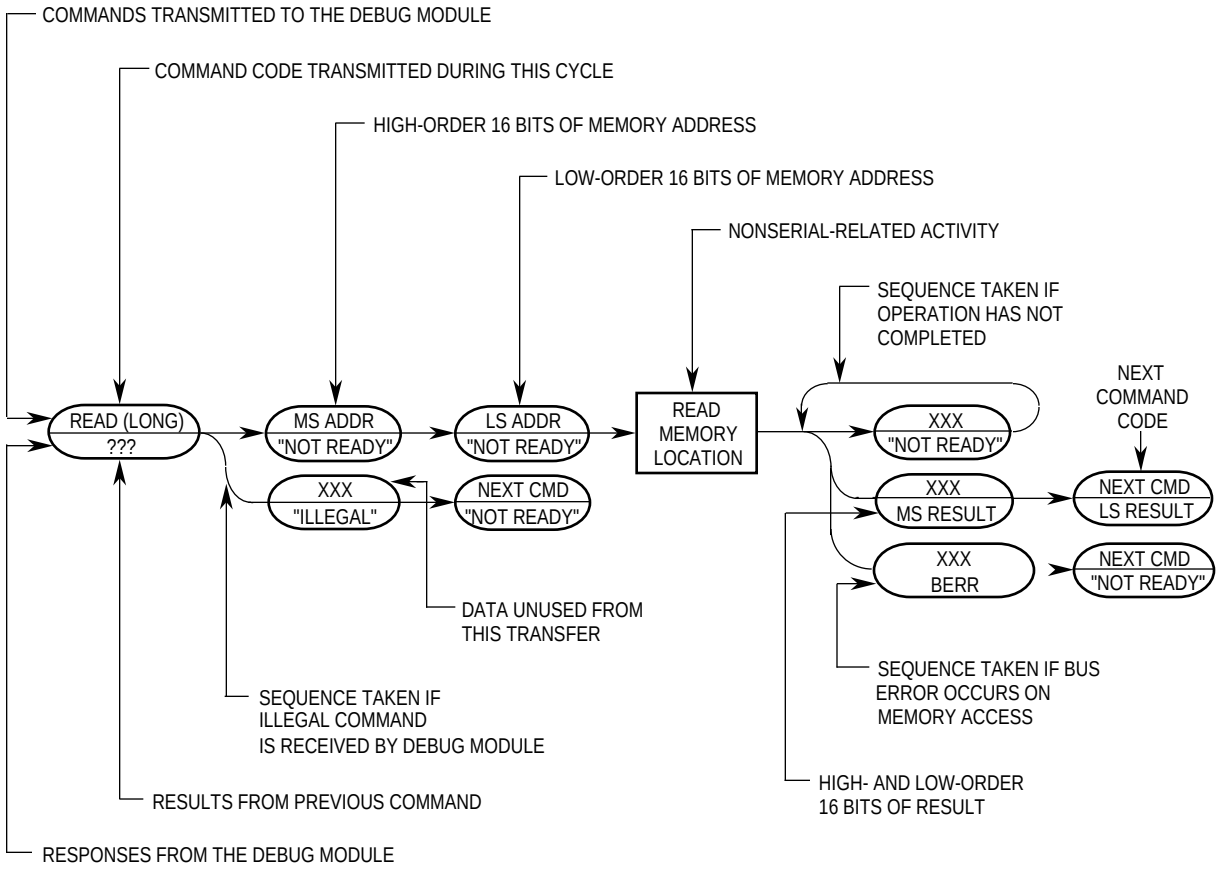


Figure 6-4. Command Sequence Diagram

6.2.3.4 Command Set Descriptions. The BDM command set is summarized on pages 6-7 and 6-8. Subsequent paragraphs contain detailed descriptions of each command.

Note

All the accompanying BDM results are defined with the most significant bit of the 17-bit response (S/C) as 0.

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.

6.2.3.4.1 Read A/D Register (RAREG/RDREG). Read the selected address or data register and return the 32-bit result. A bus error response is returned if the CPU core is not halted.

Formats:

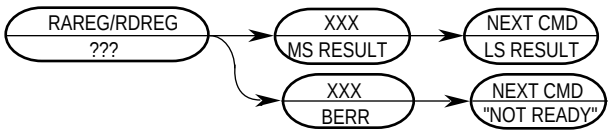
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$2				$1				$8				A/D	REGISTER		

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.

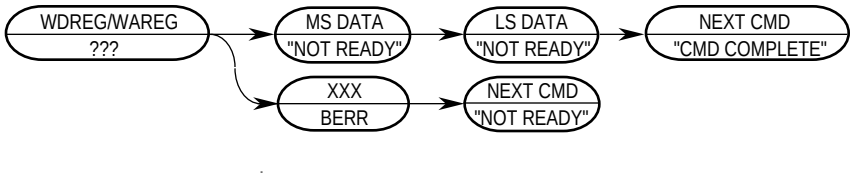
6.2.3.4.2 Write A/D Register (WAREG/WDREG). The operand (longword) data is written to the specified address or data register. All 32 register bits are altered by the write. A bus error response is returned if the CPU core is not halted.

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:



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 ($0FFFF) is returned when register write is complete.

6.2.3.4.3 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. 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

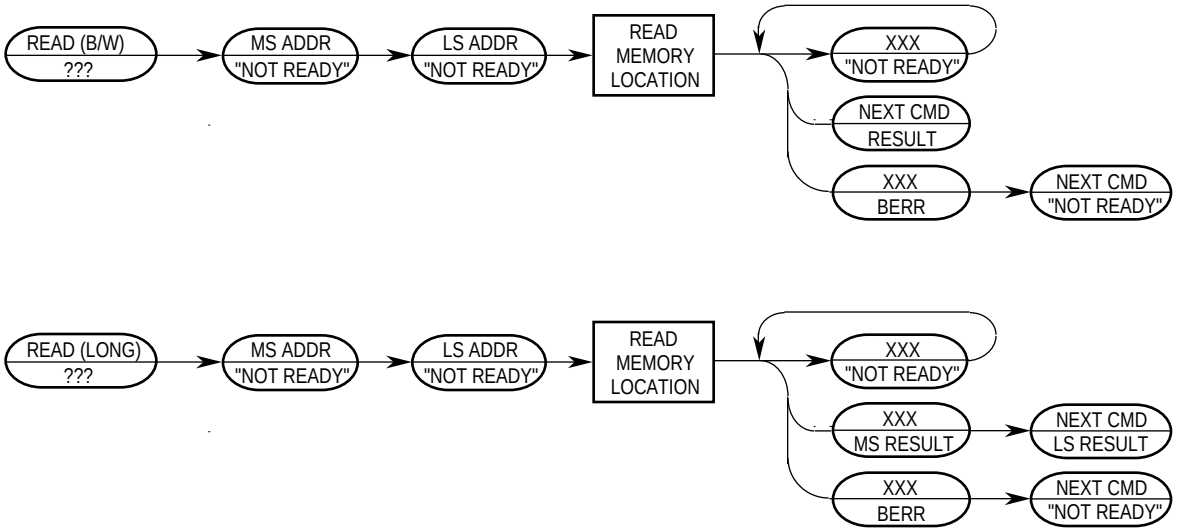
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$1				$9				$8				$0			
ADDRESS [31:16]															
ADDRESS [15:0]															

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.

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 successful read operation returns data bit 16 cleared. If a bus error is encountered, the returned data is $10001.

6.2.3.4.4 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. 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

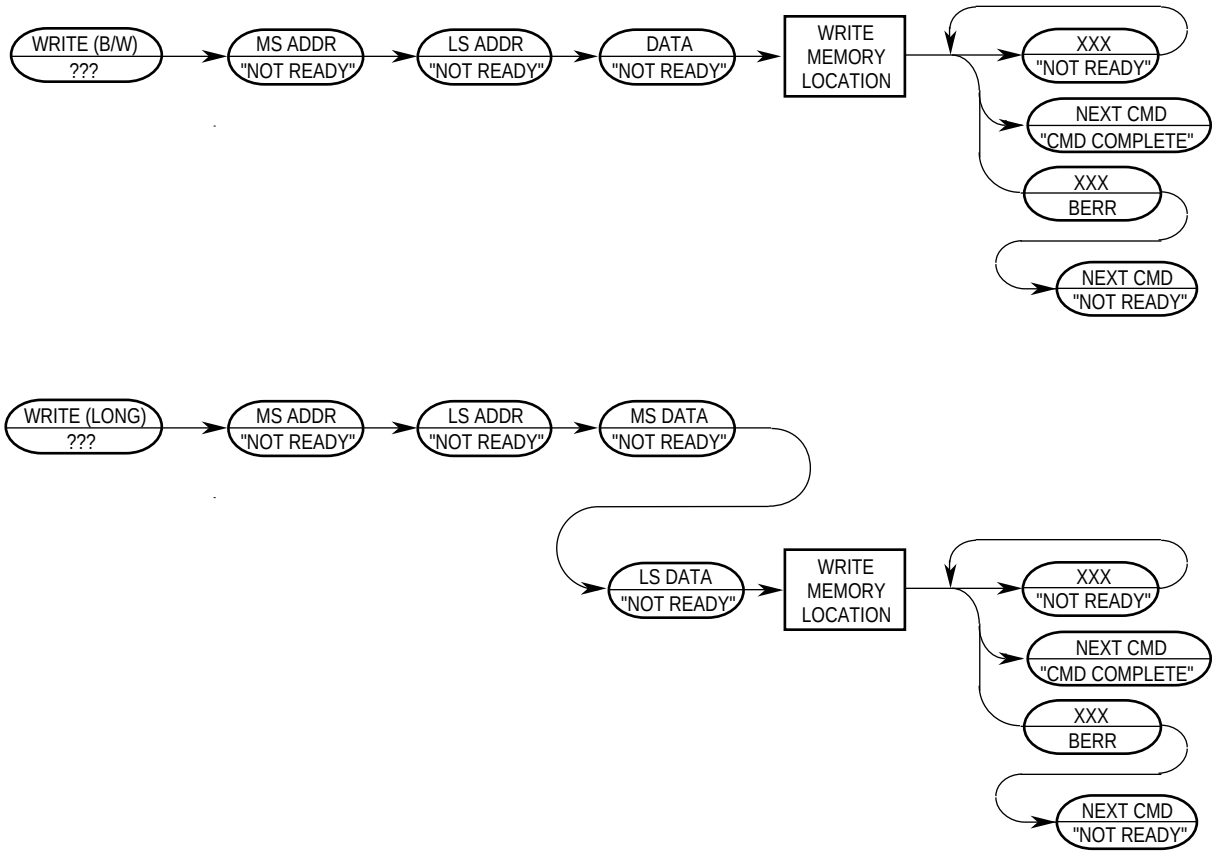
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$1				$8				$4				$0			
ADDRESS [31:16]															
ADDRESS [15:0]															
DATA [15:0]															

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 return a status of $0FFFF. A bus error on the write cycle is indicated by the assertion of bit 16 in the status message and by a data pattern of $0001.

6.2.3.4.5 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

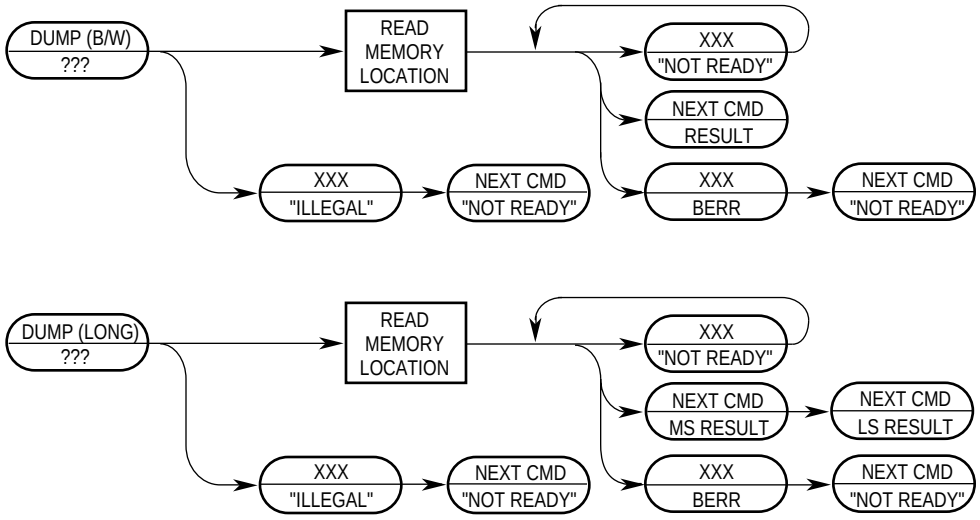
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$1				$D				$8				$0			

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. Status of the read operation is returned as in the READ command: $0xxxx for success, $10001 for a bus error.

6.2.3.4.6 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 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.

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

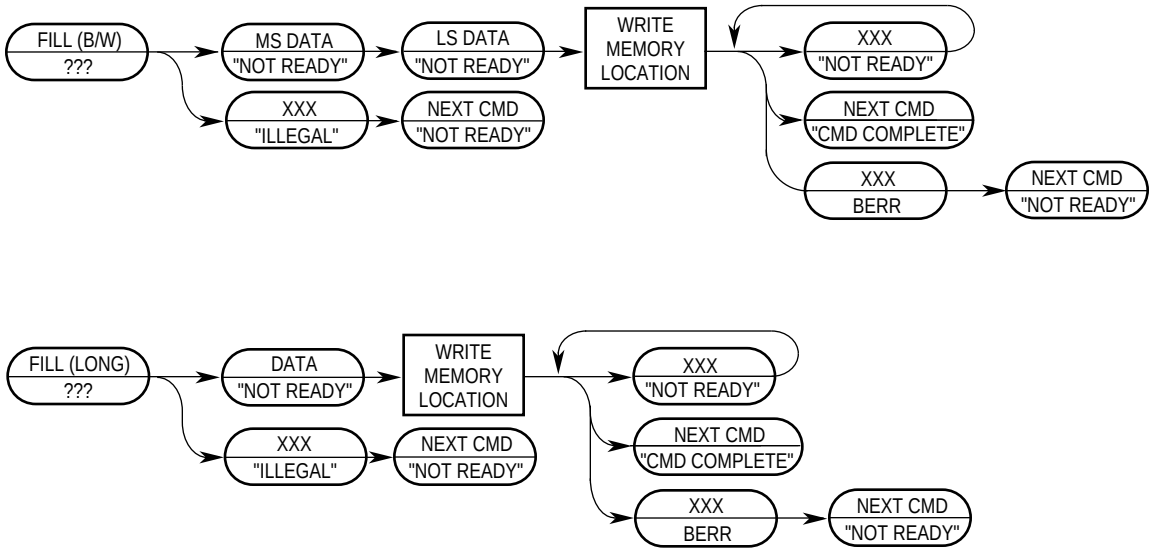
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
$1				$C				$4				$0			
DATA [15:0]															

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:

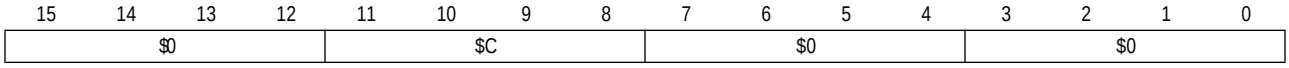
A single operand is data to be written to the memory location. Byte data is transmitted as a 16-bit word, justified in the least significant byte; 16- and 32-bit operands are transmitted as 16 and 32 bits, respectively.

Result Data:

Status is returned as in the WRITE command: $0FFFF for a successful operation and $10001 for a bus error during a write.

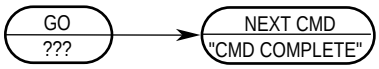
6.2.3.4.7 Resume Execution (GO). The pipeline is flushed and refilled before resuming normal instruction execution. Prefetching begins at the current PC and current privilege level. If either the PC or SR is altered during BDM, the updated value of these registers is used when prefetching begins.

Formats:



GO Command

Command Sequence:



Operand Data:

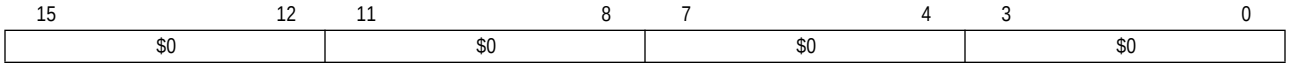
None

Result Data:

The “command complete” response ($0FFFF) is returned during the next shift operation.

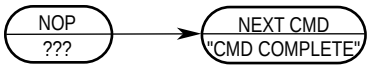
6.2.3.4.8 No Operation (NOP). NOP performs no operation and may be used as a null command where required.

Formats:



NOP Command

Command Sequence:



Operand Data:

None

Result Data:

The “command complete” response ($0FFFF) is returned during the next shift operation.

6.2.3.4.9 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]															

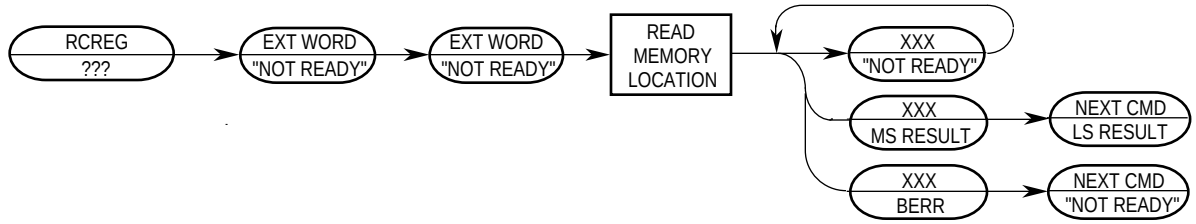
RCREG Result

Rc encoding:

Table 6-5. Control Register Map

Rc	REGISTER DEFINITION
$002	Cache Control Register (CACR)
$004	Access Control Unit 0 (ACR0)
$005	Access Control Unit 1 (ACR1)
$801	Vector Base Register (VBR)
$80E	Status Register (SR)
$80F	Program Counter (PC)

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 by most significant word first. For those control register widths less than 32 bits, only the implemented portion of the register is guaranteed to be correct. The remaining bits of the longword 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.

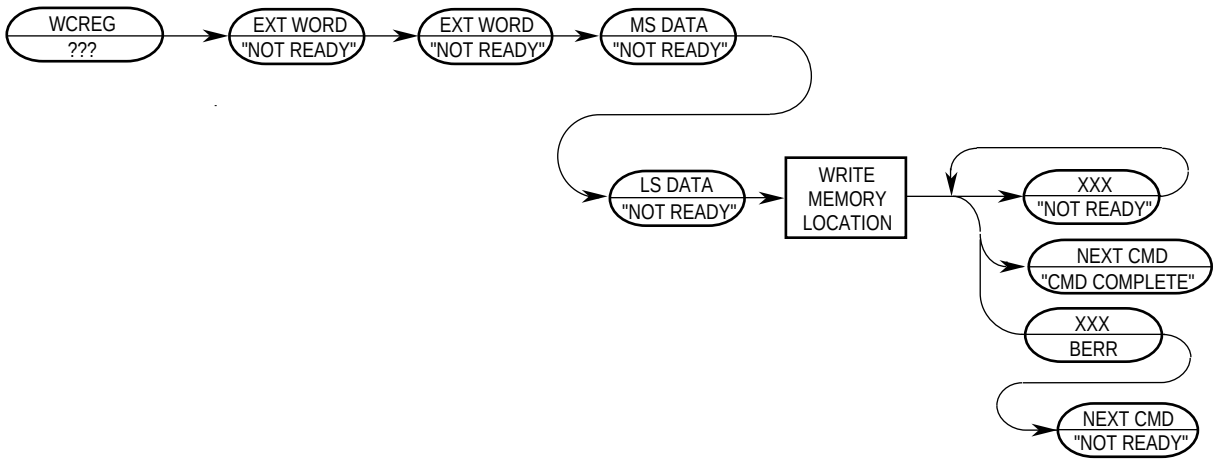
6.2.3.4.10 Write Control Register (WCREG). The operand (longword) data is written to the specified control register. The write alters all 32 register bits.

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 a status of $0FFFF. 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.

6.2.3.4.11 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]															

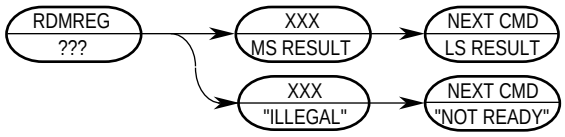
RDMREG BDM Result

DRc encoding:

Table 6-6. 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.

6.2.3.4.12 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]															

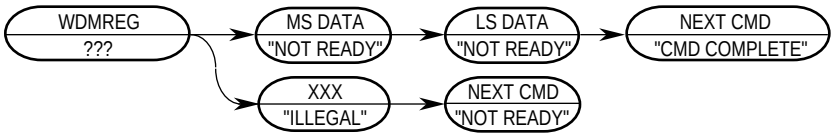
WDMREG BDM Command

DRc encoding:

Table 6-7. 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	AABR	$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.

6.2.3.4.13 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**.**

6.3 REAL-TIME DEBUG SUPPORT

ColdFire processors provide 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 debugging, 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.

6.3.1 Programming Model

In addition to the existing BDM commands that provide access to the processor's registers

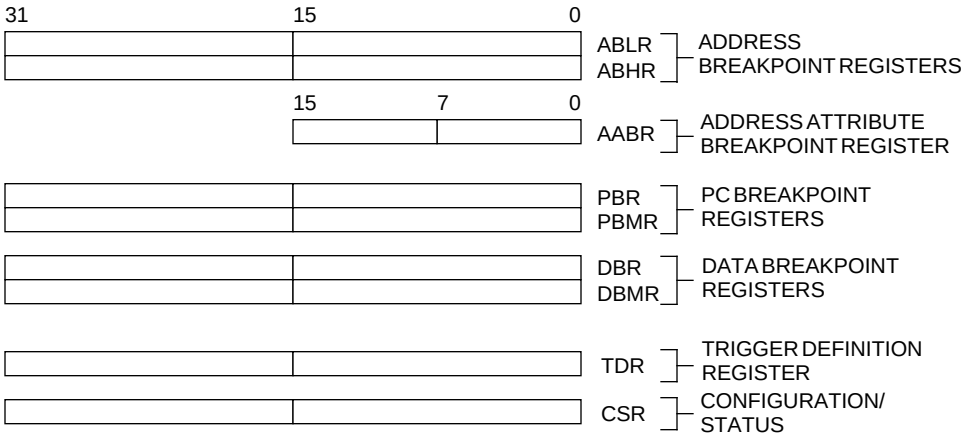


Figure 6-5. Debug Programming Model

and the memory subsystem, the debug module contains a number of 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 (DRc), are addressed using a 4-bit value as part of two new BDM commands (WDREG, RDREG).

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).

Figure 6-5 illustrates the debug module programming model.

6.3.1.1 ADDRESS BREAKPOINT REGISTERS (ABLR, ABHR). The Address Breakpoint Registers define an upper (ABHR) and a lower (ABLR) boundary for a region in the operand logical address space of the processor that can be used as part of the trigger. The ABLR and ABHR values are compared with the ColdFire CPU core address signals, as defined by the setting of the TDR.

6.3.1.2 ADDRESS ATTRIBUTE BREAKPOINT REGISTER (AABR). The Address Attribute Breakpoint Register defines the address attributes and a mask to be matched in the trigger. The AABR value is compared with the ColdFire CPU core address attribute

15	14 13	12 11	10 8	7	6 5	4 3	2 0
RM	SZM	TTM	TMM	R	SZ	TT	TM

AABR Bit Definitions

signals, as defined by the setting of the TDR. The lower 5 bits of the AABR (TT, TM) define the address space used on all BDM memory references. The initial value of the AABR is $0005.

RM–Read/Write Mask

This field corresponds to the R-field. Setting this bit causes R to be ignored in address comparisons.

SZM–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–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–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–Read/Write

This field is compared with the ColdFire CPU core R/W signal. A high level indicates a read cycle and a low level indicates a write cycle.

SZ–Size

This field is compared with the ColdFire CPU core SIZ signals.

SZ—-Size

This field is compared to the ColdFire CPU core SIZ signals. These signals indicate the data size for the bus transfer. Table 6-8 shows the definitions for the SZ encodings.

Table 6-8. SZ Encodings

SZ[1:0]	TRANSFER SIZE
00	Longword (4 bytes)
01	Byte
10	Word (2 bytes)
11	Reserved

TT–Transfer Type

This field is compared with the ColdFire CPU core TT signals. These signals indicate the transfer type for the bus transfer. Table 6-9 shows the definition of the TT encodings.

Table 6-9. Transfer Type Encodings

TT[1:0]	TRANSFER TYPE
00	Normal Access
01	Reserved
10	Alternate and Debug Access
11	Acknowledge Access

TM–Transfer Modifier

This field is compared with the ColdFire CPU core TM signals. These signals provide supplemental information for each transfer type. Table 6-10 shows encodings for normal transfers and Table 6-11 shows the encodings for alternate and debug access transfers.

For interrupt acknowledge transfers the TM signals indicate the interrupt level being acknowledged. For breakpoint acknowledge transfers, the TM signals are low.

Table 6-10. Transfer Modifier Encodings for Normal Transfers

TM[2:0]	TRANSFER MODIFIER
000	Reserved
001	User Data Access
010	User Code Access
011 - 100	Reserved
101	Supervisor Data Access
110	Supervisor Code Access
111	CPU spce - MOVEC Access

Table 6-11. Transfer Modifier Encodings for Alternate Access Transfers

TM[2:0]	TRANSFER MODIFIER
000 - 100, 111	Reserved
101	Emulator Mode Data Access
110	Emulator Mode Code Access

6.3.1.3 PROGRAM COUNTER BREAKPOINT REGISTER (PBR, PBMR). The PC Breakpoint Registers define a region in the instruction 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.

6.3.1.4 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 the ColdFire CPU core data signals, as defined in the TDR.

The data breakpoint register supports both aligned and misaligned operand references. The relationship between the processor core address, the access size, and the corresponding location within the 32-bit core data bus is shown in Table 6-12.

Table 6-12. Core Address, Access Size, and Operand Location

CORE ADDRESS[1:0]	ACCESS SIZE	OPERAND LOCATION
00	Byte	Data[31:24]
01	Byte	Data[23:16]
10	Byte	Data[15:08]
11	Byte	Data[07:00]

Table 6-12. Core Address, Access Size, and Operand Location (Continued)

CORE ADDRESS[1:0]	ACCESS SIZE	OPERAND LOCATION
0-	Word	Data[31:16]
1-	Word	Data[15:00]
-	Word	Data[31:00]

6.3.1.5 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.

Reset clears the TDR.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TRC		EBL	EDLW	EDWL	EDWU	EDLL	EDLM	EDUM	EDUU	DI	EAI	EAR	EAL	EPC	PCI

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
00		EBL	EDLW	EDWL	EDWU	EDLL	EDLM	EDUM	EDUU	DI	EAI	EAR	EAL	EPC	PCI

TDR Bit 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=reserved
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 core data bus (KD) KD[31:0] longword.

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 KD[31:0] longword.

EDWU–Enable Data Breakpoint for the Upper Data Word

If set, this bit enables the data breakpoint trigger based on the KD[31:16] word.

EDLL–Enable Data Breakpoint for the Lower Lower Data Byte

If set, this bit enables the data breakpoint trigger based on the KD[7:0] byte.

EDLM–Enable Data Breakpoint for the Lower Middle Data Byte

If set, this bit enables the data breakpoint trigger based on the KD[15:8] byte.

EDUM–Enable Data Breakpoint for the Upper Middle Data Byte

If set, this bit enables the data breakpoint trigger based on the KD[23:16] byte.

EDUU–Enable Data Breakpoint for the Upper Upper Data Byte

If set, this bit enables the data breakpoint trigger based on the KD[31:24] byte.

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. If this bit is cleared, the PC breakpoint is disabled.

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.

6.3.1.6 CONFIGURATION/STATUS REGISTER (CSR). The Configuration/Status Register 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

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
STATUS				FOF	TRG	HALT	BKPT	RESERVED							IPW

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MAP	TRC	EMU	DDC		UHE	BTB		0	NPL	IPI	SSM	0	0	0	0

Figure 6-6. CSR Bit Definitions

written by the external development system and written by the supervisor programming model.

Status–Breakpoint Status

This 4-bit field defines provides read-only status information concerning the hardware breakpoints. This field is defined as follows:

$0 = no breakpoints enabled
$1 = waiting for level 1 breakpoint
$2 = level 1 breakpoint triggered
$5 = waiting for level 2 breakpoint
$6 = level 2 breakpoint triggered

This breakpoint status is also output on the DDATA port when the bus is not displaying ColdFire CPU core captured data. A write to the TDR resets this field.

FOF–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 and is cleared on a read of the CSR.

TRG–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 on a read from the CSR and is cleared on a read of the CSR.

Halt–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 on a read from the CSR and is cleared on a read of the CSR.

BKPT–BKPT Assert

If this read-only status bit is set, the $\overline{\text{BKPT}}$ signal was asserted, forcing the processor into BDM. This bit is cleared on a read from the CSR and is cleared on a read of the CSR.

IPW–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–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 = 10, TM = 101 (data) and 110 (text). If cleared, all emulator-mode references are mapped into supervisor text and data spaces.

TRC–Force Emulation Mode on Trace Exception

If set, this bit forces the processor to enter emulator mode when a trace exception occurs.

EMU–Force Emulation Mode

If set, this bit forces the processor to begin execution in emulator mode. This bit is examined only when $\overline{\text{RSTI}}$ is negated, as the processor begins reset exception processing.

DDC–Debug Data Control

This 2-bit field provides configuration control for capturing operand data for display on the DDATA port. The encodings are:

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.

UHE-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 nonprivileged, supervisor/user instruction

BTB–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

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. This includes JMP and JSR instructions using address register indirect or indexed addressing modes, all RTE and RTS instructions as well as all exception vectors.

NPL–Nonpipelined 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–Ignore Pending Interrupts

If set, this bit forces the processor core to ignore any pending interrupt requests signalled on $\overline{\text{KIPL[2:0]}}$ while executing in single-instruction-step mode.

SSM–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 labeled Reserved or “0” are currently unused and reserved for future use. These bits should always be written as 0.

6.3.2 Theory of Operation

The breakpoint hardware can be configured to respond to triggers in several ways. The preferred 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 6-13.

Table 6-13. 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 = 01, the breakpoint trigger causes the core to halt (as reflected in the PST = $F status). 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. For these breakpoints, trigger reporting is imprecise.

If the processor core cannot be halted, the special debug interrupt can be used. With this configuration, TRC bits of the TDR = 10, the breakpoint trigger is converted into a debug

interrupt to the processor. This interrupt is treated as higher than the nonmaskable level 7 interrupt request. As with all interrupts, it is made pending 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 imprecise.

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. Depending on the state of the MAP bit in the CSR, this mode may force all memory accesses (including the exception stack frame writes and the vector fetch) into a specially mapped address space signalled by TT = 2, TM = {5, 6}. After the standard 8-byte exception stack is created, the processor fetches a unique exception vector (offset $030) 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 completed, the return-from-exception (RTE) instruction is executed and the processor exits emulator mode. The processor status output port provides a unique encoding for emulator mode entry ($D) and exit ($7). 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.

6.3.2.1 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 6-14 identifies the shared hardware structures.

Table 6-14. Shared BDM/Breakpoint Hardware

REGISTER	BDM FUNCTION	BREAKPOINT FUNCTION
AABR	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.

6.3.3 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

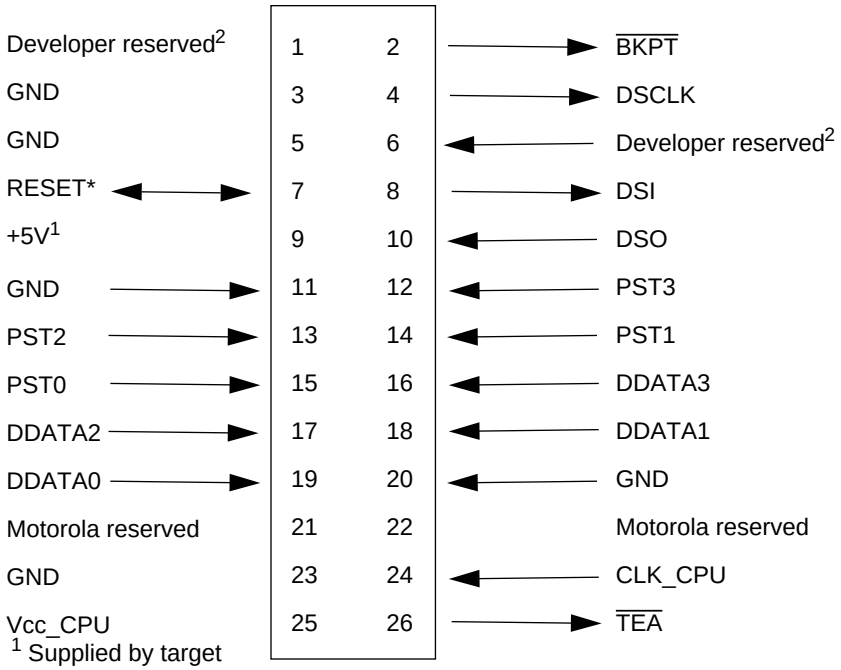
For BDM commands that access memory, the debug module requests the ColdFire core's bus. The processor responds by stalling the instruction fetch pipeline and then waiting until all current core bus activity is complete. At that time, the processor relinquishes the core bus to allow the debug module to perform the required operation. After the conclusion of the debug module core bus cycle, the processor reclaims ownership of the core bus.

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's registers (SDSCLK must be inactive).

6.4 MOTOROLA RECOMMENDED BDM PINOUT

The ColdFire BDM connector is a 26-pin Berg Connector arranged 2x13, shown below.

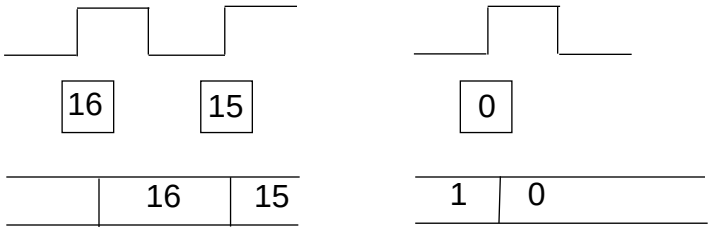


[1] Supplied by target

[2] Pins reserved for BDM developer use. Contact developer.

6.4.1 Differences Between the ColdFire BDM and a CPU32 BDM

1. DSCLK, BKPT, and DSDI 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 below.



SECTION 7
JTAG SPECIFICATION

7.1 IEEE 1149.1 TEST ACCESS PORT (JTAG) SPECIFICATION

The MCF5202 processors include dedicated user-accessible test logic that is fully compliant with the IEEE standard 1149.1 -1993 Standard test access port and boundary- scan architecture.

The following description should be used in conjunction with the supporting IEEE document listed above. This section includes the description of those chip-specific items that the IEEE standard requires as well as those items specific to the MCF5202 implementation.

The MCF5202 JTAG test architecture implementation currently supports circuit board test strategies that are based on the IEEE standard. This architecture provides access to all of the data and chip control pins from the board edge connector through the standard four-pin test access port (TAP) and the active-low JTAG reset pin, $\overline{\text{TRST}}$. The test logic itself uses a static design and is wholly independent of the system logic, except where the JTAG is subordinate to other complimentary test modes (used for manufacturing test support). When in subordinate mode, the JTAG test logic is placed in reset and the TAP pins can be used for other purposes in accordance with the rules and restrictions set forth using a JTAG compliance-enable pin.

The MCF5202 JTAG implementation can

- Perform boundary-scan operations to test circuit board electrical continuity
- Bypass the MCF5202 device by reducing the Shift Register path to a single cell
- Sample the MCF5202 system pins during operation and transparently shift out the result
- Set the MCF5202 output drive pins to fixed logic values while reducing the Shift Register path to a single cell
- Protect the MCF5202 system output and input pins from backdriving and random toggling (such as during in-circuit testing) by placing all system signal pins to high impedance

The IEEE Standard 1149.1 test logic cannot be considered completely benign to those planning not to use JTAG capability. Certain precautions must be observed to ensure that this logic does not interfere with system or debug operation. Figure 7-1 shows pin logical

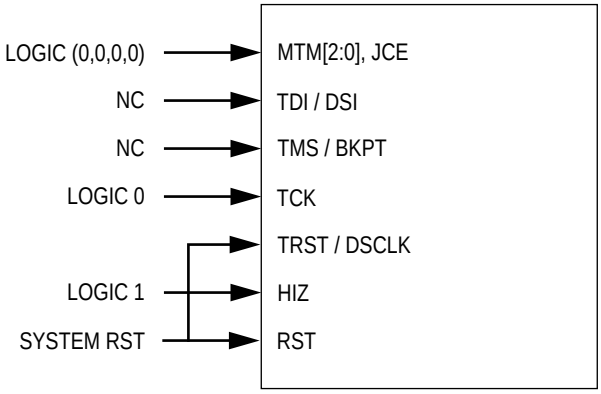


Figure 7-1. JTAG Mode, JTAG Disabled

values recommended for disabling JTAG with the part in JTAG mode.

While in JTAG mode, input pins TDI / DSI, TMS / $\overline{\text{BKPT}}$, and $\overline{\text{TRST}}$ / DSCLK have internal pullups enabled. Figure 7-2 shows pin logical values recommended for disabling JTAG with the part in Background Debug Mode.

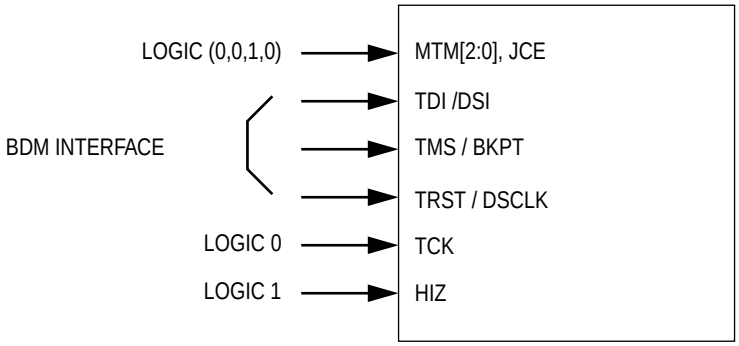


Figure 7-2. Background Debug Mode, JTAG disabled

7.2 OVERVIEW

Figure 7-3 illustrates the block diagram or programmer's model of the MCF5202 implementation of the 1149.1 IEEE Standard. The test logic includes several Test Data Registers, an Instruction Register, instruction register control decode, and a 16-state dedicated TAP controller (defined in Figure 7-3).

7.2.1 JTAG Pin Descriptions

Pins MTM[2:0](Motorola Test Mode pins), and $\overline{\text{JCE}}$ are defined to be compliance-enable inputs per section 3.8 of the IEEE Standard 1149.1a-1993 entitled Subordination of this Standard within a Higher Level Test Strategy. The compliance-enable pattern is {(0,0,0), (0)}.

Given the compliance-enable state described above, the following pin descriptions apply:

TCK - A test clock input that synchronizes test logic operations

TMS - A test mode select input with a default internal pullup resistor that is sampled on the rising edge of TCK to sequence the TAP controller

TDI - A serial test data input with a default internal pullup resistor that is sampled on the rising edge of TCK

TDO - A three-state test data output that is actively driven only in the Shift-IR and Shift-DR controller states and only updates on the falling edge of TCK

$\overline{\text{TRST}}$ - An active-low asynchronous reset with a default internal pullup resistor that forces the TAP controller into the test-logic-reset state.

As Figure 7-3 reveals, the test logic includes a 3-bit Instruction Shift Register, a 3-bit Instruction Decode Register, a Boundary Scan Register, a single-bit Bypass Register, and a TAP controller.

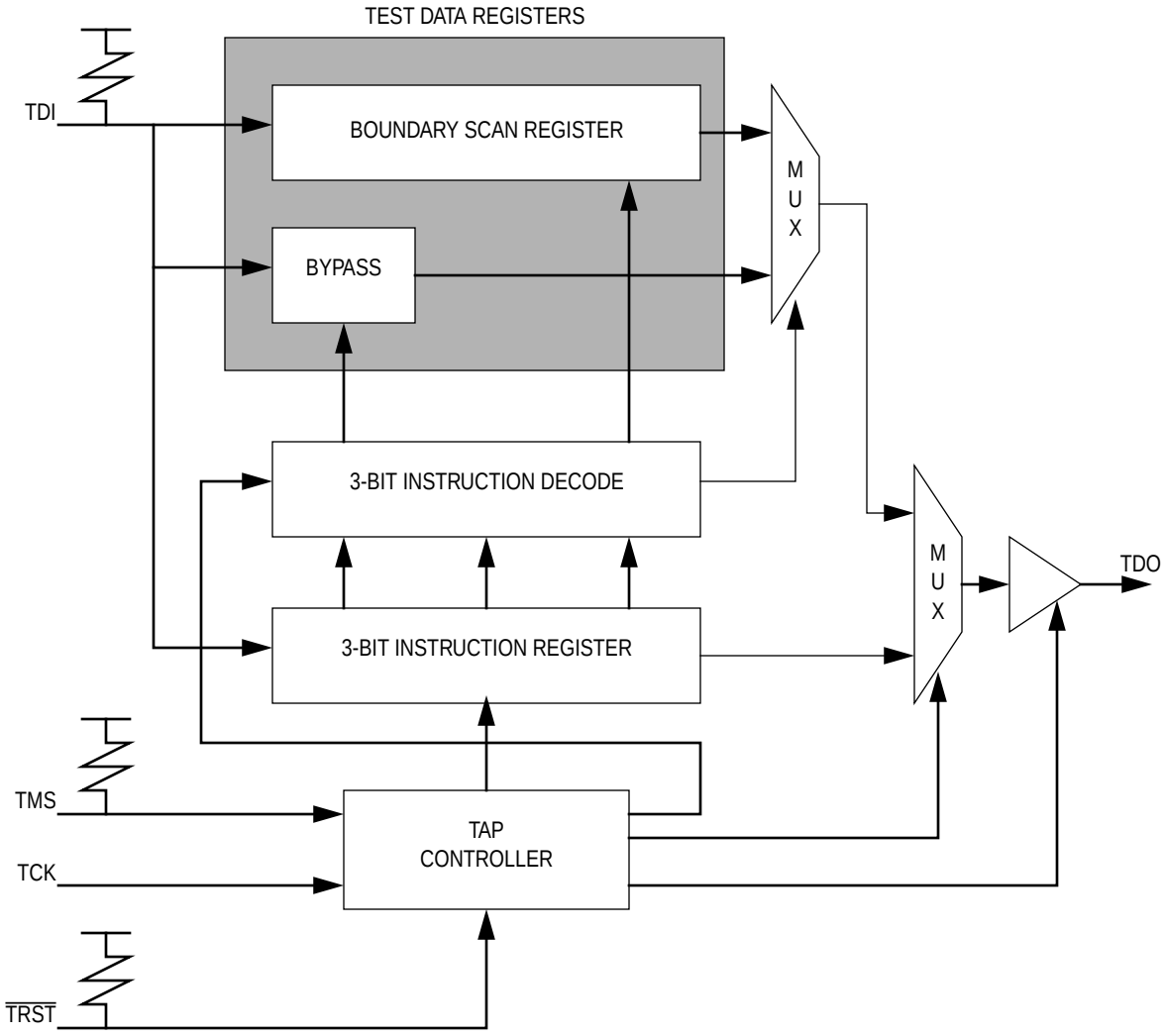


Figure 7-3. JTAG Test Logic Block Diagram

7.3 JTAG REGISTER DESCRIPTION

7.3.1 JTAG Instruction Shift Register

The MCF5202 IEEE 1149.1 implementation uses a 3-bit Instruction Shift Register without parity. This register transfers its value to a parallel hold register and applies one of eight possible instructions on the falling edge of TCK when the TAP state machine is in the update-IR state. The instructions can be loaded into the shift portion of the register by placing the serial data on the TDI pin prior to each rising edge of TCK. The MSB of the Instruction Shift Register is the bit closest to the TDI pin and the LSB is the bit closest to the TDO pin.

The public customer-usable instructions supported are listed with their encoding in Table 7-1.

Table 7-1. JTAG Instructions

INSTRUCTION	ABBR	CLASS	IR[2:0]	INSTRUCTION SUMMARY
EXTEST	EXT	Required	000	Select bs register while applying fixed values to output pins and asserting functional reset
SAMPLE/ PRELOAD	SMP	Required	100	Selects bs register for shift, sample and preload without disturbing functional operation
HIGHZ	HIZ	Optional	101	Selects the bypass register while three-stating all output pins and asserting functional reset
CLAMP	CMP	Optional	110	Selects bypass while applying fixed values to output pins and asserting functional reset
BYPASS	BYP	Required	111	Selects the bypass register for data operations

The IEEE 1149.1 requires the EXTEST, SAMPLE/PRELOAD, and BYPASS instructions. CLAMP and HIGHZ are optional standard instructions that are supported by the MCF5202 implementation and are described in the 1149.1.

7.3.1.1 EXTEST INSTRUCTION. The external test instruction (EXTEST) selects the Boundary Scan Register. The EXTEST instruction forces all output pins and bidirectional pins configured as outputs to the fixed values that are preloaded (with the SAMPLE/ PRELOAD instruction) and held in the Boundary Scan Update Registers. The EXTEST instruction can also configure the direction of bidirectional pins and establish high-impedance states on some pins. The EXTEST instruction becomes active on the falling edge of TCK in the update-IR state when the data held in the Instruction Shift Register is equivalent to octal 0.

7.3.1.2 SAMPLE/PRELOAD INSTRUCTION. The SAMPLE/PRELOAD instruction provides two separate functions. First, it provides a way to obtain a sample of the system data and control signals present at the MCF5202 input pins and just prior to the boundary scan cell at the output pins. This sampling occurs on the rising edge of TCK in the capture-DR state when an instruction encoding of octal 4 is resident in the instruction register. Users can observe this sampled data by shifting it through the Boundary Scan Register to the output TDO by using the shift-DR state. Both the data capture and the shift operation are transparent to system operation. Users are responsible for providing some form of external synchronization to achieve meaningful results because there is no internal synchronization between TCK and the system clock, CLK.

The second function of the SAMPLE/PRELOAD instruction is to initialize the Boundary Scan Register update cells before selecting EXTEST or CLAMP. This is achieved by ignoring the data being shifted out of the TDO pin while shifting in initialization data. The update-DR state in conjunction with the falling edge of TCK can then transfer this data to the update cells. This data will be applied to the external output pins when one of the instructions listed above is applied.

7.3.1.3 HIGHZ INSTRUCTION. The HIGHZ instruction anticipates the need to backdrive the output pins and protect the input pins from random toggling during circuit-board testing. The HIGHZ instruction selects the Bypass Register, forces all output and bidirectional pins to the high-impedance state.

The HIGHZ instruction becomes active on the falling edge of TCK in the update-IR state when the data held in the Instruction Shift Register is equivalent to octal 5.

7.3.1.4 CLAMP INSTRUCTION. The CLAMP instruction selects the Bypass Register and asserts functional reset while simultaneously forcing all output pins and bidirectional pins configured as outputs to the fixed values that are preloaded and held in the boundary scan update registers. This instruction enhances test efficiency by reducing the overall shift path to a single bit (the Bypass Register) while conducting an EXTEST type of instruction through the Boundary Scan Register.

The CLAMP instruction becomes active on the falling edge of TCK in the update-IR state when the data held in the Instruction Shift Register is equivalent to octal 6.

7.3.1.5 BYPASS INSTRUCTION. The BYPASS instruction selects the single-bit Bypass Register, creating a single-bit Shift Register path from the TDI pin to the bypass register to the TDO pin. This instruction enhances test efficiency by reducing the overall shift path when a device other than the MCF5202 processor becomes the device under test on a board design with multiple chips on the overall 1149.1 defined boundary scan chain. The Bypass Register has been implemented in accordance with 1149.1 so that the Shift Register stage is set to logic zero on the rising edge of TCK following entry into the capture-DR state. Therefore, the first bit to be shifted out after selecting the Bypass Register is always a logic zero (this is to differentiate a part that supports an IDCODE register from a part that supports only the Bypass Register). The BYPASS instruction becomes active on the falling edge of TCK in the update-IR state when the data held in the Instruction Shift Register is equivalent to a octal 7.

7.3.2 JTAG BOUNDARY SCAN REGISTER

An IEEE 1149.1-compliant Boundary Scan Register has been included on the MCF5202 model. This Boundary Scan Register can be connected between TDI and TDO when the EXTEST or SAMPLE/PRELOAD instructions are selected. This register captures signal pin data on the input pins, forces fixed values on the output signal pins, and selects the direction and drive characteristics (a logic value or high impedance) of the bidirectional and three-state signal pins.

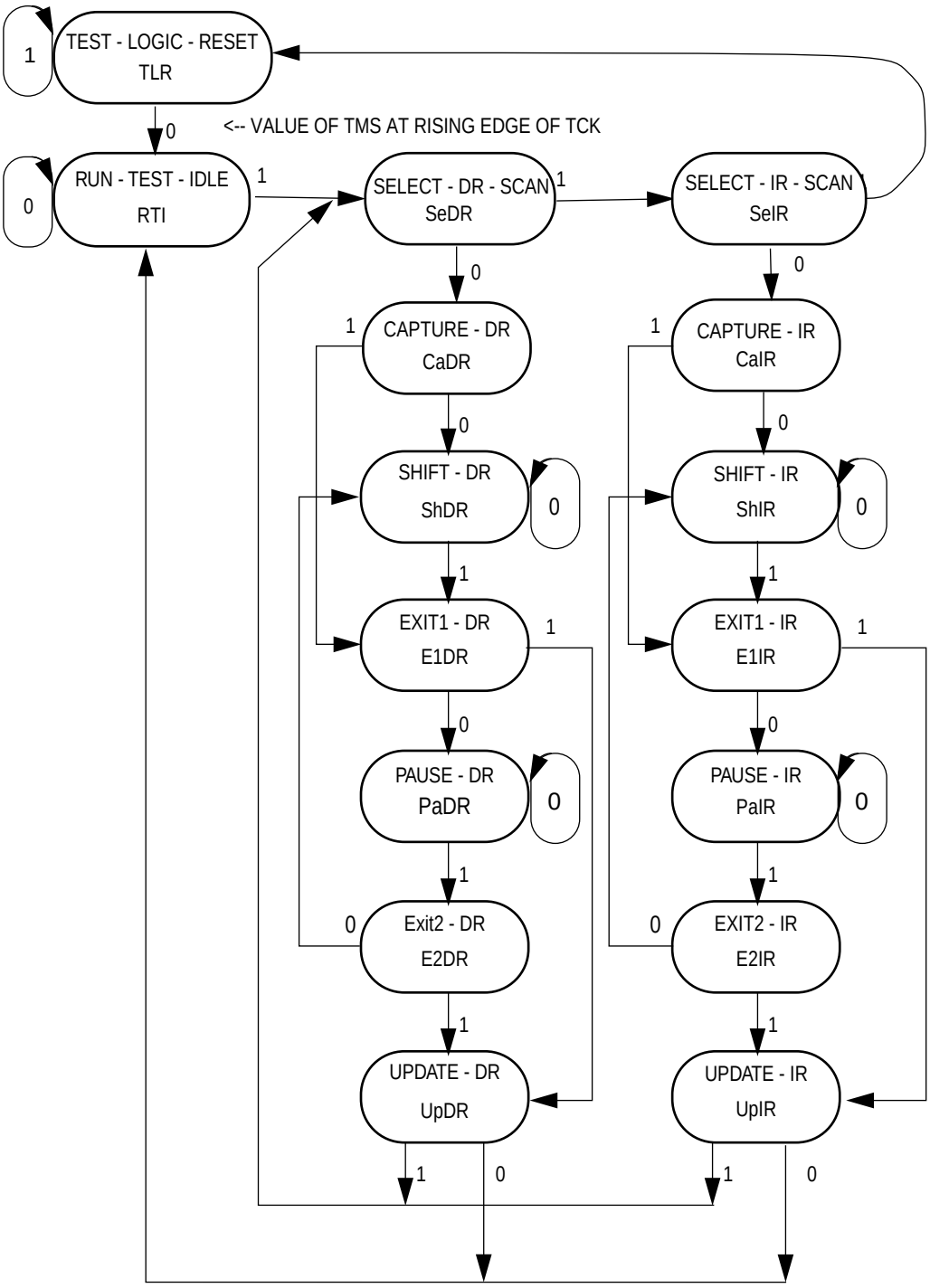


Figure 7-4. JTAG TAP Controller State Machine

7.3.3 JTAG BYPASS REGISTER

The MCF5202 includes an IEEE 1149.1-compliant Bypass Register, which creates a single bit shift register path from TDI to the Bypass Register to TDO when the BYPASS instruction is selected.

The 16 controller states are defined in detail in the IEEE 1149.1 standard.

For more information on the MCF5202, visit the following Internet address:

http://pirs.aus.sps.mot.com/aesop/aesop.html

SECTION 8
PORTING FROM M68K ARCHITECTURE

This section is an overview of the issues encountered when porting embedded development tools to work with the ColdFire processor when starting with the M68K architecture.

8.1 C COMPILERS AND HOST SOFTWARE

For the purpose of this discussion, it is assumed that an embedded software development tool chain consists of a “host” portion and a “target” portion. The host portion consists of tool chain parts that execute on a desktop computer or workstation. The target portion of the tool chain runs ColdFire executables on a physical ColdFire target board.

Compilers, assemblers, linkers, loaders, instruction set simulators, and the host portion of debuggers are examples of host tools. Many host tools such as linkers and loaders that work with the M68K architecture can also be used without modification with ColdFire.

Although an existing M68K assembler and disassembler can be used with ColdFire, Motorola recommends modifying the assembler so that nonColdFire assembly code cannot be put together in the executable. This is especially true if the assembler assembles handwritten code. Porting the disassembler is for convenience and can be performed later.

Debuggers usually are comprised of two parts. A host portion of the debugger typically issues higher level commands for the target portion of debugger target. The target portion of the debugger typically handles the exact details of the implementation of tracing, breakpoints, and other lower-level details. The debugger host portion requires little modification. Most likely, the only architectural items of concern are

- Differences in the designed supervisor registers and stack pointers (for displaying registers)
- Interpretation of exception stack frames (if not already performed by the target portion)

8.2 TARGET SOFTWARE PORT

After the compiler and assembler have been ported, porting ROM monitors and operating systems can begin. For example, consider the steps involved in porting a ROM debugger. Similar issues are encountered when porting an RTOS and target applications.

It is assumed that target software consists primarily of C and assembly source code. The first step is to create executables that will run on existing M68K hardware to test the conversion from M68K code to the proper ColdFire subset. This step verifies that the process of code conversion does not introduce new errors.

To generate a ColdFire executable of the target debugger, use a ColdFire-compliant port of the same C compiler originally used to create the M68K debugger target. This procedure prevents differences in calling convention and parameter passing from C to handwritten assembly. Another advantage to this approach is that special C flags are retained. Many C compilers have special extensions as well.

Whatever approach is used, the assembly language lines that are outside the ColdFire instruction set must be identified. Any ColdFire assembler that properly flags nonColdFire instructions can be used. During the process of conversion, architectural issues can be ignored temporarily because the target is still an M68K.

Once the instruction set differences have been resolved, the architectural differences between the ColdFire and M68K need to be addressed.

8.3 INITIALIZATION CODE

The target software and firmware often execute code that identifies the type of processor. Such a process provides one port that works with various M68K Family members and implementations. The easiest way to identify the ColdFire architecture from other M68K processors is to execute an ILLEGAL opcode ($4afc). This execution generates an exception stack frame while ensuring that the tracing is disabled. The first two bits of the exception stack frame would immediately determine whether the processor is a ColdFire processor.

Motorola suggests that ColdFire architecture testing be performed immediately to avoid having to execute potentially undefined opcodes in the ColdFire architecture. Unused opcodes in the ColdFire architecture are not guaranteed to result in an illegal instruction exception.

Another item to consider is that the ColdFire architecture will have integrated versions with modules yet to be defined. It may be a good idea to ensure that there are enough hooks to allow for initialization of routines.

8.4 EXCEPTION HANDLERS

When dealing with exceptions in debug-oriented software, it is often necessary to extract exception stack information to obtain the SR and PC. The format word (MC68010 and higher) is typically used by generalized exception routines. Their sole purpose is to catch unexpected exceptions and to easily use vector information to identify the cause of the exception. The MC68000 exception frame is different from that of other members of the M68K processors in that there is no notion of a format word. This difference would have forced target software to deal with exception stack frame differences already. The approach now in use provides guidance on handling ColdFire exception stack frame differences. In many low-level exception handlers, the extraction of the stacked SR, PC, or format word is performed in a common source file or the offsets are handled in some type of header file.

Interrupt handlers probably require no modification because in most cases, an interrupt occurs asynchronously with respect to normal program flow. Therefore, interrupt handlers cannot rely on items on the stack as it is often unnecessary to know exactly what was happening at the time of the interrupt.

System calls are often implemented by using the TRAP instruction. For trap exceptions, parameter passing is performed through data and address registers; rarely, if ever, directly through the stack. In addition, a system call typically does not need to know the stacked SR or PC information.

Breakpoints are usually implemented with the TRAP instruction or an illegal instruction such as an $A-line exception. If so, the stacked SR and PC are typically used. Other items in the stack may also need to be queried, especially if the breakpoint displays a stack trace. If so, users should examine the format closely for stack misalignments at the time of the breakpoint. This stack misalignment check would be useful in applications where stack alignment is a software design goal. These same concerns for the breakpoint implementation are applicable to trace exceptions as well.

A generalized exception handler can be implemented to catch unexpected exceptions. In addition to the SR and PC information, it is often necessary to obtain the vector information in the stack. Otherwise, the issues are similar to those found on breakpoints and tracing.

To port the ColdFire access error exception, it is best to start with an MC68000 bus error handler. The ColdFire access error recovery sequence has many similarities to the procedure recommended for the MC68000. However, users should be aware that a read bus error on the ColdFire will not advance the program counter to the next instruction. In addition, a write bus error may be taken long after an instruction has been executed and the stacked program counter may not point to the offending instruction.

The main cause of an address error exception in the M68K architecture is that program flow is forced to continue at an odd address boundary. In addition, an MC68000 reports an address error if a data byte access is initiated on an odd address. The ColdFire uses the address error for implementations that do not have the misalignment module, and that a misaligned data access is initiated. Modification of the address error exception handler to reflect a ColdFire misalignment exception is optional. The MCF5202 contains hardware support for data misalignment and therefore this is not an issue for family members.

On a ROM monitor, it is often necessary to provide a means by which a user program is executed given a certain starting address. This is often implemented by placing an exception stack frame and then performing an RTE. If this is the case, the header files that define what a stack frame looks like would require modification to reflect the ColdFire stack frame format.

8.5 SUPERVISOR REGISTERS

The target software would eventually need to communicate the contents of registers to the host software. Both the host portions and target portions of a debugger must be modified to reflect the single stack pointer architecture of ColdFire. In addition, the target debugger must keep a copy of the MOVEC register images in memory so that it can provide the host software register contents when asked to do so. A UNIX grep utility can find all instances of the MOVEC instruction and perform the appropriate modifications to accommodate the unidirectional MOVEC instruction.

The ColdFire architecture does not distinguish between a supervisor stack and a user stack. There is only a single stack pointer, A7. One way of dealing with this issue is to emulate the dual stacks by placing some code at the beginning and end portions of exception handlers to change the stack pointer contents, if necessary, during exceptions. This approach has the disadvantage that interrupt latency would be degraded because interrupts would have to be disabled during the stack-swapping process, but enable full flexibility of the 68K stack model.

Date: 7-28-98
Revision number: 0.1
Pages affected: see change bars

SECTION 9
ELECTRICAL CHARACTERISTICS

9.1 MAXIMUM RATINGS

Table 9-1. Maximum Ratings

RATING	SYMBOL	VALUE	UNIT
Supply voltage	V_{CC}	-0.3 to +7.0	V
Input voltage	V_{in}	-0.5 to V_{CC} + 0.5V	V
Storage temperature range	T_{stg}	-55 to 150	°C

The ratings in Table 9-1define maximum conditions under which the device will operate without being damaged.

This device contains circuitry that protects against damage from high static voltages or electrical fields; however, users should take normal precautions to avoid application of any voltages higher than maximum-rated voltages to this high-impedance circuit. Operational reliability improves when unused inputs are tied to an appropriate logic voltage level (e.g., either GND or V_{CC}).

Table 9-2. Operating Environment

CHARACTERISTIC	SYMBOL	VALUE	UNIT
Maximum operating junction temperature	T_J	105	°C
Maximum operating ambient temperature	-	70[a]	°C
Minimum operating ambient temperature	T_A	0	°C

a This published maximum operating ambient temperature should be used only as a system design guideline. All device operating parameters are guaranteed only when the junction temperature lies within the specified range.

NOTE

At press time, accurate, reliable power dissipation figures were not available. Please refer to the World Wide Web site at http://pirs.aus.sps.mot.com for the latest accurate power dissipation information for the MCF5202 processor.

.

Table 9-3. Thermal Characteristics

CHARACTERISTIC	SYMBOL[b]	VALUE	RATING
Thermal resistance, junction to ambient Thermal resistance, junction to top reference	θ_{ja} Ψ_{jt}	53 2.7	^{o}C/W ^{o}C/W

[b] $\theta_{ja}$and Ψ_{jt} parameters are measured in accordance with EIA/JEDEC Standard 51-2 for natural convection. Motorola recommends the use of θ_{ja} and power dissipation specifications in the system design to prevent device junction temperatures from exceeding the rated specification. System designers should be aware that device junction temperatures can be significantly influenced by the board layout and surrounding devices. Conformance to the device junction temperature specification can be verified by physical measurement in the customers' system using the Ψ_{jt} parameter, the device power dissipation, and the method described in EIA/JEDEC Standard 51-2.

9.2 CLOCK INPUT SPECIFICATION

Table 9-4. Clock Input Specifications

NUM	CHARACTERISTIC	16.67 MHz		25 MHz		33 MHz		UNIT
		MIN	MAX	MIN	MAX	MIN	MAX	
1	CLK cycle time	60	—	40	—	30	—	ns
2[a]	CLK rise time	—	2	—	2	—	2	ns
3[a]	CLK fall time	—	2	—	2	—	2	ns
4	CLK duty cycle measured at 1.5 V	40	—	40	—	40	—	%
4a	CLK pulse width high measured at 1.5 V	24	—	16	—	12	—	ns
4b	CLK pulse width low measured at 1.5 V	24	—	16	—	12	—	ns

a. Specification values not tested

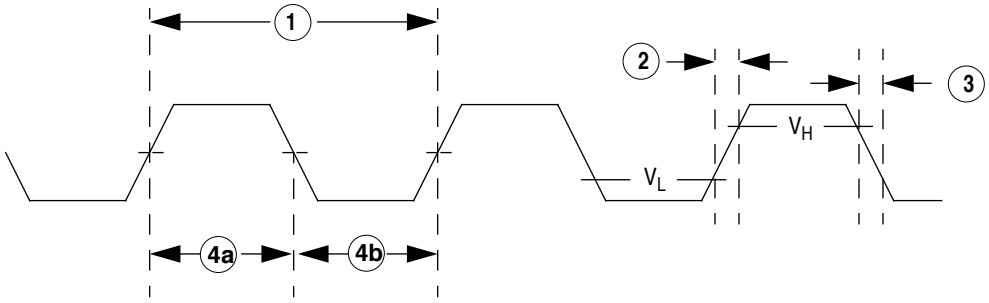


Figure 9-1. Clock Input Timing

9.3 DC ELECTRICAL SPECIFICATIONS

Table 9-5. DC Electrical Specifications (V_{CC} = 5.0 VDC ±5%)

CHARACTERISTIC	SYMBOL	MIN	MAX	UNIT
Input high voltage	V_{IH}	2	VCC	V
Input low voltage	V_{IL}	GND	0.8	V
Input leakage current @ GND, V_{CC}	I_{in}	—	20	uA
HI-Z (three-state) leakage current @ GND, V_{CC}	I_{TSI}	—	20	uA
Output high voltage, I_{OH} = 5 mA	V_{OH}	2.4	—	V
Output low voltage, I_{OL} = 5 mA	V_{OL}	—	0.5	V
Pin capacitance[a]	C_{in}	—	10	pF

a. This specification periodically sampled but not 100% tested.

9.4 OUTPUT AC TIMING SPECIFICATIONS

Table 9-6. Output AC Timing Specifications[a]

NUM	CHARACTERISTIC	16.67 MHz		25 MHz		33 MHz		UNITS
		MIN	MAX	MIN	MAX	MIN	MAX	
10	CLK to $\overline{TS}$, R/$\overline{W}$, SIZ, TT, ATM, $\overline{DTIP}$, $\overline{BR}$, $\overline{BD}$, PST, DDATA, DSO valid	—	30	—	25	—	20	ns
11	CLK to $\overline{TS}$, R/$\overline{W}$, SIZ, TT, ATM, $\overline{DTIP}$, $\overline{BR}$, $\overline{BD}$, PST, DDATA. DSO Invalid (hold)	5	—	5	—	5	—	ns
12	CLK to $\overline{TS}$, R/$\overline{W}$, SIZ, TT, ATM, $\overline{DTIP}$ high impedence	—	30	—	25	—	20	ns
13	CLK to A/D-Out valid	—	30	—	25	—	20	ns
14	CLK to A/D-OUT invalid (hold)	5	—	5	—	5	—	ns
15	CLK to A/D-OUT high impedence	—	30	—	25	—	20	ns
21[b]	$\overline{HIZ}$ to outputs high impedence ($\overline{HIZ}$ asserted)	—	60	—	60	—	60	ns
22[b]	$\overline{HIZ}$ to outputs valid ($\overline{HIZ}$ negated)	—	60	—	60	—	60	ns

a. Output timing is measured at the pin. These specifications assume a capacitive load of 50 pF.
b. Specification values not tested.

9.5 INPUT AC TIMING SPECIFICATIONS

Table 9-7. Input AC Timing Specifications

NUM	CHARACTERISTIC	16.67 MHz		25 MHz		33 MHz		UNITS
		MIN	MAX	MIN	MAX	MIN	MAX	
16	A/D - valid to CLK (setup)	10	—	5	—	5	—	ns
17	CLK to A/D - invalid (hold)	5	—	3	—	3	—	ns
18	CLK to A/D - high impedence	—	50	—	30	—	25	ns
19[a]	$\overline{DA}$, $\overline{TEA}$, $\overline{TBI}$, $\overline{AVEC}$, $\overline{BG}$, $\overline{AA}$, $\overline{IPL}$, $\overline{RST}$, $\overline{BKPT}$, DSCLK[b], DSI valid to CLK (setup)	15	—	8	—	8	—	ns
20	CLK to $\overline{DA}$, $\overline{TEA}$, $\overline{TBI}$, $\overline{AVEC}$, $\overline{BG}$, $\overline{AA}$, $\overline{IPL}$, $\overline{RST}$, $\overline{BKPT}$, DSCLK, DSI invalid (hold)	5	—	3	—	3	—	ns

a. $\overline{IPL}$ and $\overline{RST}$ are internally synchronized. This setup time must be met only if recognition on a particular clock is required.
b. Maximum frequency of operation for DSCLK is 1/2 the frequency of CLK.

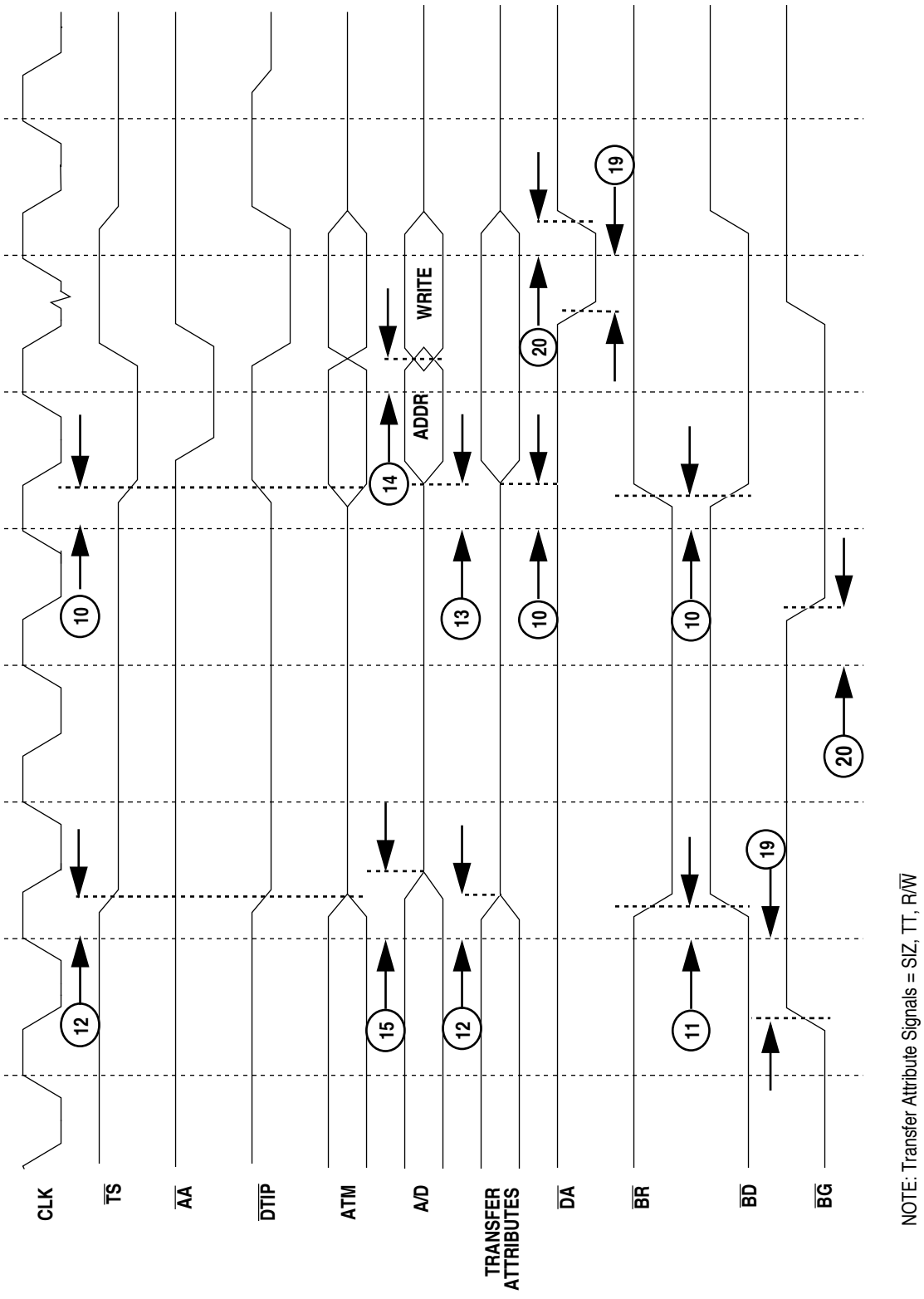


Figure 9-2. Bus Arbitration Timing

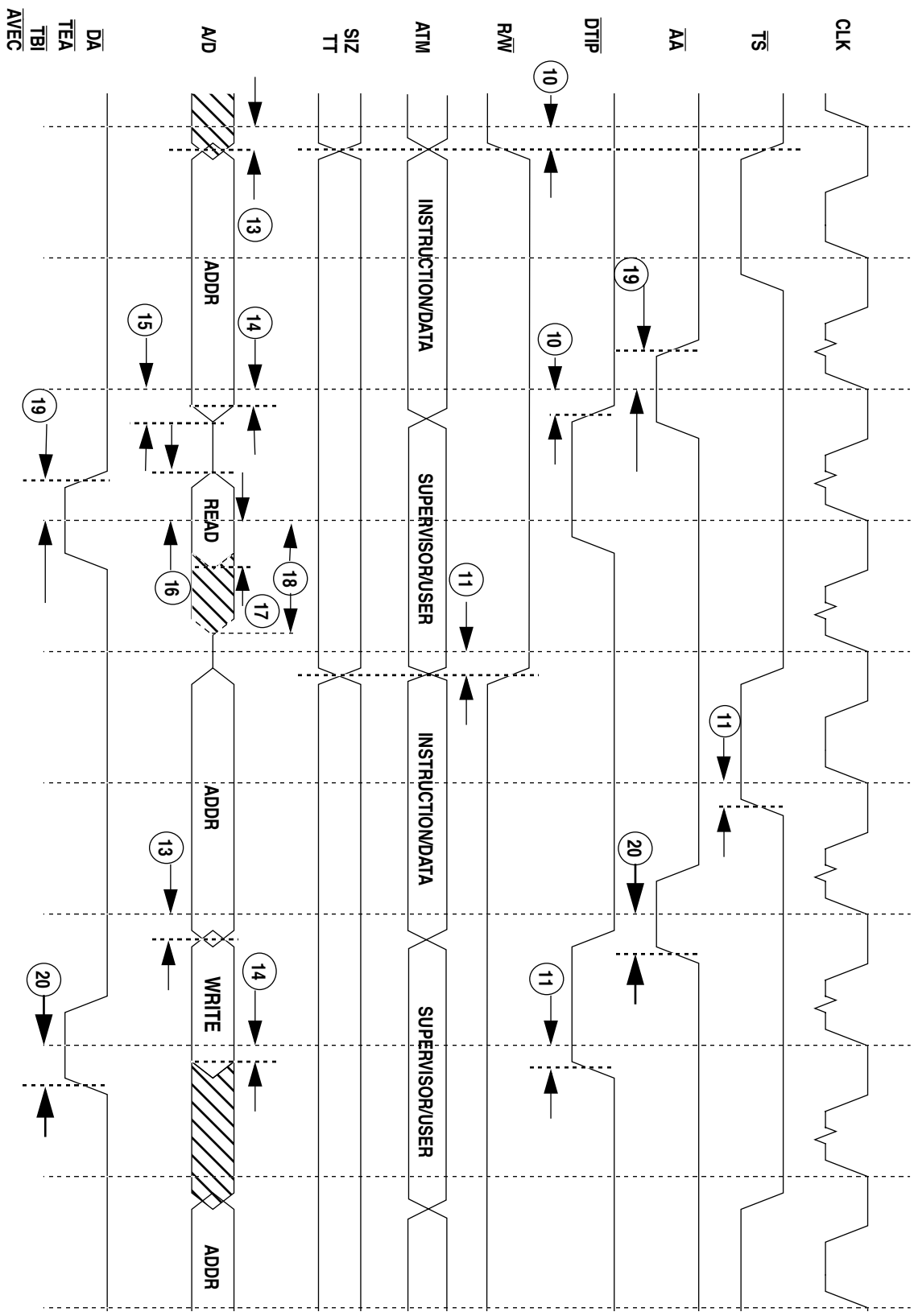


Figure 9-3. Read/Write Timing

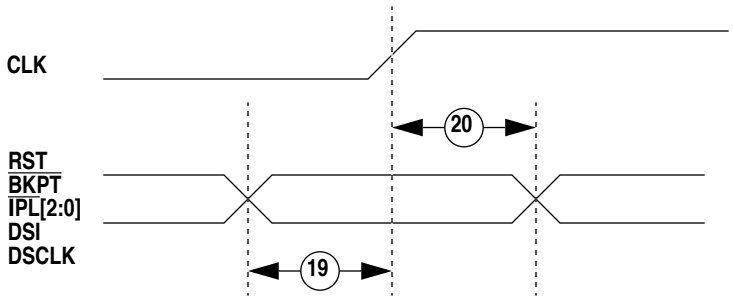


Figure 9-4. Other Signals, Input Timing

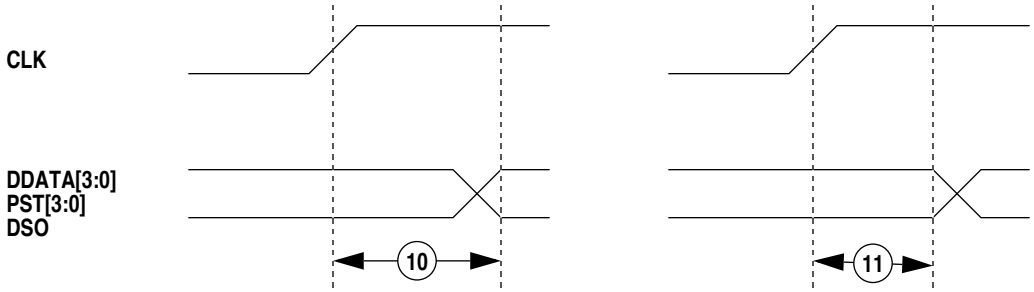


Figure 9-5. Other Signals, Output Timing

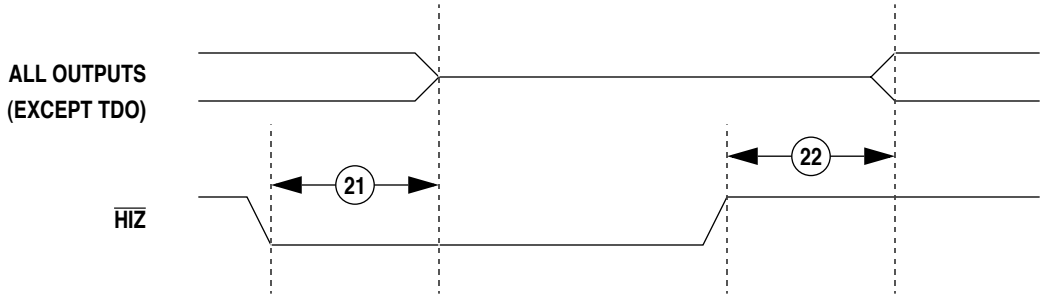


Figure 9-6. $\overline{\text{HIZ}}$ Output Timing

9.6 JTAG AC Timing Specification

Table 9-8. JTAG AC Input Timing Specification

NUM	CHARACTERISTIC	MIN	MAX	UNITS
—	TCK frequency of operation	0	10	MHz
J1	TCK cycle time	100	—	ns
J2	TCK clock pulse width measured at 1.5V	40	—	ns
J3	TCK rise and fall times	—	2	ns
J4	TDI, TMS to TCK rising edge (setup)	10	—	ns
J5	TCK rising edge to TDI, TMS invalid (hold)	15	—	ns
J6	Boundary scan data valid to TCK rising edge (setup)	10	—	ns
J7	Boundary scan data invalid to TCK rising edge (hold)	15	—	ns
J8	TRST pulse width	15	—	ns

Table 9-9. JTAG AC Output Timing Specification

NUM	CHARACTERISTIC	MIN	MAX	UNITS
J9	TCK falling edge to TDO valid	—	30	ns
J10	TCK falling edge to TDO high impedance	—	30	ns
J11	TCK falling edge to boundary scan data valid	—	30	ns
J12	TCK falling edge to boundary scan data high impedance	—	30	ns

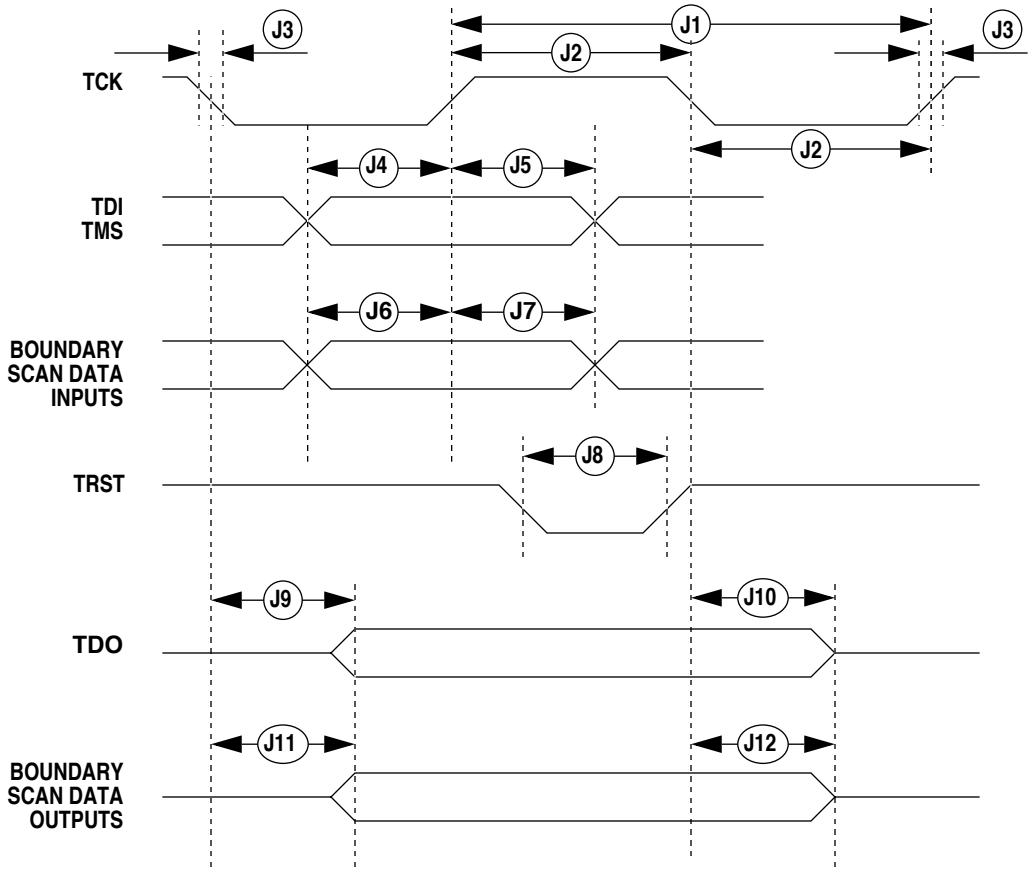


Figure 9-7. JTAG Timing

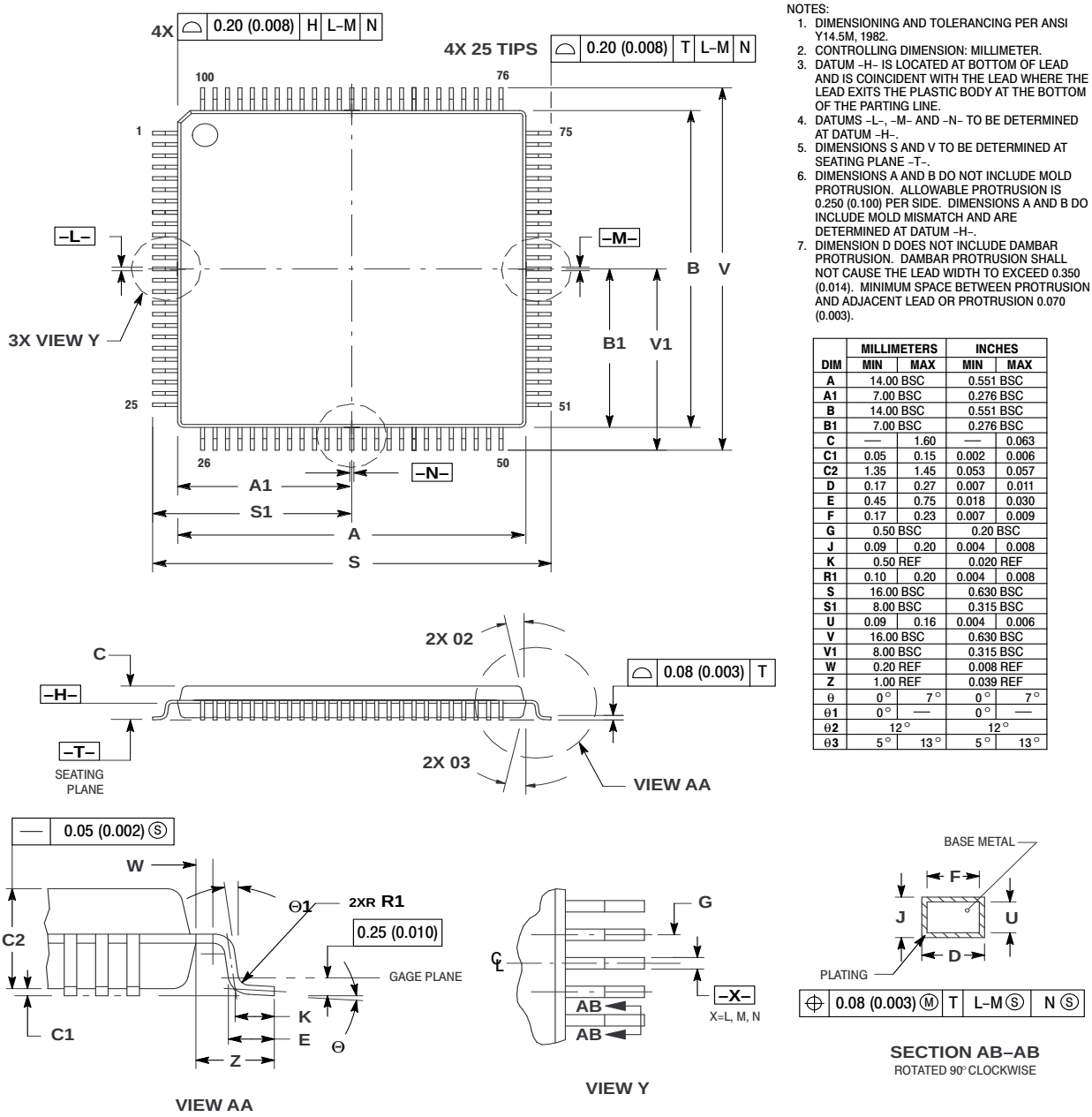
9.7 Power Consumption

The power consumption figures stated are for 5.0 V and 50 pf loads on all pins, room temperature. The code which was used was Dhrystone 2.1. The data is as follows:

Table 9-10. MCF5202 Power Consumption

16MHZ	25MHz	33MHz	UNITS
302	448	589	mW

SECTION 10
MECHANICAL DATA



NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: MILLIMETER.
3. DATUM –H– IS LOCATED AT BOTTOM OF LEAD AND IS COINCIDENT WITH THE LEAD WHERE THE LEAD EXITS THE PLASTIC BODY AT THE BOTTOM OF THE PARTING LINE.
4. DATUMS –L–, –M– AND –N– TO BE DETERMINED AT DATUM –H–.
5. DIMENSIONS S AND V TO BE DETERMINED AT SEATING PLANE –T–.
6. DIMENSIONS A AND B DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE PROTRUSION IS 0.250 (0.100) PER SIDE. DIMENSIONS A AND B DO INCLUDE MOLD MISMATCH AND ARE DETERMINED AT DATUM –H–.
7. DIMENSION D DOES NOT INCLUDE DAMBAR PROTRUSION. DAMBAR PROTRUSION SHALL NOT CAUSE THE LEAD WIDTH TO EXCEED 0.350 (0.014). MINIMUM SPACE BETWEEN PROTRUSION AND ADJACENT LEAD OR PROTRUSION 0.070 (0.003).

	MILLIMETERS		INCHES	
DIM	MIN	MAX	MIN	MAX
A	14.00 BSC		0.551 BSC	
A1	7.00 BSC		0.276 BSC	
B	14.00 BSC		0.551 BSC	
B1	7.00 BSC		0.276 BSC	
C	—	1.60	—	0.063
C1	0.05	0.15	0.002	0.006
C2	1.35	1.45	0.053	0.057
D	0.17	0.27	0.007	0.011
E	0.45	0.75	0.018	0.030
F	0.17	0.23	0.007	0.009
G	0.50 BSC		0.20 BSC	
J	0.09	0.20	0.004	0.008
K	0.50 REF		0.020 REF	
R1	0.10	0.20	0.004	0.008
S	16.00 BSC		0.630 BSC	
S1	8.00 BSC		0.315 BSC	
U	0.09	0.16	0.004	0.006
V	16.00 BSC		0.630 BSC	
V1	8.00 BSC		0.315 BSC	
W	0.20 REF		0.008 REF	
Z	1.00 REF		0.039 REF	
θ	0°	7°	0°	7°
θ1	0°	—	0°	—
θ2	12°		12°	
θ3	5°	13°	5°	13°

Figure 10-1. MC5202 Mechanical Specs

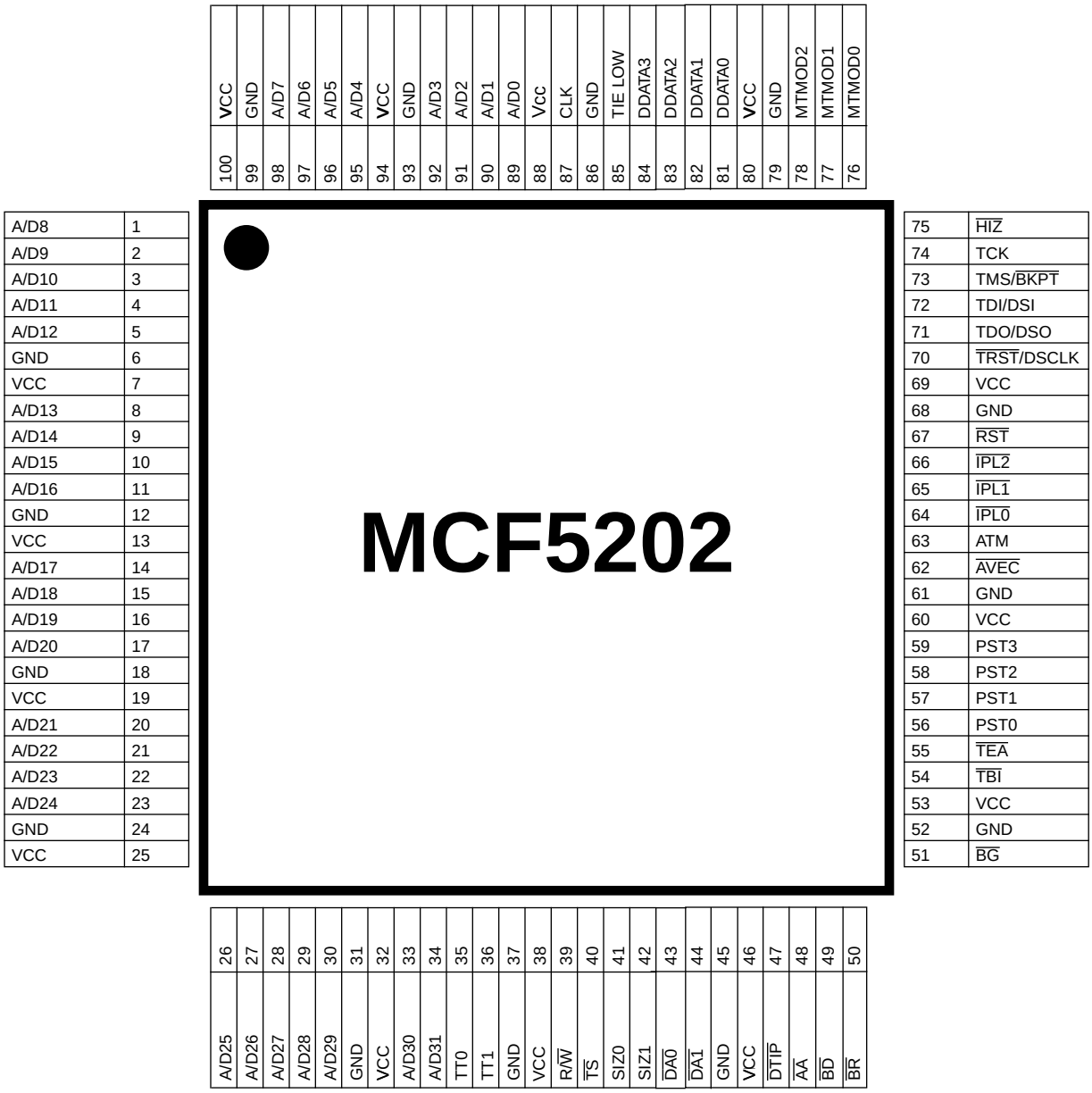


Figure 10-2. MCF5202 Pinout